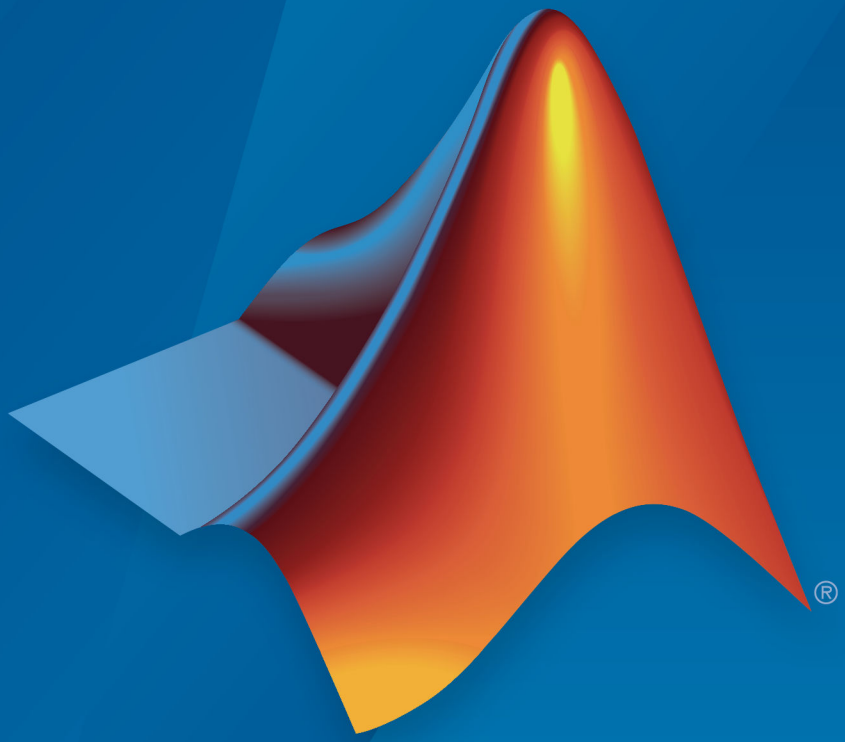
Simscape™

User's Guide



MATLAB® & SIMULINK®

R2018b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

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

Phone: 508-647-7000

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

Simscape™ User's Guide

© COPYRIGHT 2007–2018 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)
October 2015	Online only	Rereleased for Version 3.13.1 (Release 2015aSP1)
March 2016	Online only	Revised for Version 4.0 (Release 2016a)
September 2016	Online only	Revised for Version 4.1 (Release 2016b)
March 2017	Online only	Revised for Version 4.2 (Release 2017a)
September 2017	Online only	Revised for Version 4.3 (Release 2017b)
March 2018	Online only	Revised for Version 4.4 (Release 2018a)
September 2018	Online only	Revised for Version 4.5 (Release 2018b)

Contents

1 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 **1-10**

Simscape Block Libraries **1-12**
Library Structure Overview **1-12**
Accessing 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 **1-17**

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 **1-35**
Grounding Rules **1-35**
Avoiding Numerical Simulation Issues **1-39**

Domain-Specific Line Styles **1-42**

Plot Lookup Tables **1-45**

2 Thermal Liquid Models

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 **2-7**
Why Use Thermal Liquid Blocks? **2-7**
Representing Thermal Liquid Components **2-7**
Specifying Thermal Liquid Medium **2-9**
Modeling Multidomain Systems **2-9**

Thermal Liquid Modeling Framework **2-11**
How Blocks Represent Components **2-11**
How Ports Represent Interfaces **2-12**
Full Flux Scheme **2-13**

Heat Transfer in Insulated Oil Pipeline **2-15**
Oil Pipelines **2-15**
Modeling Considerations **2-16**
Simscape Model **2-18**
Run Simulation **2-20**
Run Optimization Script **2-27**

3 Two-Phase Fluid Models

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

4 Gas System Models

Modeling Gas Systems 4-2
Intended Applications 4-2
Network Variables 4-2
Gas Property Models 4-3
Blocks with Gas Volume 4-5
Reference Node and Grounding Rules 4-6
Initial Conditions for Blocks with Finite Gas Volume 4-6
Choked Flow 4-7
Flow Reversal 4-13

Simple Gas Model 4-14

Change Flow Boundary Conditions 4-20

5 Moist Air System Models

Modeling Moist Air Systems 5-2
Intended Applications 5-2
Network Variables for Moist Air Domain 5-3
Moist Air Properties 5-3
Humidity and Trace Gas Property Definitions 5-5
Blocks with Moist Air Volume 5-8
Reference Node and Grounding Rules 5-8
Initial Conditions for Blocks with Finite Moist Air Volume 5-9
Saturation and Condensation 5-10

Choked Flow 5-12

Modeling Moisture and Trace Gas Levels 5-18
Moist Air Source Domain 5-18
Trace Gas Modeling Options 5-19
Using Moisture and Trace Gas Sources 5-20
Measuring Moisture and Trace Gas Levels 5-21

6 Model Simulation

How Simscape Models Represent Physical Systems 6-2
Representations of Physical Systems 6-2
Differential, Differential-Algebraic, and Algebraic Systems . . . 6-2
Stiffness 6-3
Events and Zero Crossings 6-3
Working with Simscape Representation 6-3
Managing Zero Crossings in Simscape Models 6-4
References 6-5

How Simscape Simulation Works 6-7
Simscape Simulation Phases 6-7
Model Validation 6-9
Network Construction 6-9
Equation Construction 6-10
Initial Conditions Computation 6-10
Transient Initialization 6-11
Transient Solve 6-12

Setting Up Solvers for Physical Models 6-13
About Simulink and Simscape Solvers 6-13
Choosing Simulink and Simscape Solvers 6-13
Harmonizing Simulink and Simscape Solvers 6-15

Important Concepts and Choices in Physical Simulation . . . 6-18
Variable-Step and Fixed-Step Solvers 6-18
Explicit and Implicit Solvers 6-19
Full and Sparse Linear Algebra 6-19
Event Detection and Location 6-19
Unbounded, Bounded, and Fixed-Cost Simulation 6-20

Global and Local Solvers 6-20

Making Optimal Solver Choices for Physical Simulation 6-22
Simulating with Variable Time Step 6-22
Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers 6-22
Simulating with Fixed Cost 6-24
Troubleshooting and Improving Solver Performance 6-24
Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System 6-25

Filtering Input Signals and Providing Time Derivatives 6-27

System Scaling by Nominal Values 6-30
Enable or Disable System Scaling by Nominal Values 6-30
Possible Sources of Nominal Values and Their Evaluation Order 6-31
Specify Nominal Value-Unit Pairs for a Model 6-31
Modify Nominal Values for a Block Variable 6-34

Frequency and Time Simulation Mode 6-37
Speeding Up Model Simulation 6-37
Variable Initialization for Frequency-and-Time Simulation . . . 6-38
Limitations 6-38
Perform Sinusoidal Steady-State Analysis of a Model 6-38

Troubleshooting Simulation Errors 6-45
Troubleshooting Tips and Techniques 6-45
System Configuration Errors 6-46
Numerical Simulation Issues 6-48
Initial Conditions Solve Failure 6-49
Transient Simulation Issues 6-50

Limitations 6-52
Sample Time and Solver Restrictions 6-52
Algebraic Loops 6-52
Unsupported Simulink Tools and Features 6-53
Restricted Simulink Tools 6-54
Simulink Tools Not Compatible with Simscape Blocks 6-55
Code Generation 6-56

7 Variable Initialization and Operating Points

Block-Level Variable Initialization 7-2
Initializing Block Variables for Model Simulation 7-2
Variable Initialization Priority 7-3
Suggested Workflow 7-4

Set Priority and Initial Target for Block Variables 7-5

Initialize Variables for a Mass-Spring-Damper System 7-7

Variable Viewer 7-25
About Variable Viewer 7-25
Advanced Configuration 7-28
Switching Between Tree View and Flat View 7-30
Useful Filtering Techniques 7-32
Saving Viewer Configuration 7-33
Link to Block Diagram 7-33
Interaction with Model Updates and Simulation 7-35

Using Operating Point Data for Model Initialization 7-37
Using Operating Points to Initialize Model Variables 7-37
Suggested Workflow 7-38
Extracting Variable Initialization Data into an Operating Point 7-38
Manipulating Operating Point Data 7-39
Applying Operating Point Data to Initialize Model 7-39

Initialize Model Using Operating Point from Logged Simulation Data 7-42

8 Linearization and Trimming

Finding an Operating Point 8-2
What Is an Operating Point? 8-2
Finding Operating Points in Physical Models 8-3

Linearizing at an Operating Point **8-6**
What Is Linearization? **8-6**
Linearizing a Physical Model **8-8**

Linearize an Electronic Circuit **8-12**
Explore the Model **8-12**
Linearize with Steady-State Solver and linmod Function **8-16**
Linearize with Simulink Control Design Software **8-18**
Use Control System Toolbox Software for Bode Analysis **8-19**

Linearize a Plant Model for Use in Feedback Control Design **8-22**
Explore the Model **8-22**
Trim Using the Controller and Linearize with Simulink linmod Function **8-25**
Linearize with Simulink Control Design Software **8-26**

9 Simscape Run-Time Parameters

About Simscape Run-Time Parameters **9-2**
Limitations **9-3**

Show Simscape Run-Time Parameter Settings **9-5**

Manage Simscape Run-Time Parameters **9-7**
Show Simscape Run-Time Parameter Settings **9-7**
Specify Simscape Run-Time Parameters **9-7**
Set the Default Parameter Behavior for Generated Code **9-8**
Selectively Override the Inline Default Behavior **9-8**

Specify and Change a Simscape Run-Time Parameter **9-10**
Prerequisites **9-10**
Specify a Parameter as Run-Time Configurable **9-10**
Change a Simscape Run-Time Parameter Using Fast Restart **9-11**

Troubleshoot Simscape Run-Time Parameter Issues **9-13**
Simscape Run-Time Parameter Setting Not Visible **9-13**

Simulation Does Not Respond to Simscape Run-Time Parameter Change 9-13
Unsupported Data Types 9-13

How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ 9-15

Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters 9-18
Model Referencing with Run-Time Configurable Parameters 9-18
Code Generation with Run-Time Configurable Parameters . . 9-18
Fast Restart with Simscape Run-Time Parameters 9-19

Decrease Computational Cost by Inlining Simscape Run-Time Parameters 9-21

10 Real-Time Simulation

Model Preparation Objectives 10-3
Obtain Reference Results 10-3
Determine Step Size 10-3
Adjust Model Fidelity or Scope 10-4

Real-Time Model Preparation Workflow 10-6
Prepare Your Model for Real-Time Simulation 10-8
Insufficient Computational Capability for Workflow Completion 10-10

Improving Speed and Accuracy 10-12
Why Speed and Accuracy Matter for Real-Time Simulation 10-12
Balancing Speed and Accuracy 10-13
Eliminating Effects That Require Intensive Computation . . . 10-14
Optimizing Local and Global Solver Configurations 10-15
Upgrading Target Hardware 10-15
Simulating Parts of the System in Parallel 10-15

Determine Step Size 10-17

Increase Simulation Speed Using the Partitioning Solver . . **10-27**
Limitations **10-27**
Options **10-28**
Simulate a Simscape Model Using the Partitioning Solver . . **10-28**

Reduce Computation Costs **10-34**
Data Logging and Monitoring Guidelines **10-34**
Improve Data Logging and Monitoring Efficiency **10-35**
Additional Methods for Reducing Computational Cost **10-38**

Reduce Fast Dynamics **10-40**
Why Reduce Fast Dynamics **10-40**
Frequency-Response Analysis **10-40**
Pole Analysis **10-41**
Linearize the Model **10-42**
Perform Frequency-Response and Pole-Speed Analyses . . . **10-46**
Identify and Eliminate the Sources of Fast Dynamics **10-50**

Reduce Numerical Stiffness **10-63**
Why Reduce Stiffness? **10-63**
Review Reference Results **10-64**
Identify and Modify a Stiff Element **10-66**
Analyze Results **10-68**

Reduce Zero Crossings **10-73**
Why Reduce Zero Crossings? **10-73**
Obtain Reference Results and Plot Simulation Step Size . . . **10-73**
Identify and Modify Elements That Cause Zero Crossings . . **10-78**
Analyze the Results of the Modified Model **10-82**

Partition a Model **10-86**

Manage Model Variants **10-95**
Limitations **10-95**

Fixed-Cost Simulation for Real-Time Viability **10-96**

Real-Time Simulation Workflow **10-98**
Make Your Model Real-Time Viable **10-101**
Insufficient Computational Capability for Real-Time Viability **10-103**

Solvers for Real-Time Simulation **10-104**
Choosing Between Discrete and Continuous Solvers **10-105**
Computational Cost for Continuous Solvers **10-105**
How Numerical Stiffness Affects Solver Choice **10-106**
Using Simscape Local Fixed-Step Solvers **10-107**

Troubleshooting Real-Time Simulation Issues **10-109**
Avoid Computer Overloads and Unacceptable Simulation Results **10-109**
Optimize Real-Time Application Execution Using Simscape Checks **10-109**

Determine System Stiffness **10-111**
Obtain Reference Results **10-111**
Simulate with an Implicit Fixed-Step Solver **10-112**
Simulate with an Explicit Fixed-Step Solver **10-114**
Analyze the Results **10-116**

Estimate Computation Costs **10-118**

Choose Step Size and Number of Iterations **10-121**
Obtain Reference Results **10-122**
Determine Maximum Step Size for Accurate Results **10-124**
Parameterize Global and Local Solver Settings **10-126**
Perform Fixed-Step, Fixed-Cost Simulation **10-127**
Adjust Solver Settings to Improve Accuracy **10-131**

What Is Hardware-In-The-Loop Simulation? **10-138**
Why Perform Hardware-In-The-Loop Simulation? **10-139**

Hardware-In-The-Loop Simulation Workflow **10-142**
Perform Hardware-In-The-Loop Simulation **10-146**
Insufficient Computational Capability for Hardware-In-The-Loop Simulation **10-147**

Code Generation Requirements **10-149**
Hardware Requirements **10-149**
Software Requirements **10-149**

Software and Hardware Configuration **10-151**

Signal and Parameter Visualization and Control **10-153**

Troubleshoot Hardware-in-the-Loop Simulation Issues . . . **10-155**

Generate, Download, and Execute Code **10-157**
Requirements for Building and Executing Simulink Real-Time Applications **10-157**
Create, Build, and Download a Real-Time Application **10-157**
Execute Real-Time Application **10-158**

Check for Target Hardware Overruns **10-160**
Prerequisites **10-160**
About Simulation Overruns **10-160**
Generate Reference Results **10-161**
Configure Model for Deployment **10-162**
Evaluate Task-Execution Time **10-164**
Adjust Step Size Based on Maximum Task-Execution Time **10-166**

Change Parameter Values on Target Hardware **10-170**
Prerequisites **10-170**
Configure the Simscape Model for Deployment **10-170**
Deploy the Model to the Real-Time Target Machine **10-172**
Change Parameters and See Results Using Simulink Real-Time Explorer **10-173**

Requirements for Using Alternative Platforms **10-179**
Hardware Requirements **10-179**
Software Requirements **10-179**

Extending Embedded and Generic Real-Time System Target Files **10-181**

Precompiled Static Libraries **10-183**

Initialization Cost **10-184**

Generate HDL Code Using the Simscape HDL Workflow Advisor **10-185**
Simscape HDL Workflow Advisor Steps **10-185**
Limitations **10-186**
Generate HDL Code for a Simscape Model Using the Simscape HDL Workflow Advisor **10-187**

11 Code Generation

About Code Generation from Simscape Models 11-2

Reasons for Generating Code 11-3

Using Code-Related Products and Features 11-4

How Simscape Code Generation Differs from Simulink 11-5
- Simscape and Simulink Code Generated Separately 11-5
- Compiler and Processor Architecture Requirements 11-5
- Precompiled Libraries Provided for Selected Compilers 11-5
- Simscape Code Reuse Not Supported 11-6
- Tunable Parameters Not Supported 11-6
- Simscape Run-Time Parameter Inlining Override of Global Exceptions 11-6

12 Data Logging

About Simulation Data Logging 12-2
- Suggested Workflows 12-2
- Limitations 12-3

Enable Data Logging for the Whole Model 12-5

Log Data for Selected Blocks Only 12-6

Data Logging Options 12-7

Log and Plot Simulation Data 12-9

Log Simulation Statistics 12-15

Log and View Simulation Data for Selected Blocks 12-20

Log, Navigate, and Plot Simulation Data 12-25

About the Simscape Results Explorer **12-30**
Selecting Nodes to Plot Data **12-30**
Link to MATLAB Session **12-34**
Link to Block Diagram **12-35**

Plot Simulation Data in Different Units **12-39**

Use Custom Units to Plot Simulation Data **12-45**

View Sparkline Plots of Simulation Data **12-49**

Stream Logging Data to Disk **12-56**
Streaming to Disk and parfor Loop **12-57**
Streaming to Disk with parsim **12-58**

Saving and Retrieving Logged Simulation Data **12-60**
Data Logged to Memory **12-61**
Data Logged to Temporary File on Disk **12-61**
Data Recorded in Simulation Data Inspector **12-61**

13 Model Statistics

Simscape Model Statistics **13-2**
Updating the Statistics Viewer **13-3**

1-D Physical System Statistics **13-5**

3-D Multibody System Statistics **13-9**

1-D/3-D Interface Statistics **13-12**

View Model Statistics **13-14**

Access Block Variables Using Statistics Viewer **13-20**

Model Statistics Available when Using the Partitioning Solver **13-25**
Estimate the Memory Budget for Exhaustive Partitioning Storage **13-26**

14 Physical Units

How to Work with Physical Units **14-2**

Unit Definitions **14-4**

How to Specify Units in Block Dialogs **14-11**

Thermal Unit Conversions **14-13**
About Affine Units **14-13**
When to Apply Affine Conversion **14-13**

How to Apply Affine Conversion **14-15**

Angular Units **14-17**
References **14-17**

Units for Angular Velocity and Frequency **14-18**

Working with Simulink Units **14-20**

15 Add-On Product License Management

About the Simscape Editing Mode **15-2**
Suggested Workflows **15-2**
What You Can Do in Restricted Mode **15-3**
What You Can Do in Full Mode **15-4**
Switching Between Modes **15-4**
Working with Block Libraries **15-7**

Set the Model Loading Preference **15-9**

Save a Model in Restricted Mode **15-11**
Example of Saving a Model in Restricted Mode **15-12**

Work with a Model in Restricted Mode **15-15**
How to Simulate and Fine-Tune a Model in Restricted Mode **15-15**
How to Add and Delete Simulink Blocks in Restricted Mode **15-20**
Performing an Operation Disallowed in Restricted Mode . . . **15-25**

Switch from Restricted to Full Mode **15-29**

Get Editing Mode Information **15-31**
What Is the Current Mode? **15-31**
Which Licenses Are Checked Out? **15-31**

Model Construction

- “Basic Principles of Modeling Physical Networks” on page 1-2
- “Connecting Simscape Diagrams to Simulink Sources and Scopes” on page 1-10
- “Simscape Block Libraries” on page 1-12
- “Essential Physical Modeling Techniques” on page 1-15
- “Creating and Simulating a Simple Model” on page 1-18
- “Modeling Best Practices” on page 1-35
- “Domain-Specific Line Styles” on page 1-42
- “Plot Lookup Tables” on page 1-45

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

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.

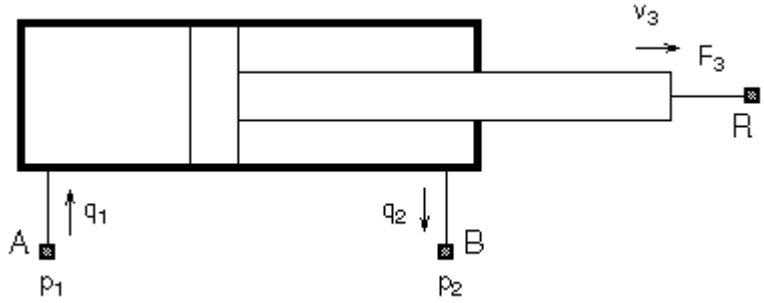
These connection ports are nondirectional. They mimic physical connections between elements. Connecting Simscape blocks together is analogous to connecting real components, such as pumps, valves, and so on. In other words, Simscape diagrams mimic the physical system layout. If physical components can 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:

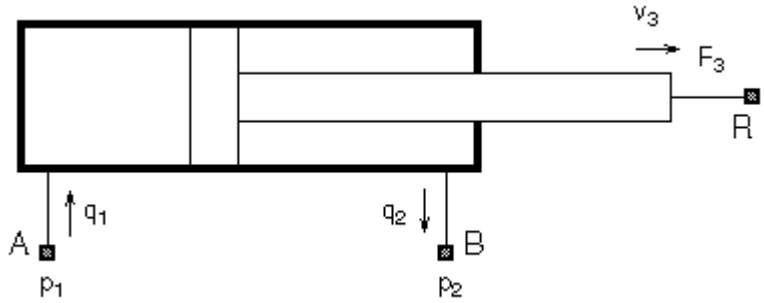
Physical Domain	Across Variable	Through Variable
Electrical	Voltage	Current
Hydraulic	Pressure	Flow rate
Magnetic	Magnetomotive force (mmf)	Flux
Mechanical rotational	Angular velocity	Torque
Mechanical translational	Translational velocity	Force
Gas	Pressure and temperature	Mass flow rate and energy flow rate
Moist Air	Pressure, temperature, specific humidity (water vapor mass fraction), and trace gas mass fraction	Mixture mass flow rate, mixture energy flow rate, water vapor mass flow rate, and trace gas mass flow rate
Thermal	Temperature	Heat flow
Thermal liquid	Pressure and temperature	Mass flow rate and energy flow rate
Two-phase fluid	Pressure and specific internal energy	Mass flow rate and energy 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 magnetic domain (where the product of mmf and flux is not power, but energy), and the thermodynamic domains (gas,

moist air, thermal liquid, and two-phase fluid), where products of 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:

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

$$q_1 = A_1 \bullet v_3$$

$$q_2 = A_2 \bullet v_3$$

where

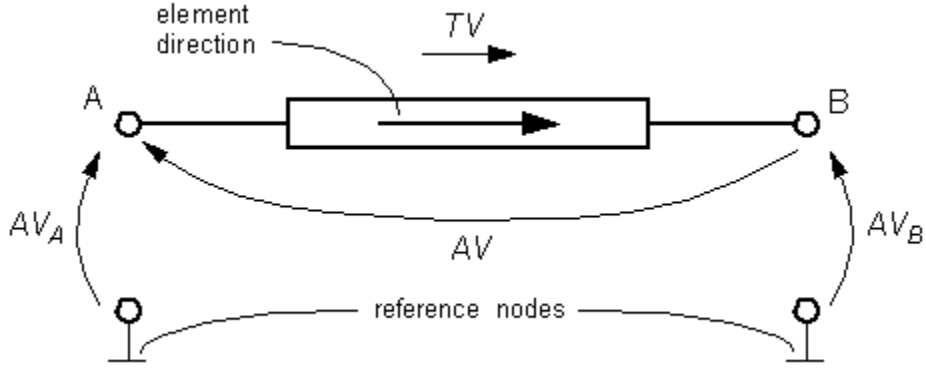
q_1, q_2	Flow rates through ports A and B, respectively (Through variables)
p_1, p_2	Gauge pressures at ports A and B, respectively (Across variables)
A_1, A_2	Piston effective areas
F_3	Rod force (Through variable)
v_3	Rod velocity (Across variable)

The model could be considerably more complex, for example, it could account for friction, fluid compressibility, inertia of the moving parts, and so on. For all these different mathematical models, however, the element configuration (that is, the number and type of ports and the associated Through and Across variables) would remain the same, meaning that the Physical Network approach lets you substitute models of different levels of complexity without introducing any changes to the schematic. For example, you can start developing your system by using the Resistive Tube block from the Foundation library, which accounts only for friction losses. At a later stage in development, you may want to account for fluid compressibility. You can then replace it with a Hydraulic Pipeline block, available with Simscape Fluids™ block libraries, or, depending on your application, even with a Segmented Pipeline block if you also need to account for fluid inertia. This modeling principle is called incremental modeling.

Direction of Variables

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

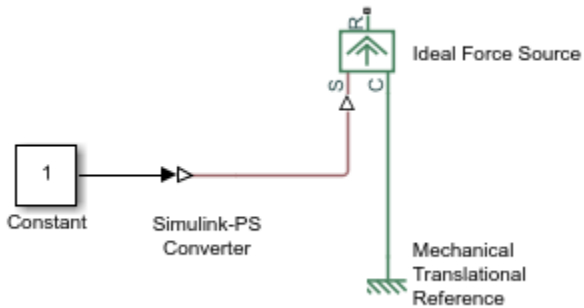
Elements with only two ports are characterized with one pair of variables, a Through variable and an Across variable. Since these variables are closely related, their orientation is defined with one direction. For example, if an element is oriented from port A to port B, it implies that the Through variable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



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

- Provides a simple and consistent way to determine whether an element is active or passive. Energy is one of the most important characteristics to be determined during simulation. If the variables direction, or sign, is determined as described above, their product (that is, the energy) is positive if the element consumes energy, and is negative if it provides energy to a system. This rule is followed throughout the Simscape software.
- Simplifies the model description. Symbol `A → B` is enough to specify variable polarity for both the Across and the Through variables.
- Lets you apply the oriented graph theory to network analysis and design.

As an example of variables direction rules, let us consider the Ideal Force Source block. In this block, as in many other mechanical blocks, port C is associated with the source reference point (case), and port R is associated with the rod.



The block positive direction is from port C to port R. This means that the force is positive if it acts in the direction from C to R, and causes bodies connected to port R to accelerate in the positive direction. The relative velocity is determined as $v = v_C - v_R$, where v_R, v_C are the absolute velocities at ports R and C, respectively, and it is negative if velocity at port R is greater than that at port C. The power generated by the source is computed as the product of force and velocity, and is negative if the source provides energy to the system.

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

All the elements in a network are divided into active and passive elements, depending on whether they deliver energy to the system or dissipate (or store) it. Active elements (force

and velocity sources, flow rate and pressure sources, etc.) must be oriented strictly in accordance with the line of action or function that they are expected to perform in the system, while passive elements (dampers, resistors, springs, pipelines, etc.) can be oriented either way.

Connector Ports and Connection Lines

Simscape blocks may have the following types of ports:

- Physical conserving ports — Nondirectional ports (for example, hydraulic or mechanical) that represent physical connections and relate physical variables based on the Physical Network approach.
- Physical signal ports — Unidirectional ports transferring signals that use an internal Simscape engine for computations.

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

Physical Conserving Ports

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

- You can connect conserving ports only to other conserving ports of the same type.
- The physical connection lines that connect conserving ports together are nondirectional lines that carry physical variables (Across and Through variables, as described above) rather than signals. You cannot connect physical lines to Simulink ports or to physical signal ports.
- Two directly connected conserving ports must have the same values for all their Across variables (such as pressure or angular velocity).
- You can branch physical connection lines. When you do so, components directly connected with one another continue to share the same Across variables. Any Through variable (such as flow rate or torque) transferred along the physical connection line is divided among the multiple components connected by the branches. How the Through variable is divided is determined by the system dynamics.

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

Each type of physical conserving ports used in Simscape blocks uniquely represents a physical modeling domain. For a list of port types, along with the Through and Across variables associated with each type, see the table in “Variable Types” on page 1-4.

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 1-42.

Physical Signal Ports

Physical signal ports ▷ carry signals between Simscape blocks. You connect them with regular connection lines, similar to Simulink signal connections. Physical signal ports are used in Simscape block diagrams instead of Simulink input and output ports to increase computation speed and avoid issues with algebraic loops. Physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs, and 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 1-42.

See Also

Related Examples

- “Creating and Simulating a Simple Model” on page 1-18

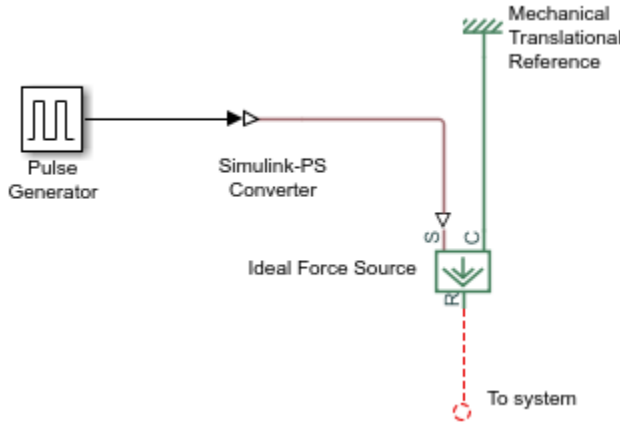
More About

- “Connecting Simscape Diagrams to Simulink Sources and Scopes” on page 1-10

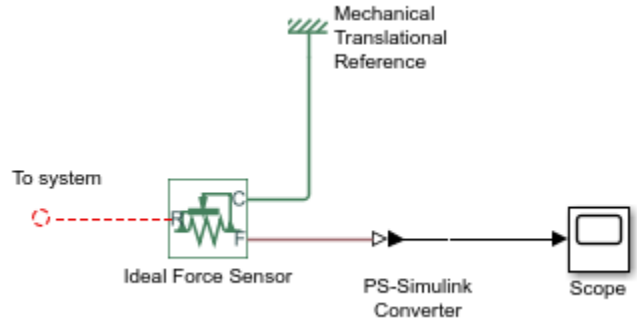
Connecting Simscape Diagrams to Simulink Sources and Scopes

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

See Also

Related Examples

- “Creating and Simulating a Simple Model” on page 1-18

More About

- “Physical Signal Ports” on page 1-9

Simscape Block Libraries

In this section...
“Library Structure Overview” on page 1-12
“Accessing 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, mechanical, electrical, magnetic, thermal, thermal liquid, two-phase fluid, gas, moist air, 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
- 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
- Gas building blocks that let you model gas systems with various levels of idealization: perfect gas, semiperfect gas, or real gas
- Moist air building blocks that let you model two- and three-species gas systems, keeping track of humidity and trace gas levels in the air mixture

- 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 outports to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outports to Simulink inports.

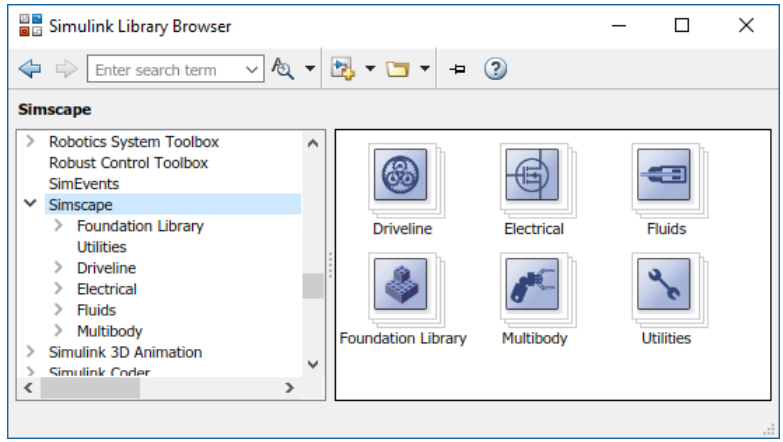
For examples of using these blocks in a Simscape model, see the tutorial “Creating and Simulating a Simple Model” on page 1-18.

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 1-10 for more information on how to do this.

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

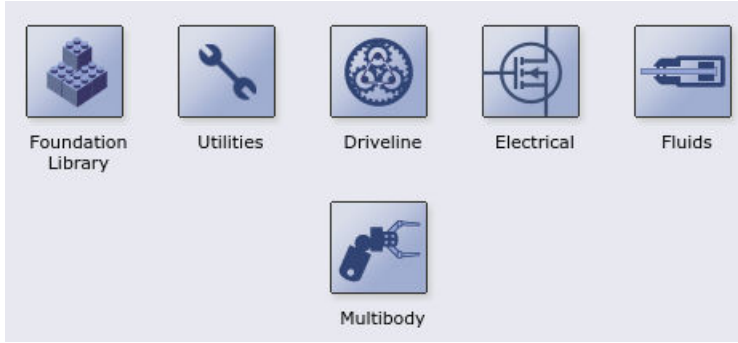
Accessing the Block Libraries

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



When you create a new model using the `ssc_new` command, or type `simscape` in the MATLAB Command Window, the main Simscape library opens in a separate window.

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



Essential Physical Modeling Techniques

Building Your Model

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

- Build your physical model by using a combination of blocks from the Simscape Foundation and Utilities libraries. Simscape software lets you create a network representation of the system under design, based on the Physical Network approach. According to this approach, each system is represented as consisting of functional elements that interact with each other by exchanging energy through their ports.
- Each Simscape diagram (or each topologically distinct physical network in a diagram) must contain a Solver Configuration block from the Simscape Utilities library.
- 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 Simscape Fluids 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 gas elements in your model, default gas properties are for dry air. Attach a Gas Properties (G) block to each topologically distinct circuit to change gas properties.
- If you have moist air elements in your model, default properties correspond to dry air, water vapor, and carbon dioxide (the optional trace gas). Attach a Moist Air Properties (MA) block to each topologically distinct circuit to change the air mixture properties.
- 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 1-17.
- 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 Simscape Fluids 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, 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 1-42.
- The Physical connection lines that connect Conserving ports together are nondirectional lines that carry physical variables (Across and Through variables, as described above) rather than signals. You cannot connect Physical lines to Simulink ports or to Physical Signal ports.
- Two directly connected Conserving ports must have the same values for all their Across variables (such as voltage or angular velocity).
- You can branch Physical connection lines. When you do so, components directly connected with one another continue to share the same Across variables. Any Through variable (such as current or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches. How the Through variable is divided is determined by the system dynamics.

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

Using the Physical Signal Ports

The following rules apply to Physical Signal ports:

- You can connect Physical Signal ports to other Physical Signal ports with regular connection lines, similar to Simulink signal connections. These connection lines carry physical signals between Simscape blocks.
- You can connect Physical Signal ports to Simulink ports through special converter blocks. Use the Simulink-PS Converter block to connect Simulink outports to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outports to Simulink inports.
- Physical Signals can have units associated with them. Simscape block dialogs 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

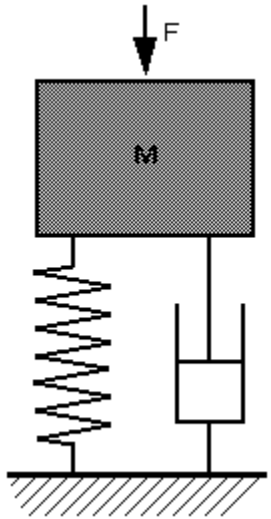
In this section...
“Building a Simscape Diagram” on page 1-18
“Modifying Initial Settings” on page 1-26
“Running the Simulation” on page 1-28
“Adjusting the Parameters” on page 1-30

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 on page 1-15 and makes you familiar with using the basic Simscape blocks.

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

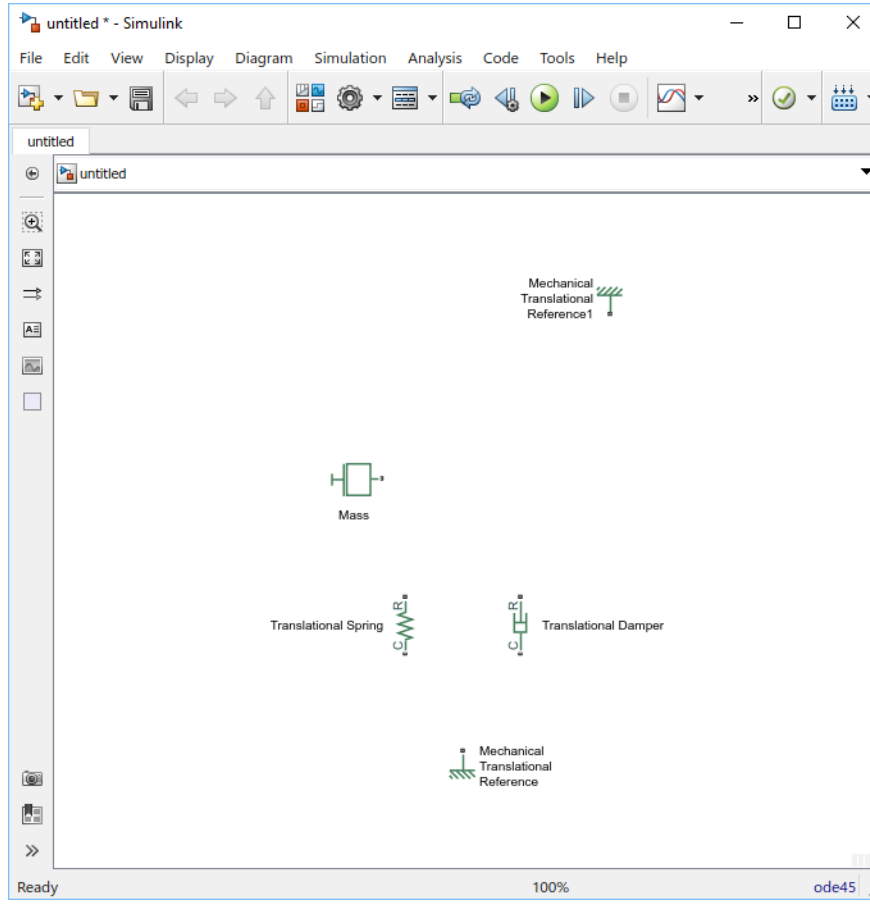
The following schematic represents a simple model of a car suspension. It consists of a spring and damper connected to a body (represented as a mass), which is agitated by a force. You can vary the model parameters, such as the stiffness of the spring, the mass of the body, or the force profile, and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



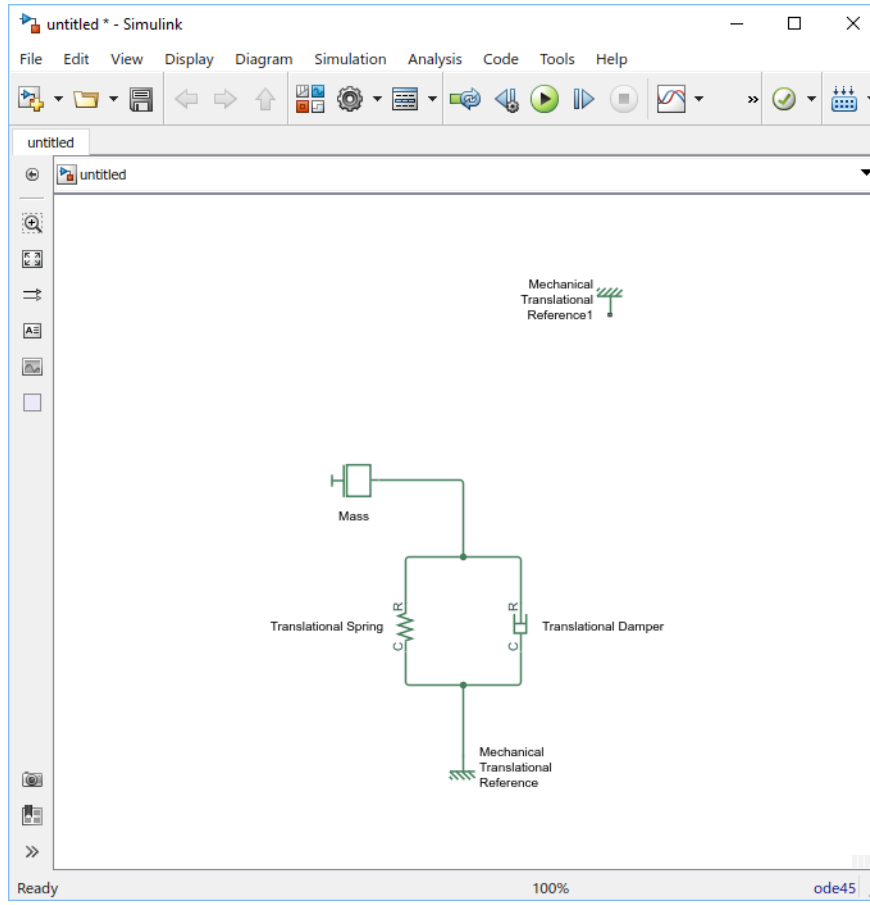
To create an equivalent Simscape diagram, follow these steps:

1. Open the Simulink Library Browser, as described in “Simscape Block Libraries” on page 1-12.
2. Create a new Simulink model using the **Blank Model** template. The software creates an empty model in memory and displays it in a new model editor window.

   Note 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. By default, Simulink Editor hides the automatic block names in model diagrams. To display hidden block names for training purposes, select **Display** and clear the **Hide Automatic Names** check box.
4. Open the Simscape > Foundation Library > Mechanical > Translational Elements library.
5. Drag the Mass, Translational Spring, Translational Damper, and two Mechanical Translational Reference blocks into the model window.
6. Orient the blocks as shown in the following illustration. To rotate a block, select it and press **Ctrl+R**.

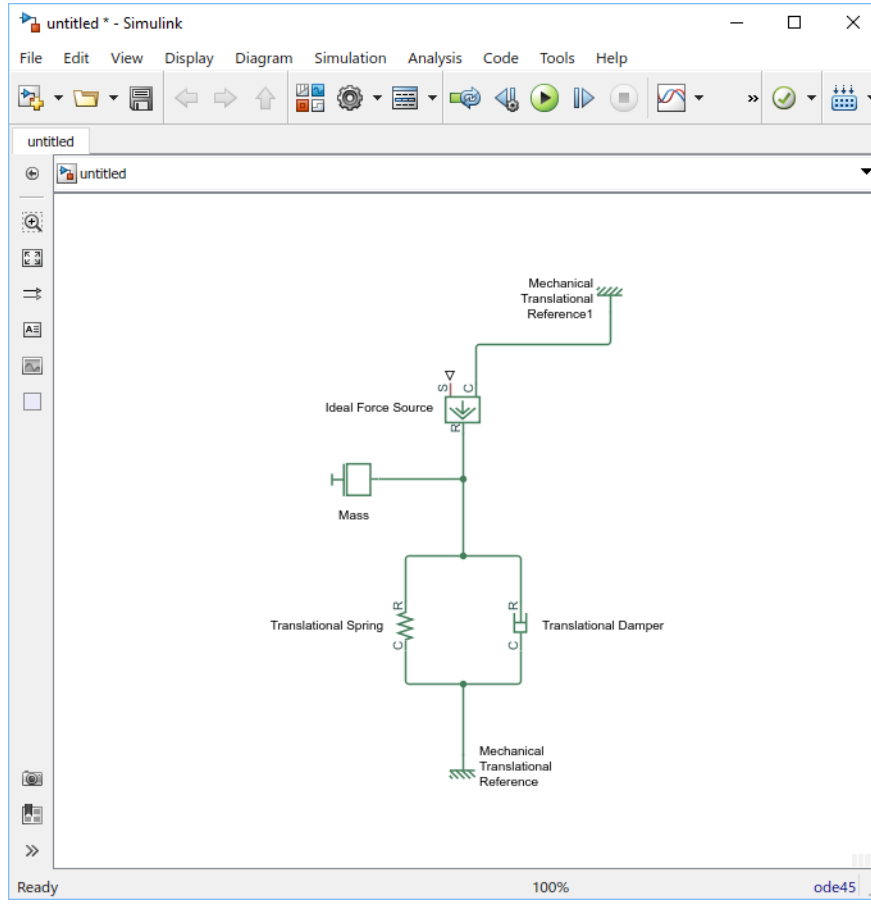


7 Connect the Translational Spring, Translational Damper, and Mass blocks to one of the Mechanical Translational Reference blocks as shown in the next illustration.

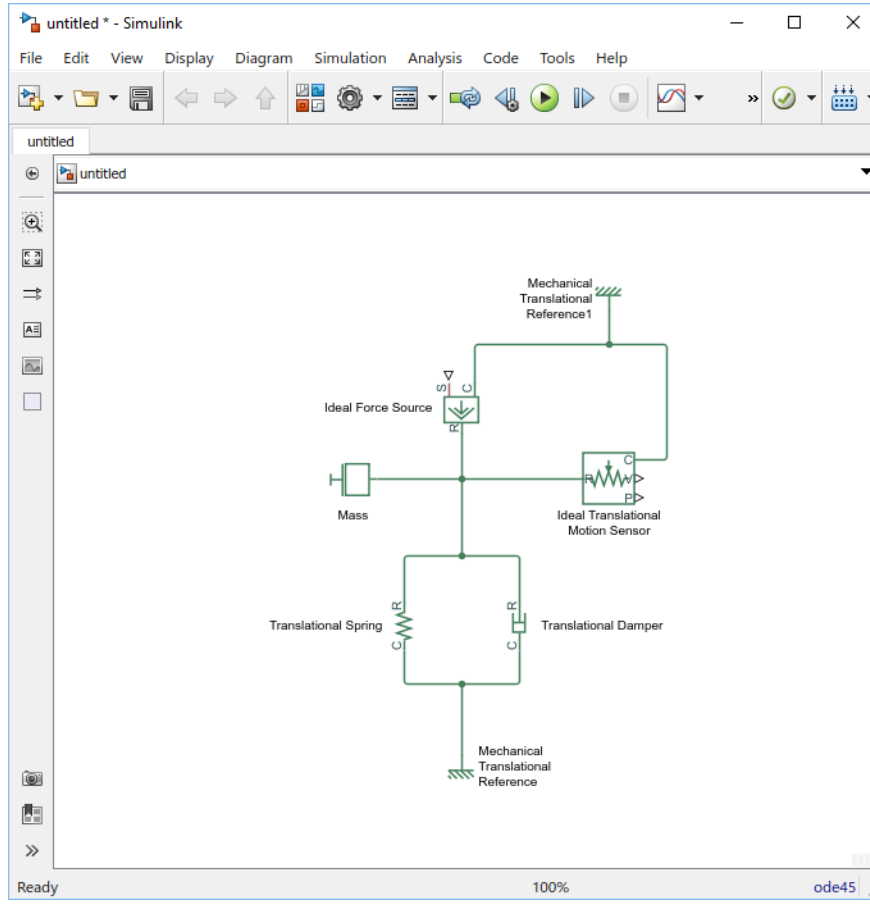


8 To add the representation of the force acting on the mass, open the Simscape > Foundation Library > Mechanical > Mechanical Sources library and add the Ideal Force Source block to your diagram.

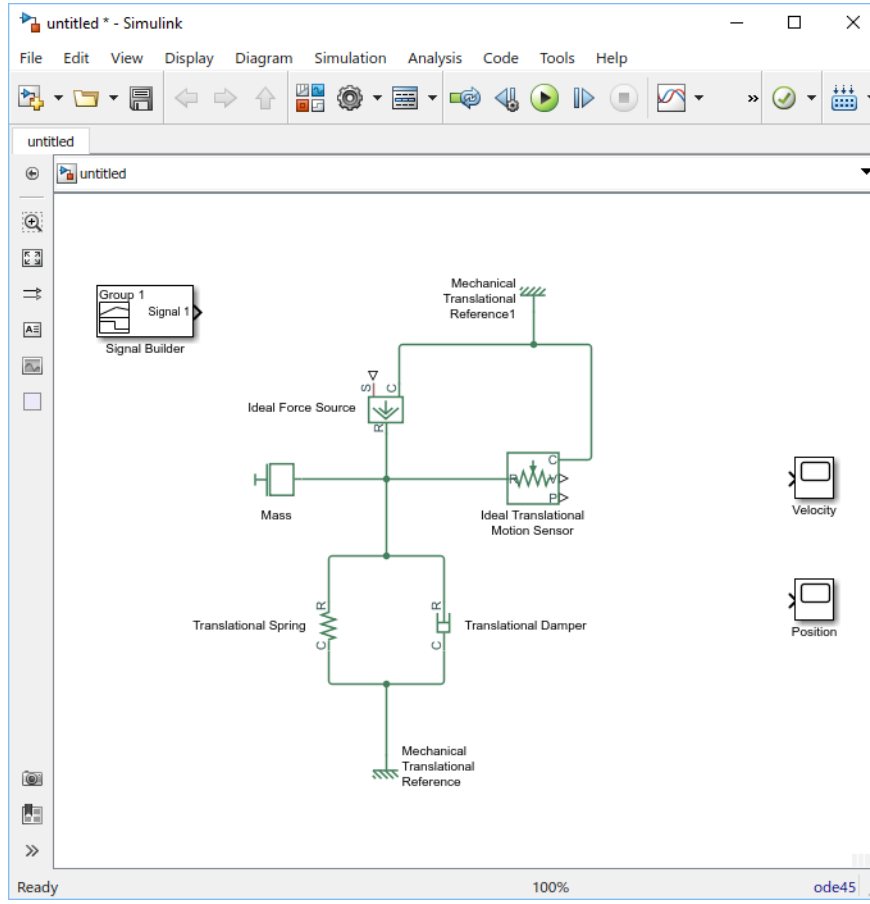
To reflect the correct direction of the force shown in the original schematic, flip the block by selecting **Diagram > Rotate & Flip > Flip Block > Up-Down** from the top menu bar of the model window. Connect the block's port C (for “case”) to the second Mechanical Translational Reference block, and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



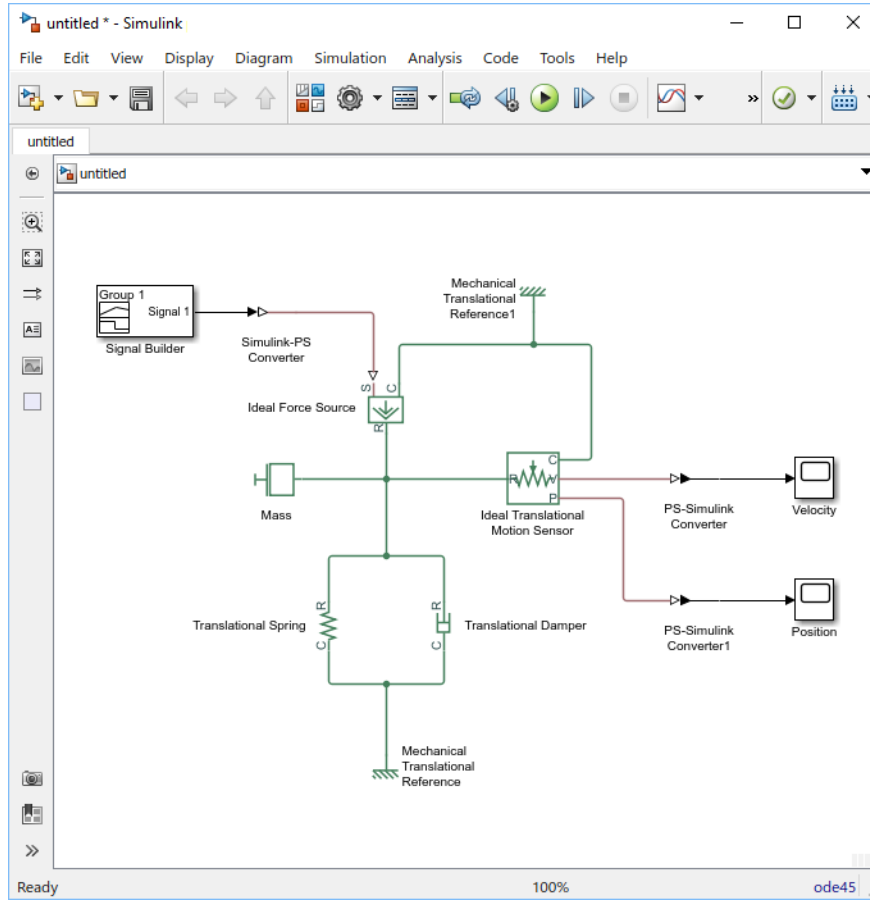
9 Add the sensor to measure speed and position of the mass. Place the Ideal Translational Motion Sensor block from the Mechanical Sensors library into your diagram and connect it as shown below.



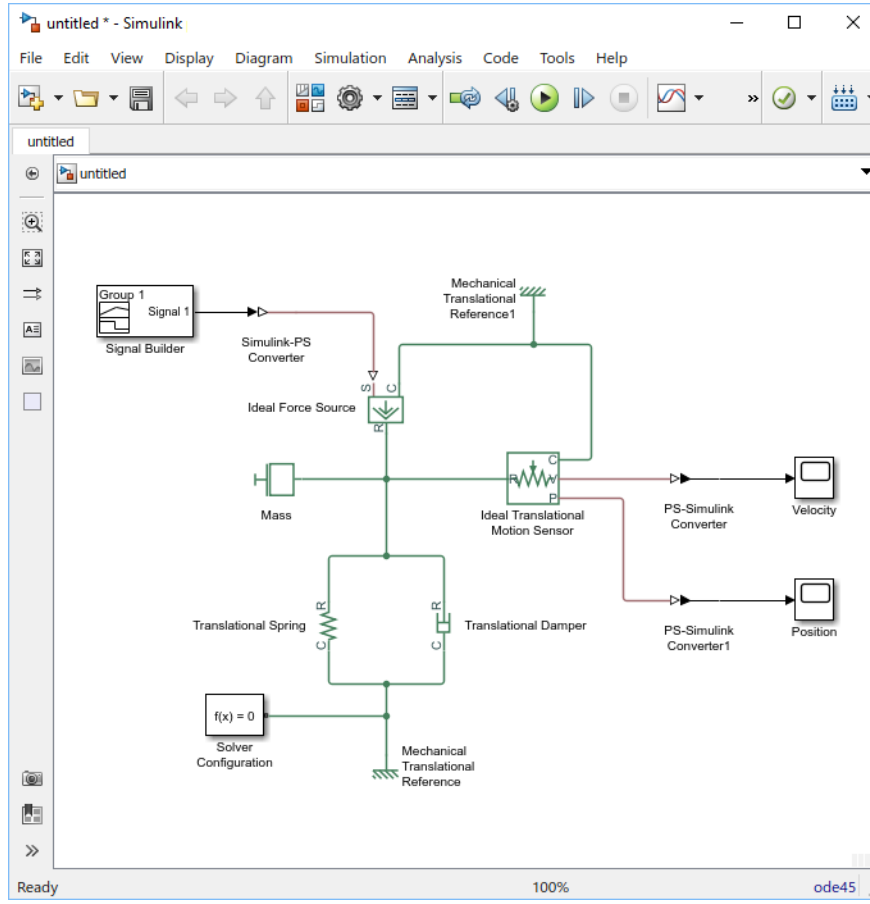
10 Now you need to add the sources and scopes. They are found in the regular Simulink libraries. Open the Simulink > Sources library and copy the Signal Builder block into the model. Then open the Simulink > Sinks library and copy two Scope blocks. Rename one of the Scope blocks to `Velocity` and the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



11 Every time you connect a Simulink source or scope to a Simscape diagram, you have to use an appropriate converter block, to convert Simulink signals into physical signals and vice versa. Open the Simscape > Utilities library and copy a Simulink-PS Converter block and two PS-Simulink Converter blocks into the model. Connect the blocks as shown below.



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



13 Your block diagram is now complete. Save it as `mech_simple`.

Modifying Initial Settings

After you have put together a block diagram of your model, as described in the previous section on page 1-18, you need to select a solver and provide the correct values for configuration parameters.

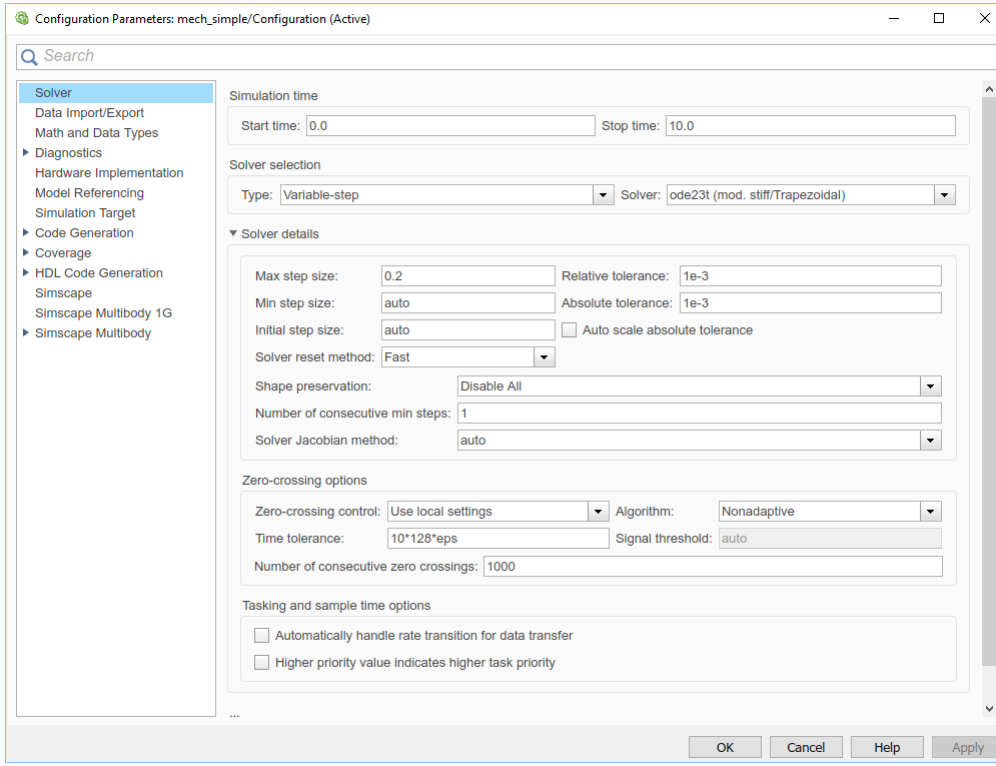
To prepare for simulating the model, follow these steps:

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

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

   Also note that **Simulation time** is specified to be between 0 and 10 seconds. You can adjust this setting later, if needed.



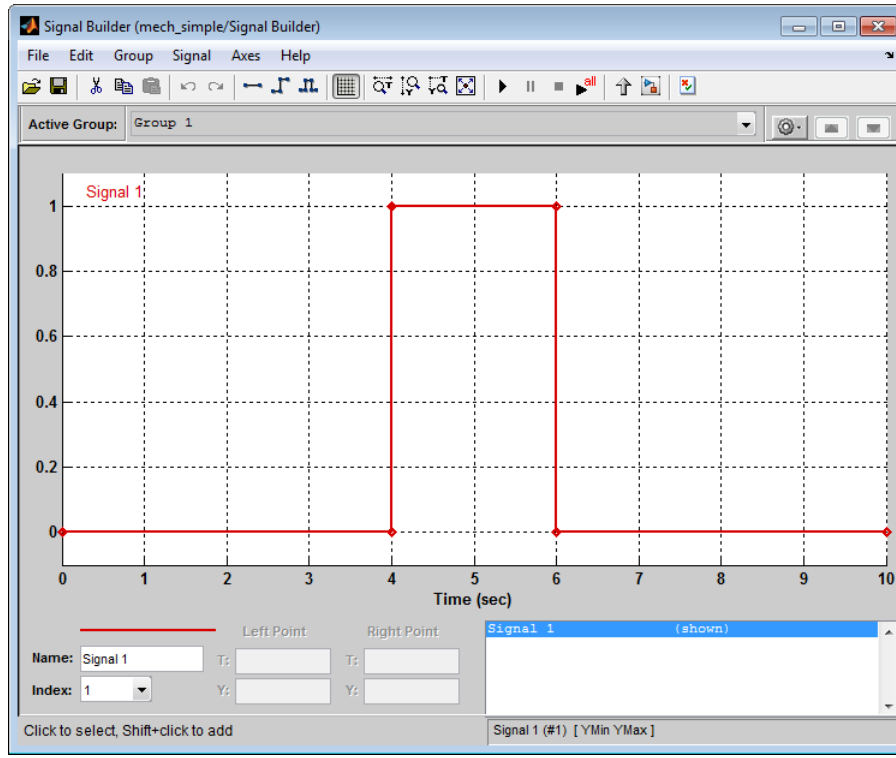
   Click **OK** to close the Configuration Parameters dialog box.

2. Save the model.

Running the Simulation

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

1 The input signal for the force is provided by the Signal 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.

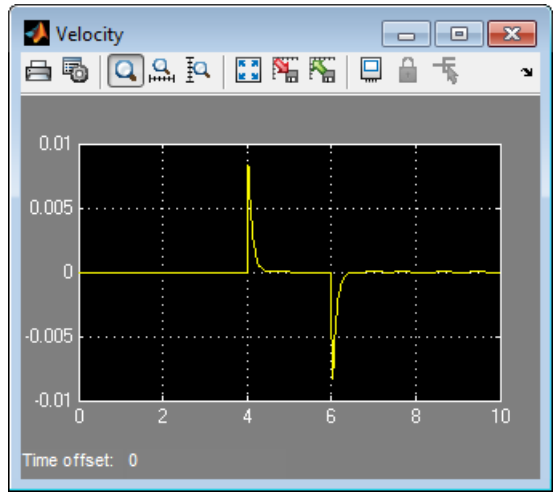


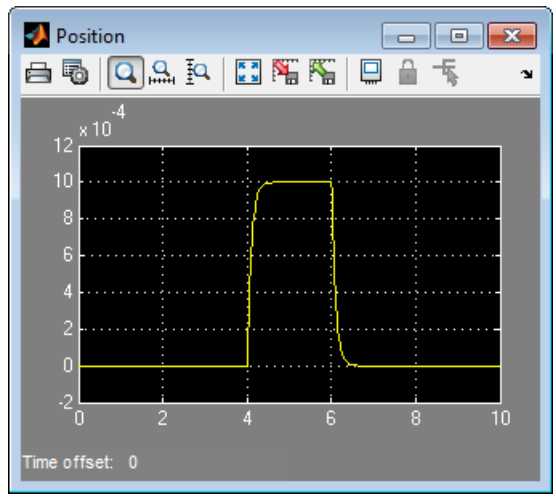
The Velocity scope outputs the mass velocity, and the Position scope outputs the mass displacement as a function of time. Double-click both scopes to open them.

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 6-7. Completion of this step may take a few seconds. The message in the bottom-left corner of the model window provides the status update.

3 Once the simulation starts running, the Velocity and Position scope windows display the simulation results, as shown in the next illustration.





In the beginning, the mass is at rest. Then at 4 seconds, as the input signal changes abruptly, the mass velocity spikes in the positive direction and gradually returns to zero. The mass position at the same time changes more gradually, on account of inertia and damping, and stays at the new value as long as the force is acting upon it. At 6 seconds, when the input signal changes back to zero, the velocity gets a mirror spike, and the mass gradually returns to its initial position.

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

Adjusting the Parameters

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

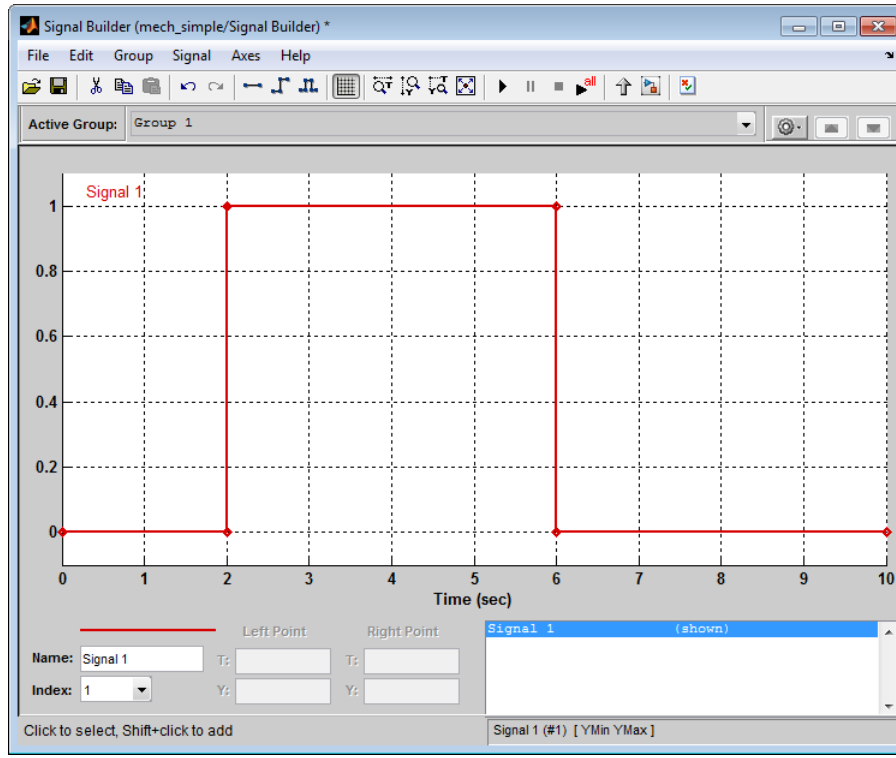
Try the following adjustments:

1. Change the force profile on page 1-30.
2. Change the model parameters. on page 1-32
3. Change the mass position output units. on page 1-34

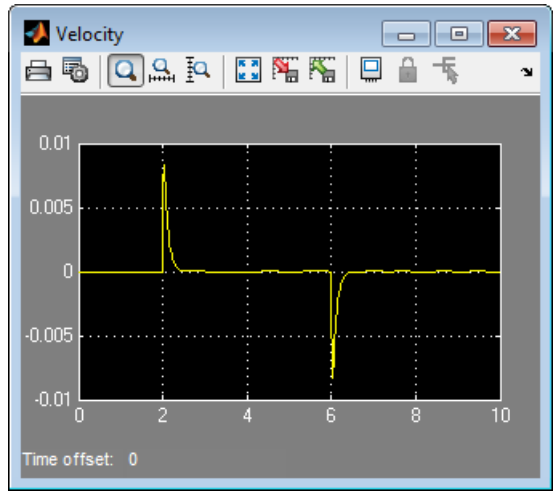
Changing the Force Profile

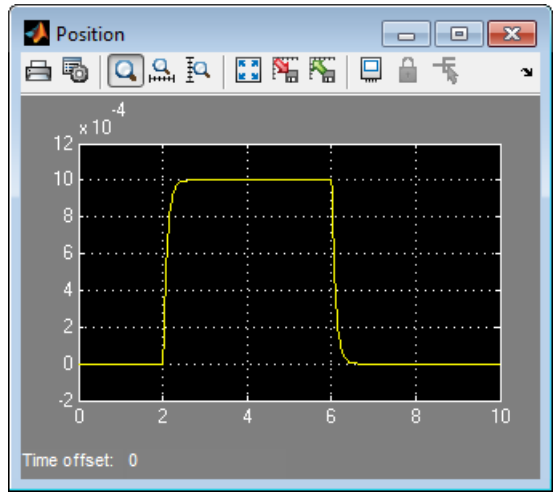
This example shows how a change in the input signal affects the force profile, and therefore the mass displacement.

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.



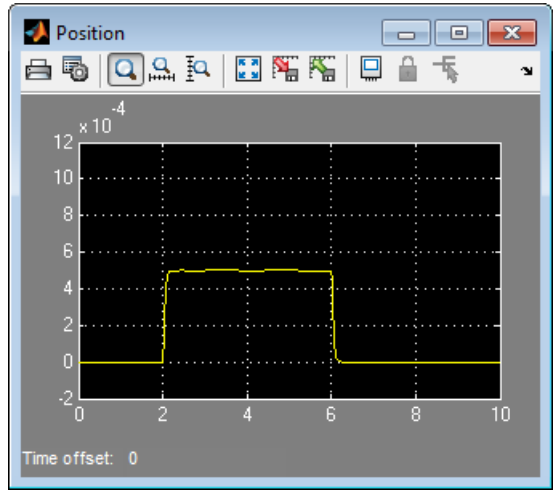


Changing the Model Parameters

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.

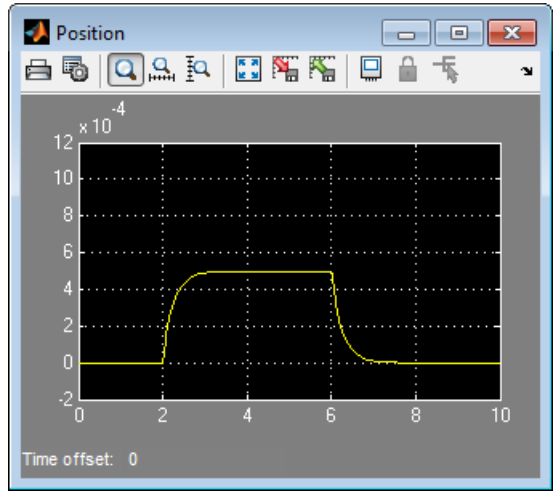
1 Double-click the Translational Spring block. Set its **Spring rate** to `2000 N/m`.

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)`.

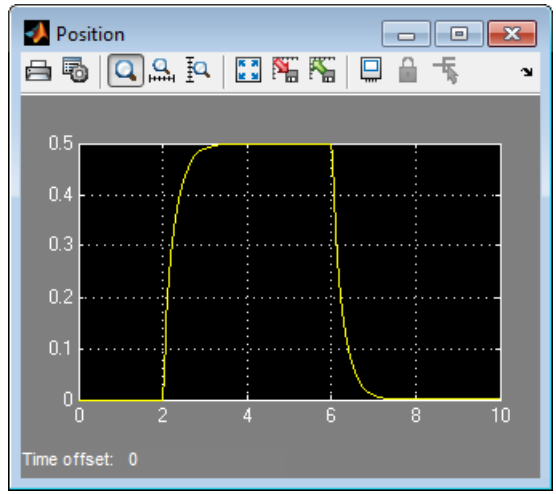
4 Run the simulation. Because of the increase in viscosity, the mass is slower both in reaching its maximum displacement and in returning to the initial position, as shown in the following illustration.



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.



See Also

More About

- “Basic Principles of Modeling Physical Networks” on page 1-2
- “Modeling Best Practices” on page 1-35

Modeling Best Practices

In this section...
“Grounding Rules” on page 1-35 “Avoiding Numerical Simulation Issues” on page 1-39

Grounding Rules

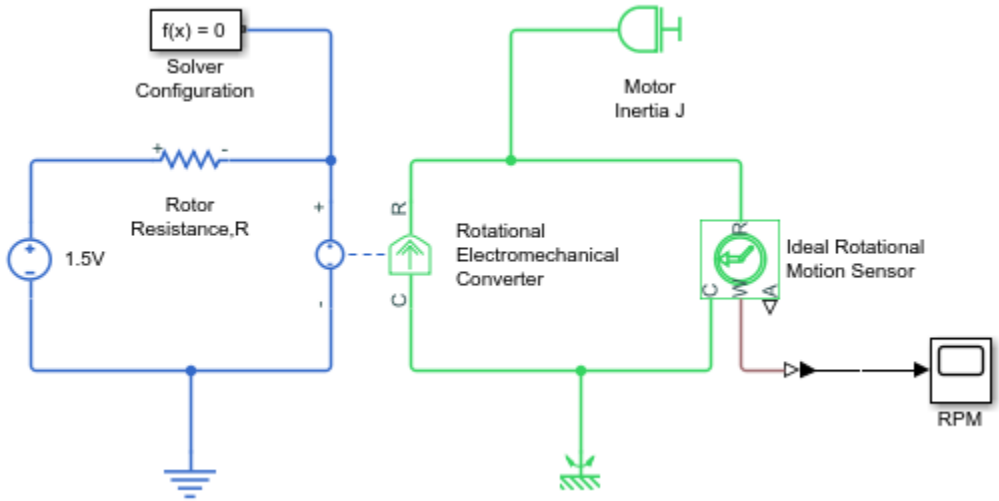
This section contains guidelines for using domain-specific reference blocks (such as Electrical Reference, Mechanical Translational Reference, and so on) in Simscape diagrams, along with examples of correct and incorrect configurations.

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

- “Each Domain Requires at Least One Reference Block” on page 1-35
- “Each Circuit Requires at Least One Reference Block” on page 1-36
- “Multiple Connections to the Domain Reference Are Allowed Within a Circuit” on page 1-38

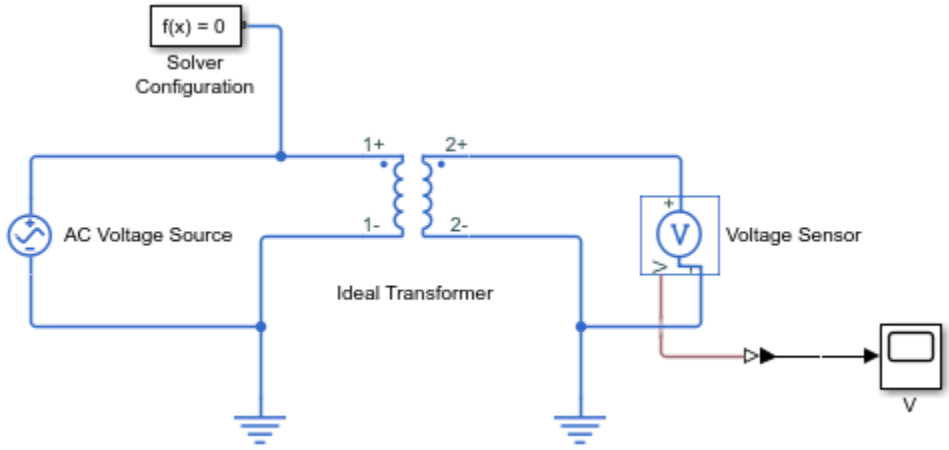
Each Domain Requires at Least One Reference Block

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.

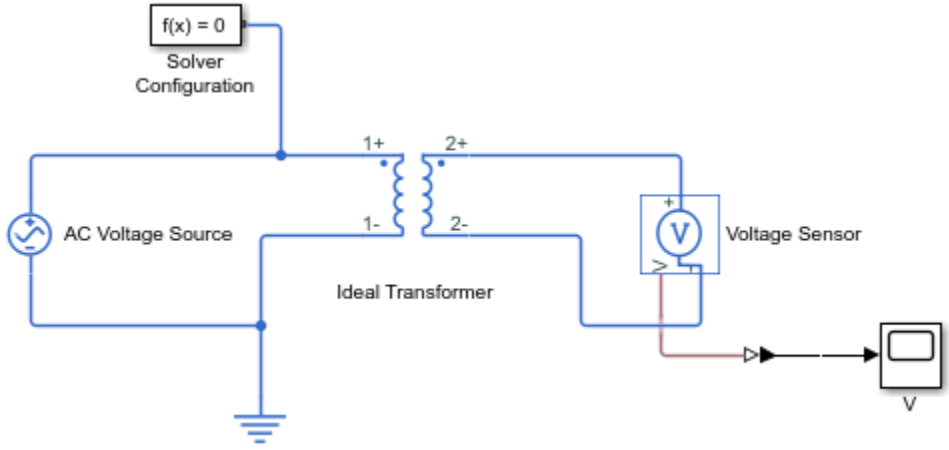


Each Circuit Requires at Least One Reference Block

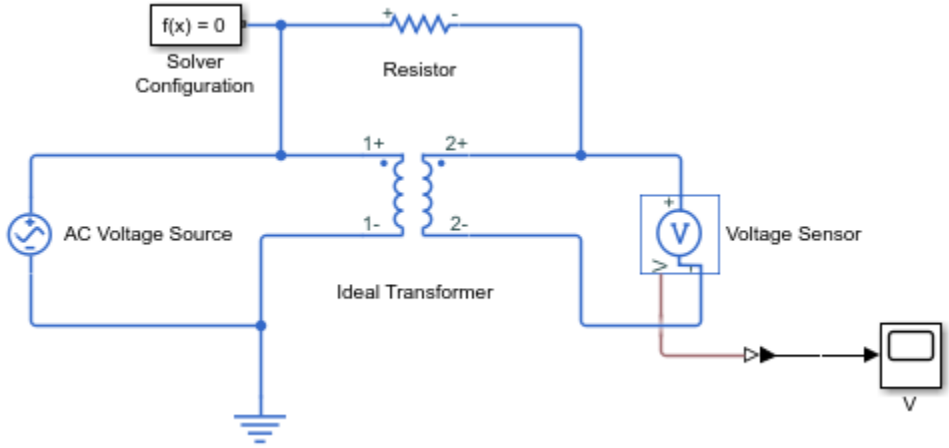
Each topologically distinct circuit within a domain must contain at least one reference block. Some blocks, such as an Ideal Transformer, interface two parts of the network but do not convey information about signal levels relative to the reference block. In the following diagram, there are two separate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

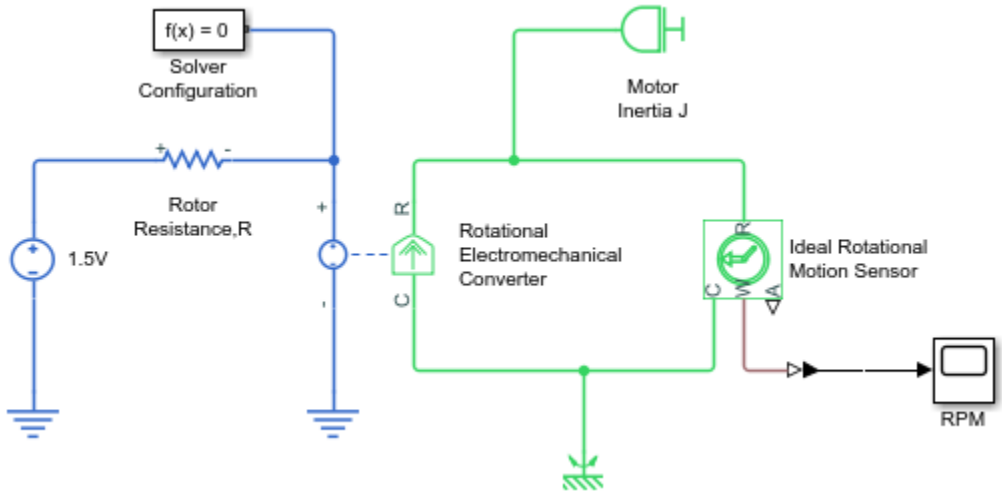


Multiple Connections to the Domain Reference Are Allowed Within a Circuit

More that 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.

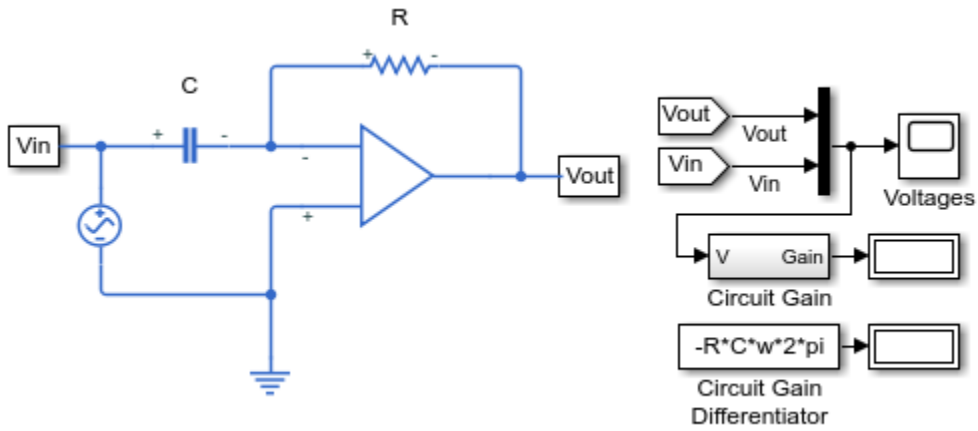
Note Mathematically, these topologies result in Index-2 differential algebraic equations (DAEs). Their solution requires two differentiations of the constraint equations and, as such, it is numerically better to avoid these component topologies where possible.

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.

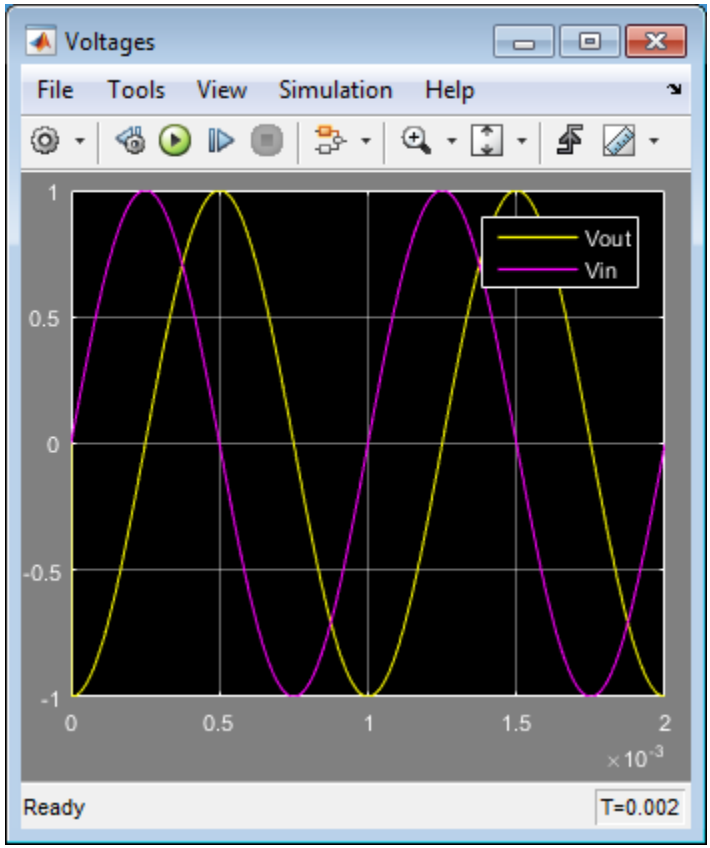
The second approach is to include small parasitic resistances in the component. In the Simscape Foundation library, the Capacitor and Inductor blocks include such parasitic terms, so that you can connect capacitances in parallel with voltage sources and inductors in series with current sources. If your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

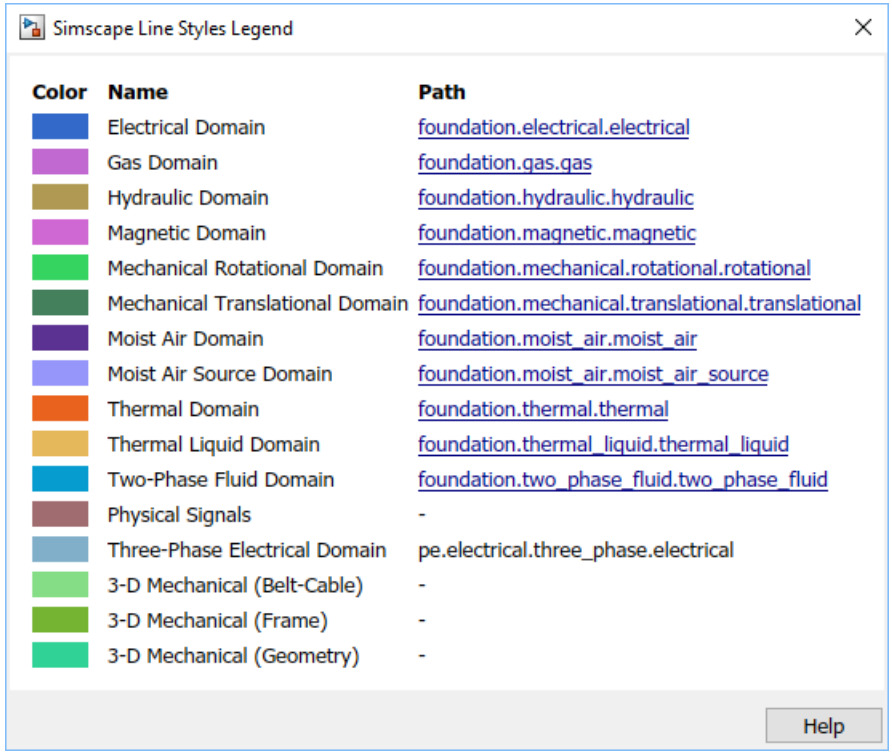
The cause of the problems is that the circuit effectively connects the voltage source in parallel with the capacitor. This is because an ideal op-amp satisfies `V+ = V-` , where `V+` and `V-` are the noninverting and inverting inputs, respectively. This is an example where it is not possible to replace 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.

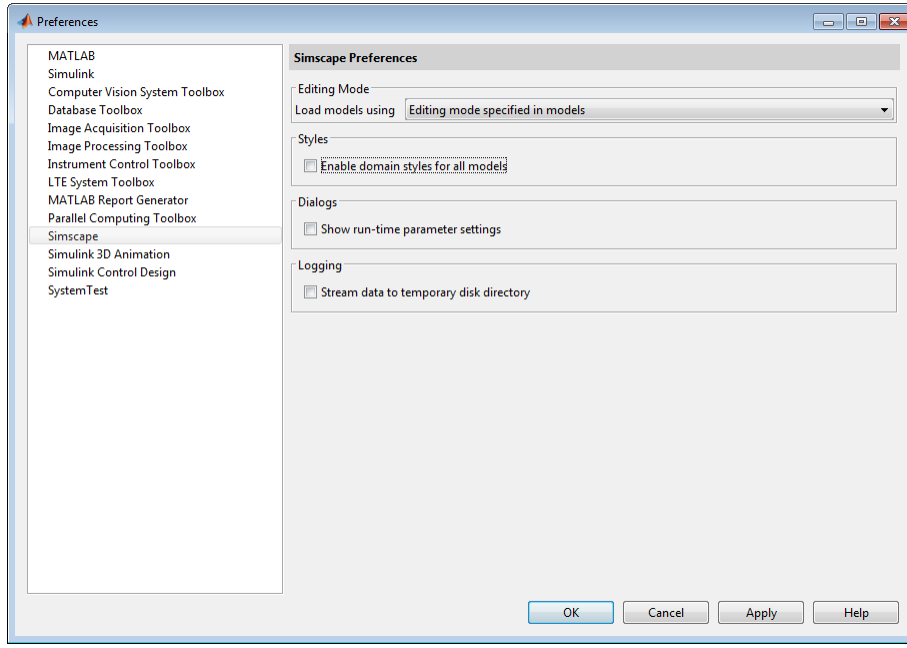
Domain-specific line styles apply to the block icons as well. If all the block ports belong to the same domain, then the whole block icon assumes the line style and color of that domain. If a block has multiple port types, such as the Rotational Electromechanical Converter, then relevant parts of the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

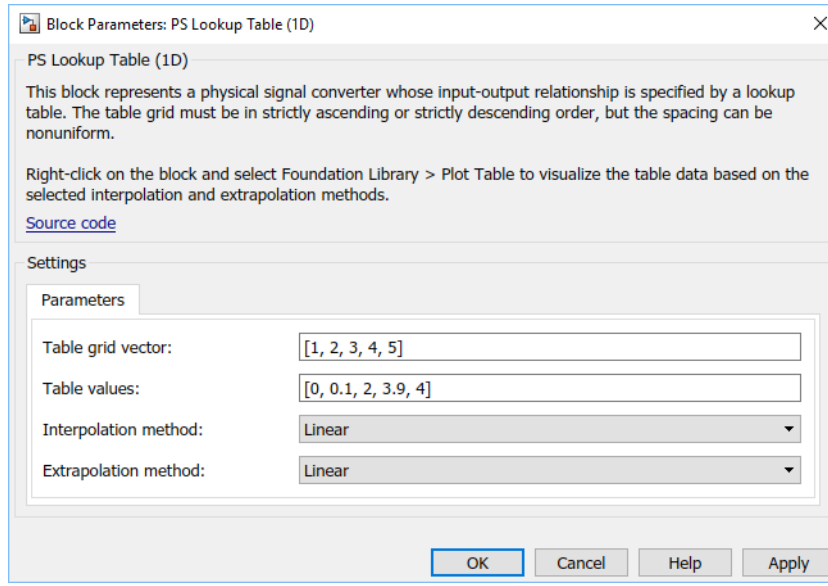

Preferences
MATLAB
Simulink
Computer Vision System Toolbox
Database Toolbox
Image Acquisition Toolbox
Image Processing Toolbox
Instrument Control Toolbox
LTE System Toolbox
MATLAB Report Generator
Parallel Computing Toolbox
Simscape
Simulink 3D Animation
Simulink Control Design
SystemTest
Simscape Preferences
Editing Mode
Load models using
Editing mode specified in models
Styles
Enable domain styles for all models
Dialogs
Show run-time parameter settings
Logging
Stream data to temporary disk directory
OK
Cancel
Apply
Help

Plot Lookup Tables

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

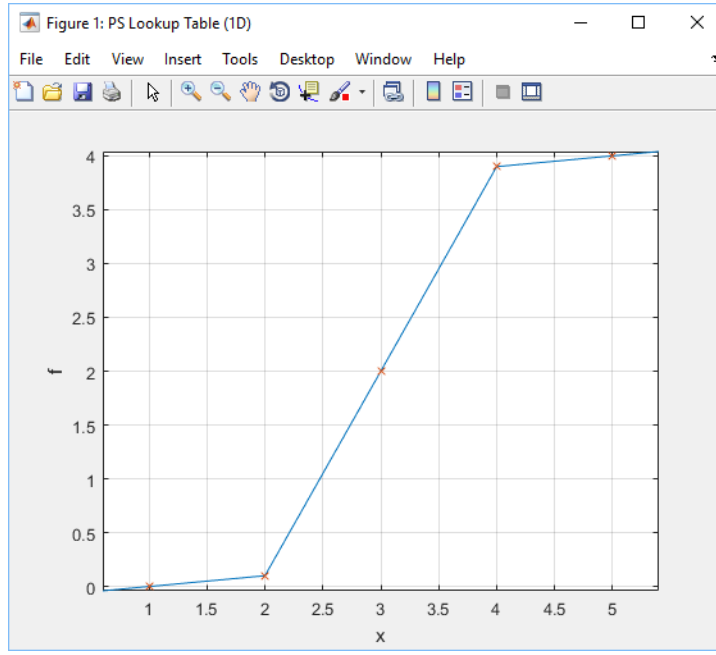
If you change the underlying table data, plotting it again opens a new window. This way, you can compare the plots side by side and see how the block parameter values affect the resulting lookup function.

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



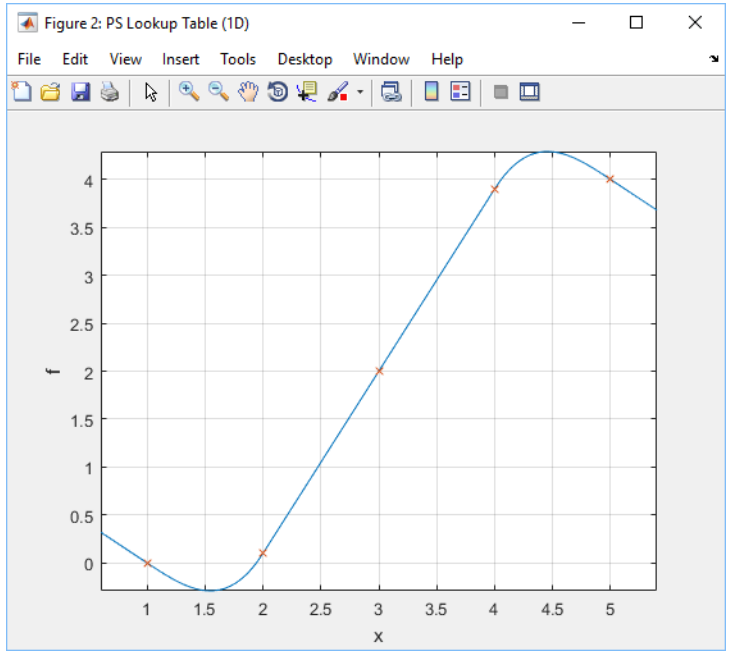
2. Right-click the block in your model. From the context menu, select **Foundation Library > Plot Table**.

   A figure window containing the plot of the data opens.



3. In the block dialog box, change the **Interpolation method** parameter value to `Smooth`.
4. Plot the data again by right-clicking the block and selecting **Foundation Library** > **Plot Table**.

   A new figure window opens.

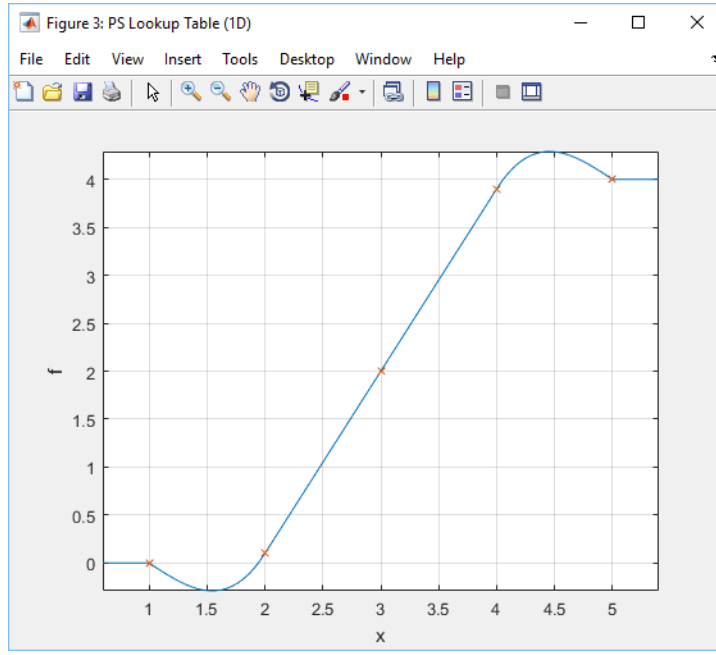


The curve shape has changed because of the new interpolation method.

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

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

A new figure window opens.



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

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

See Also

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

2

Thermal Liquid Models

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-7
- “Thermal Liquid Modeling Framework” on page 2-11
- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

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 Gas—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 Gas 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 and energy flow rates), the other for Across variables (pressure and temperature). By using the PS-Simulink Converter block, you can specify the physical units of the sensed variable.

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

For an overview of Simscape data logging, see “About Simulation Data Logging” on page 12-2. For an example of how to plot logged data, see “Log and Plot Simulation Data” on page 12-9.

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.

See Also

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

More About

- “Thermal Liquid Library” on page 2-7
- “Thermal Liquid Modeling Framework” on page 2-11

Thermal Liquid Library

In this section...
“Why Use Thermal Liquid Blocks?” on page 2-7
“Representing Thermal Liquid Components” on page 2-7
“Specifying Thermal Liquid Medium” on page 2-9
“Modeling Multidomain Systems” on page 2-9

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	Output measurement data for dynamic variables such as mass flow rate, energy flow rate, pressure, and temperature.	Pressure & Temperature Sensor (TL), Mass & Energy Flow Rate Sensor (TL), Thermodynamic Properties Sensor (TL), Volumetric Flow Rate Sensor (TL)

Component Type	Description	Thermal Liquid Blocks
Thermal liquid	Specify thermodynamic properties and pressure-temperature validity region of thermal liquid medium.	Thermal Liquid Settings (TL)

Specifying Thermal Liquid Medium

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

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

Modeling Multidomain Systems

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

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

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

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

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

See Also

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

More About

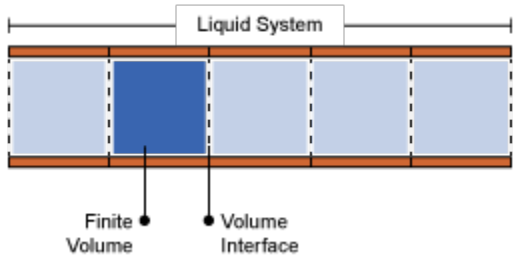
- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Modeling Framework” on page 2-11

Thermal Liquid Modeling Framework

In this section...
“How Blocks Represent Components” on page 2-11 “How Ports Represent Interfaces” on page 2-12 “Full Flux Scheme” on page 2-13

How Blocks Represent Components

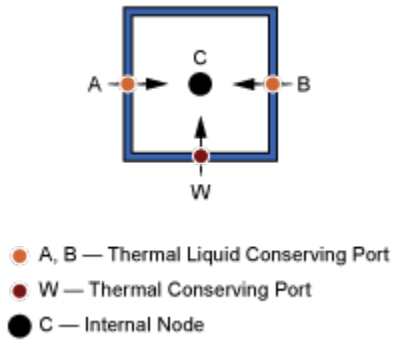
Thermal Liquid models are based on the finite volume method. This method discretizes a thermal liquid system into multiple control volumes that interact via shared interfaces. An oil pipeline system is one example: you can model this system as a set of pipeline segments that connect serially along the 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 12-2.



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.

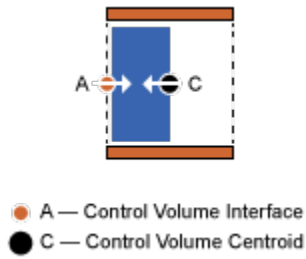
How Ports Represent Interfaces

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.

See Also

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

More About

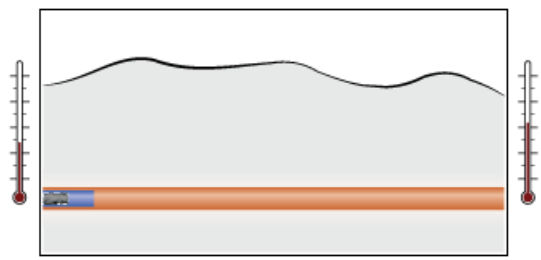
- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-7

Heat Transfer in Insulated Oil Pipeline

In this section...
“Oil Pipelines” on page 2-15
“Modeling Considerations” on page 2-16
“Simscape Model” on page 2-18
“Run Simulation” on page 2-20
“Run Optimization Script” on page 2-27

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.

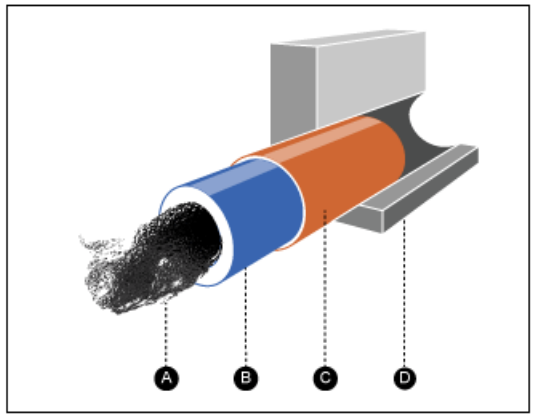
Viscous dissipation provides an additional heat source. As adjacent parcels of oil flow against each other, they experience energy losses that appear in the form of heat. The warming effect is small, but sufficient to at least partially offset the conductive heat losses that occur through the insulant 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, *TUpstream*, with a fixed mass flow rate, *Vdot* * *rho0*, where:

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

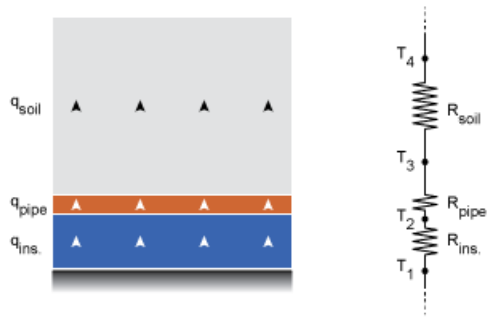
- *rho0* is the mass density of oil entering the pipeline segment.

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

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

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

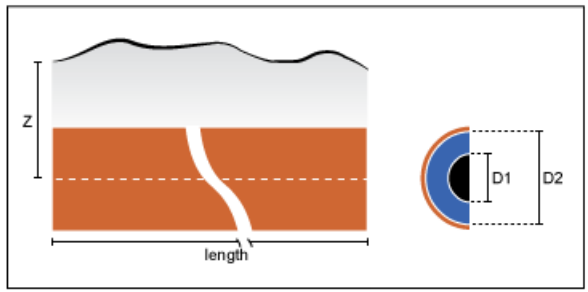
$$R_{\text{combined}} = R_{\text{wall}} + R_{\text{ins.}} + R_{\text{soil}}$$



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, $R_{\text{ins.}}$ and R_{soil}.

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

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



z — Soil layer thickness

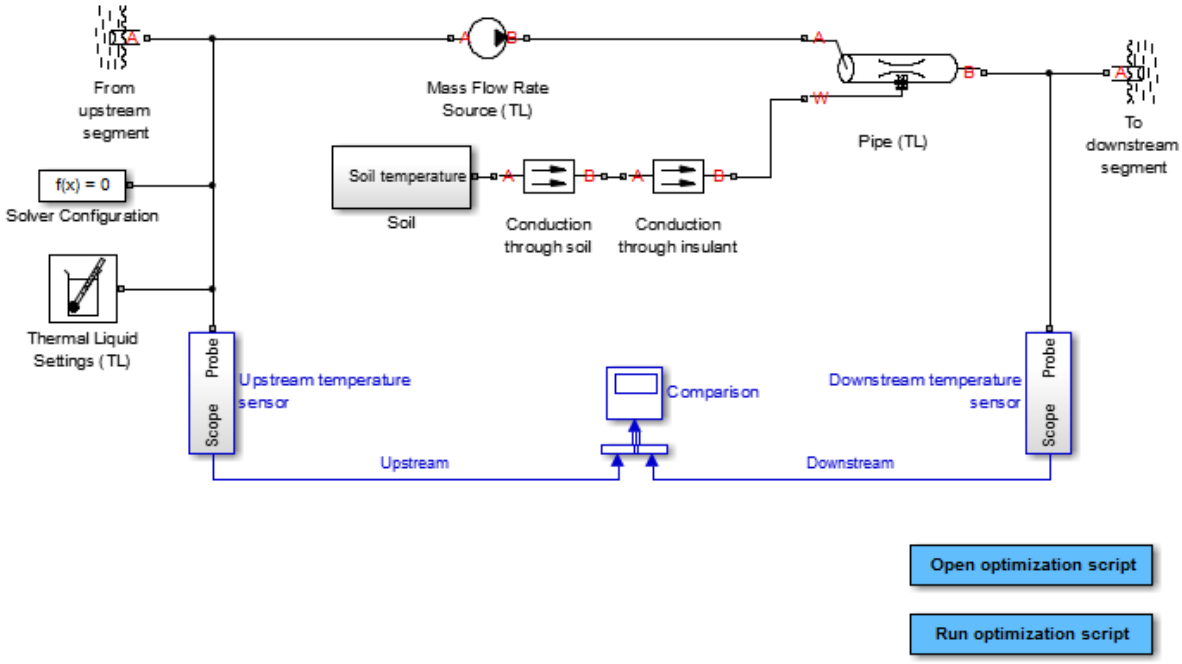
length — Pipeline segment length

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.

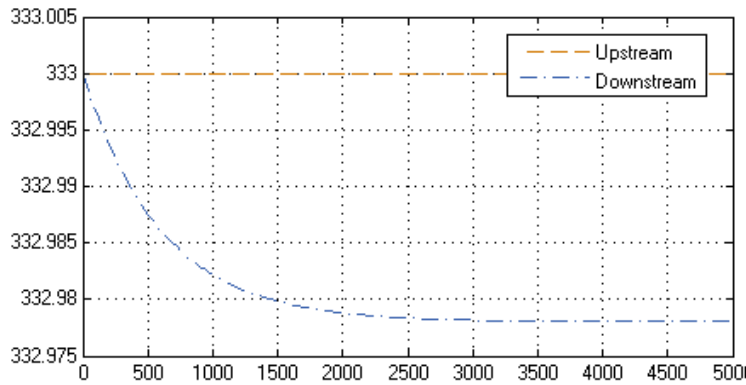
The Conduction through insulant and Conduction through soil blocks represent thermal conduction through insulant and soil layers, respectively. These blocks appear in the Simscape Thermal library as Conductive Heat Transfer. The Soil subsystem block provides the temperature boundary condition at the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

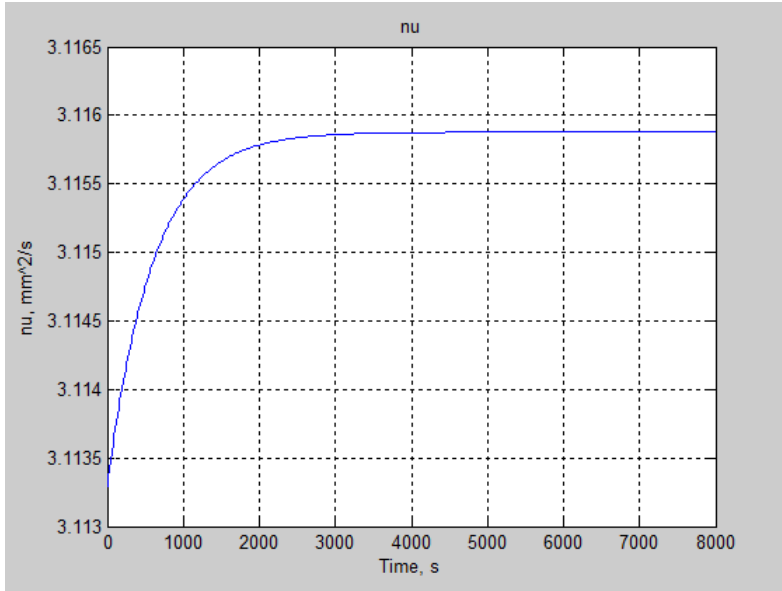
physical property. One example is the kinematic viscosity of oil in the pipeline segment, represented by the Pipe (TL) block.

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
```

2 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 12-2.

Simulate Effects of Changing Insulation Diameter

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

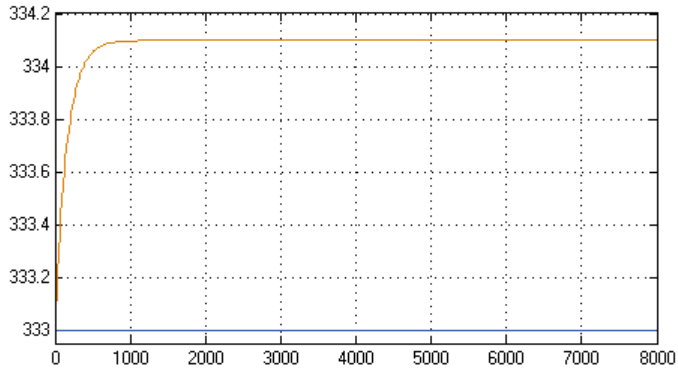
1 Open Model Explorer.

2 In the **Model Hierarchy** pane, select **Base Workspace**.

3 In the **Contents** pane, click the value of parameter D1.

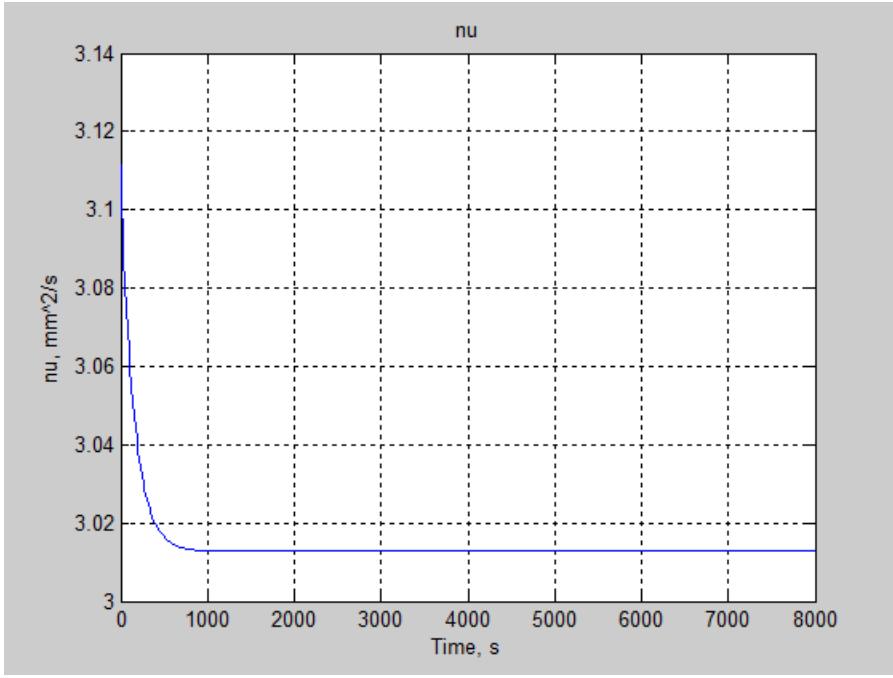
4 Enter `0.20`.

By reducing the inner diameter of the insulation layer to 0.20, you increase the insulation thickness, slowing down heat loss through the pipe wall via thermal conduction. Run the simulation. Then, open the Comparison scope and autoscale to view full plot.

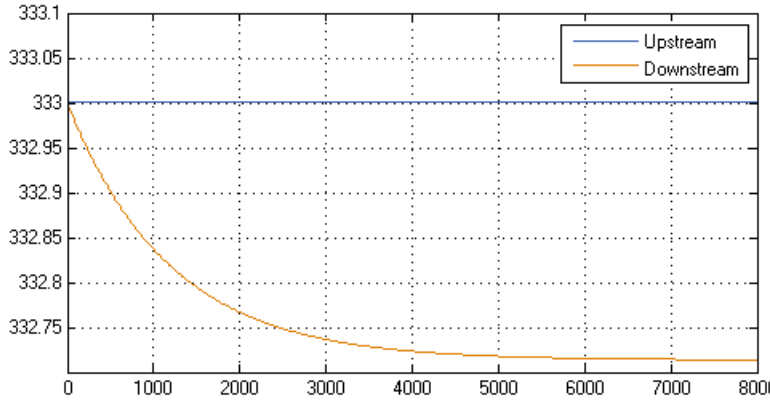


The new plot shows an oil temperature at the pipe outlet (top curve) that significantly exceeds that at the pipe inlet (bottom line). Viscous dissipation now dominates the thermal energy balance in the pipeline segment. The new insulation thickness poses a design problem: in a long pipeline, a 1.1 K/km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

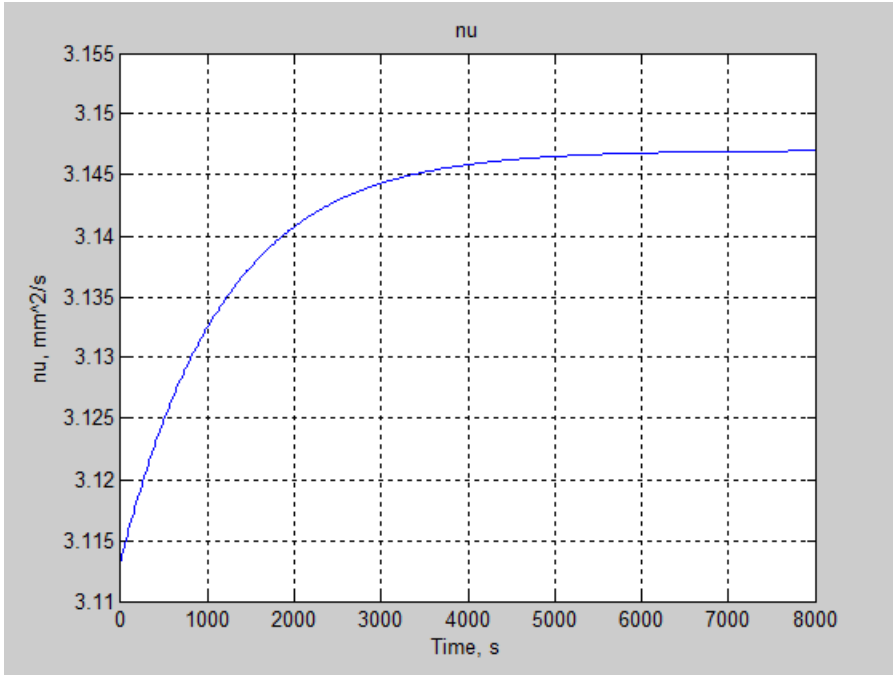


Try increasing the inner diameter of the insulation layer, D1, to 0.55. By increasing this value, you decrease the insulation thickness, accelerating heat loss through the pipe wall via thermal conduction. Then, run the simulation. Open the Comparison scope and autoscale to view the full plot.



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.

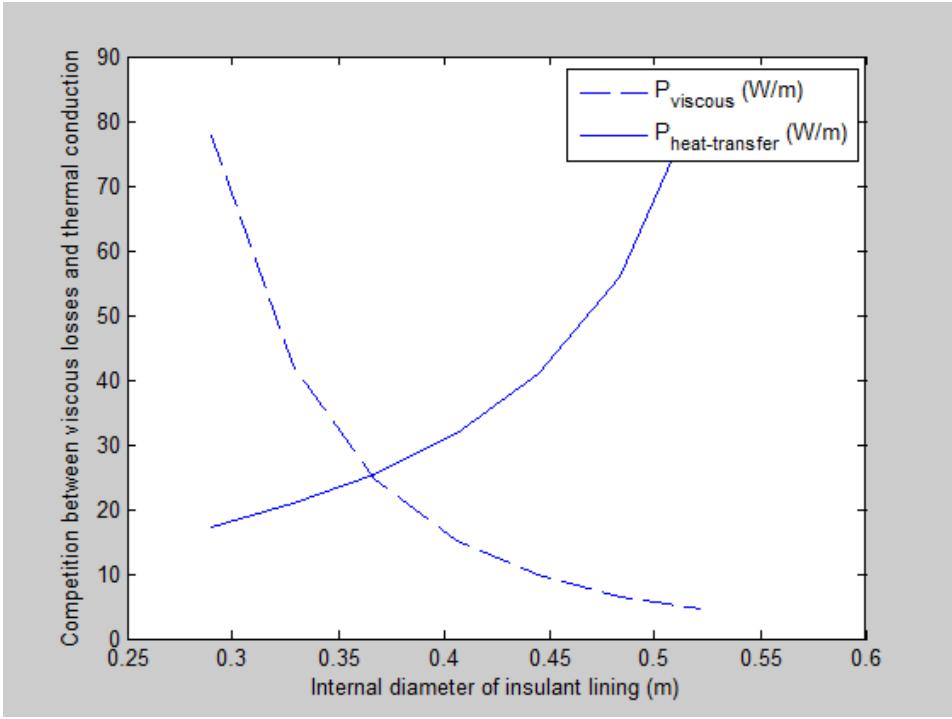
Plot the kinematic viscosity as a function of time using Simscape logging. Because the temperature change is now more modest, changes in viscosity are less significant.



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**.

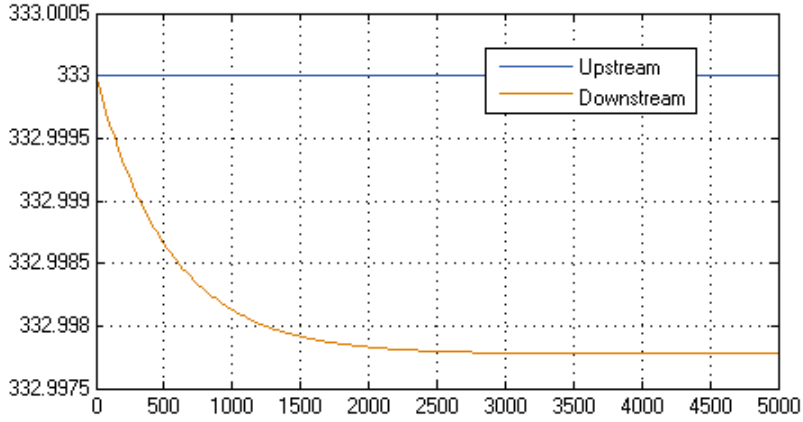


2 In the plot that opens, visually determine the horizontal-axis value for the intersection point between the two curves.

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.



See Also

More About

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-7
- “Thermal Liquid Modeling Framework” on page 2-11

3

Two-Phase Fluid Models

Manually Generate Fluid Property Tables

In this section...

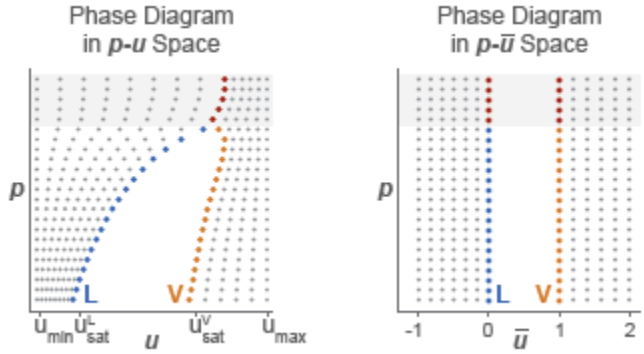
"Fluid Property Tables" on page 3-2
"Steps for Generating Property Tables" on page 3-3
"Before Generating Property Tables" on page 3-3
"Create Fluid Property Functions" on page 3-3
"Set Property Table Criteria" on page 3-4
"Create Pressure-Normalized Internal Energy Grids" on page 3-5
"Map Grids Onto 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 `twoPhaseFluidTables` 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 = vaporTemperature(u, p)
% Returns artificial temperature data as a function
% of specific internal energy and pressure.
T = -1000 + 0.6*u + 5*p;
```

- Name — `saturatedLiquidInternalEnergy`

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

- Name — `saturatedVaporInternalEnergy`

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

Set Property Table Criteria

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

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

% Minimum specific internal energy, kJ/kg
uMin = 30;
% Maximum specific internal energy, kJ/kg
uMax = 4000;
```

```
% Minimum pressure, MPa
pMin = 0.01;
% Maximum pressure, MPa
pMax = 15;

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

Create Pressure-Normalized Internal Energy Grids

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

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

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

Map Grids Onto Pressure-Specific Internal Energy Space

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

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

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

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

- `saturatedLiquidInternalEnergy`
- `saturatedVaporInternalEnergy`

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

```
% Map pressure-specific internal energy grid onto
% pressure-normalized internal energy space
fluidTables.liquid.u = (fluidTables.liquid.unorm + 1)*...
(fluidTables.liquid.u_sat - uMin) + uMin;
fluidTables.vapor.u = (fluidTables.vapor.unorm - 2)*...
(uMax - fluidTables.vapor.u_sat) + uMax;
```

Obtain Fluid Properties at Grid Points

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

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

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

- `liquidTemperature`
- `vaporTemperature`

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

```
fluidTables =

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

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

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

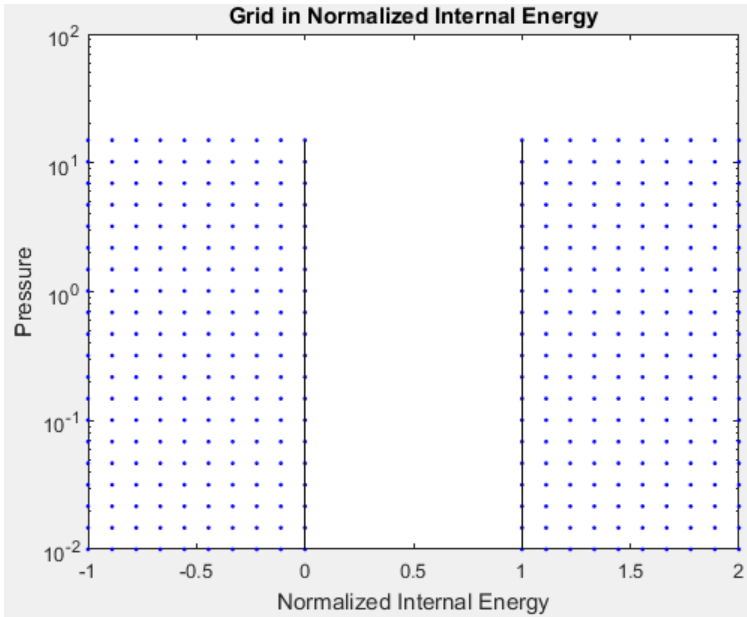
Visualize Grids

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

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

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

% Plot grid
figure;
hold on;

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

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

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

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

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

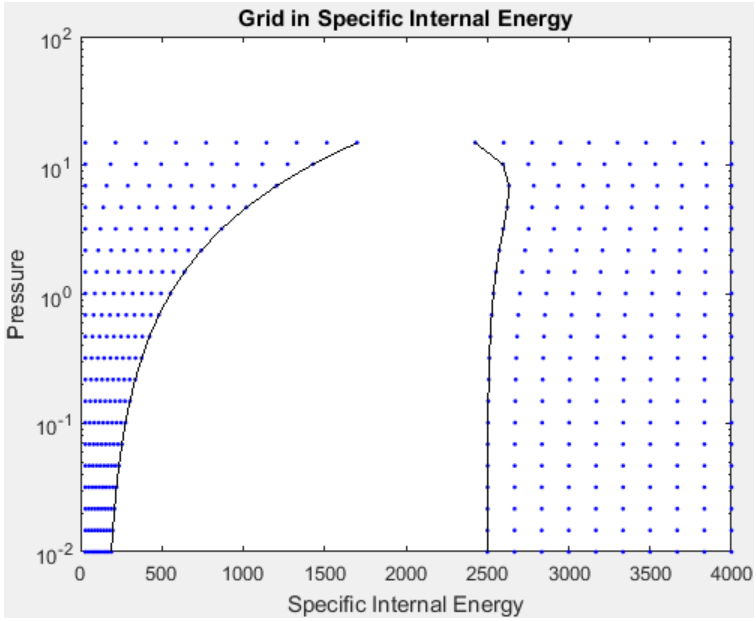


To visualize the transformed grid in pressure-specific internal energy space, at the MATLAB command prompt enter this code:

```
% Define horizontal and vertical axes

% Plot grid
figure;
hold on;

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

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

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

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



4

Gas System Models

- “Modeling Gas Systems” on page 4-2
- “Simple Gas Model” on page 4-14
- “Change Flow Boundary Conditions” on page 4-20

Modeling Gas Systems

In this section...
“Intended Applications” on page 4-2
“Network Variables” on page 4-2
“Gas Property Models” on page 4-3
“Blocks with Gas Volume” on page 4-5
“Reference Node and Grounding Rules” on page 4-6
“Initial Conditions for Blocks with Finite Gas Volume” on page 4-6
“Choked Flow” on page 4-7
“Flow Reversal” on page 4-13

Intended Applications

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

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

You specify the gas properties in the connected loop by using the Gas Properties (G) block. This block lets you choose between three idealization levels: perfect gas, semiperfect gas, or real gas (see “Gas Property Models” on page 4-3).

Network Variables

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

Gas Property Models

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

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

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

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

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

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

$$p = Z\rho RT$$

where:

- p is the pressure.
- Z is the compressibility factor.

- R is the specific gas constant.
- T is the temperature.

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

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

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

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

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

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

Blocks with Gas Volume

Components in the gas domain are modeled using control volumes. The control volume encompasses the gas inside the component and separates it from the surrounding environment and other components. Gas flows and heat flows across the control surface are represented by ports. The gas volume inside the component is represented using an internal node, which provides the gas pressure and temperature inside the component. This internal node is not visible, but you can access its parameters and variables using Simscape data logging. For more information, see About Simulation Data Logging on page 12-2.

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

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

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

Reference Node and Grounding Rules

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

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

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

Initial Conditions for Blocks with Finite Gas Volume

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

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

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

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

By default, **Pressure of gas volume** and **Temperature of gas volume** have high priority, with target values equal to the standard condition (`0.101325 MPa` and `293.15 K`). You can adjust the target values to represent the appropriate initial state of the gas volume for the block. **Density of gas volume** has the default priority `None` because only

the initial conditions of two of the three variables are needed to completely determine the initial state of the gas volume. If desired, an alternative way to specify the initial conditions is to change **Density of gas volume** to high priority with an appropriate target value, and then change either **Pressure of gas volume** or **Temperature of gas volume** to a priority of none.

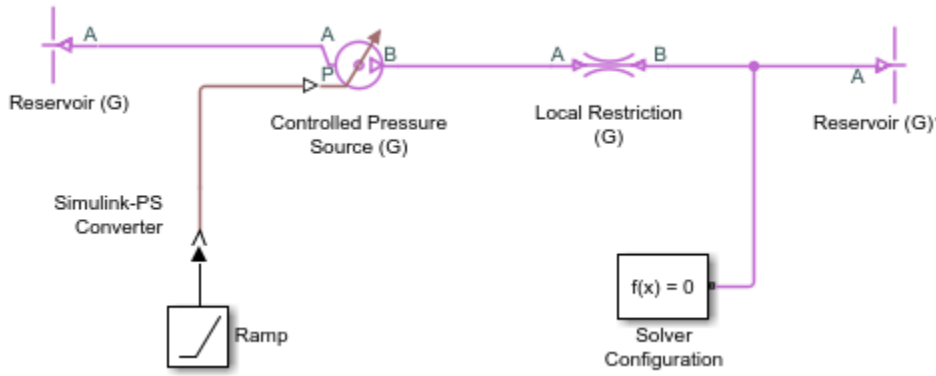
It is important that only two of the three variables have their priorities set to `High` for each block with a finite gas volume. Placing high-priority constraints on all three variables results in over-specification, with the solver unable to find an initialization solution that satisfies the desired initial values. Conversely, placing high-priority constraint only on one variable makes the system under-specified, and the solver might resolve the variables with arbitrary and unexpected initial values. For more information on variable initialization and dealing with over-specification, see “Initialize Variables for a Mass-Spring-Damper System” on page 7-7.

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

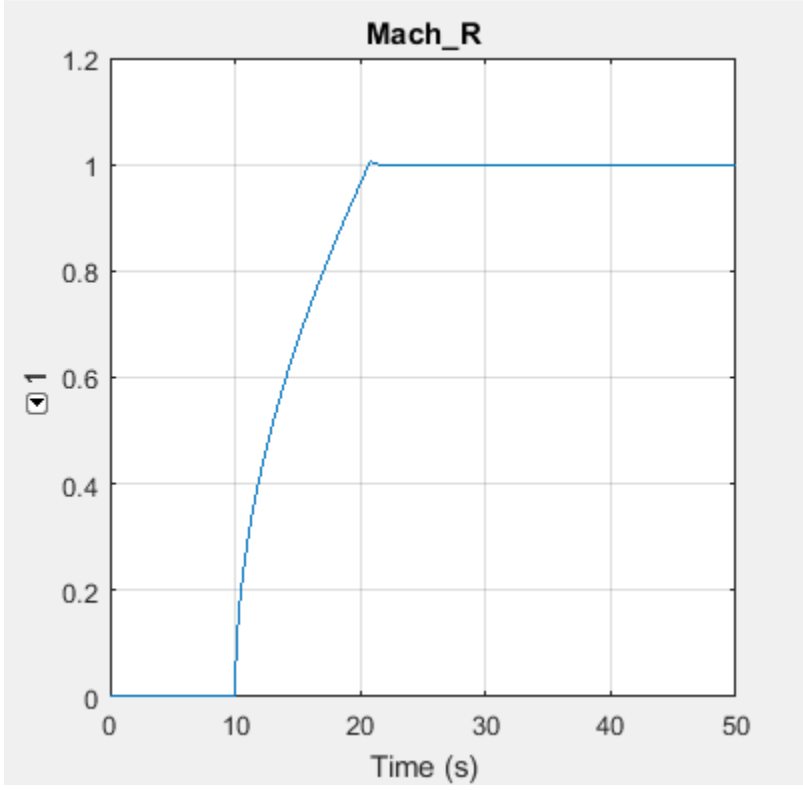
Choked Flow

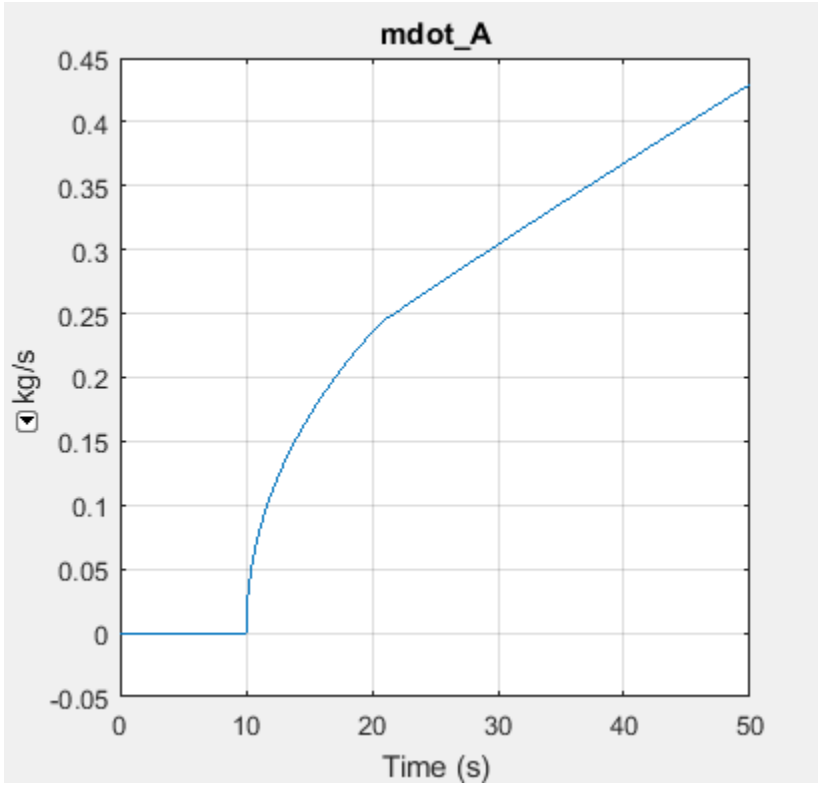
Gas flow through Local Restriction (G), Variable Local Restriction (G), or Pipe (G) blocks can become choked. Choking occurs when the flow velocity reaches the local speed of sound. When the flow is choked, the velocity at the point of choking cannot increase any further. However, the mass flow rate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

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

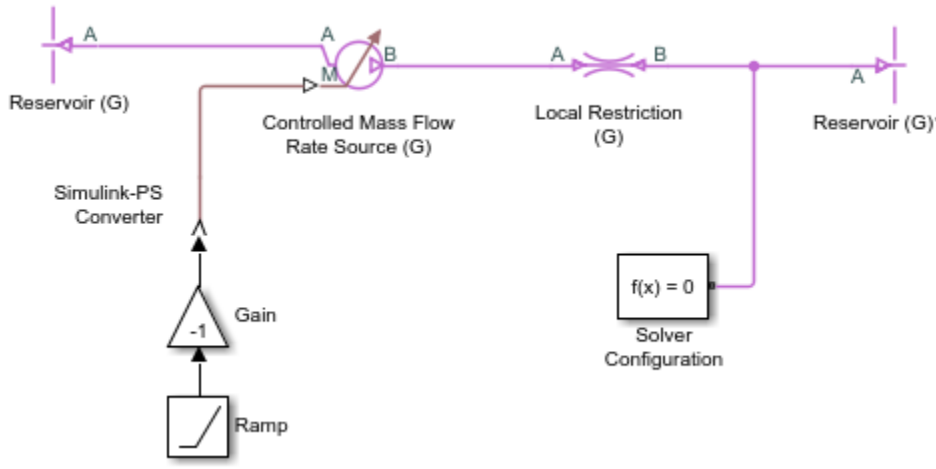


The following illustration shows the logged simulation data for the Local Restriction (G) block. The Mach number at the restriction (**Mach_R**) reaches 1 at around 20 s, indicating that the flow is choked. The mass flow rate (**mdot_A**) before the flow is choked follows the typical quadratic behavior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

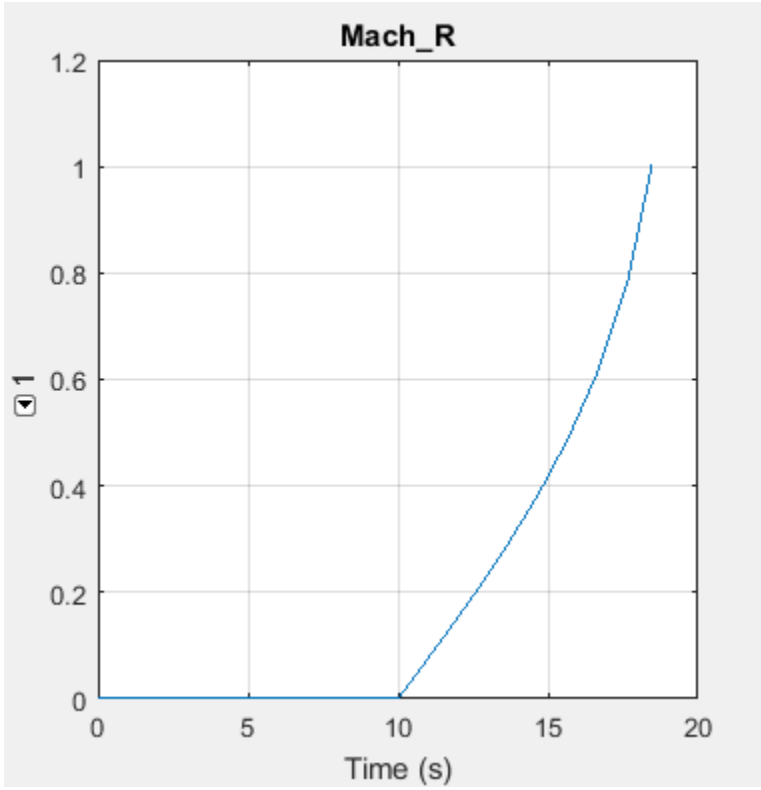

Mach_R
1.2
1
0.8
0.6
0.4
0.2
0
1
0
10
20
30
40
50
Time (s)



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



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



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

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

Flow Reversal

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

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

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

See Also

Related Examples

- “Simple Gas Model” on page 4-14
- “Change Flow Boundary Conditions” on page 4-20
- Choked Flow in Gas Orifice
- Pneumatic Actuation Circuit
- Pneumatic Motor Circuit

Simple Gas Model

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

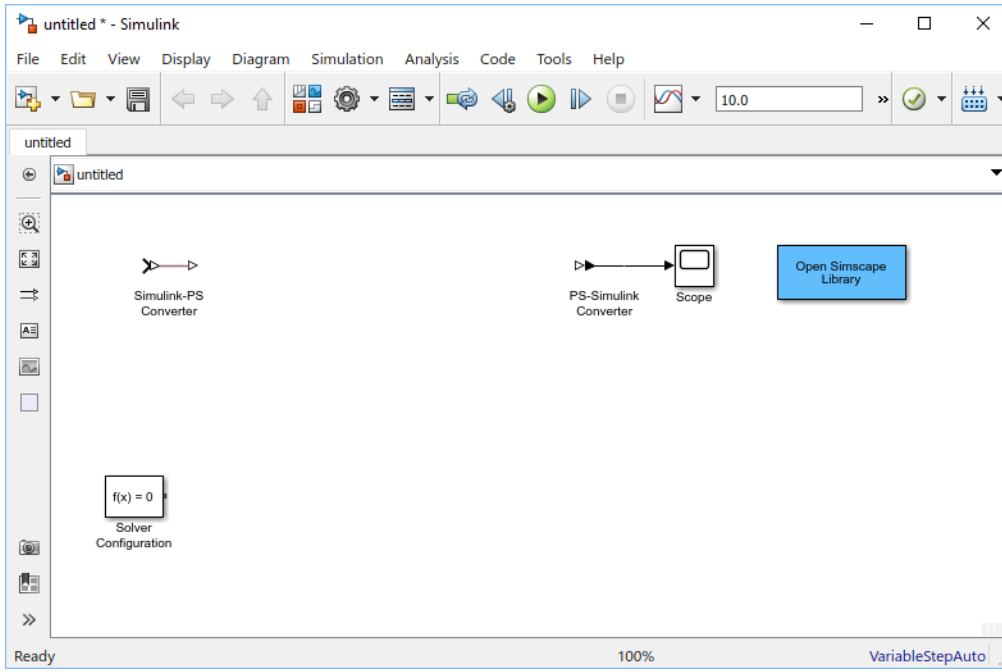
Reservoir blocks are useful for setting up pressure and temperature boundary conditions. If you want the pressure and temperature boundary conditions to change over time, use controlled reservoir blocks.

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

To create this model:

1. In the MATLAB Command Window, type:

   ```
   ssc_new
   ```



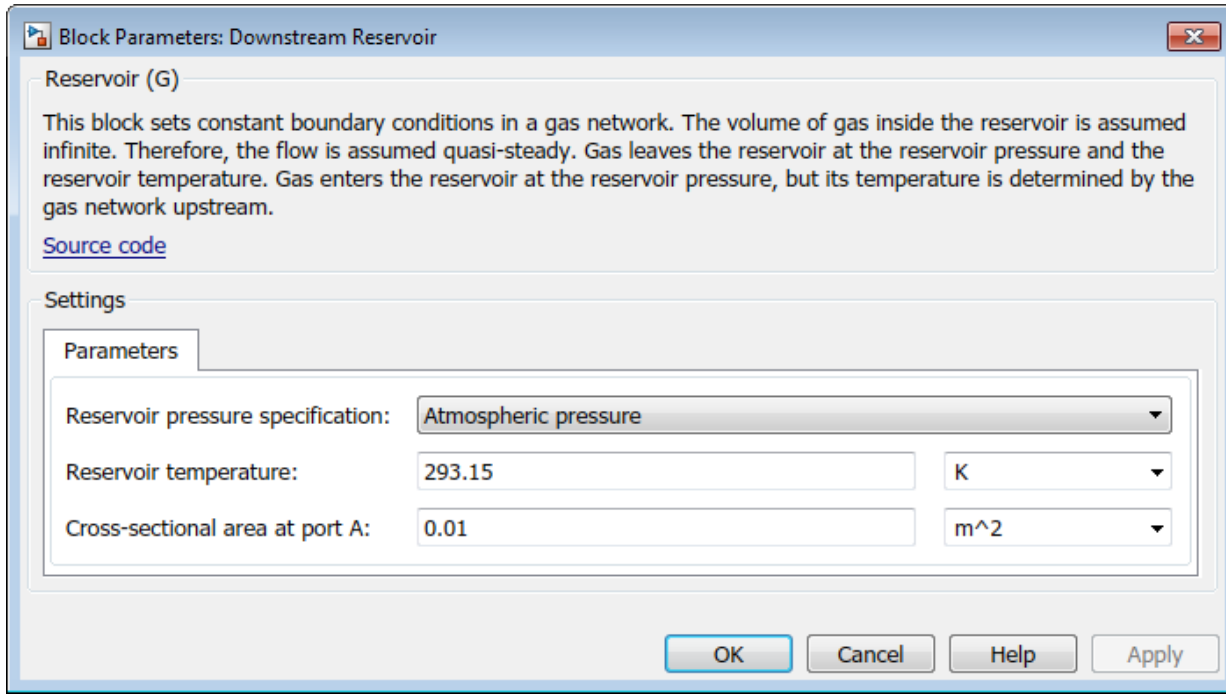
Note By default, Simulink Editor hides the automatic block names in model diagrams. To display hidden block names for training purposes, select **Display** and clear the **Hide Automatic Names** check box.

2. Delete the Simulink-PS Converter block and the Open Simscape Library block.
3. To reduce diagram clutter, right-click the PS-Simulink Converter block and, from the context menu, select **Format > Show Block Name > Off**.
4. Add the following blocks.

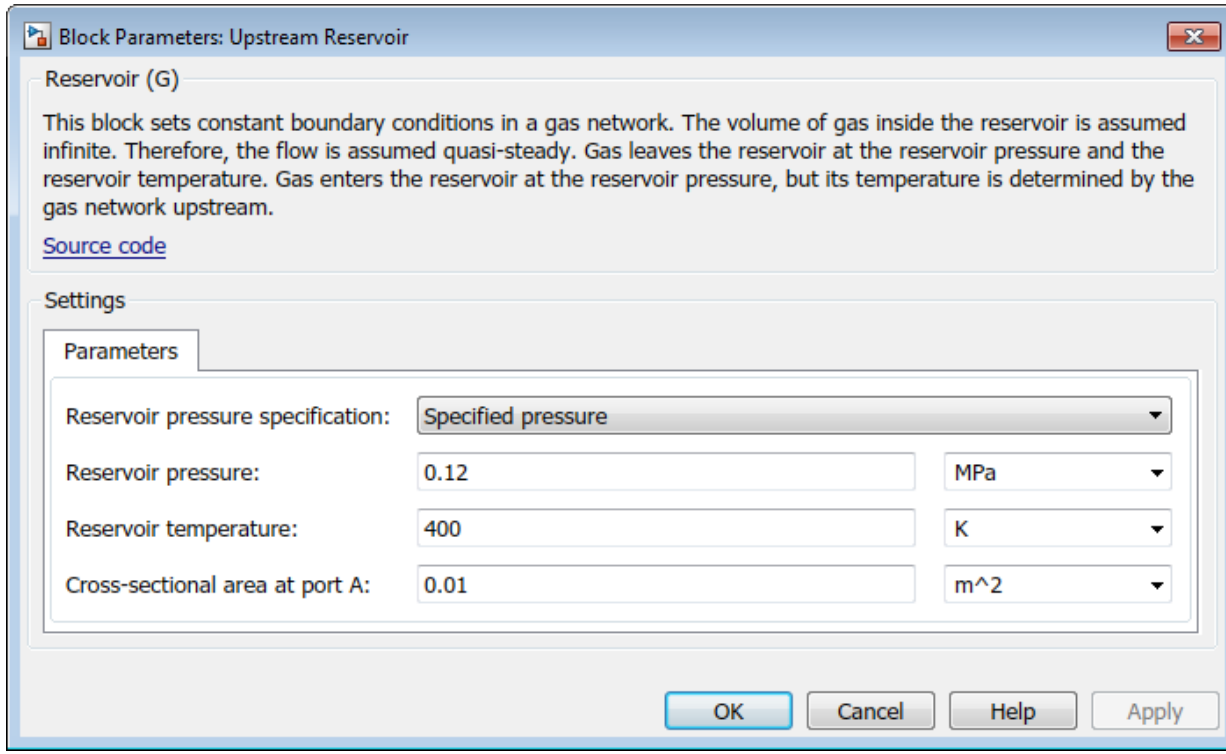
Block Name	Library	Quantity
Local Restriction (G)	Gas/Elements	1
Reservoir (G)	Gas/Elements	2
Gas Properties (G)	Gas/Utilities	1
Mass & Energy Flow Rate Sensor (G)	Gas/Sensors	1

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

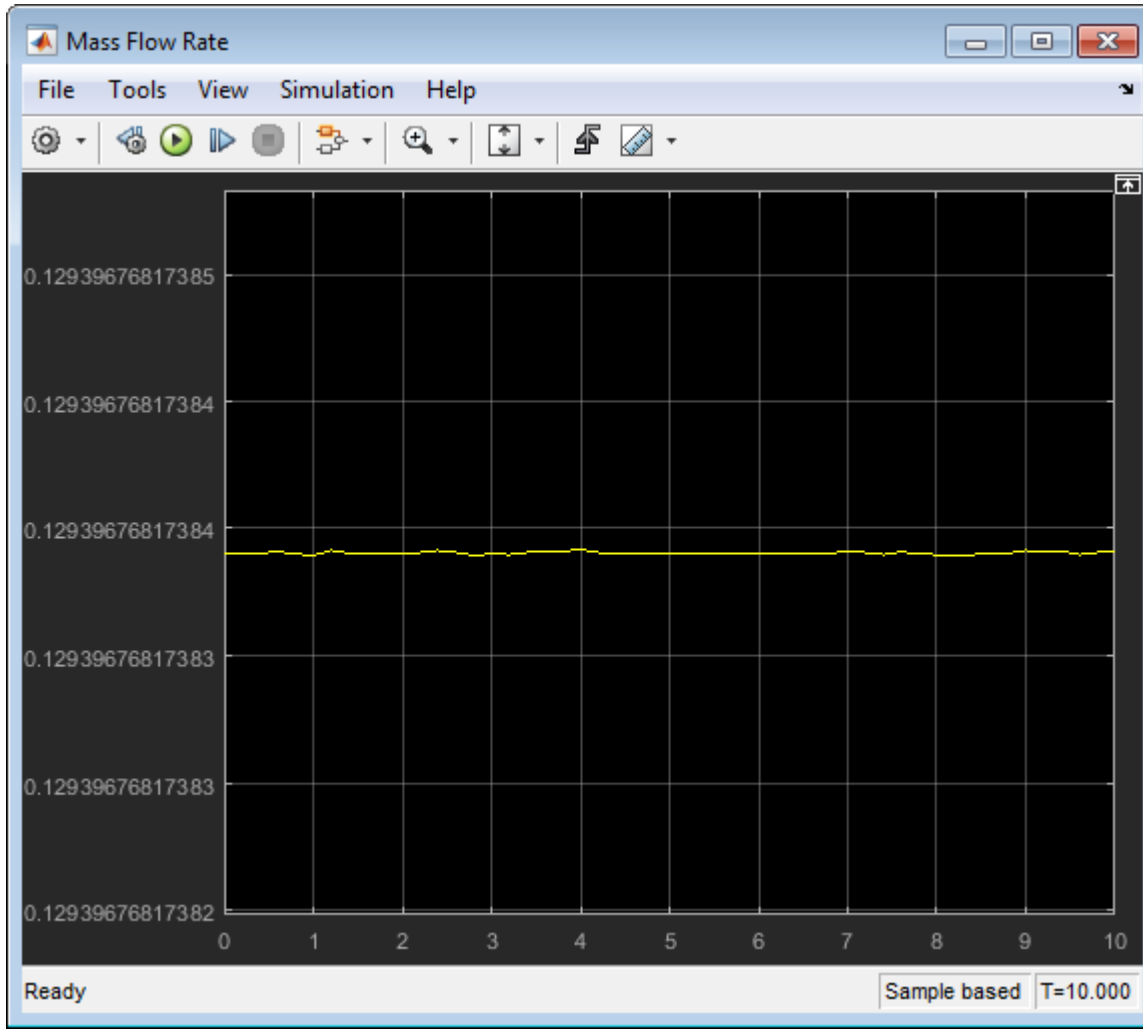
6. Leave the Downstream Reservoir block at standard atmospheric conditions.



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



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



See Also

Related Examples

- “Change Flow Boundary Conditions” on page 4-20

More About

- “Modeling Gas Systems” on page 4-2

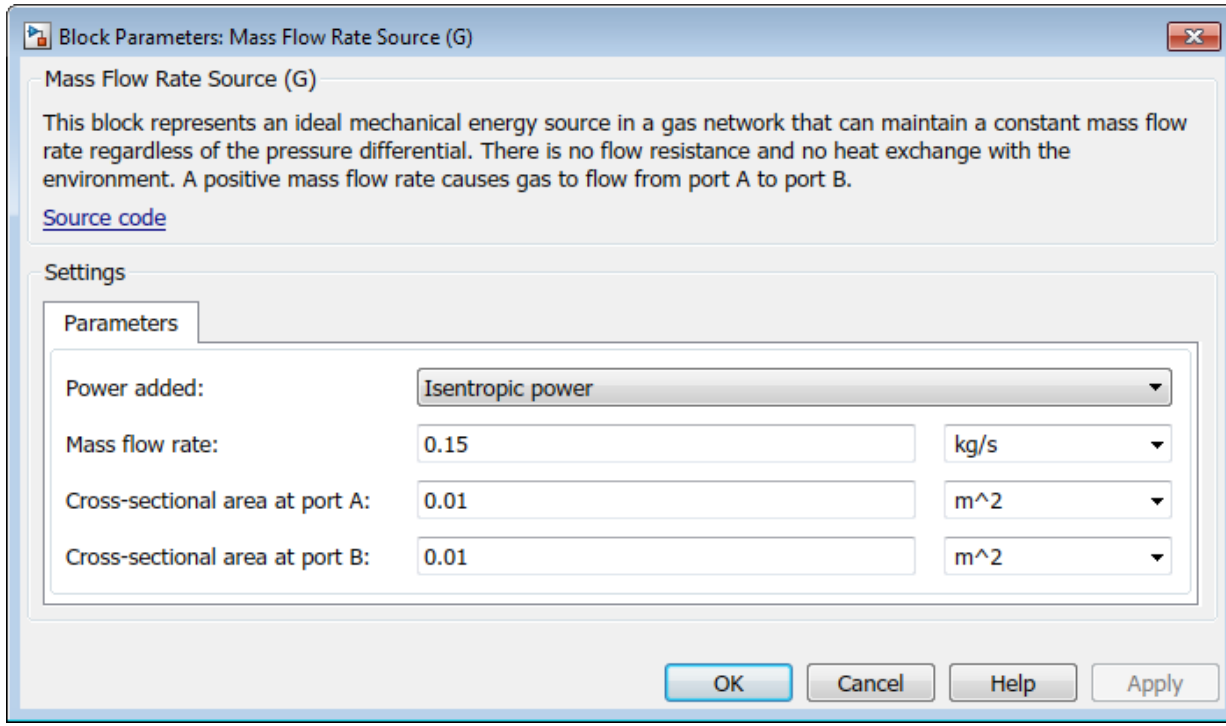
Change Flow Boundary Conditions

In the “Simple Gas Model” on page 4-14 tutorial, you created a simple open-loop gas model. This example shows how to modify this model by changing the gas flow boundary conditions without affecting temperature. To open the completed model, in the MATLAB Command Window, type `ssc_gas_tutorial_step2`.

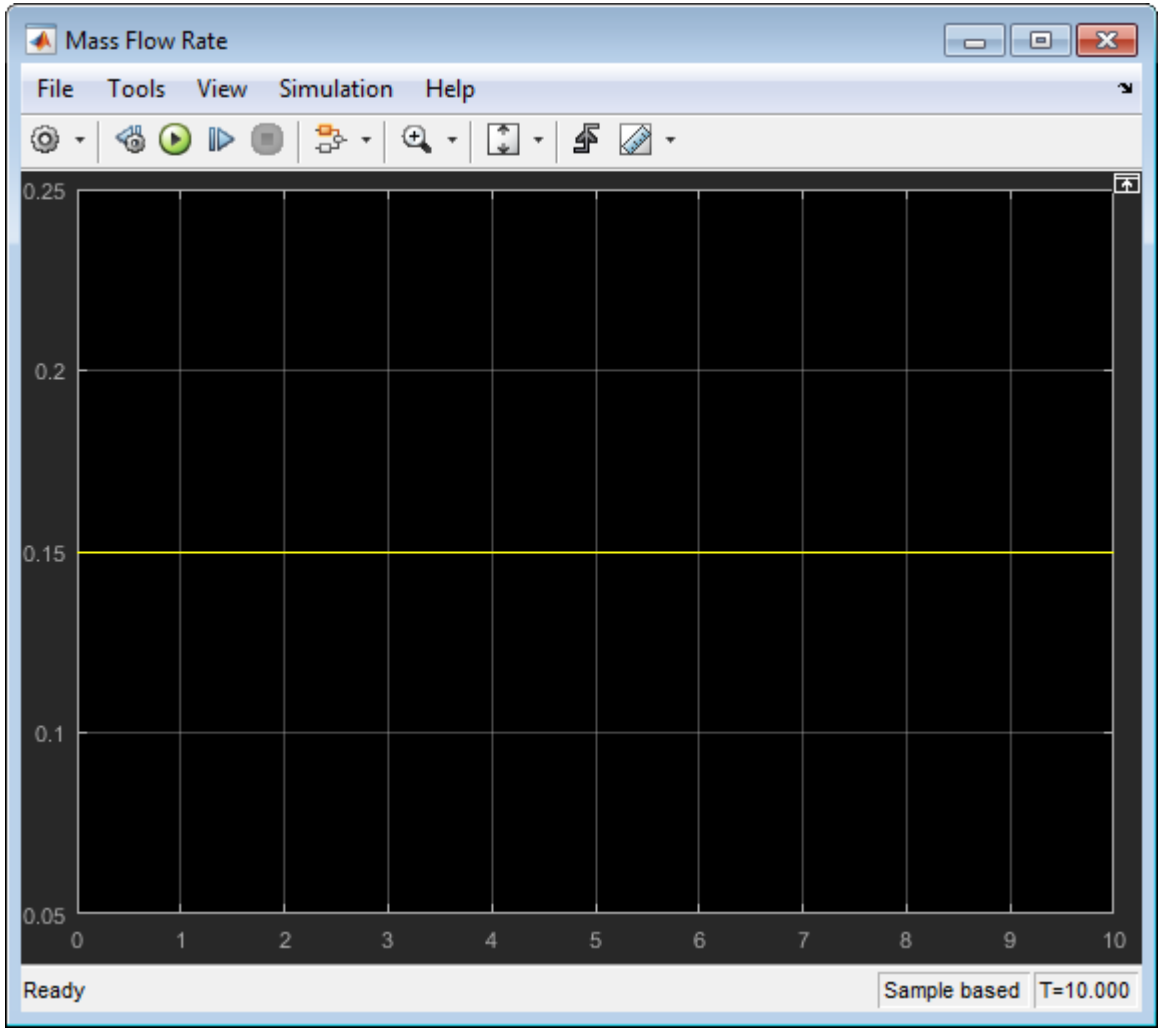
To change the upstream boundary conditions from specified pressure and temperature to specified mass flow rate and temperature:

1. Open the model created in the “Simple Gas Model” on page 4-14 tutorial, by typing `ssc_gas_tutorial_step1`.

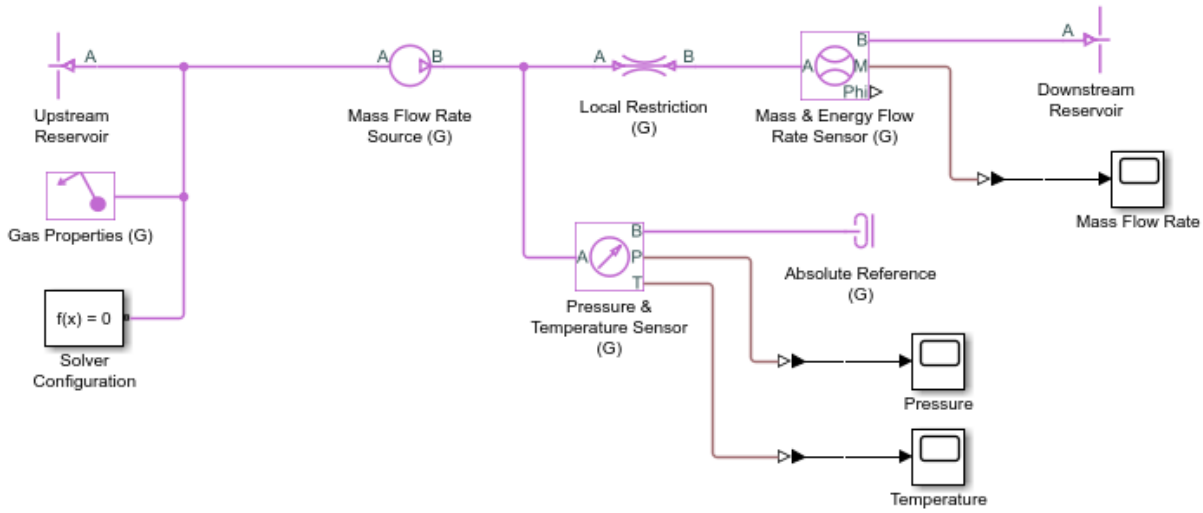
2. Change the Upstream Reservoir block back to `Atmospheric pressure`, but keep the temperature of 400 K.
3. Add a Mass Flow Rate Source (G) block upstream from the local restriction. Set the **Mass flow rate** parameter `0.15` kg/s.



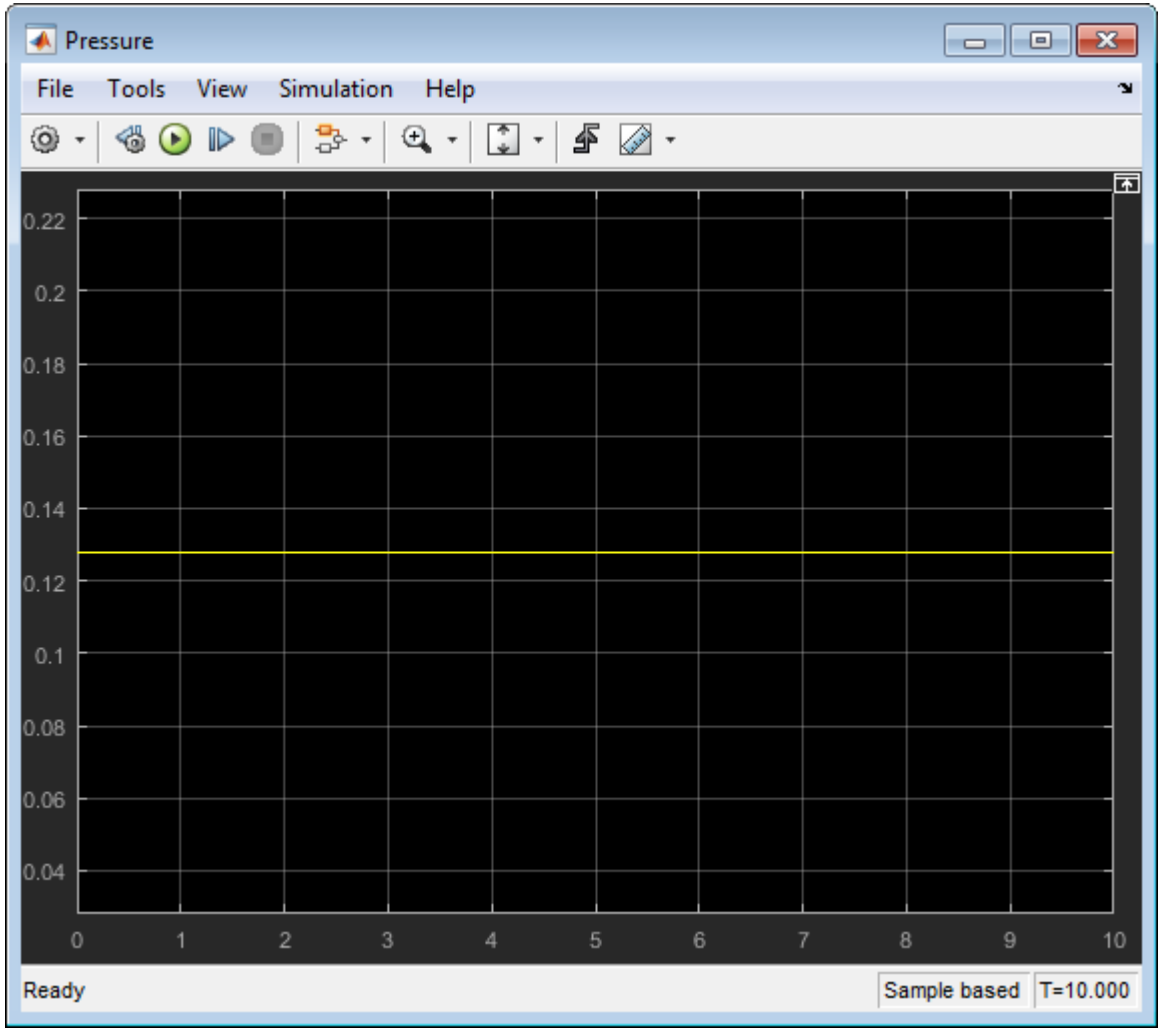
4 Simulate the model. The mass flow rate through the restriction is now 0.15 kg/s.



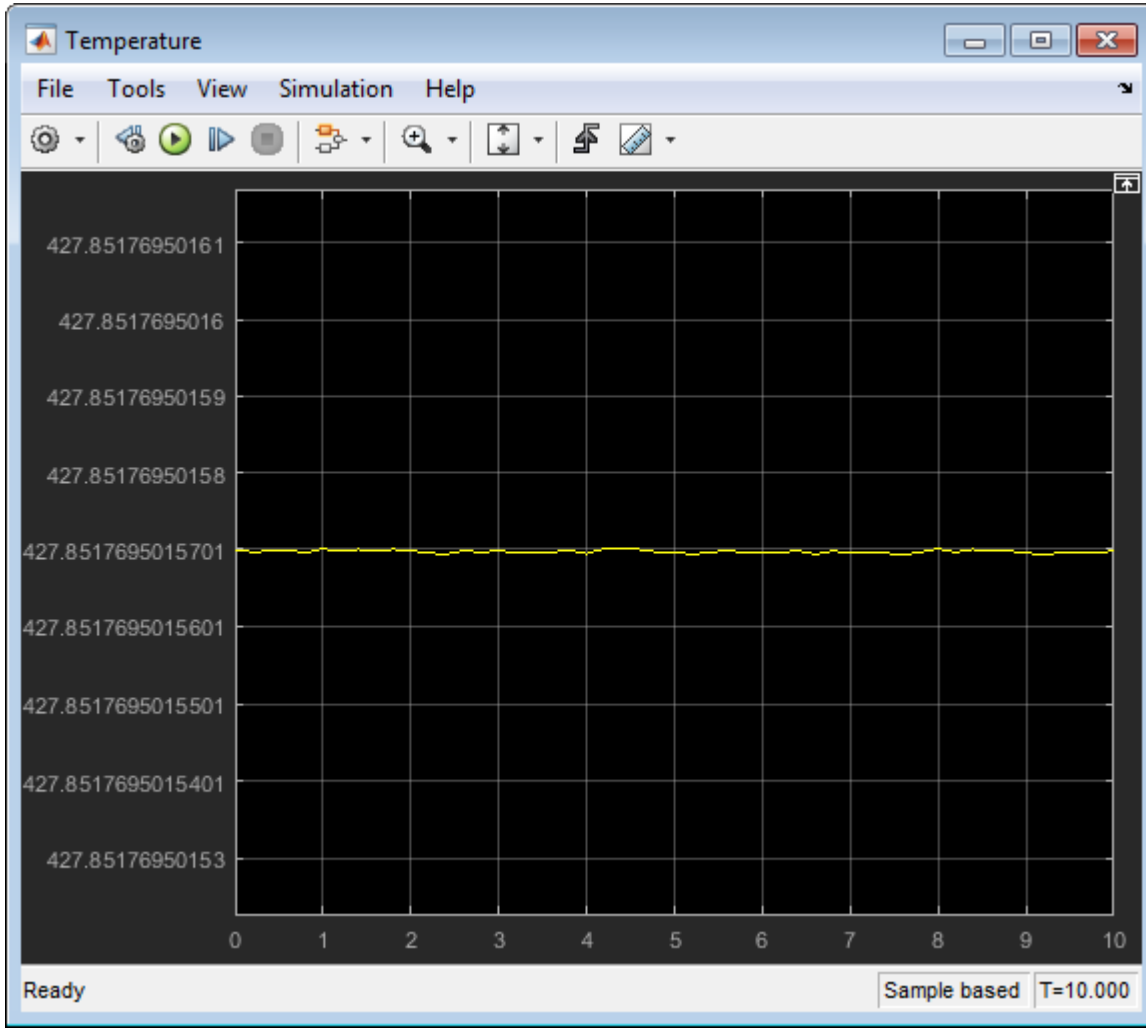
5 To measure the absolute pressure and temperature upstream of the local restriction, add a Pressure & Temperature Sensor (G) block and connect an Absolute Reference (G) block to the B node of the sensor. Duplicate the converter-scope block pair to add the Pressure and Temperature scopes to the model, as shown in the diagram.



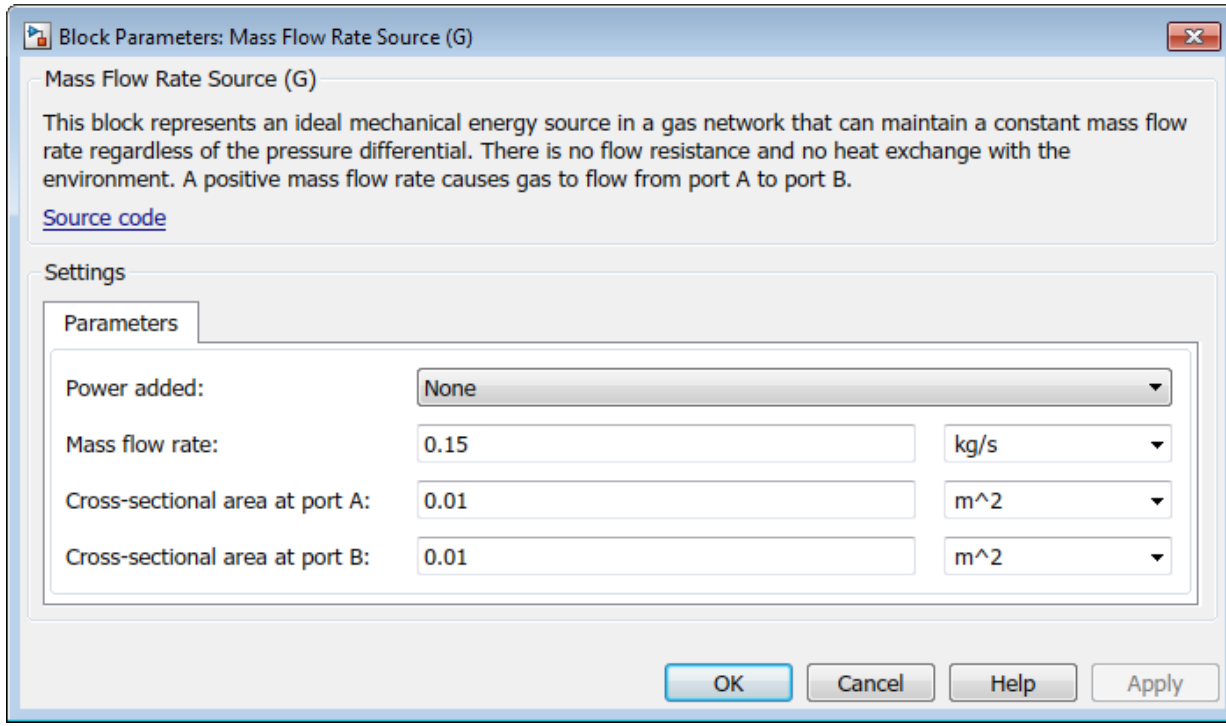
6 Simulate the model. To drive 0.15 kg/s of gas through the restriction, the Mass Flow Rate Source (G) block increased the pressure from atmospheric (as specified by the Upstream Reservoir block) to almost 0.13 MPa.



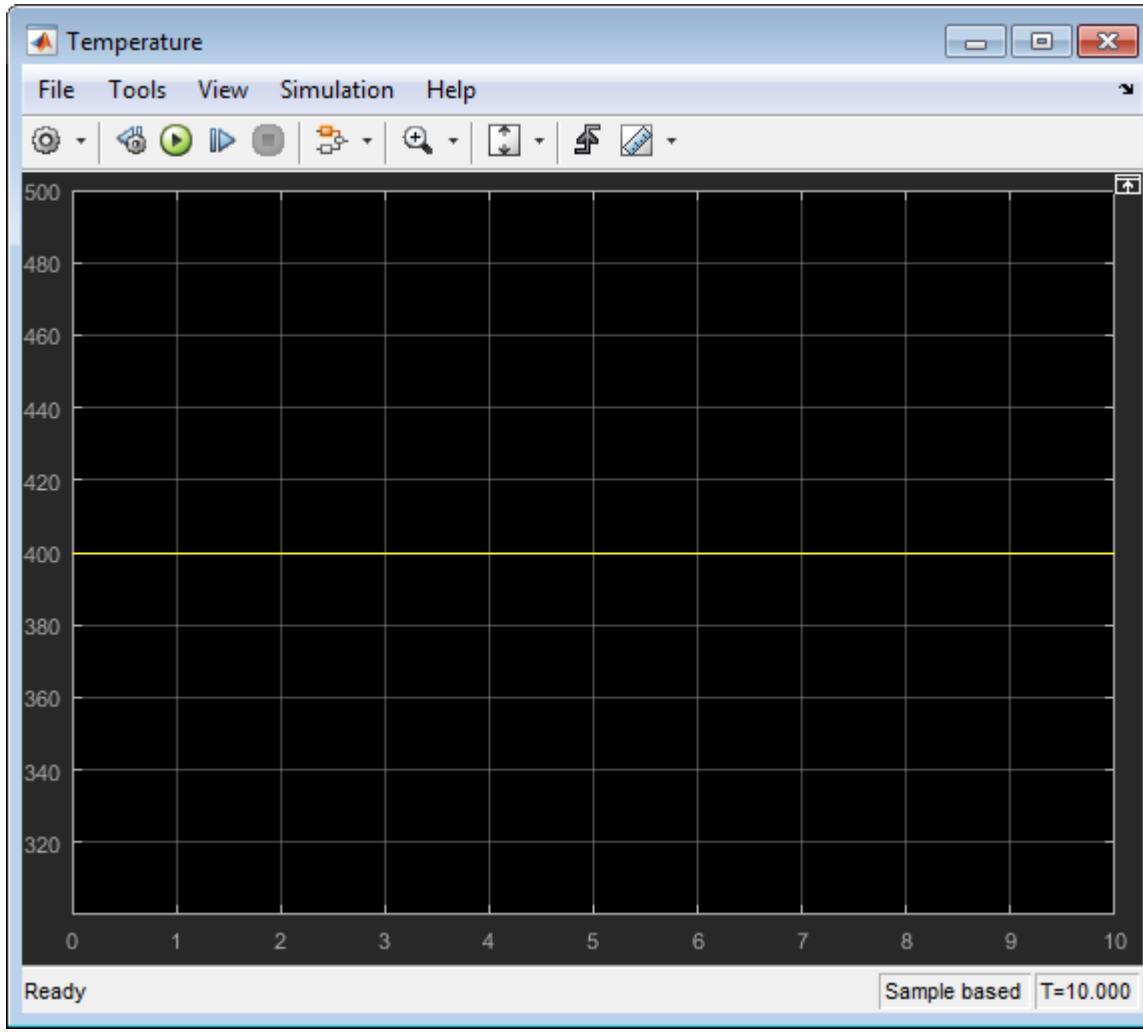
The temperature upstream of the restriction is approximately 427 K, not 400 K (as specified by the Upstream Reservoir block).



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



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



See Also

Related Examples

- “Simple Gas Model” on page 4-14

More About

- “Modeling Gas Systems” on page 4-2

5

Moist Air System Models

- “Modeling Moist Air Systems” on page 5-2
- “Modeling Moisture and Trace Gas Levels” on page 5-18

Modeling Moist Air Systems

In this section...
“Intended Applications” on page 5-2
“Network Variables for Moist Air Domain” on page 5-3
“Moist Air Properties” on page 5-3
“Humidity and Trace Gas Property Definitions” on page 5-5
“Blocks with Moist Air Volume” on page 5-8
“Reference Node and Grounding Rules” on page 5-8
“Initial Conditions for Blocks with Finite Moist Air Volume” on page 5-9
“Saturation and Condensation” on page 5-10
“Choked Flow” on page 5-12

Intended Applications

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

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

The moist air mixture is composed of dry air and water vapor. Trace gas is an optional third species in the moist air mixture. Example of the trace gas usage is to track carbon dioxide and pollutants such as nitrogen oxides (NOx). You specify the moist air properties in the connected loop by using the Moist Air Properties (MA) block. This block also gives you several options for modeling trace gas properties. You increase and decrease levels of moisture and trace gas in the air mixture by using the blocks in the Moisture & Trace Gas Sources library (see “Modeling Moisture and Trace Gas Levels” on page 5-18).

All gas species in the mixture are assumed to be semiperfect gas. This means that pressure, temperature, and density obey the ideal gas law. Other properties—specific

enthalpy, specific heat, dynamic viscosity, and thermal conductivity—are functions of temperature only.

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

- Develop requirements of an HVAC system for an environment, such as a building, automobile, or aircraft
- Ensure acceptable temperature, pressure, humidity, and condensation within the environment
- Determine capacity of an HVAC system to match heating, cooling, and dehumidification requirements
- Analyze HVAC system performance, efficiency, and cost
- Validate HVAC system model against test data
- Design and simulate HVAC components and tune component models to test rig data
- Simulate models including an HVAC system, environment model, and controller
- Design controllers for valves, fans, and compressors to ensure safe and optimal operation
- Perform HIL testing

Network Variables for Moist Air Domain

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

There is a separate domain for modeling moisture and trace gas levels in moist air systems. For more information, see “Moist Air Source Domain” on page 5-18.

Moist Air Properties

The default fluid properties for the moist air library correspond to dry air, water vapor, and carbon dioxide (the optional trace gas). However, you can modify the fluid properties in the Moist Air Properties (MA) block to model mixtures of other gases and vapors. You can replace dry air and carbon dioxide with other gas species. You can also change water vapor to another condensing vapor (or even to another noncondensing gas species, by

supplying large enough values for the saturation pressure, so that it would never reach saturation during simulation). In this way, you can model any three-species gas mixture.

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

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

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

Dalton's law applies to ideal gases:

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

Therefore, the mixture also obeys the ideal gas law:

$$p = \rho R T,$$

where:

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

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

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

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

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

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

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

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

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

Therefore, the mixture specific entropy is

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

Humidity and Trace Gas Property Definitions

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

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

Symbol	Property	Definition
φ_w	Relative humidity	Moles of water vapor as a fraction of the moles of water vapor needed to saturate at the same temperature. Water vapor saturation pressure is a property of water and is a function of temperature only, $p_{ws}(T)$. The ideal gas law (due to the assumed semiperfect gas) means that mole fraction is equivalent to partial pressure fraction. The mole fraction y_w cannot be greater than 1. Therefore, at high temperature or low pressure, it may not be possible to achieve relative humidity of 1. $$\varphi_w = \left.\frac{y_w}{y_{ws}}\right\|_T = \frac{y_w p}{p_{ws}(T)}$$
x_w	Specific humidity	Mass of water vapor as a fraction of the total mass of the moist air mixture. It is another term for water vapor mass fraction. It may not be possible to achieve specific humidity of 1 due to saturation. $$x_w = \frac{M_w}{M} = \frac{\dot{m}_w}{\dot{m}}$$
y_w	Water vapor mole fraction	Moles of water vapor as a fraction of the total moles of the moist air mixture. It may not be possible to achieve water vapor mole fraction of 1 due to saturation. $$y_w = \frac{N_w}{N} = \frac{p_w}{p}$$

Symbol	Property	Definition
r_w	Humidity ratio	Ratio of mass of water vapor to mass of dry air and trace gas. For conditions in typical HVAC applications, it is close to the specific humidity. $$r_w = \frac{M_w}{M_a + M_g}$$
ρ_w	Absolute humidity	Mass of water vapor over the volume of the moist air mixture. It is another term for water vapor density. $$\rho_w = \frac{M_w}{V}$$
x_g	Trace gas mass fraction	Mass of trace gas as a fraction of the total mass of the moist air mixture. $$x_g = \frac{M_g}{M} = \frac{\dot{m}_g}{\dot{m}}$$
y_g	Trace gas mole fraction	Moles of trace gas as a fraction of the total moles of the moist air mixture. $$y_g = \frac{N_g}{N} = \frac{p_g}{p}$$

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

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

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

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

Blocks with Moist Air Volume

Components in the moist air domain are modeled using control volumes. A control volume encompasses the moist air inside the component and separates it from the surrounding environment and other components. Air flows and heat flows across the control surface are represented by ports. The moist air volume inside the component is represented using an internal node. This internal node is not visible, but you can access its parameters and variables using Simscape data logging. For more information, see About Simulation Data Logging on page 12-2.

For a moist air volume, you must specify the pressure, temperature, moisture level, and trace gas level. For more information, see “Initial Conditions for Blocks with Finite Moist Air Volume” on page 5-9.

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

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

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

Reference Node and Grounding Rules

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

Blocks with a moist air volume contain an internal node, which provides the pressure, temperature, moisture level, and trace gas level inside the component and therefore serves as a reference node for the moist air network. Each connected moist air network must have at least one reference node. This means that each connected moist air network must have at least one of the blocks listed in “Blocks with Moist Air Volume” on page 5-8. In other words, a moist air network that contains no air volume is an invalid network.

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

Initial Conditions for Blocks with Finite Moist Air Volume

This section discusses the specific initialization requirements for blocks modeled with finite moist air volume. These blocks are listed in “Blocks with Moist Air Volume” on page 5-8.

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

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

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

It is important that only four variables, as described, have their priorities set to `High` for each block with a finite moist air volume. Placing high-priority constraints on additional variables results in overspecification, with the solver being unable to find an initialization solution that satisfies the desired initial values. Set the priority of remaining variables to `None`. You can use the equations in "Humidity and Trace Gas Property Definitions" on page 5-5 and "Trace Gas Modeling Options" on page 5-19 to convert values from one moisture or trace gas measure to another. For more information on variable initialization and dealing with overspecification, see "Initialize Variables for a Mass-Spring-Damper System" on page 7-7.

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

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

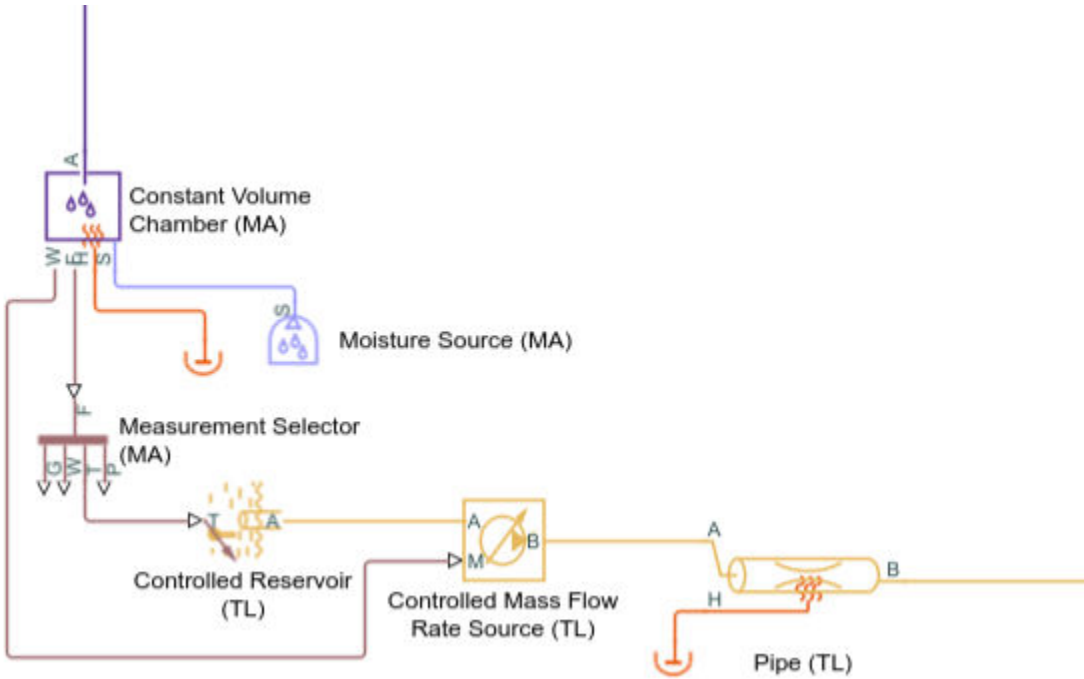
Saturation and Condensation

Blocks with finite moist air volume (listed in "Blocks with Moist Air Volume" on page 5-8) can become saturated when the relative humidity φ_w reaches the relative humidity at saturation φ_{ws}. The saturated state represents the maximum amount of moisture that the moist air volume can hold at that pressure and temperature. Any additional moisture condenses into liquid water.

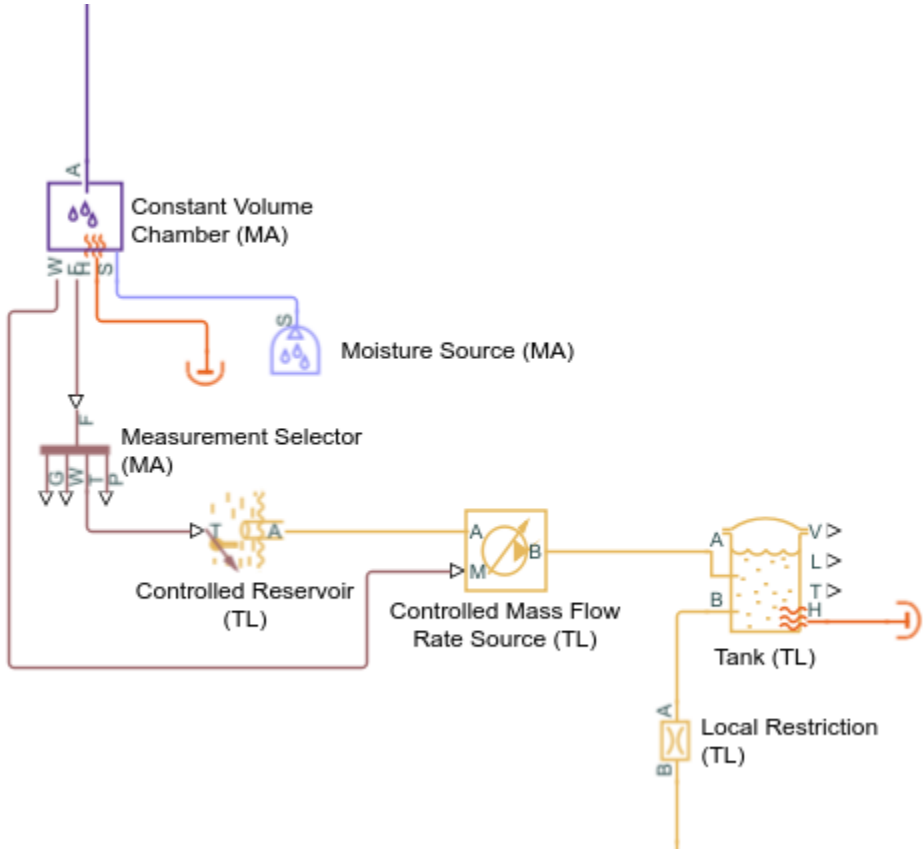
By definition, the relative humidity at saturation is 1. However, you can specify a different value for φ_{ws} to model some empirical effect or other phenomenon. When $\varphi_{ws} > 1$, water vapor partial pressure can become greater than the water vapor saturation pressure. When $\varphi_{ws} < 1$,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

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

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



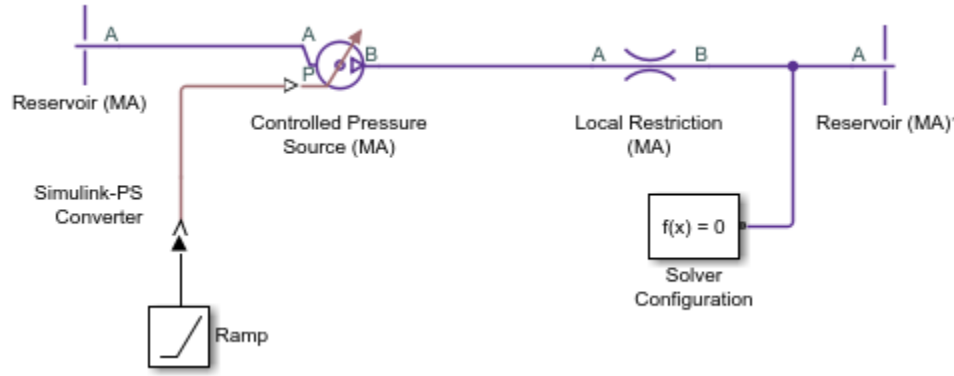
If you have a Simscape Fluids license, you can also use the Tank (TL) block to model a condensation collection tray. The liquid level in the tank represents the amount of condensation collected but not yet drained from the tank.



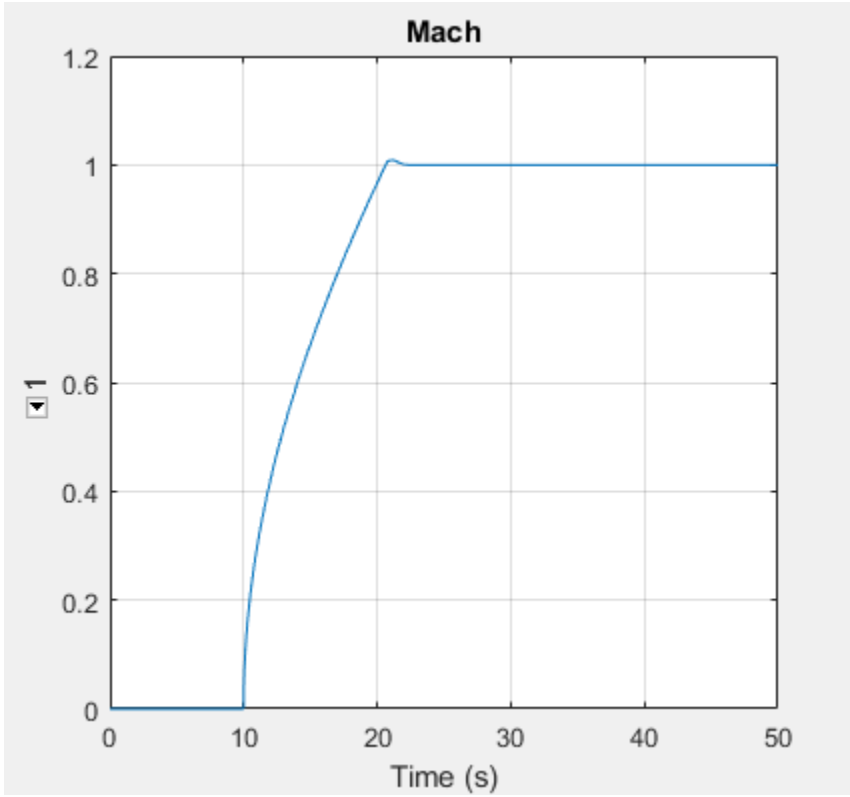
Choked Flow

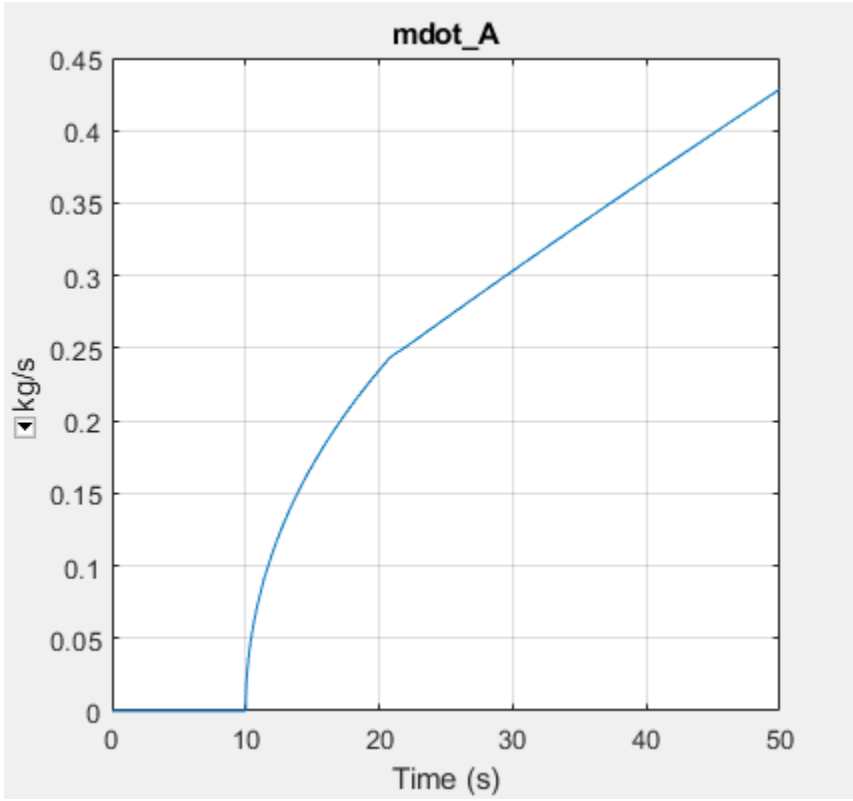
Moist air flow through Local Restriction (MA), Variable Local Restriction (MA), or Pipe (MA) blocks can become choked. Choking occurs when the flow velocity reaches the local speed of sound. When the flow is choked, the velocity at the point of choking cannot increase any further. However, the mass flow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

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

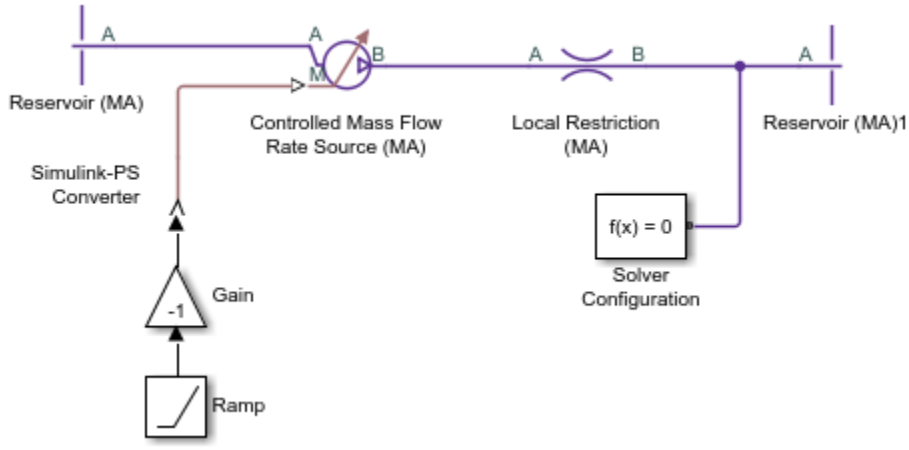


The following two plots show the logged simulation data for the Local Restriction (MA) block. The Mach number at the restriction (**Mach**) reaches 1 at around 20 s, indicating that the flow is choked. The mass flow rate (**mdot_A**) before the flow is choked follows the typical quadratic behavior with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

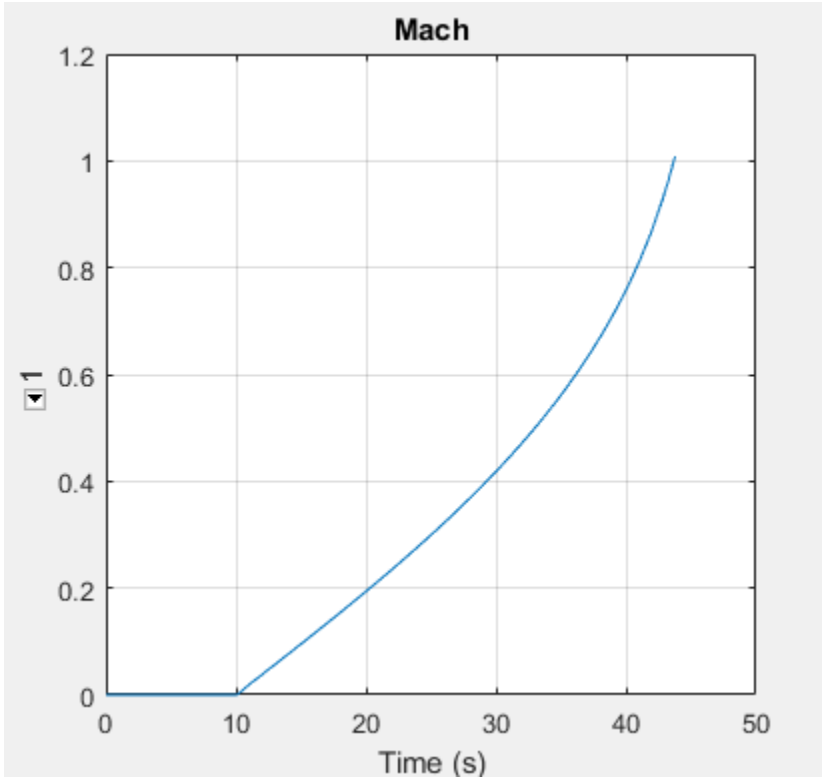

Mach
1.2
1
0.8
0.6
0.4
0.2
0
1
0
10
20
30
40
50
Time (s)



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



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



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

See Also

More About

- “Modeling Moisture and Trace Gas Levels” on page 5-18

Modeling Moisture and Trace Gas Levels

In this section...
“Moist Air Source Domain” on page 5-18
“Trace Gas Modeling Options” on page 5-19
“Using Moisture and Trace Gas Sources” on page 5-20
“Measuring Moisture and Trace Gas Levels” on page 5-21

Moist Air Source Domain

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

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

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

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

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

Blocks in the Moisture & Trace Gas Sources library, which inject or extract moisture and trace gas, also have a port **S** that belongs to the moist air source domain. This naming convention helps to distinguish the two different domain types. For more information, see “Using Moisture and Trace Gas Sources” on page 5-20.

Trace Gas Modeling Options

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

- `None` — No trace gas is present. The moist air mixture consists only of dry air and water vapor.
- `Track mass fraction only` — The trace gas level can be nonzero and vary during simulation. However, the amount of trace gas is assumed to be small enough to have a negligible impact on the fluid properties of the moist air mixture.
- `Track mass fraction and gas properties` — The trace gas level can be nonzero and vary during simulation. The fluid properties of the moist air mixture depend on the amount of trace gas in the mixture.

Trace Gas — None

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

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

Trace Gas — Track mass fraction only

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

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

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

Trace Gas — Track mass fraction and gas properties

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

Using Moisture and Trace Gas Sources

Moisture and trace gas can be injected into blocks with finite moist air volume. (For a complete list of these blocks, see “Blocks with Moist Air Volume” on page 5-8.) For example, adding moisture to a Constant Volume Chamber (MA) block can represent respiration of occupants in a room. Moisture and trace gas can also be extracted from these blocks. For example, removing trace gas from a Pipe (MA) block can represent a filter that extracts CO2 from moist air flow.

Use the blocks in the Moisture & Trace Gas Sources library to inject or extract moisture and trace gas. The moisture and trace gas sources can be connected only to the conserving port **S** of the blocks with moist air volume, which is the port associated with the moist air source domain. For more information, see “Moist Air Source Domain” on page 5-18.

You can connect multiple moisture and trace gas sources to the same port **S** of blocks with moist air volume. For example, a Constant Volume Chamber (MA) block can have all of these sources connected to its port **S**:

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

If no moisture or trace gas is injected or extracted from a block with moist air volume, connect a Moisture & Trace Gas Cap (MA) block to the unused port **S**.

Note Using moisture source blocks to remove moisture is a different effect than condensation. Condensation occurs independently whenever $\varphi_w \geq \varphi_{ws}$. For more information, see “Saturation and Condensation” on page 5-10.

Measuring Moisture and Trace Gas Levels

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

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

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

where:

- *p_I* is the pressure of the moist air volume, in Pa
- *T_I* is the temperature of the moist air volume, in K
- *RH_I* is the relative humidity of the moist air volume
- *x_w_I* is the specific humidity of moist air volume
- *y_w_I* is the water vapor mole fraction of the moist air volume
- *HR_I* is the humidity ratio of the moist air volume
- *x_g_I* is the trace gas mass fraction of the moist air volume
- *y_g_I* is the trace gas mole fraction of the moist air volume

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

See Also

More About

- “Modeling Moist Air Systems” on page 5-2

Model Simulation

- “How Simscape Models Represent Physical Systems” on page 6-2
- “How Simscape Simulation Works” on page 6-7
- “Setting Up Solvers for Physical Models” on page 6-13
- “Important Concepts and Choices in Physical Simulation” on page 6-18
- “Making Optimal Solver Choices for Physical Simulation” on page 6-22
- “Filtering Input Signals and Providing Time Derivatives” on page 6-27
- “System Scaling by Nominal Values” on page 6-30
- “Frequency and Time Simulation Mode” on page 6-37
- “Troubleshooting Simulation Errors” on page 6-45
- “Limitations” on page 6-52

How Simscape Models Represent Physical Systems

In this section...
“Representations of Physical Systems” on page 6-2
“Differential, Differential-Algebraic, and Algebraic Systems” on page 6-2
“Stiffness” on page 6-3
“Events and Zero Crossings” on page 6-3
“Working with Simscape Representation” on page 6-3
“Managing Zero Crossings in Simscape Models” on page 6-4
“References” on page 6-5

Representations of Physical Systems

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

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

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

Differential, Differential-Algebraic, and Algebraic Systems

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

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

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

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

Stiffness

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

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

Events and Zero Crossings

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

A *zero crossing* is a specific event type, represented by the value of a mathematical function changing sign. Variable-step solvers take smaller steps when they detect a zero-crossing event. Smaller steps help to capture the dynamics that cause the zero crossing, but they also significantly slow down the simulation. Various methods of zero crossing detection and analysis help you strike the right balance between the simulation speed and accuracy. For more information, see “Managing Zero Crossings in Simscape Models” on page 6-4.

Working with Simscape Representation

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

- Start by assuming that your physical network is a DAE system: a mix of differential and algebraic equations and variables on page 6-2.

  Remember that some physical networks are represented by ODEs only.
- Physical networks may contain stiff differential equations on page 6-3.
- Identify discrete and continuous components that might change discontinuously on page 6-3 during a simulation.

Managing Zero Crossings in Simscape Models

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

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

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

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

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

Detecting and Minimizing Zero Crossings in Simscape Models

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

- Prior to simulation, you can use the Statistics Viewer to identify the potential zero-crossing signals in the model. These signals are typically generated from operators and functions that contain discontinuities, such as comparison operators, `abs`, `sqrt` functions, and so on. During simulation it is possible for none of these signals to produce a zero-crossing event or for one or more of these signals to have multiple zero-crossing events. For more information, see “View Model Statistics” on page 13-14.

- When logging simulation data for a model, you can select the **Log simulation statistics** option. The data log then includes the actual zero-crossing data during simulation. For more information, see “Log Simulation Statistics” on page 12-15.

  You can access and analyze zero-crossing data logged during simulation by using the Simscape Results Explorer. For more information, see “About the Simscape Results Explorer” on page 12-30.
- The `sscprintzcs` function prints information about zero crossings detected during simulation, based on logged simulation data. Before you call this function, you must have the simulation log variable, which includes simulation statistics data, in your current workspace. For more information and examples, see `sscprintzcs`.

Managing zero crossing is especially important when you prepare your models for real-time simulation. See “Reduce Zero Crossings” on page 10-73 for a detailed example of this workflow.

Enabling and Disabling Zero-Crossing Conditions in Simscape Language

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

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

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

References

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

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

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

How Simscape Simulation Works

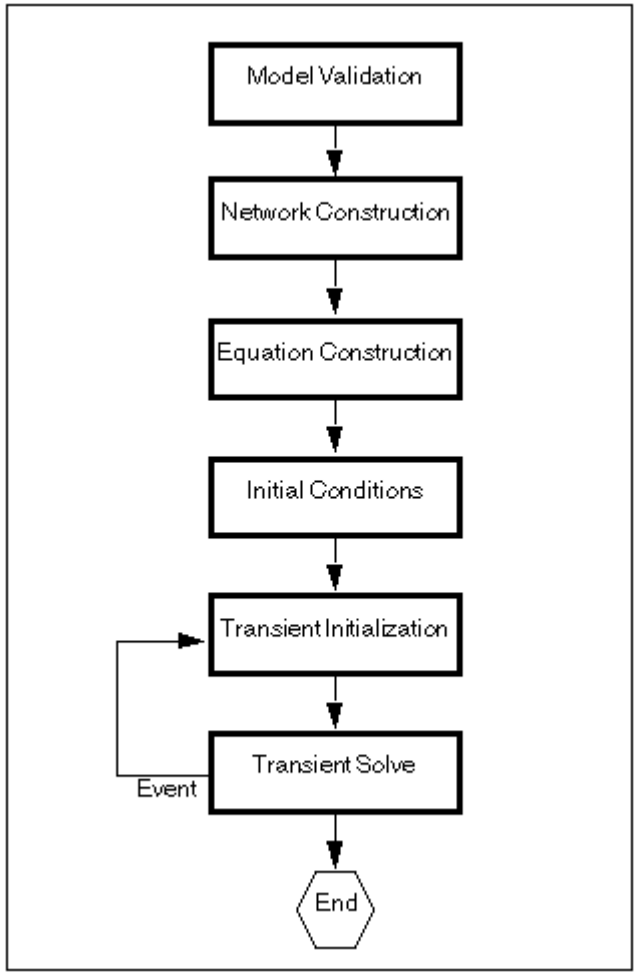
In this section...
“Simscape Simulation Phases” on page 6-7
“Model Validation” on page 6-9
“Network Construction” on page 6-9
“Equation Construction” on page 6-10
“Initial Conditions Computation” on page 6-10
“Transient Initialization” on page 6-11
“Transient Solve” on page 6-12

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 6-2.

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 6-9
2. “Network Construction” on page 6-9
3. “Equation Construction” on page 6-10
4. “Initial Conditions Computation” on page 6-10
5. “Transient Initialization” on page 6-11

6 “Transient Solve” on page 6-12

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 Simscape Fluids 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 gas elements, default gas properties are for dry air. If you attach a Gas Properties (G) block to a topologically distinct circuit, you can change gas properties for all the blocks connected to the circuit. However, more than one Gas Properties (G) block in a 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 outport.

Network Construction

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

- Two directly connected Conserving ports have the same values for all their Across variables (such as voltage or angular velocity).
- Any Through variable (such as current or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches.

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

Equation Construction

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

These equations contain system variables of the following types:

- Dynamic — Time derivatives of these variables appear in equations. Dynamic, or differential, variables add dynamics to the system and require the solver to use numerical integration to compute their values. Dynamic variables can produce either independent or dependent states for simulation.
- Algebraic — Time derivatives of these variables do not appear in equations. These variables appear in algebraic equations but add no dynamics, and this typically occurs in physical systems due to conservation laws, such as conservation of mass and energy. The states of algebraic variables are always dependent on dynamic variables, other algebraic variables, or inputs.

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

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

Initial Conditions Computation

The Simscape solver computes the initial conditions only once, at the beginning of simulation ($t = 0$). In the Solver Configuration block 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 6-11.

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. You can also initialize variables for a whole model from a saved operating point.

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

- At first, the solver tries to find a solution where all the high-priority variable targets are met exactly, and the low-priority targets are approximated as closely as possible. If the solution is found during this stage, it satisfies all the high-priority targets. Some of the low-priority targets might also be met exactly, the others are approximated.
- If the solver cannot find a solution that exactly satisfies all the high-priority targets, it issues a warning and enters the second stage, where `High` priority is relaxed to `Low`. That is, the solver tries to find a solution by approximating both the high-priority and the low-priority targets as closely as possible.

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

Finding an Initial Steady State

When you select the **Start simulation from steady state** check box, 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 6-13
“Choosing Simulink and Simscape Solvers” on page 6-13
“Harmonizing Simulink and Simscape Solvers” on page 6-15

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 6-22.

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 6-15.

- “Working with Global Simulink Solvers” on page 6-13
- “Working with Local Simscape Solvers” on page 6-14

Working with Global Simulink Solvers

In the Configuration Parameters dialog box of your model, on the **Solver** pane, the solver and related settings that you select are global choices. For more information, see “Solver Types” (Simulink).

When you first create a model, the default Simulink solver is `VariableStepAuto`. For more information, see “Use Auto Solver to Select a Solver” (Simulink). 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.
- You can also select from among explicit and implicit solvers. 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 6-16.

- If all the Simulink and Simscape states in your model are discrete, Simulink automatically switches to a discrete solver and issues a warning. Otherwise, a continuous solver is the default.
- By default, Simulink variable-step solvers attempt to locate events in time by zero-crossing detection. See “Managing Zero Crossings in Simscape Models” on page 6-4.

Working with Local Simscape Solvers

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

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

Choosing Local Solvers and Sample Times

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

Choosing Fixed-Cost Simulation

You can select a fixed-cost simulation for one or more physical networks by selecting **Use fixed-cost runtime consistency iterations**, as well as **Use local solver**, and fixing the number of nonlinear and mode iterations. Fixed-cost simulation requires a global fixed-step solver.

Choosing Multirate Simulation

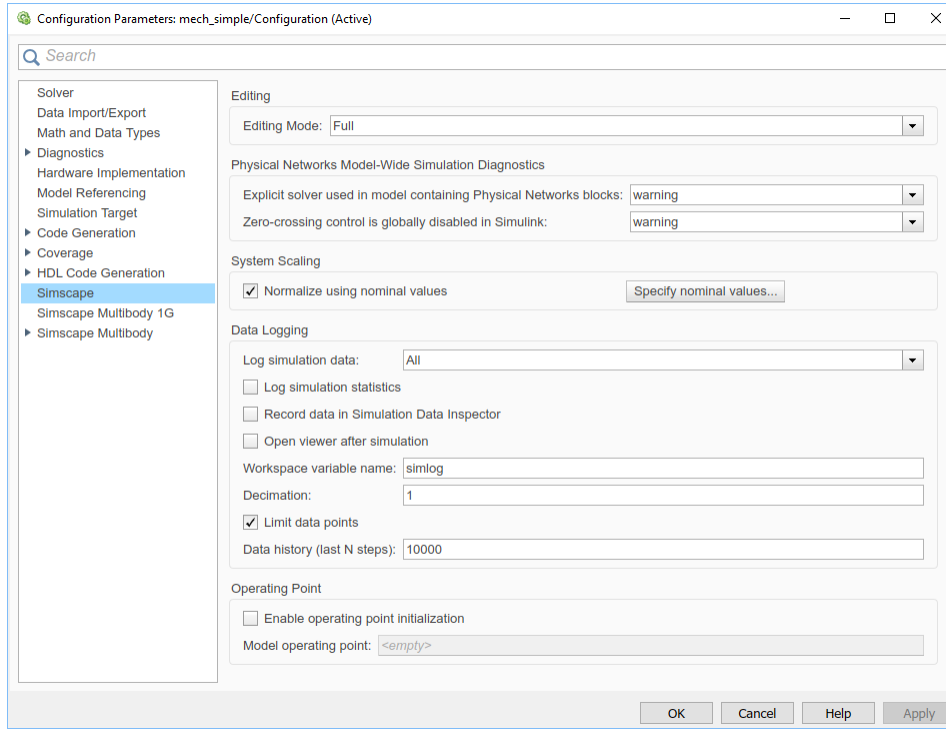
With the local solver option, you can perform multirate simulations, with:

- 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 6-16
- “Enabling or Disabling Simulink Zero-Crossing Detection” on page 6-16
- “Making Multirate Simulation Consistent” on page 6-17



Simscape Pane of the Configuration Parameters Dialog Box

Switching from the Default Explicit Solver to Other Simulink Solvers

When you first create a model, the default Simulink solver is `VariableStepAuto`. Auto solver chooses a suitable solver as described in "Use Auto Solver to Select a Solver" (Simulink), and for some types of models it can choose an explicit solver, `ode45`. If you do not modify the default (explicit) solver, your performance may not be optimal. Implicit solvers are better for most physical simulations. For more information about implicit solvers and physical systems, see "Important Concepts and Choices in Physical Simulation" on page 6-18.

Diagnostic Messages About Explicit Solvers

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

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

1. From the **Explicit solver used in model containing Physical Networks blocks** drop-down list, select the option that you want:
   - `warning` — If the model uses an explicit solver, the system issues a warning upon simulation. This is the default option that alerts you to a potential problem if you use the default solver.
   - `error` — If the model uses an explicit solver, the system issues an error message upon simulation. If your model is stiff, and you do not want to use explicit solvers, select this option to avoid future errors.
   - `none` — If the model uses an explicit solver, the system issues no warning or error message upon simulation. If you want to work with explicit solvers, in particular for models that are not stiff, select this option.
2. Click **OK**.

Enabling or Disabling Simulink Zero-Crossing Detection

By default, Simulink tracks an important class of simulation events by detecting zero crossings. With a global variable-step solver and without a local solver, Simulink attempts to locate the simulated times of zero crossings, if present. See "Managing Zero Crossings in Simscape Models" on page 6-4.

Diagnostic Messages About Globally Disabling Zero-Crossing Detection

You can globally disable zero-crossing detection in the **Solver** pane of the Configuration Parameters dialog box, under **Zero-crossing options**. If you do, and if you are using a

global variable-step solver without a local solver, the system issues a warning or error when you simulate with Simscape blocks.

You can choose between warning and error messages in the **Simscape** pane of the Configuration Parameters dialog box.

1. From the **Zero-crossing control is globally disabled in Simulink** drop-down list, select the option that you want, if you globally disable zero-crossing detection:
   - `warning` — The system issues a warning message upon simulation. This option is the default.
   - `error` — The system issues an error message upon simulation, which stops.
2. Click **OK**.

Making Multirate Simulation Consistent

The sample time or step size of the global Simulink solver must be the smallest time step of all the solvers in a multirate Simscape simulation.

To avoid simulation errors in sample time propagation, go to the **Solver** pane in the Configuration Parameters dialog box and select the **Automatically handle rate transition for data transfer** check box.

Important Concepts and Choices in Physical Simulation

In this section...
“Variable-Step and Fixed-Step Solvers” on page 6-18
“Explicit and Implicit Solvers” on page 6-19
“Full and Sparse Linear Algebra” on page 6-19
“Event Detection and Location” on page 6-19
“Unbounded, Bounded, and Fixed-Cost Simulation” on page 6-20
“Global and Local Solvers” on page 6-20

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 6-22. For background information, consult “How Simscape Models Represent Physical Systems” on page 6-2 and “How Simscape Simulation Works” on page 6-7.

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

In this section...
“Simulating with Variable Time Step” on page 6-22
“Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers” on page 6-22
“Simulating with Fixed Cost” on page 6-24
“Troubleshooting and Improving Solver Performance” on page 6-24
“Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System” on page 6-25

For the key simulation concepts to consider before making these choices, see “Important Concepts and Choices in Physical Simulation” on page 6-18.

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:

- The Backward Euler tends to damp out oscillations, but is more stable, especially if you increase the time step.
- The Trapezoidal Rule solver captures oscillations better but is less stable.
- The Partitioning solver lets you increase real-time simulation speed by partitioning the entire system of equations corresponding to a Simscape network into a cascade of

smaller equation systems. Not all networks can be partitioned. However, when a system can be partitioned, this solver provides a significant increase in real-time simulation speed. For more information, see “Increase Simulation Speed Using the Partitioning Solver” on page 10-27.

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.

Note Input filtering may introduce continuous states. If you are using a combination of discrete and local solvers and get an error message about the model containing continuous states, check the Simulink-PS Converter blocks in the model and turn off input filtering, if needed. For more information, see “Filtering Input Signals and Providing Time Derivatives” on page 6-27.

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 6-20 and "Fixed-Cost Simulation for Real-Time Viability" on page 10-96.

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 6-2.) 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.

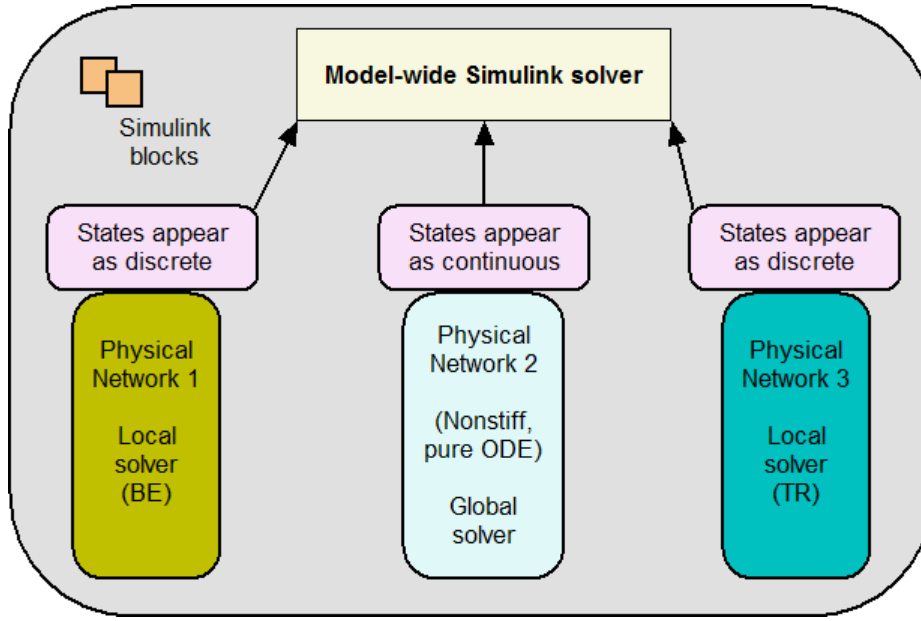
Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System

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.



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.

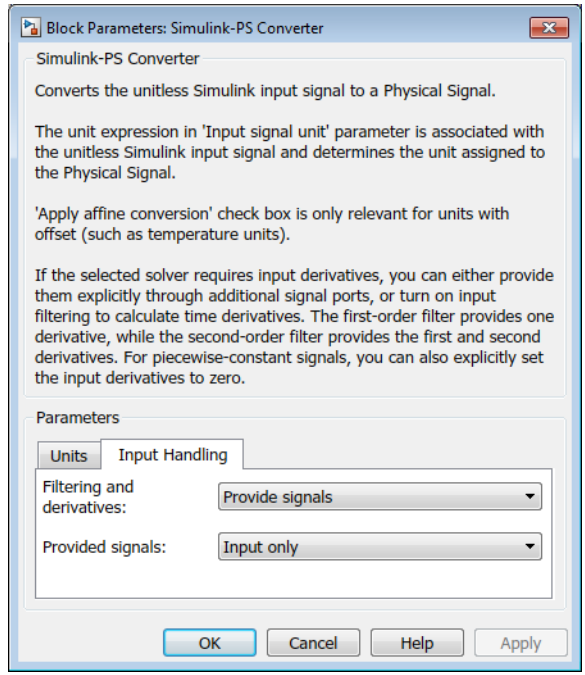
The filter time constant controls the filtering of the input signal. The filtered input follows the true input but is smoothed, with a lag on the order of the time constant that you choose. Set the time constant to a value no larger than the smallest time interval in the system that interests you. If 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.

For piecewise-constant signals, you can also explicitly set the input derivatives to zero.

You can control the way you provide time derivatives for each input signal by configuring the Simulink-PS Converter block connected to that input signal:

1. Open the Simulink-PS Converter block dialog box.
2. Click the **Input Handling** tab.



3. When you add a new Simulink-PS Converter block to your model, the default input handling options are `Provide signals` and `Input only`, and the block has one Simulink input port and one physical signal output port.
4. To turn on input filtering, set the **Filtering and derivatives** parameter to `Filter input, derivatives calculated`. Select the first-order or second-order filter, by using the **Input filtering order** parameter, and set the appropriate **Input filtering time constant (in seconds)** parameter value for your model.
5. To avoid filtering the input signal, keep the **Filtering and derivatives** parameter as `Provide signals`. Then set the **Provided signals** parameter value:

   - `Input and first derivative` — If you select this option, an additional Simulink input port appears on the Simulink-PS Converter block, to let you connect the signal providing input derivative.
   - `Input and first two derivatives` — If you select this option, two additional Simulink input ports appear on the Simulink-PS Converter block, to let you connect the signals providing input derivatives.

6 Finally, if your input signal is piecewise constant (such as step), you can also explicitly set the input derivatives to zero by selecting the `Zero derivatives (piecewise constant)` value for the **Filtering and derivatives** parameter.

System Scaling by Nominal Values

In this section...
“Enable or Disable System Scaling by Nominal Values” on page 6-30
“Possible Sources of Nominal Values and Their Evaluation Order” on page 6-31
“Specify Nominal Value-Unit Pairs for a Model” on page 6-31
“Modify Nominal Values for a Block Variable” on page 6-34

Nominal values provide a way to specify the expected magnitude of a variable in a model, similar to specifying a transformer rating, or setting a range on a voltmeter. Using system scaling based on nominal values increases the simulation robustness. This functionality provides a way to fine-tune scaling of individual variables in a model. It is especially helpful for initial conditions convergence and maintaining a minimum step size.

Enable or Disable System Scaling by Nominal Values

Using system scaling based on nominal values is a best practice for Simscape models because it improves simulation robustness. Therefore, when you create a new model, scaling by nominal values is enabled by default.

System scaling by nominal values is controlled by the **Normalize using nominal values** configuration parameter.

1. In the 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 **Normalize using nominal values** check box:

    - If the check box is selected, the model provides the scaling information to the solver based on the specified nominal values. To view, add, and edit the value-unit pairs for the model, click the **Specify nominal values** button next to the **Normalize using nominal values** check box.
    - If the check box is cleared, the scaling by nominal values is disabled.

Possible Sources of Nominal Values and Their Evaluation Order

The scaling of each variable is determined by its nominal value and physical units. Nominal values can come from different sources:

- **Block** — You can specify nominal value and unit as variable declaration attributes in a Simscape component file underlying the block. These attributes translate into block parameters *x*`_nominal_value` and *x*`_nominal_unit` (where *x* is the variable name). You can also override these values on individual blocks in the model by setting the corresponding block parameter *x*`_nominal_specify` to `'on'` and supplying different values for *x*`_nominal_value` and *x*`_nominal_unit`. These parameters are not visible in the block dialog box, but you can use either the Property Inspector or `set_param` and `get_param` functions to view and change their values. For more information, see “Modify Nominal Values for a Block Variable” on page 6-34.
- **Model** — In absence of a nominal value specified for the block, a variable uses the nominal value for the commensurate physical unit specified in the model table. All models have a default table of nominal values and units (factory default). To view, add, and edit the value-unit pairs for the model, click the **Specify nominal values** button next to the **Normalize using nominal values** check box. For more information, see “Specify Nominal Value-Unit Pairs for a Model” on page 6-31.
- **Derived** — If the model table of nominal values does not contain a row for a unit commensurate with the physical unit of a variable, then the nominal value for this variable is derived from fundamental dimensions. For example, if the variable's initial value is in `lbf`, and there is no entry in the table for force, but the table contains `{10,'lbm'}`, `{12,'ft'}`, and `{2,'min'}`, then the nominal value for that variable is `{10*12/2^2,'lbm*ft/min^2'}`.
- **Fixed** — Event variables, top-level model inputs, and Simscape Multibody™ variables cannot be scaled according to nominal values.

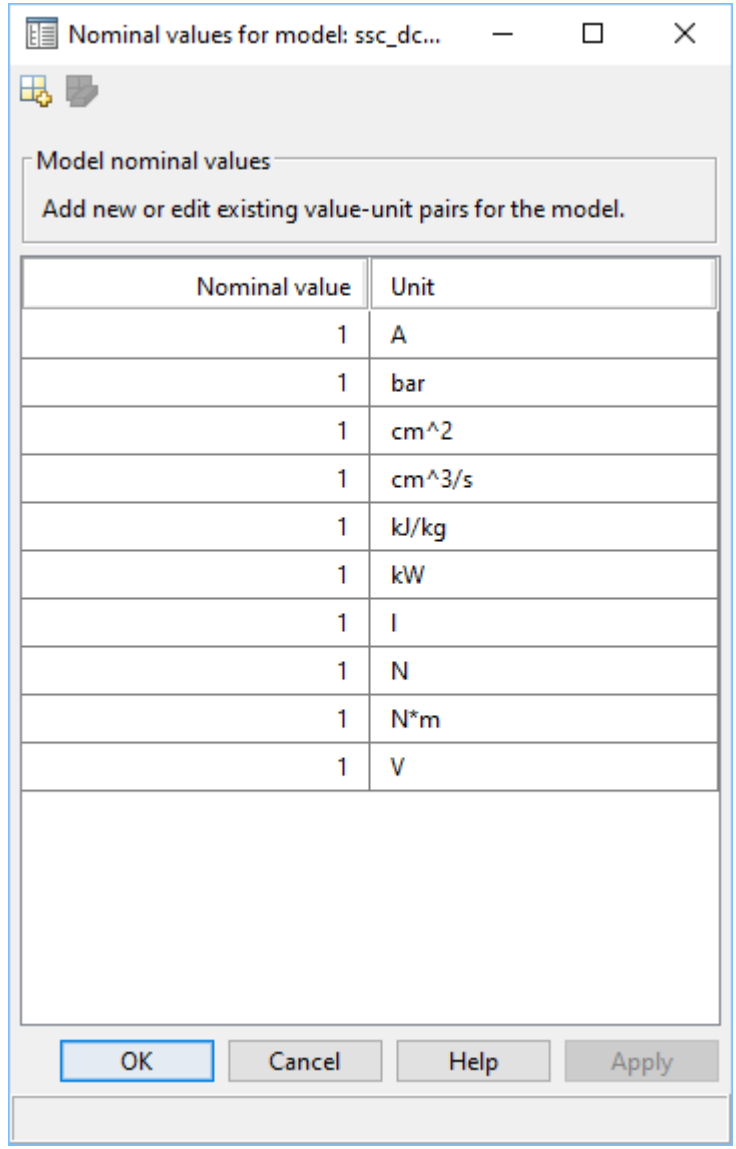
The Variable Viewer in advanced configuration shows the nominal value and unit for each variable, along with the source. For more information, see “Variable Viewer” on page 7-25.

Specify Nominal Value-Unit Pairs for a Model

All models have a default table of nominal values and units (factory default). To view, add, and edit the value-unit pairs for a model:

1. In the 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**.
3. Make sure the **Normalize using nominal values** check box is selected.
4. Click the **Specify nominal values** button next to the **Normalize using nominal values** check box.

   The model table of nominal values opens in a new window. It contains all the value-unit pairs currently defined for the model.



5 To edit a value-unit pair, select the corresponding cell and enter the new value or unit.

6 To add a new value-unit pair, in the top toolbar of the window containing the table, click . This action adds a new empty row at the bottom of the table. Select the cells in this row and enter the nominal value and unit for the new value-unit pair.

7 To delete a value-unit pair, select the corresponding row and click .

8 When finished editing the table, click **OK**. Table data is saved when you save the model.

Modify Nominal Values for a Block Variable

Each variable in a block has three associated block parameters (where *x* is the variable name):

- *x*`_nominal_specify` — Lets you override the system default nominal value for variable *x* in this particular block. The default parameter value is `'off'`, in which case the variable nominal value is determined according to the evaluation order described in “Possible Sources of Nominal Values and Their Evaluation Order” on page 6-31. Set this parameter to `'on'` to use the *x*`_nominal_value` and *x*`_nominal_unit` parameter values for scaling.
- *x*`_nominal_value` — If the *x*`_nominal_specify` parameter is set to `'on'`, then this value, in conjunction with the nominal unit parameter, determines the scaling of variable *x* in this particular block. The parameter value must be a numeric value, specified as a character vector. The default parameter value is `'1'`.
- *x*`_nominal_unit` — If the *x*`_nominal_specify` parameter is set to `'on'`, then this unit, in conjunction with the nominal value parameter, determines the scaling of variable *x* in this particular block. The parameter value must be a valid physical unit name, specified as a character vector. The unit must be commensurate with the unit specified for the initial value of the variable. The default unit is the same as for the initial value.

Note Nominal value and unit can be specified as variable declaration attributes in a Simscape component file underlying the block. For more information, see “Nominal Value and Unit for a Variable”. In this case, the nominal value and unit parameters for that variable get their default values from the variable declaration attributes.

These parameters are not visible in the block dialog box, but you can use `set_param` and `get_param` functions to view and change their values.

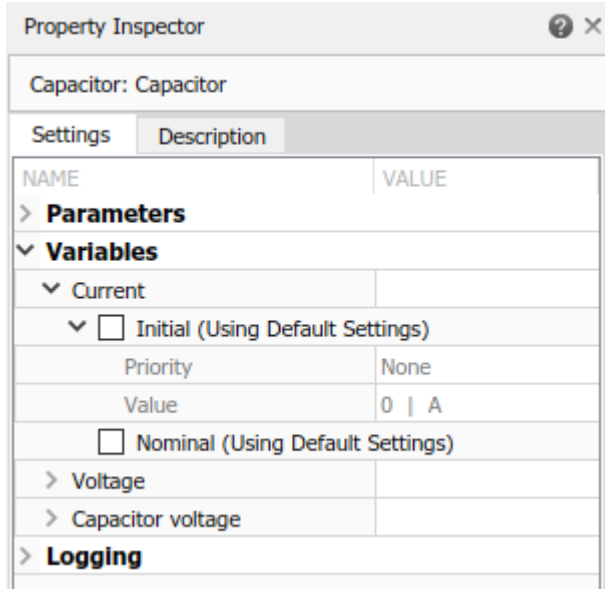
For example, to change the nominal value and unit for variable `i` (current) for an individual block, select this block in the model and type:

```
set_param(gcb,'i_nominal_specify','on')
set_param(gcb,'i_nominal_value','10')
set_param(gcb,'i_nominal_unit','mA')
```

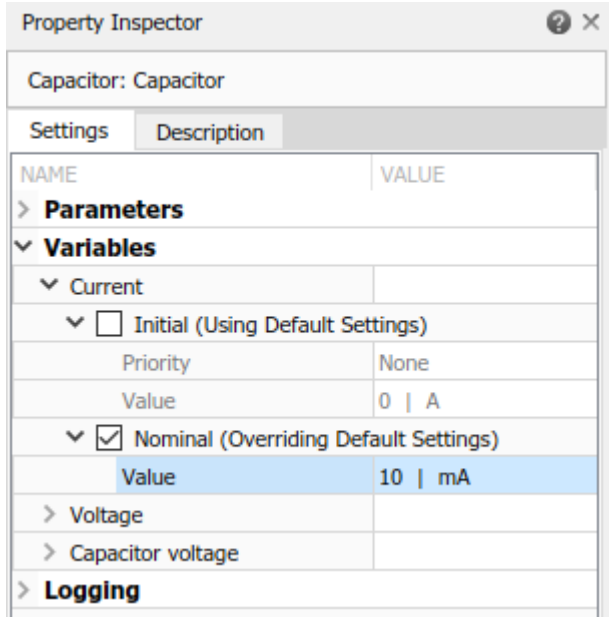
This sequence of commands overrides the default nominal value for the block variable and sets it to 10 mA.

To perform the same actions using the Property Inspector:

1. Select the block in the model.
2. In the model window, from the top menu bar, select **View** > **Property Inspector**.
3. In the Property Inspector pane showing the block properties, expand the **Variables** node, and then expand the nodes for the **Current** variable.



4. Select the check box next to **Nominal**. This action is equivalent to setting the `i_nominal_specify` parameter to `'on'`.
5. Once the **Nominal** check box is selected, its **Value** field becomes editable. Enter `10` and select `mA` from the unit drop-down list.



See Also

`variables`

More About

- “Variable Viewer” on page 7-25

Frequency and Time Simulation Mode

In this section...
“Speeding Up Model Simulation” on page 6-37
“Variable Initialization for Frequency-and-Time Simulation” on page 6-38
“Limitations” on page 6-38
“Perform Sinusoidal Steady-State Analysis of a Model” on page 6-38

Frequency and time simulation mode speeds up simulation of systems with a single nominal frequency by letting you increase the maximum step size for variable solvers. This mode also lets you perform phasor analysis of such systems by using the blocks in the Periodic Operators sublibrary of the Physical Signals library.

Depending on your task, you can switch between time and frequency-and-time simulation modes without modifying the model. For example, use the time simulation mode to study transient effects, and then switch to the frequency-and-time mode to perform the phasor analysis of a model.

Speeding Up Model Simulation

Frequency-and-time equation formulation is intended for linear and linear parameter-varying (LPV) systems. It speeds up the simulation using a variable-step solver because the solver step size is no longer limited by the period of the nominal frequency.

The frequency-and-time simulation mode is based on changing the equation formulation for a physical network with a nominal frequency ω_0 by dividing its variables into two categories:

- Time variables, which vary slowly relative to the nominal period $2\pi/\omega_0$
- Frequency variables, which are sinusoidal and represent forced response at the nominal frequency, $x = d_x + a_x\cos(\omega_0 t) + b_x\sin(\omega_0 t)$

In time simulation mode, the solver step size is typically limited to a small fraction of a period of the nominal frequency. In frequency-and-time simulation mode, the representation of frequency, or fast, variables as sinusoids allows the variable solver to take much larger steps. The speeding-up effect is especially pronounced in complex machine systems that use three-phase Simscape Electrical™ blocks.

When you run a model in frequency-and-time simulation mode, the software automatically detects the nominal frequency and determines which of the variables are fast (frequency) and which are slow (time).

To benefit from improved performance, the time variables in the system should have slow dynamics. If time variables have time constants comparable to, or smaller than, the nominal frequency period, frequency-and-time simulation of such a system will be slow (due to the large number of timesteps required to resolve these dynamics) and possibly inaccurate. In such cases, use the time simulation mode instead.

Variable Initialization for Frequency-and-Time Simulation

Variable initialization for frequency-and-time equation formulation follows these rules:

- For time variables and algebraic frequency variables, initialization targets and priorities are preserved.
- For dynamic frequency variables, initialization priority is switched to `None` because the solver is using the sinusoidal steady-state approximation for these variables.

Limitations

Frequency-and-time equation formulation is intended for systems with a single nominal frequency. In other words:

- The model must have at least one sinusoidal source in its physical network.
- In case of multiple sinusoidal sources, they must all operate at the same frequency.
- Blocks outside the physical network, such as a Sine Wave block, are not considered valid sinusoidal sources.

If you try to run a frequency-and-time simulation on a model that does not meet the above criteria, you get an error message.

Perform Sinusoidal Steady-State Analysis of a Model

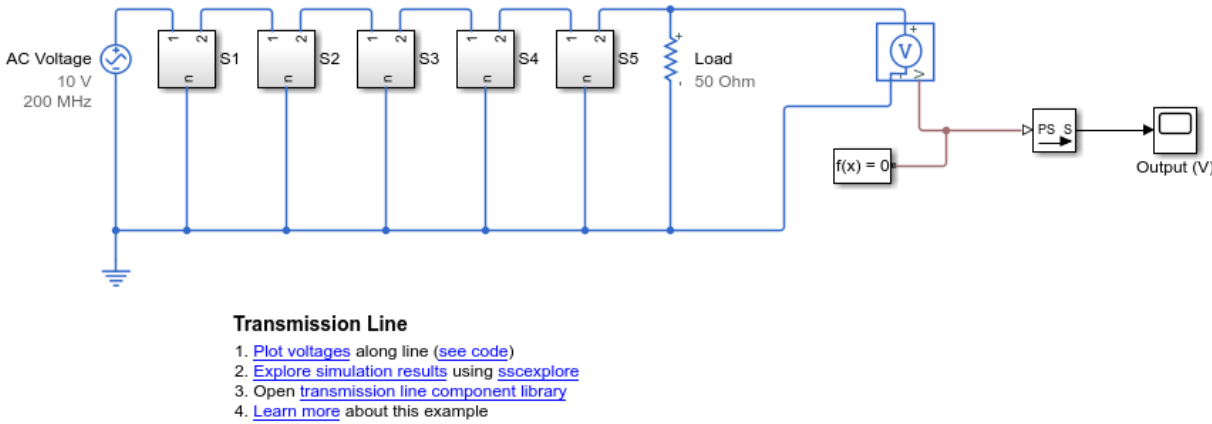
This example shows how you can deploy different simulation modes on the same model, depending on the type of analysis you want to perform.

The transmission line model used in this example is built from 50 identical blocks, each block representing a single T-section segment. For more information, see “Transmission

Line”. The model has one sinusoidal source (AC Voltage) and operates at a nominal frequency 200 MHz, which makes it a good candidate for frequency-and-time simulation.

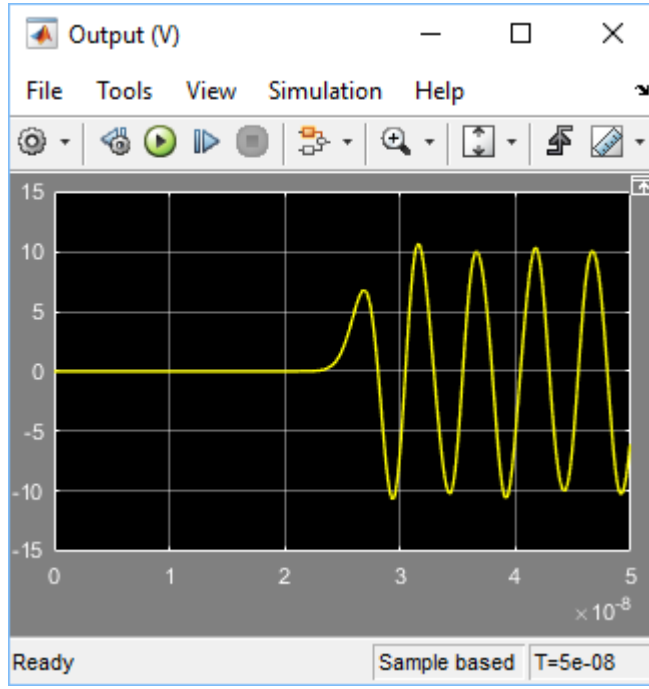
1 Open the Transmission Line example model by typing `ssc_transmission_line` in the MATLAB Command Window.

Expand the Voltage Sensor subsystem, which consists of a Voltage Sensor block, a Solver Configuration block, and a PS-S Converter block connected to the scope.



2 To analyze the transient behavior of the model, run it in the time simulation mode.

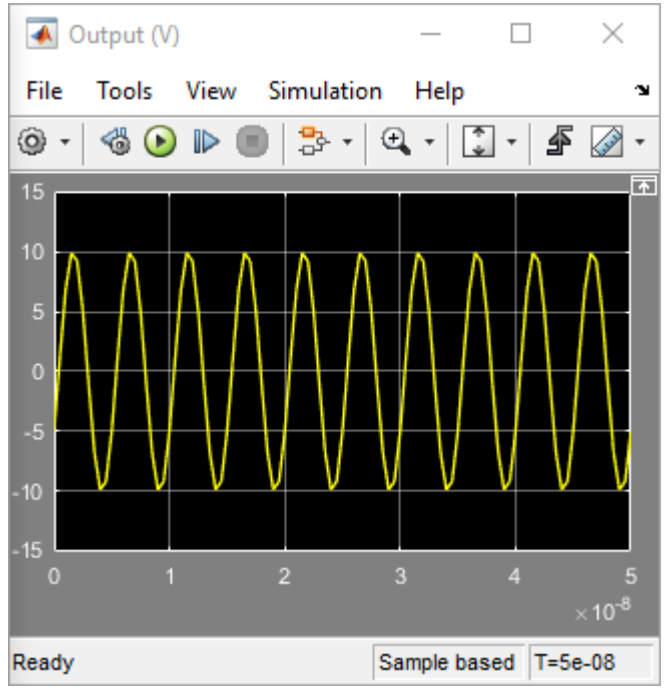
Open the Solver Configuration block dialog box and verify that the **Equation formulation** parameter is set to `Time`. Simulate the model.



You can observe the transmission delay from the simulation results.

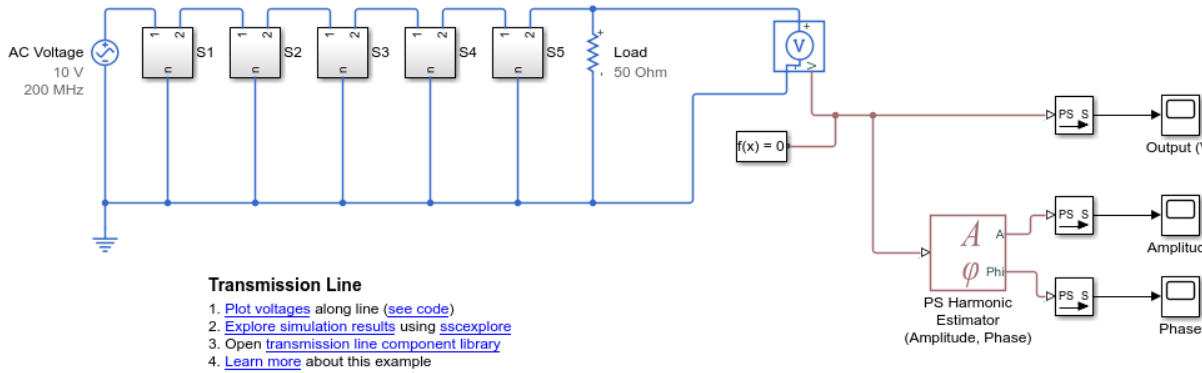
3 To perform the phasor analysis, switch to frequency-and-time simulation mode.

Open the Solver Configuration block dialog box and set the **Equation formulation** parameter to `Frequency and time`. Simulate the model.



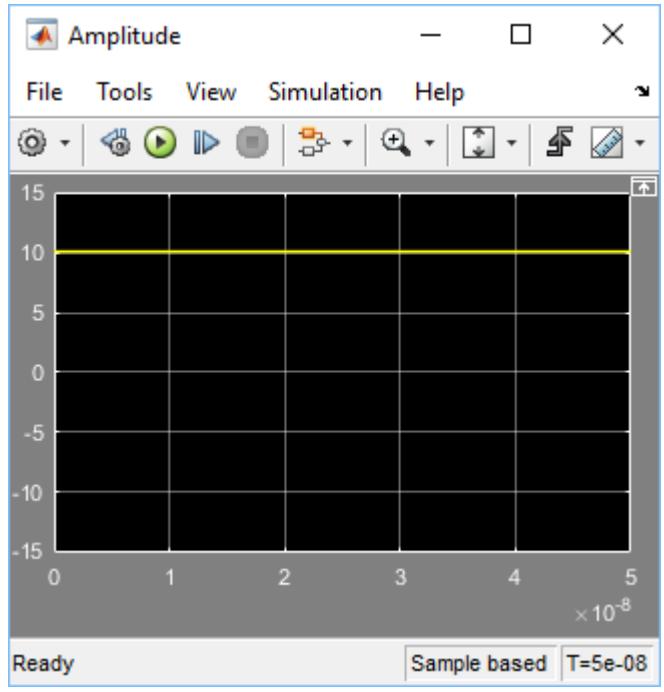
Notice that in frequency-and-time mode the simulation starts in sinusoidal steady state.

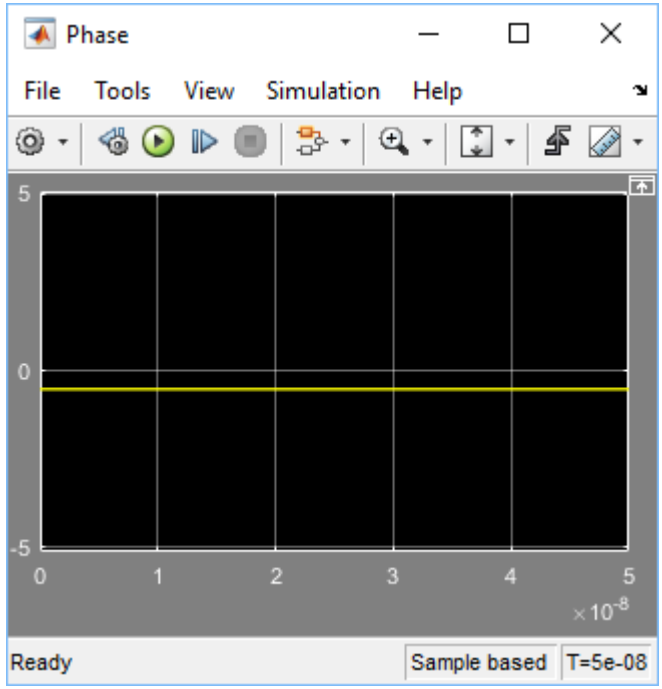
4 To determine the amplitude and phase of the base frequency, connect the PS Harmonic Estimator (Amplitude, Phase) block to the voltage sensor output. Add the respective scopes.



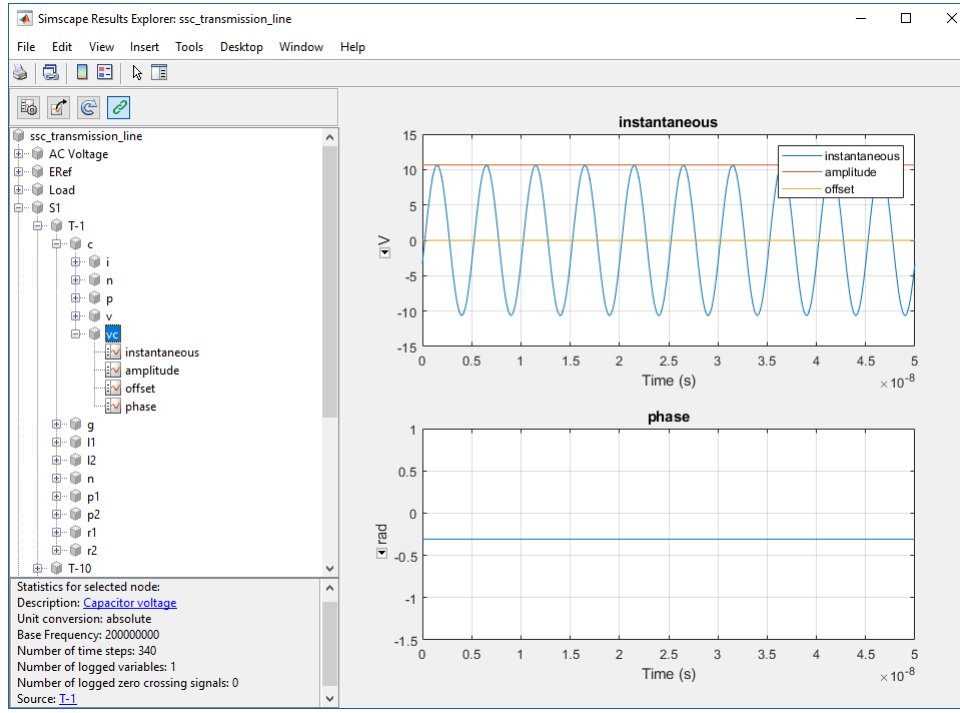
5 Open the PS Harmonic Estimator (Amplitude, Phase) block dialog box and set the **Base frequency** parameter to `200 MHz`, to match the nominal frequency of the model.

6 Simulate the model.





7 The logged simulation data for frequency variables contains subnodes that let you examine the variable instantaneous value, amplitude, phase, and offset data separately.



Note If you use the workflow of live-streaming the data to Simulation Data Inspector, the recorded simulation data does not contain these subnodes. To view the additional subnodes for frequency variables, clear the **Record data in Simulation Data Inspector** check box and rerun the simulation.

See Also

PS Harmonic Estimator (Amplitude, Phase) | Solver Configuration

More About

- “Phasor-Mode Simulation in Simscape Components” (Simscape Electrical)
- “About the Simscape Results Explorer” on page 12-30

Troubleshooting Simulation Errors

In this section...
“Troubleshooting Tips and Techniques” on page 6-45
“System Configuration Errors” on page 6-46
“Numerical Simulation Issues” on page 6-48
“Initial Conditions Solve Failure” on page 6-49
“Transient Simulation Issues” on page 6-50

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 6-7, 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 6-46
- "Extra Fluid or Gas Properties Block" on page 6-46
- "Missing Reference Block" on page 6-47
- "Basic Errors in Physical System Representation" on page 6-47

Missing Solver Configuration Block

Each topologically distinct Simscape block diagram requires exactly one Solver Configuration block to be connected to it. The Solver Configuration block specifies the global environment information and provides parameters for the solver that your model needs before you can begin simulation.

If you get an error message about a missing Solver Configuration block, open the Simscape Utilities library and add the Solver Configuration block anywhere on the circuit.

Extra Fluid or Gas Properties Block

If your model contains hydraulic elements, each topologically distinct hydraulic circuit in a diagram requires a Custom Hydraulic Fluid block (or Hydraulic Fluid block, available with Simscape Fluids block libraries) to be connected to it. These blocks define the fluid properties that act as global parameters for all the blocks connected to the hydraulic circuit. If no hydraulic fluid block is attached to a loop, the hydraulic blocks in this loop use the default fluid. However, more than one hydraulic fluid block in a loop generates an error.

Similarly, more than one Thermal Liquid Settings (TL) block in a thermal liquid circuit, Two-Phase Fluid Properties (2P) block in a two-phase fluid circuit, or Gas Properties (G) block in a gas circuit generates an error.

If you get an error message about too many domain-specific global parameter blocks attached to the network, look for an extra Hydraulic Fluid block, Custom Hydraulic Fluid

block, Thermal Liquid Settings (TL) block, Two-Phase Fluid Properties (2P) block, or Gas Properties (G) block and remove it.

Missing Reference Block

Simscape libraries contain domain-specific reference blocks, which represent reference points for the conserving ports of the appropriate type. For example, each topologically distinct electrical circuit must contain at least one Electrical Reference block, which represents connection to ground. Similarly, hydraulic conserving ports of all the blocks that are referenced to atmosphere (for example, suction ports of hydraulic pumps, or return ports of valves, cylinders, pipelines, if they are considered directly connected to atmosphere) must be connected to a Hydraulic Reference block, which represents connection to atmospheric pressure. Mechanical translational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Translational Reference block, and so on.

If you get an error message about a missing reference block, or node, check your system configuration and add the appropriate reference block based on the rules described above. The missing reference node diagnostic messages include information about the particular block and variable that needs a reference node. This is especially helpful when multiple domains are involved in the model. For more information and examples of best modeling practices, see “Grounding Rules” on page 1-35.

Basic Errors in Physical System Representation

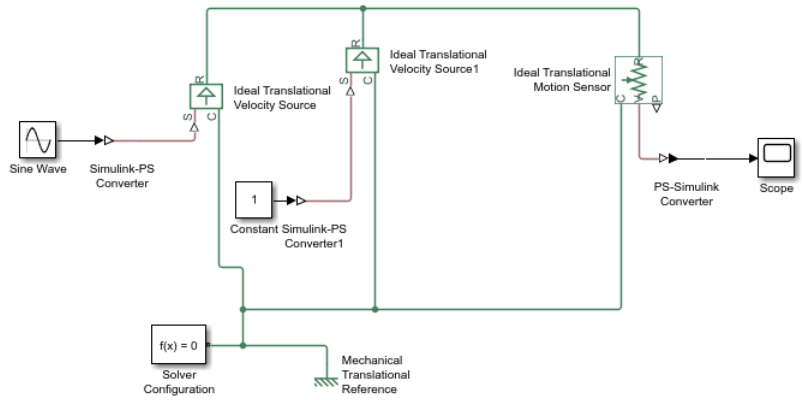
Physical systems are represented in the Simscape modeling environment as Physical Networks according to the Kirchhoff's generalized circuit laws. Certain model configurations violate these laws and are therefore illegal. There are two broad violations:

- 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 6-48
- “Parameter Discontinuities” on page 6-49

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 6-50).

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 6-46 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 6-48 for more information.
- The residual tolerance may be too tight to produce a consistent solution to the algebraic constraints at the beginning of simulation. You can try to increase the **Consistency Tolerance** parameter value (that is, relax the tolerance) in the Solver Configuration block.

If the Simulation Diagnostics window has other, more specific, error messages, address them first and try rerunning the simulation. See also “Troubleshooting Tips and Techniques” on page 6-45.

Transient Simulation Issues

- “Transient Initialization Not Converging” on page 6-50
- “Step-Size-Related Errors — Dependent States — High Stiffness” on page 6-50

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 6-45.

You can also try to decrease the **Consistency Tolerance** parameter value (that is, tighten the tolerance) in the Solver Configuration block.

Step-Size-Related Errors — Dependent States — High Stiffness

A typical step-size-related error message may state that the system is unable to reduce the step size without violating the minimum step size for a certain number of consecutive times. This error message indicates numerical difficulties in solving the Differential Algebraic Equations (DAEs) for the model. This might be caused by dependent dynamic states (higher-index DAEs) or by the high stiffness of the system. You can try the following:

- Tighten the solver tolerance (decrease the **Relative Tolerance** parameter value in the Configuration Parameters dialog box)
- Specify a value, other than `auto`, for the **Absolute Tolerance** parameter in the Configuration Parameters dialog box. Experiment with this parameter value.
- Tighten the residual tolerance (decrease the **Consistency Tolerance** parameter value in the Solver Configuration block)
- Increase the value of the **Number of consecutive min step size violations allowed** parameter in the Configuration Parameters dialog box (set it to a value greater than the number of consecutive step size violations given in the error message)

- Review the model configuration and try to simplify the circuit, or add small parasitic terms to your circuit to avoid dependent dynamic states. For more information, see “Numerical Simulation Issues” on page 6-48.

Limitations

In this section...
“Sample Time and Solver Restrictions” on page 6-52
“Algebraic Loops” on page 6-52
“Unsupported Simulink Tools and Features” on page 6-53
“Restricted Simulink Tools” on page 6-54
“Simulink Tools Not Compatible with Simscape Blocks” on page 6-55
“Code Generation” on page 6-56

Sample Time and Solver Restrictions

The default sample times of Simscape blocks are continuous. You cannot simulate Simscape blocks with discrete solvers using the default sample times.

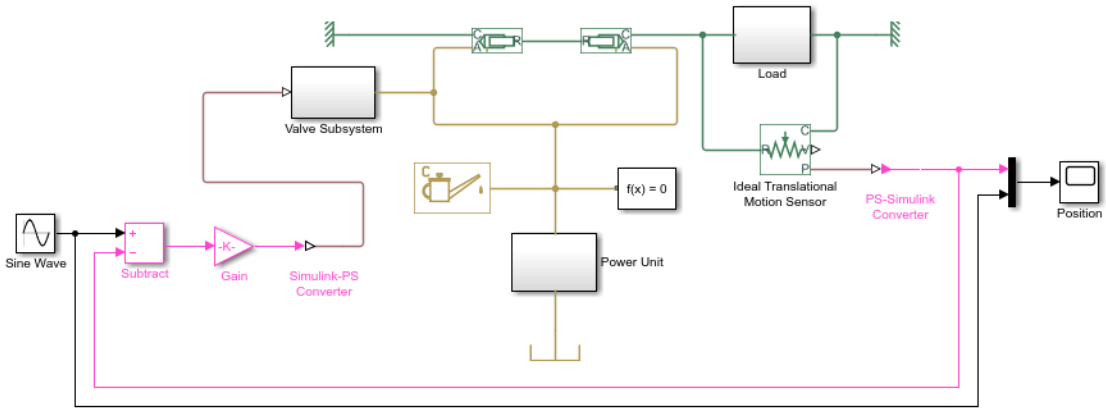
If you switch to a local solver in the Solver Configuration block, the states of the associated physical network become discrete. If there are no continuous Simulink or Simscape states anywhere in a model, you are free to use a discrete solver to simulate the model.

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

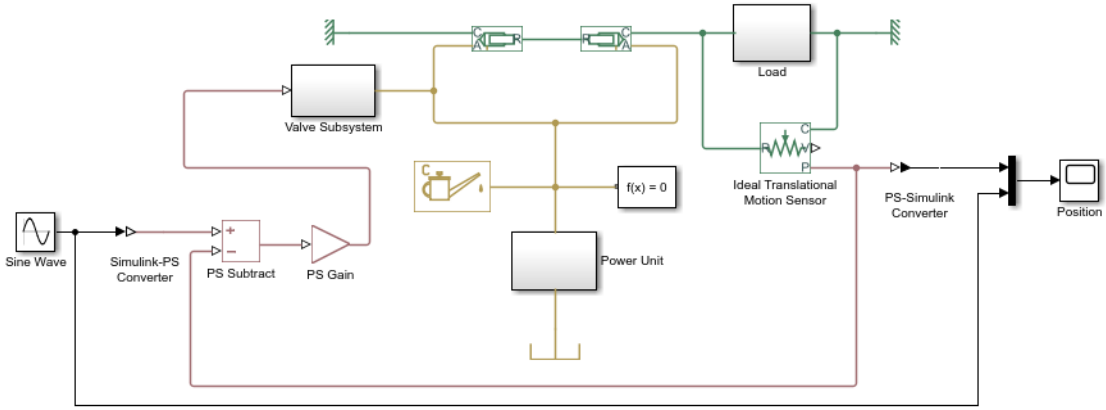
Algebraic Loops

A Simscape physical network should not exist within a Simulink algebraic loop. This means that you should not directly connect an output of a PS-Simulink Converter block to an input of a Simulink-PS Converter block of the same physical network.

For example, the following model contains a direct feedthrough between the PS-Simulink Converter block and the Simulink-PS Converter block (highlighted in magenta). To avoid the algebraic loop, you can insert a Transfer Function block anywhere along the highlighted loop.



A better way to avoid an algebraic loop without introducing additional dynamics is shown in the modified model below.



Unsupported Simulink Tools and Features

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

- Exporting a model to a format used by an earlier version (**File > Export Model to > Previous Version**) is not supported for models containing Simscape blocks.
- The Simulink Profiler tool does not work with Simscape models.

- Physical signals and physical connection lines between conserving ports are different from Simulink signals. Therefore, the Signal and Scope Manager tool and the signal label functionality are not supported.

Restricted Simulink Tools

Certain Simulink tools are restricted for use with Simscape software:

- You can use the Simulink `set_param` and `get_param` commands to set or get Simscape block parameters, if the parameters correspond to fields in the block dialog box. 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” (Simulink)). 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.

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.
- `MaxStackSize` is not supported.
- Simulation of Simscape models on fixed-point processors is not supported.
- Block diagnostics in error messages are not supported. This means that if you get an error message from simulating generated code, it does not contain a list of blocks involved.
- Conversion of models or subsystems containing Simscape blocks to S-functions is not supported.

"Code Generation" describes Simscape code generation features. "Restricted Simulink Tools" on page 6-54 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.

7

Variable Initialization and Operating Points

- “Block-Level Variable Initialization” on page 7-2
- “Set Priority and Initial Target for Block Variables” on page 7-5
- “Initialize Variables for a Mass-Spring-Damper System” on page 7-7
- “Variable Viewer” on page 7-25
- “Using Operating Point Data for Model Initialization” on page 7-37
- “Initialize Model Using Operating Point from Logged Simulation Data” on page 7-42

Block-Level Variable Initialization

In this section...
“Initializing Block Variables for Model Simulation” on page 7-2
“Variable Initialization Priority” on page 7-3
“Suggested Workflow” on page 7-4

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 6-10. 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 7-25.

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 7-7.

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 7-5.

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 and refresh the Variable Viewer to see which of the initial targets have been satisfied. Although the viewer does not simulate the model, it runs the simulation for 0 seconds to initialize it, and therefore the model must be in an executable state.
3. If initialization fails, or you are not satisfied with the results, iterate by changing the block variable target values and priority, then refreshing the viewer.
4. When satisfied with initialization, run the simulation to see the results.

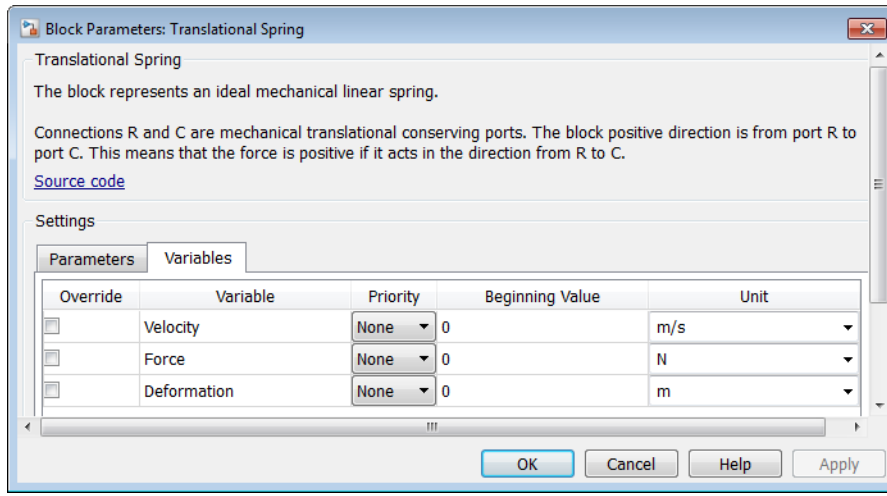
See Also

More About

- “Set Priority and Initial Target for Block Variables” on page 7-5
- “Initialize Variables for a Mass-Spring-Damper System” on page 7-7
- “Variable Viewer” on page 7-25

Set Priority and Initial Target for Block Variables

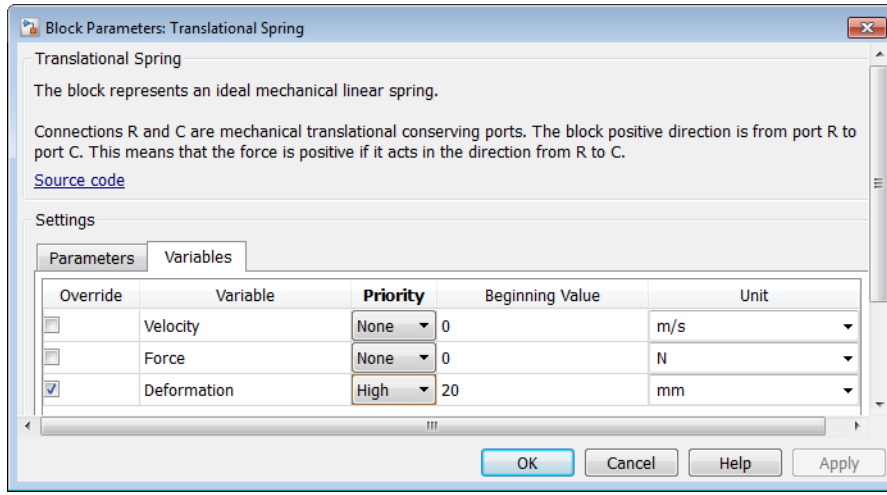
When you open the **Variables** tab of a block dialog box, it lists all the public variables specified in the underlying component file, along with priority, beginning (target) value, and unit. For example, if you add a Translational Spring block to your model, double-click it to open its dialog box, 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.

Note The **Source code** link is available for all the Foundation library blocks that have a **Variables** tab. Blocks from the add-on products, like Simscape Electrical or Simscape Fluids, do not have a **Source code** link in the block dialog box. See the block reference page for information on relevant equations and specific initialization considerations.

To specify the initial deformation of the spring, select the **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.

See Also

More About

- “Initialize Variables for a Mass-Spring-Damper System” on page 7-7
- “Block-Level Variable Initialization” on page 7-2

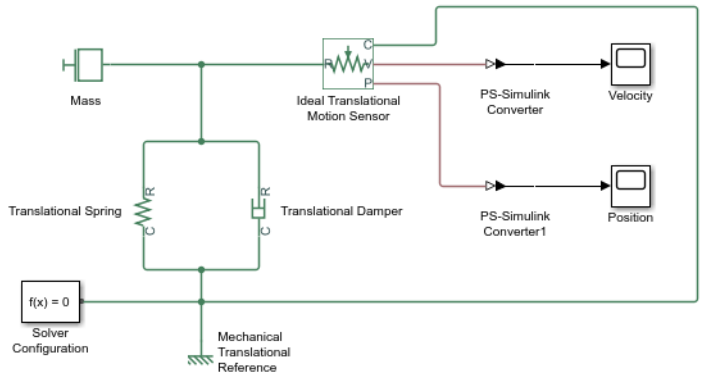
Initialize Variables for a Mass-Spring-Damper System

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

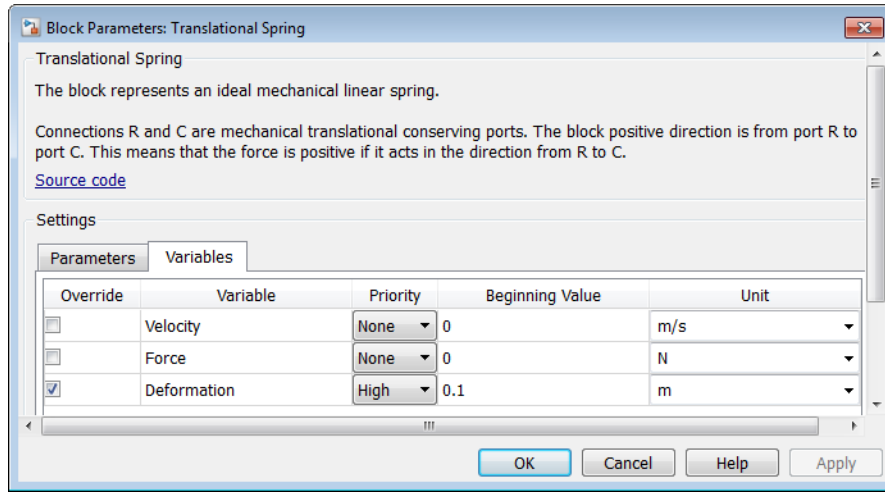
The model is a classical unforced mass-spring-damper system, with the oscillations of the mass caused by the initial deformation of the spring.

Create and Set Up the Model

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

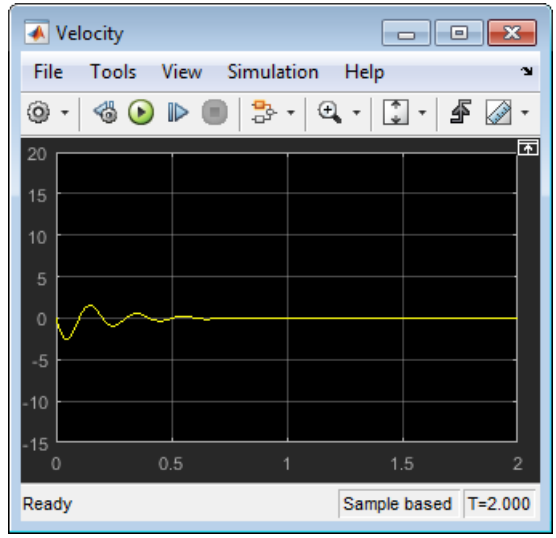


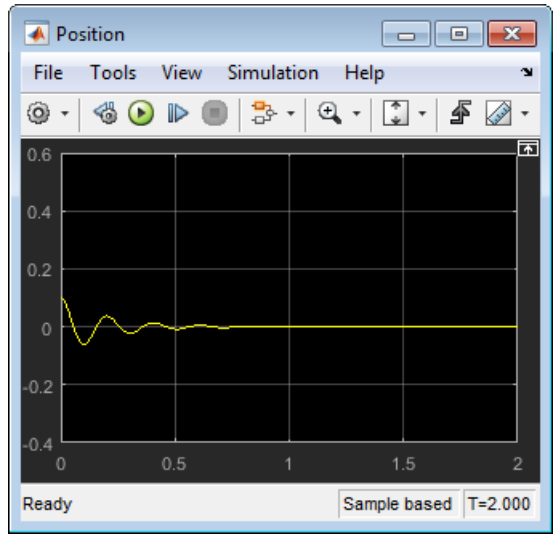
2. Prepare the model for simulation. On the top menu bar of the model window, select **Simulation > Model Configuration Parameters**. On the **Solver** pane of the Configuration Parameters dialog box, set **Solver** to `ode23t (mod.stiff/ Trapezoidal)` and **Max step size** to `0.2`. Also adjust the **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`.



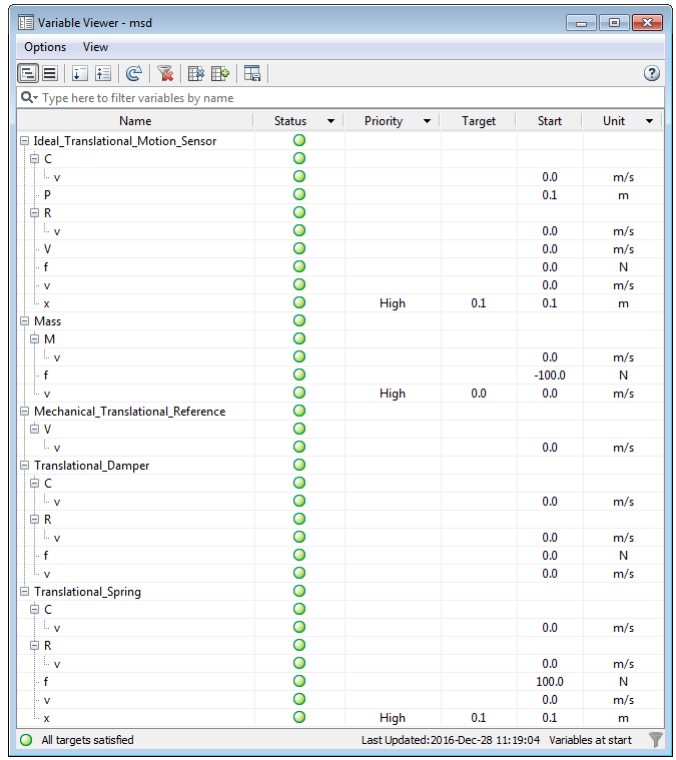
4 Adjust the initial position of the sensor, to compensate for the spring deformation. Double-click the Ideal Translational Motion Sensor block and set its **Initial position** parameter value to `0.1 m` as well. This way, when you simulate the model, mass oscillations center around 0.

5 Simulate the model.





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



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.

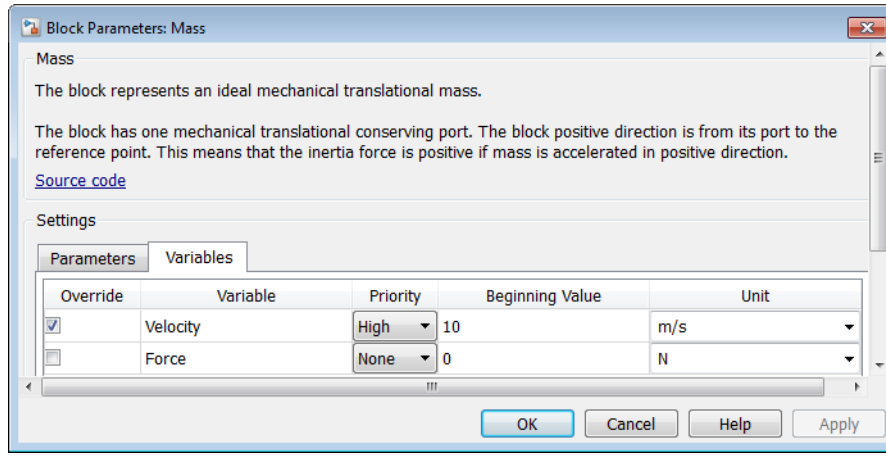
The other high-priority variable in this model is the position, *x*, of the Ideal Translational Motion Sensor block, which is set inside the component file because it is necessary for the correct operation of the sensor. Its actual start value also matches its target value, and its **Status** column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

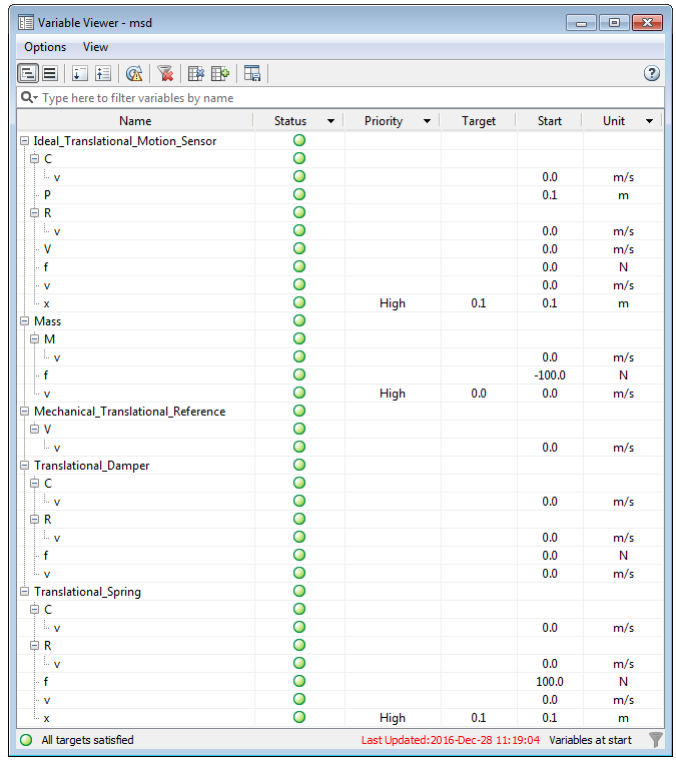
Change Initialization Targets

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

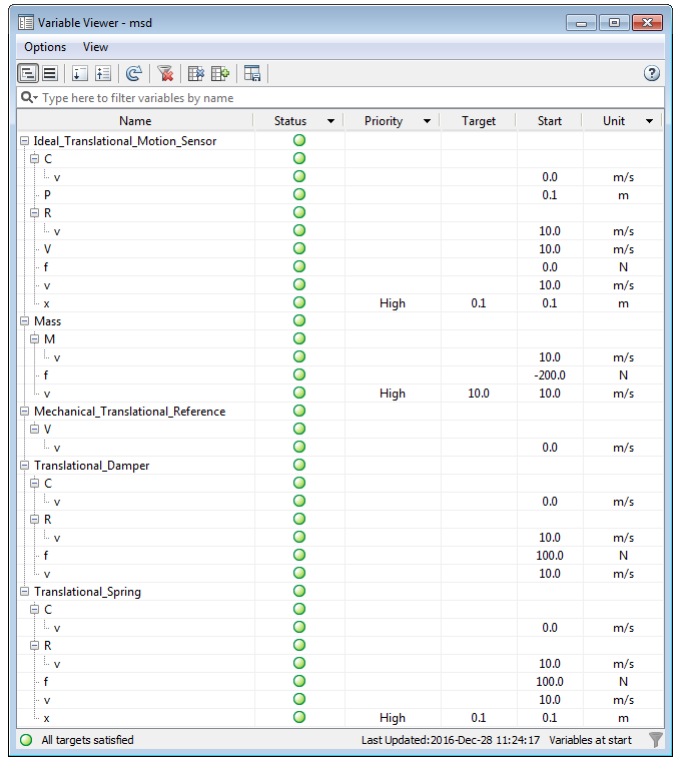
1 Specify the initial velocity of the mass. Double-click the Mass block, go to the **Variables** tab, select the check box next to the **Velocity** variable, change its **Priority** to `High`, and enter a beginning value of `10`. Keep the unit `m/s`.



When you change variable priorities and targets or adjust the block parameters, the results in the Variable Viewer are not updated automatically. Instead, the **Refresh** button displays a warning symbol (yellow triangle), and the timestamp at the bottom of the viewer window turns red to indicate that the data in the viewer does not reflect the latest model changes.



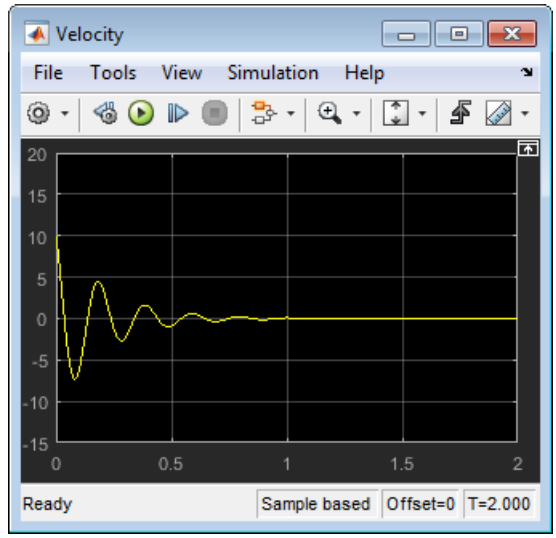
2 Refresh the Variable Viewer by clicking .

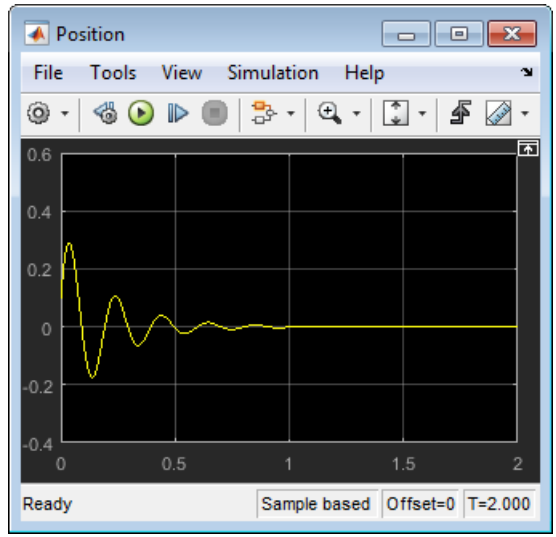


You can see that the solver has found a different initial solution, which satisfies your variable targets for spring deformation and mass velocity. The **Status** column displays green circles, and the overall status at the bottom of the Variable Viewer window also displays a green circle and says that 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.

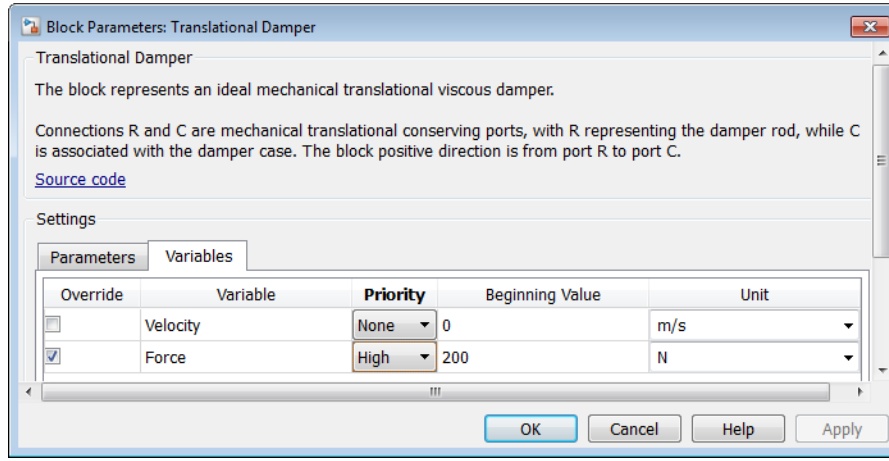




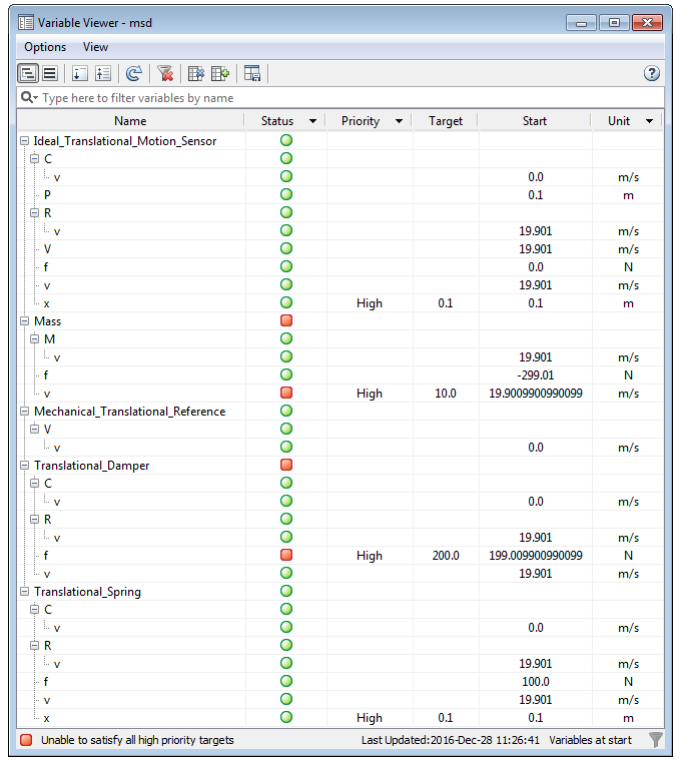
Deal with Over-Specification

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`.

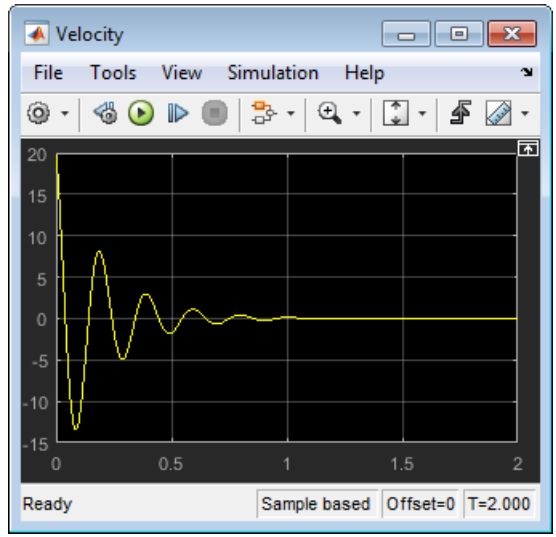


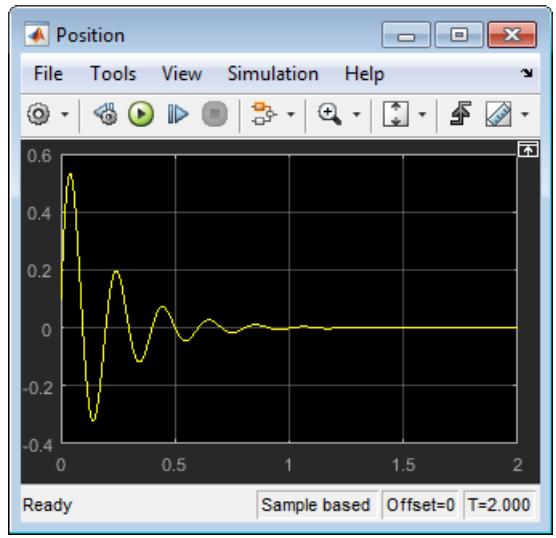
2 Refresh the Variable Viewer.



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.

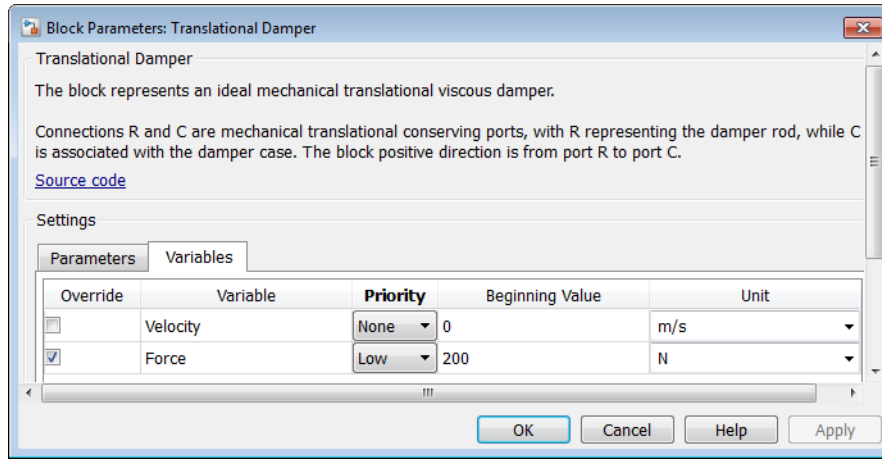
Notice that the solver has been able to find a solution for model initialization. If you rerun the simulation, it runs without errors and you can see the new simulation results.



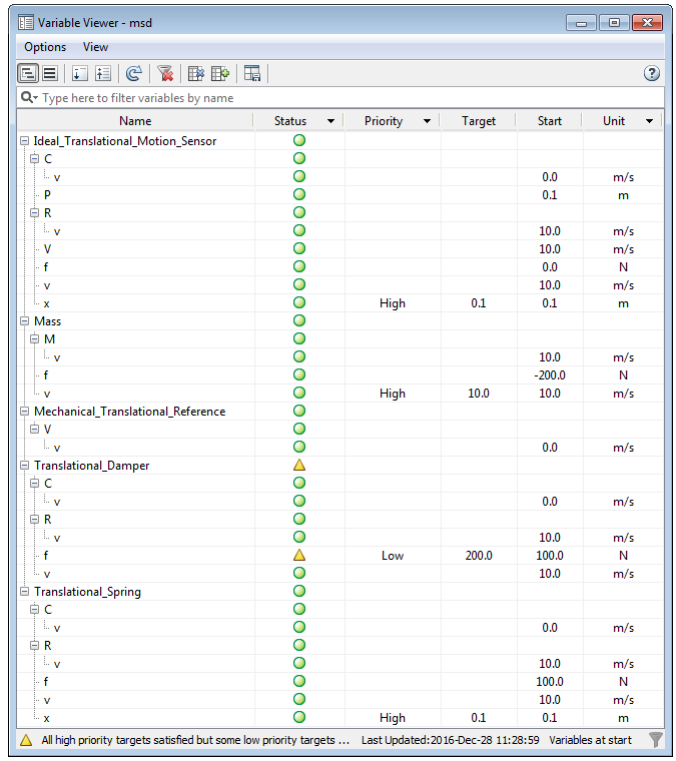


However, the Variable Viewer shows that the model initialization solution does not satisfy your target values for block variables. This happens because placing high-priority constraints on all three elements of the mass-spring-damper system results in a conflict. You can resolve the over-specification 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`.

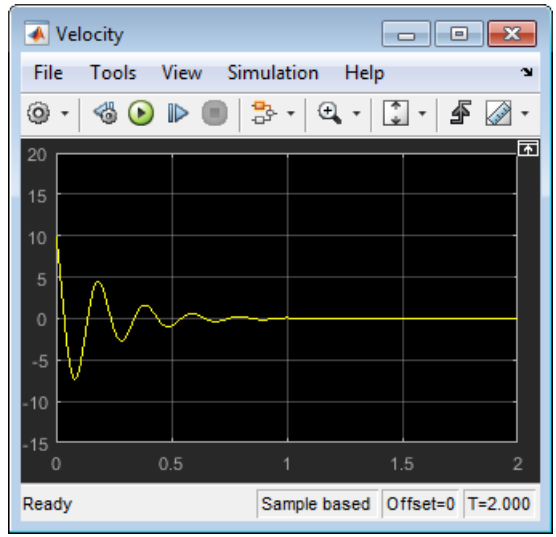


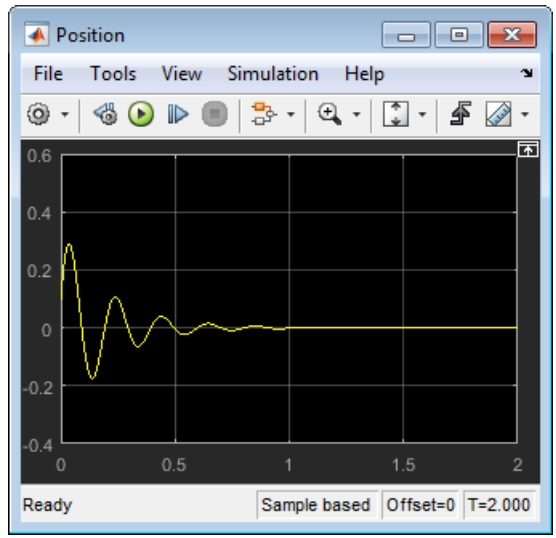
4 Refresh the Variable Viewer.



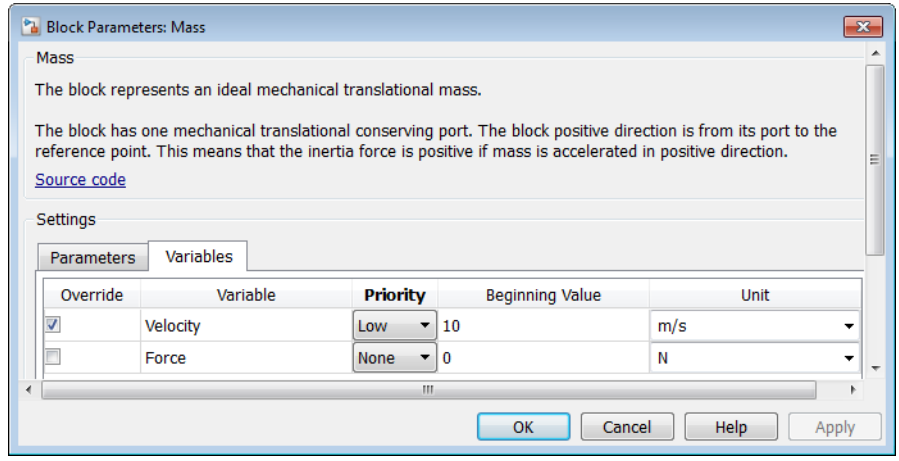
The overall status at the bottom of the Variable Viewer window now displays a yellow triangle and says that all the high-priority targets are satisfied, but some of the low-priority targets are not satisfied. There are now two yellow triangles in the status column: one for the low-priority force variable *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 on page 7-0 , and the simulation results are the same.

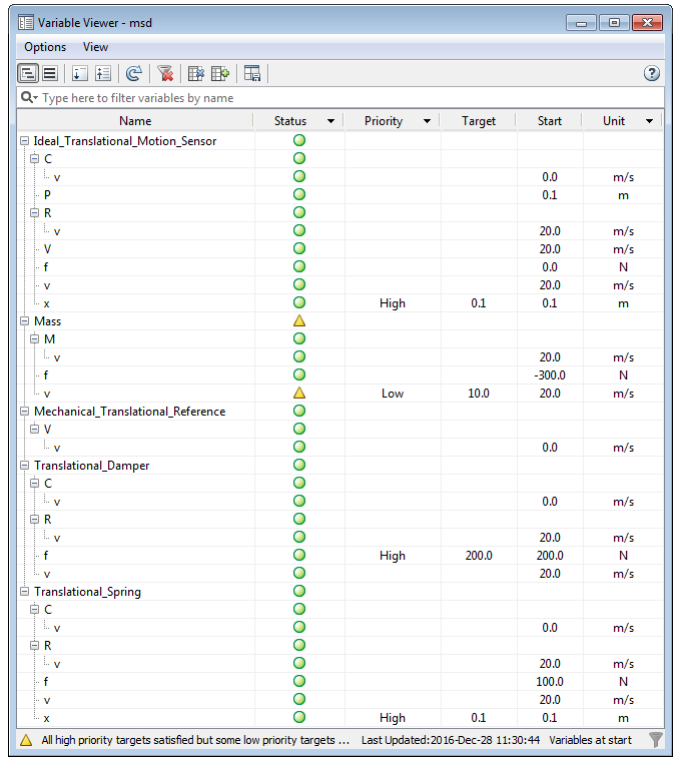




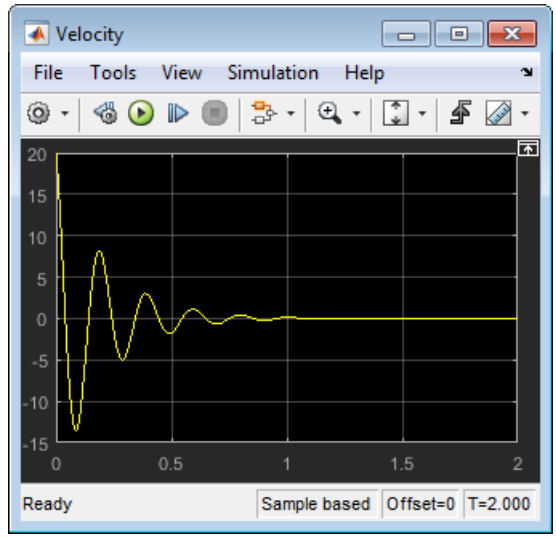
5 Another way to deal with over-specification is to keep the high priority on the damper force and relax the priority on mass initial velocity. Double-click the Translational Damper block again, go to the **Variables** tab, and change the priority of the **Force** variable back to `High`. Then double-click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

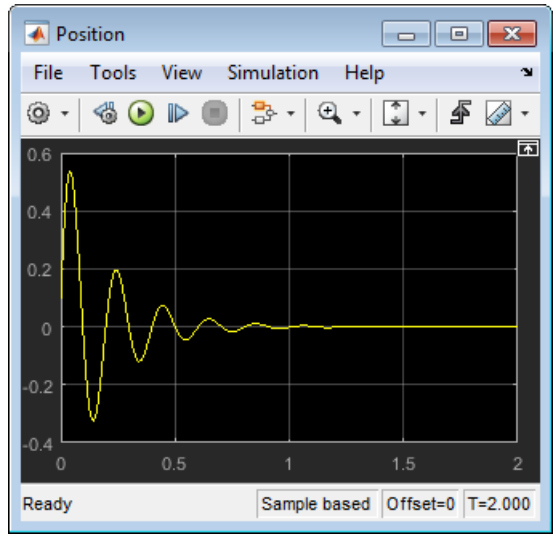


6 Refresh the Variable Viewer.



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.


Velocity
File
Tools
View
Simulation
Help
20
15
10
5
0
-5
-10
-15
0
0.5
1
1.5
2
Ready
Sample based
Offset=0
T=2.000



See Also

More About

- “Block-Level Variable Initialization” on page 7-2
- “Set Priority and Initial Target for Block Variables” on page 7-5
- “Variable Viewer” on page 7-25

Variable Viewer

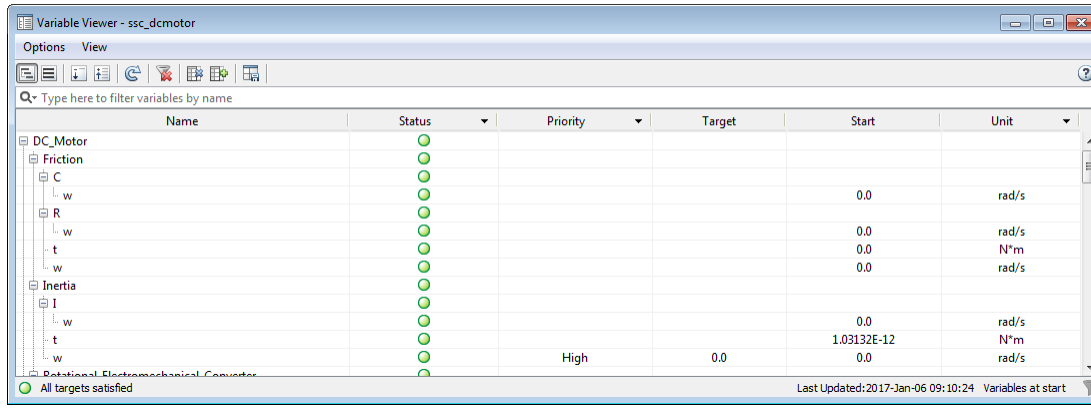
In this section...
“About Variable Viewer” on page 7-25
“Advanced Configuration” on page 7-28
“Switching Between Tree View and Flat View” on page 7-30
“Useful Filtering Techniques” on page 7-32
“Saving Viewer Configuration” on page 7-33
“Link to Block Diagram” on page 7-33
“Interaction with Model Updates and Simulation” on page 7-35

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**.

Note If you open a model, and then open the Variable Viewer before simulating the model, then the viewer does not contain any data. The **Refresh** button displays a warning symbol ( ), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.



The Variable Viewer is a table, its rows listing all the blocks in the model and all the public variables under each block, and the columns providing the initialization status, priority, target and actual start values, and other information for each variable.

By default, the Variable Viewer opens in basic configuration, unless you specified another configuration as a preferred one. (For information on specifying a preferred configuration, see “Saving Viewer Configuration” on page 7-33.) In basic configuration, the Variable Viewer has the following columns:

Name	Description
Status	Initialization status of each variable, can be one of: • 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 7-35.

Name	Description
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 7-5 and "Variable Priority for Model Initialization". If the variable has no initialization priority (`None` or `priority.none`), 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 7-32.

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. If the data in the Variable Viewer is out of sync with the model, he **Refresh** button displays a warning symbol ( ), and the timestamp at the bottom of the viewer window turns red. For more information, see “Interaction with Model Updates and Simulation” on page 7-35.

Clears all the column filtering options and displays all the rows in the table. For more information, see “Useful Filtering Techniques” on page 7-32.

Shows the Variable Viewer in its default, basic, configuration, with only the following columns displayed: **Status**, **Priority**, **Target**, **Start**, and **Unit**.

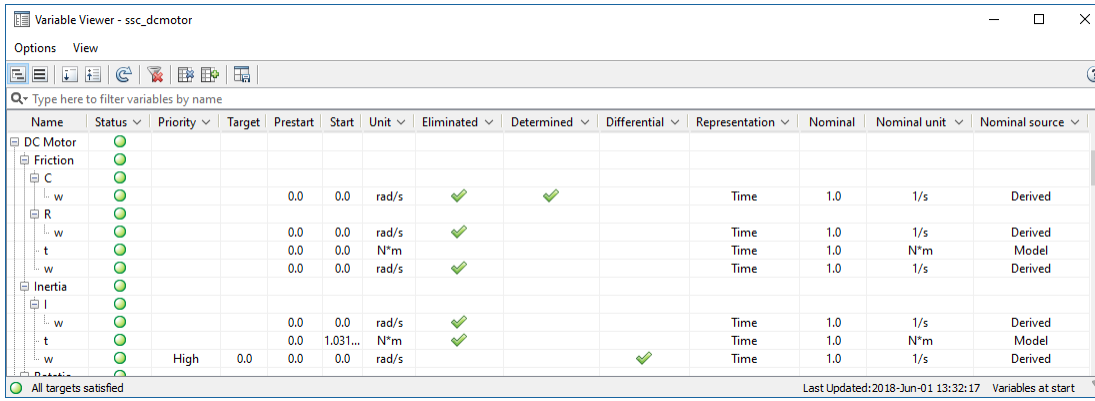
Shows the Variable Viewer in advanced configuration, with all the columns displayed. Use this view for troubleshooting your model, for example, if the model initialization failed.

Saves the current Variable Viewer configuration. For more information, see “Saving Viewer Configuration” on page 7-33.

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.

Name	Description
Determined	The values of these variables depend on the system inputs, or their values are predetermined based on the analysis of equations. Therefore, specifying initialization priority and targets for these variables has little or no impact on system solution. Also, if you specify a high-priority target for a predetermined variable, the solver most likely will not be able to satisfy this target but will spend extra time trying to find a second-stage solution.
Differential	Time derivatives of these variables appear in equations. These variables add dynamics to the system and can produce independent states. Therefore, these variables are more likely to require high initialization priority.
Representation	If frequency-and-time simulation mode is turned on, indicates how the solver marks the variables: **Frequency** ("fast") or **Time** ("slow"). For more information, see “Frequency and Time Simulation Mode” on page 6-37. In regular simulation, all variables are marked **Time**.
Nominal	Nominal value of the variable. For more information, see “System Scaling by Nominal Values” on page 6-30.
Nominal unit	Physical unit associated with the nominal value of the variable. For more information, see “System Scaling by Nominal Values” on page 6-30.
Nominal source	Source of the nominal value and unit: **Block**, **Model**, **Derived**, or **Fixed**. For more information, see “Possible Sources of Nominal Values and Their Evaluation Order” on page 6-31.

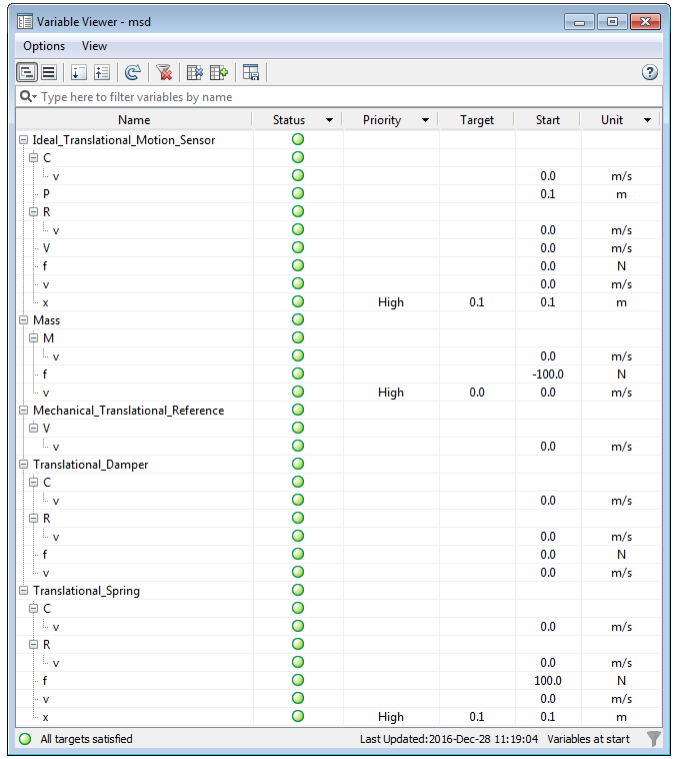
You can change the default order of columns by clicking a column heading and dragging it, while holding down the mouse button, to the desired location. You can also hide columns by right-clicking their headers and selecting **Hide This Column** from the context menu, or clearing the check mark next to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

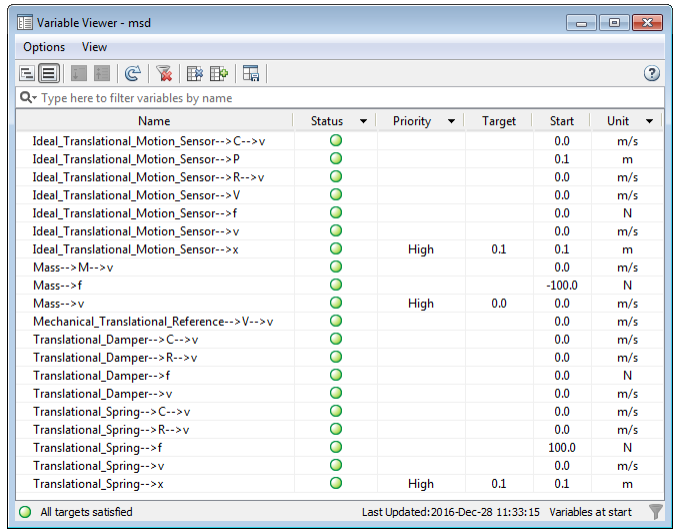
You can control the number of rows in the Variable Viewer by switching between the tree view (the default) and the flat view. By default, the Variable Viewer opens in tree view, with variable nodes grouped under the parent port, block, and subsystem nodes.

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).



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



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.

Saving Viewer Configuration

The **Save Viewer Configuration** button ( ) in the Variable Viewer toolbar lets you save the following configuration preferences:

- Variable Viewer view type (tree or flat)
- Visible columns
- Ordering of columns
- Filters applied for all columns (both visible and hidden)
- Sorting on a specific column

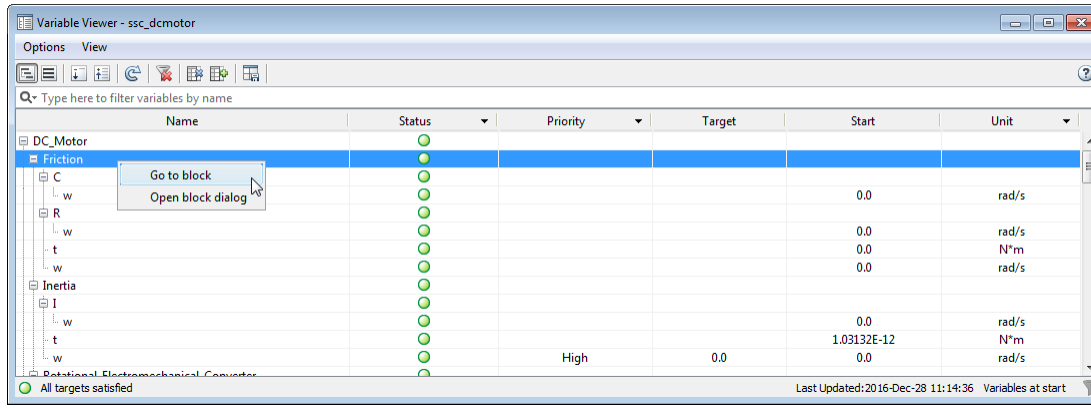
If you save viewer configuration, then the next time you open Variable Viewer, for this or another model, it will open with the same configuration. This behavior is consistent with saving other MATLAB preferences.

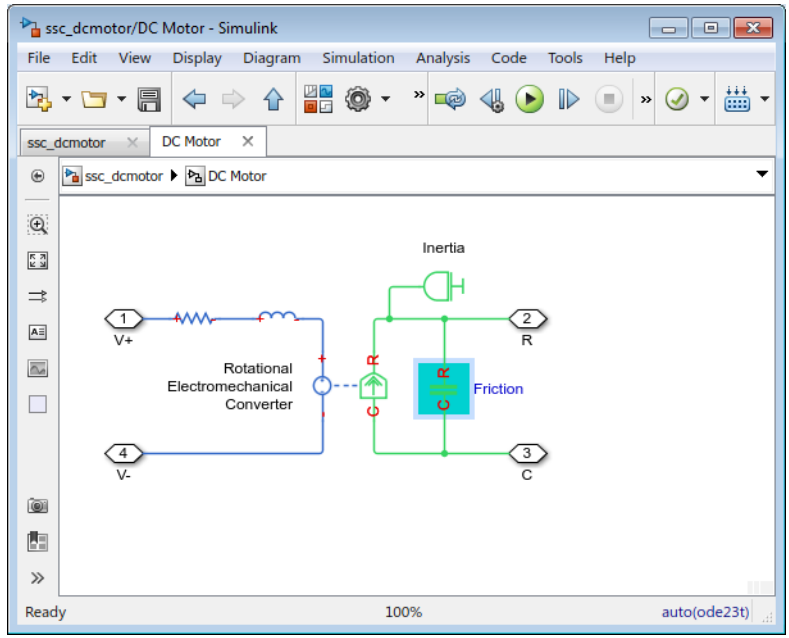
Link to Block Diagram

The Variable Viewer tool provides direct linking to the block diagram. This link lets you highlight the appropriate block, or easily go from a variable listed in the Variable Viewer to the **Variables** tab in the corresponding block dialog box, to modify the variable priorities and targets.

When you right-click in the **Name** column of any row in the Variable Viewer table, a context menu opens with the following options:

- **Go to block** — Highlights the corresponding block in the block diagram, opening the appropriate subsystem if needed. If the row represents a variable, highlights the parent block for this variable.
- **Open block dialog** — Opens the corresponding block dialog box (for a variable, opens the parent block dialog box). In the block dialog box, 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.





Interaction with Model Updates and Simulation

Opening the Variable Viewer does not trigger an automatic update. For complex models, computing initial values for all the variables can last several minutes, and unnecessary updates could lead to loss of productivity. You have to update the data explicitly by clicking the **Refresh** button ( ).

When you open the Variable Viewer, it gets populated with the data from the last simulation. The status at the bottom of the viewer window displays the timestamp of its last update. If you have modified the model since the viewer has last been updated, the **Refresh** button displays a warning symbol ( ), and the timestamp at the bottom of the viewer window turns red to indicate that the data in the viewer might not reflect the latest model changes.

If you open a model, and then open the Variable Viewer before simulating the model, then the viewer does not contain any data. The **Refresh** button displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.

The Variable Viewer computes the actual initial values of the variables by running the simulation for 0 seconds. Therefore:

- The model must be in an executable state when you refresh the viewer, otherwise you get an error message.
- If the scopes are open, they turn blank every time you refresh the viewer. Rerun the simulation to see the new results.
- If you rerun the simulation while the Variable Viewer is open, the results in the viewer are automatically refreshed when the simulation starts running.
- If you change variable priorities and targets or adjust the block parameters, the results in the viewer are not updated automatically. Refresh the viewer (by clicking 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.

See Also

More About

- “Block-Level Variable Initialization” on page 7-2
- “Initialize Variables for a Mass-Spring-Damper System” on page 7-7

Using Operating Point Data for Model Initialization

In this section...
“Using Operating Points to Initialize Model Variables” on page 7-37
“Suggested Workflow” on page 7-38
“Extracting Variable Initialization Data into an Operating Point” on page 7-38
“Manipulating Operating Point Data” on page 7-39
“Applying Operating Point Data to Initialize Model” on page 7-39

Using Operating Points to Initialize Model Variables

Block-level variable initialization lets you specify the priority and target for individual block variables. You can also initialize variables for a whole model from the saved operating point data.

You can use `OperatingPoint` objects to save sets of data necessary to initialize a model, manipulate this data, and then use it to initialize another model, or the same model before another simulation run. These sets of data contain a hierarchy of variable initialization targets. Each target consists of a variable value, unit, and initialization priority, as described in “Variable Initialization Priority” on page 7-3.

The `OperatingPoint` data hierarchy is a tree, with nodes corresponding to subsystems and blocks in a model. At the lowest level of the data tree, inside the block nodes, are the variable initialization targets for that block.

When you use an `OperatingPoint` to initialize a model, the solver matches the `OperatingPoint` data hierarchy to the model hierarchy and applies the initialization targets from the operating point to the respective model variables. If there is no variable matching an operating point target, this target is ignored. After applying all the data from the operating point, the solver performs model initialization as described in “Initial Conditions Computation” on page 6-10.

After you initialize the variables and prior to simulating the model, you can open the Variable Viewer to see which of the variable targets have been satisfied. For details, see “Variable Viewer” on page 7-25.

Suggested Workflow

1. Create an `OperatingPoint` object by extracting data from the model or from the simulation log. For more information, see “Extracting Variable Initialization Data into an Operating Point” on page 7-38.
2. Modify the operating point data, if needed, by changing, adding, or removing targets and nodes. For more information, see “Manipulating Operating Point Data” on page 7-39.
3. When satisfied with the operating point data, apply it to initialize another model, or the same model for another simulation run. For more information, see “Applying Operating Point Data to Initialize Model” on page 7-39.

Extracting Variable Initialization Data into an Operating Point

You can create an `OperatingPoint` object by extracting data from an existing model or from logged simulation data. For more information, see `simscape.op.create`.

You can extract variable initialization targets from a model in these ways:

- Start values — Initialize the model and use the variable targets corresponding to the `Start` values in the Variable Viewer.
- Prestart values — Update the model and use the variable targets corresponding to the `Prestart` values in the Variable Viewer.
- Cached data — Extract cached values of variable targets from a model that has been previously initialized or simulated. You can specify `Start` or `Prestart` values. This method lets you save time by avoiding repeated initialization of the model if the data that you want to extract has not changed.

Alternatively, you can simulate the model while logging simulation data, and then extract variable targets from the simulation log at a specified time, t:

- If the set of times recorded in the simulation data log contains an exact match for time t, then the `simscape.op.create` function extracts these variable target values into the operating point data.
- If there is no exact match, but t is between the minimum and maximum times in the simulation data log, then the function uses linear interpolation to determine the target values.
- If t is less than the minimum time, then the function extracts the first value for each variable in the simulation data log.

- If t is greater than the maximum time, then the function extracts the last value for each variable in the simulation data log.

When you extract data from a model into an operating point, the elements in the data hierarchy of the `OperatingPoint` object match the structure of the model. The operating point data tree has nodes corresponding to subsystems and blocks in the model, with variable initialization targets for each block at the lowest level of the data tree hierarchy. Similarly, when you extract an operating point from logged simulation data, the operating point data tree matches the data tree of the simulation log. For an example, see “Find Relative Path to Block Node in Operating Point Data Tree”.

Manipulating Operating Point Data

You can create an empty `OperatingPoint` object, or populate it with the data extracted from an existing model or from logged simulation data.

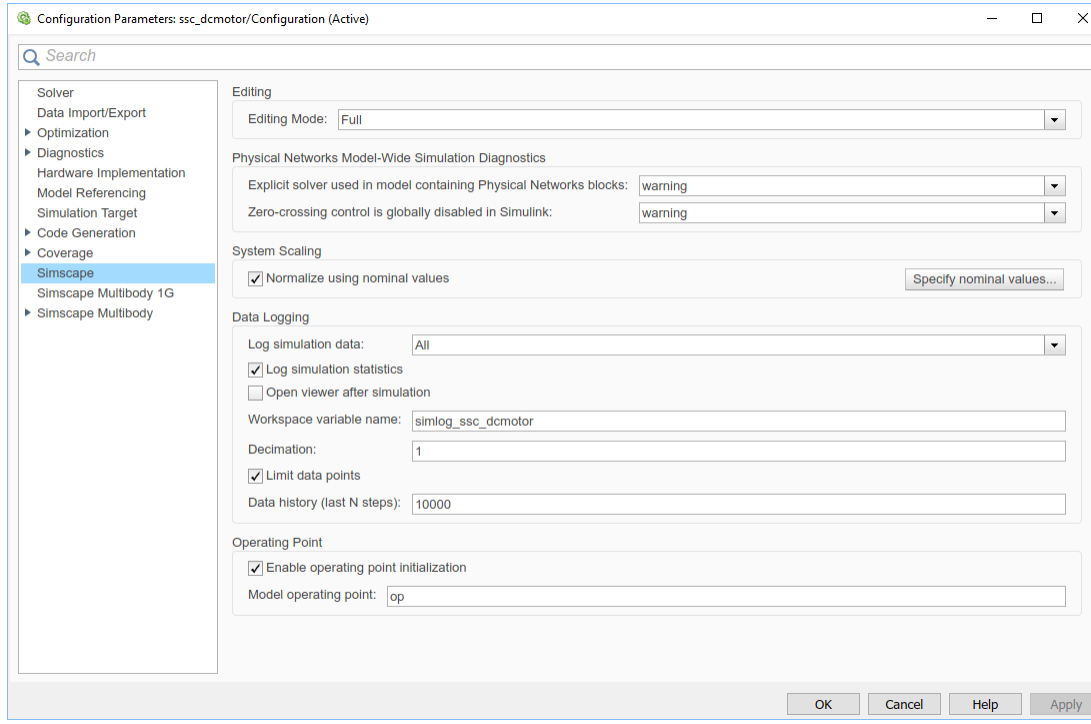
Once you create an `OperatingPoint` object, you can modify it in these ways:

- Add targets one-by-one. For an example, see “Add Element to an Operating Point”.
- Copy and insert elements. For an example, see “Copy Element from an Operating Point”. You can then insert the copied element into another operating point using the `set` function.
- Remove elements. For an example, see “Remove an Element from Operating Point Data”.
- Rename or move elements. For an example, see “Rename Element to Match New Block Name”.
- Merge operating points. For an example, see “Merge Two Operating Points”.

Applying Operating Point Data to Initialize Model

To initialize a model from an operating point:

1. Open the Configuration Parameters dialog box.
2. On the **Simscape** pane, select the **Enable operating point initialization** check box.
3. In the **Model operating point** textbox, enter the name of the workspace variable associated with an `OperatingPoint` object.



You can also use the equivalent command-line interface to set the model configuration parameters:

- `set_param('model_name','SimscapeUseOperatingPoints','on');`
- `set_param('model_name','SimscapeOperatingPoint','op_name');`

where *`model_name`* is the name of the model and *`op_name`* is the name of the `OperatingPoint` object.

See Also

`simscape.op.OperatingPoint` | `simscape.op.Target`

More About

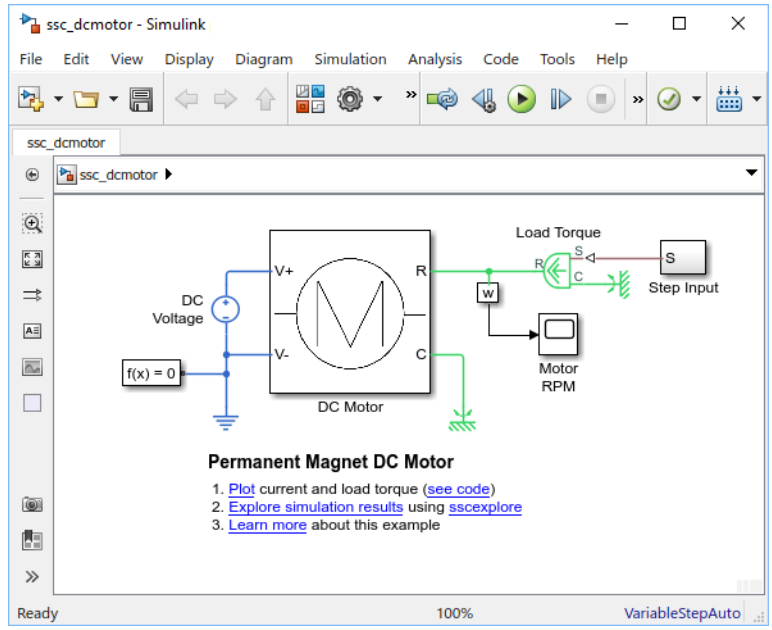
- “Initialize Model Using Operating Point from Logged Simulation Data” on page 7-42

- “Variable Viewer” on page 7-25

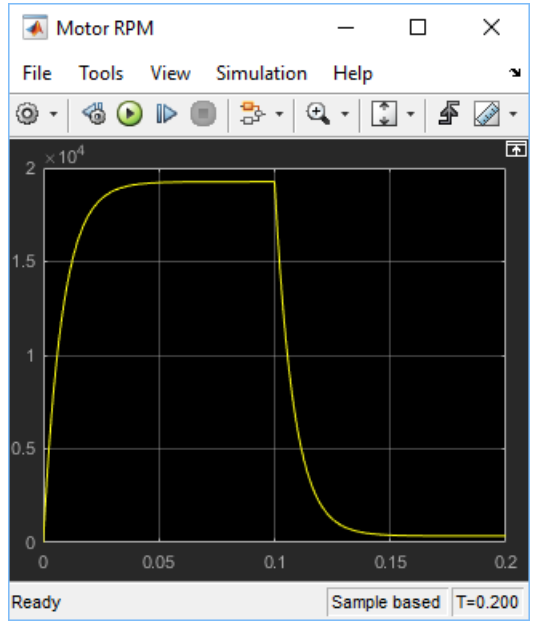
Initialize Model Using Operating Point from Logged Simulation Data

This example shows how you can create an `OperatingPoint` object from logged simulation data and then use this operating point to initialize the model for a subsequent simulation run.

1. Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This model has data logging enabled for the whole model, with the **Workspace variable name** parameter set to `simlog_ssc_dcmotor`.



2. Simulate the model to log the simulation data.
3. Examine the simulation results in the Motor RPM scope window.



For the first 0.1 seconds, the motor has no external load, and the speed builds up to the no-load value. Then at 0.1 seconds, the stall torque is applied as a load to the motor shaft.

4 Create an operating point from logged simulation data at 0.1 seconds after the start of simulation:

```
op = simscape.op.create(simlog_ssc_dcmotor, 0.1)
```

```
op =

  OperatingPoint with children:
  -----------------------------
   DC Motor
   DC Voltage
   ERef
   Load Torque
   MRRef Motor
   MRRef Torque
   Sensing
  -----------------------------
```

5 Enable model initialization from operating point:

```
set_param(gcs,'SimscapeUseOperatingPoints','on');
```

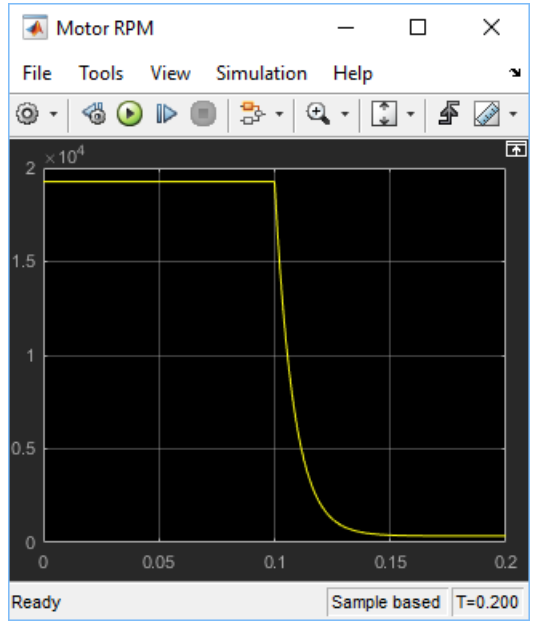
This command is equivalent to selecting the **Enable operating point initialization** check box in the **Simscape** pane of the Configuration Parameters dialog box.

6 Specify the name of operating point:

```
set_param(gcs,'SimscapeOperatingPoint','op');
```

This command is equivalent to entering `op` in the **Model operating point** textbox.

7 Simulate the model. The simulation now starts with the full no-load speed.



See Also

More About

- “Log, Navigate, and Plot Simulation Data” on page 12-25

- “Using Operating Point Data for Model Initialization” on page 7-37

Linearization and Trimming

- “Finding an Operating Point” on page 8-2
- “Linearizing at an Operating Point” on page 8-6
- “Linearize an Electronic Circuit” on page 8-12
- “Linearize a Plant Model for Use in Feedback Control Design” on page 8-22

Finding an Operating Point

In this section...
“What Is an Operating Point?” on page 8-2
“Finding Operating Points in Physical Models” on page 8-3

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 $\boldsymbol{x}$ and inputs $\boldsymbol{u}$. 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, $\boldsymbol{y}$, in addition to inputs.

See “Linearizing at an Operating Point” on page 8-6.

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 $\boldsymbol{x}_0$ and input vector $\boldsymbol{u}_0$.

- "Simulating in Time to Search for an Operating Point" on page 8-3
- "Using the Simscape Initial Condition Solver" on page 8-4
- "Using Simulink Control Design Techniques to Find Operating Points" on page 8-4
- "Using Sources to Find Operating Points Not Recommended" on page 8-5
- "Simulink trim Function Not Supported with Simscape Models" on page 8-5

Simulating in Time to Search for an Operating Point

One way to identify operating points is to simulate your model and inspect its state $\boldsymbol{x}$ and output $\boldsymbol{y}$ as a time series.

1. In your Simscape model, set up sensor outputs for whatever block outputs you want to observe.
2. Connect Scope blocks, To Workspace blocks, or both, to your Simscape block outputs to observe and record simulation behavior.
3. In the **Data Import/Export** pane of your model Configuration Parameters settings, select the **Time**, **States**, and **Output** check boxes to record this simulation information in your workspace.

Using the Simscape Initial Condition Solver

Simscape software provides two workflows to initialize a physical model. The first solves for steady state, where all differential variables have zero derivative. Using this approach you can search for multiple steady states with the steady-state solver by varying the model inputs, parameters, and initial conditions. The second approach is to directly specify initial conditions by specifying initialization priority and targets for block variables. For more information on this approach, see “Variable Initialization”.

To use the first approach, enable the steady-state solver:

1. In each, some, or all of the physical networks in your Simscape model, open the Solver Configuration block.
2. In each 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 $\boldsymbol{x}$ values in your workspace.

   If you also have input signals $\boldsymbol{u}$ in the model, you can capture those inputs by connecting To Workspace blocks to the input Simulink signal lines.
4. Close these dialog boxes and start simulation.

The first vector of values $\boldsymbol{x}(t=0)$ that you capture during simulation reflects the steady state $\boldsymbol{x}_0$ that the Simscape solver identified.

Tip Finding an initial steady state is part of the nondefault Simscape simulation sequence. See “Initial Conditions Computation” on page 6-10.

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

Using Simulink Control Design Techniques to Find Operating Points

You can use Simulink Control Design software to find operating points for models with Simscape components. Simulink Control Design provides both command-line and graphical interfaces for finding and analyzing operating points.

For more information, see “Find Steady-State Operating Points for Simscape Models” (Simulink Control Design).

Using Sources to Find Operating Points Not Recommended

You can impose an operating specification on part of a Simscape model by inserting source blocks from the Simscape Foundation Library. These impose specified values of system variables in parts of the model. You can simulate and save the state vector.

However, you cannot obtain an operating point for the original system (without the source blocks) by saving the state values from the model and then removing the source blocks. In general, the number, order, and identity of state components change after adding and removing Simscape blocks in a model.

Simulink trim Function Not Supported with Simscape Models

The Simulink `trim` function is not supported for models containing Simscape components.

Linearizing at an Operating Point

In this section...
“What Is Linearization?” on page 8-6
“Linearizing a Physical Model” on page 8-8

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 8-2.

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

What Is a Linearized Model?

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

If the system dynamics do not explicitly depend on time and the operating point is a steady state, the system response to state and input perturbations near the steady state is approximately governed by a *linear time-invariant* (LTI) state space model:

$$d\boldsymbol{x}/dt = A\cdot\boldsymbol{x} + B\cdot\boldsymbol{u}$$

$$\boldsymbol{y} = C\cdot\boldsymbol{x} + D\cdot\boldsymbol{u}.$$

The matrices A, B, C, D have components and structures that are independent of the simulation time. A system is stable to changes in state at an operating point if the eigenvalues of A are negative.

If the operating point is not a steady state or the system dynamics depend explicitly on time, the linearized dynamics near the operating point are more complicated. The

matrices A, B, C, D are not constant and depend on the simulation time t_0 , as well as the operating point $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$.

Tip While you can linearize a closed system with no inputs or outputs and obtain a nonzero A matrix, obtaining a nontrivial linearized input-output model requires at least one input component in $\boldsymbol{u}$ and one output component in $\boldsymbol{y}$.

Example

A pilot is flying, or simulating, an aircraft in level, constant-velocity, and constant-altitude flight relative to the ground. A crucial question for the aircraft pilot and designers is: will the aircraft return to the steady state if perturbed from it by a disturbance, such as a wind gust — in other words, is this steady state stable? If the operating point is unstable, the aircraft trajectory can diverge from the steady state, requiring human or automatic intervention to maintain steady flight.

Choosing a Good Operating Point for Linearization

Although steady-state and other operating points (state $\boldsymbol{x}_0$ and inputs $\boldsymbol{u}_0$) 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 8-8
- "Linearizing with Simulink Control Design Software" on page 8-9
- "Linearizing with the Simulink linmod and dlinmod Functions" on page 8-9
- "Linearizing with Simulink Linearization Blocks" on page 8-11

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 6-7.

The complete, unreduced LTI A, B, C, D matrices have the following structure.

- The `A` matrix, of size `n_states` by `n_states`, is all zeros except for a submatrix of size `n_ind` by `n_ind`, where `n_ind` is the number of independent states.
- The `B` matrix, of size `n_states` by `n_inputs`, is all zeros except for a submatrix of size `n_ind` by `n_inputs`.
- The `C` matrix, of size `n_outputs` by `n_states`, is all zeros except for a submatrix of size `n_outputs` by `n_ind`.
- The `D` matrix, of size `n_outputs` by `n_inputs`, can be nonzeros everywhere.

Obtaining the Independent Subset of States

A minimal linearized solution uses only an independent subset of system states. From the matrices *A*, *B*, *C*, *D*, you can obtain a minimal input-output linearized model with:

- The `minreal` and `sminreal` functions from Control System Toolbox™ software
- Automatically with the Simulink Control Design approach

Linearizing with Simulink Control Design Software

Note The techniques described in this section require the Simulink Control Design product.

You must use the features of this product on the Simulink lines in your model, not directly on Simscape physical network lines or blocks.

This approach requires that you start with an operating point object saved from trimming the model to an operating specification.

To linearize a model with an operating point object, use the `linearize` function, customizing where necessary. The resulting state-space object contains the matrices *A*, *B*, *C*, *D*.

You can also use the graphical user interface, through the model menu bar: **Analysis > Control Design > Linear Analysis**.

For more information on linearizing Simscape models using Simulink Control Design, see "Linearize Simscape Networks" (Simulink Control Design).

Linearizing with the Simulink linmod and dlinmod Functions

You have several ways that you can use the Simulink functions `linmod` and `dlinmod`, and the linearization results can differ depending on the method chosen. To use these functions, you do not have to open the model, just have the model file on your MATLAB path.

For more information about Simulink linearization, see "Linearizing Models" (Simulink).

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 $\boldsymbol{x}_0$ and

inputs $\boldsymbol{u}_0$ 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

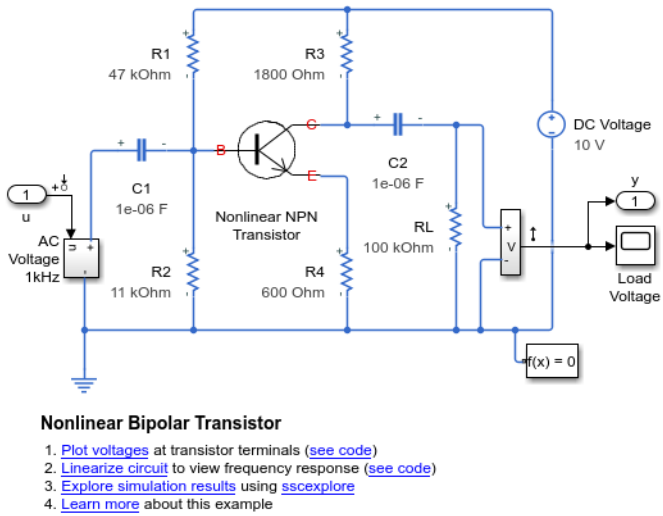
This example shows how to linearize a model of a nonlinear bipolar transistor circuit and create a Bode plot for small-signal frequency-domain analysis.

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 8-12
“Linearize with Steady-State Solver and linmod Function” on page 8-16
“Linearize with Simulink Control Design Software” on page 8-18
“Use Control System Toolbox Software for Bode Analysis” on page 8-19

Explore the Model

To open the Nonlinear Bipolar Transistor example model, type `ssc_bipolar_nonlinear` in the MATLAB Command Window.



The model represents a single-transistor audio amplifier. The transistor is an NPN bipolar device, and as such has a nonlinear set of current-voltage characteristics. Therefore the

overall behavior of the amplifier is dependent on the operating point of the transistor. The transistor itself is represented by and Ebers-Moll equivalent circuit implemented using a masked subsystem. The circuit has a sinusoidal input test signal with amplitude 10 mV and frequency 1 kHz. The Load Voltage scope displays the resulting collector output voltage after the DC is filtered out by the output decoupling capacitor.

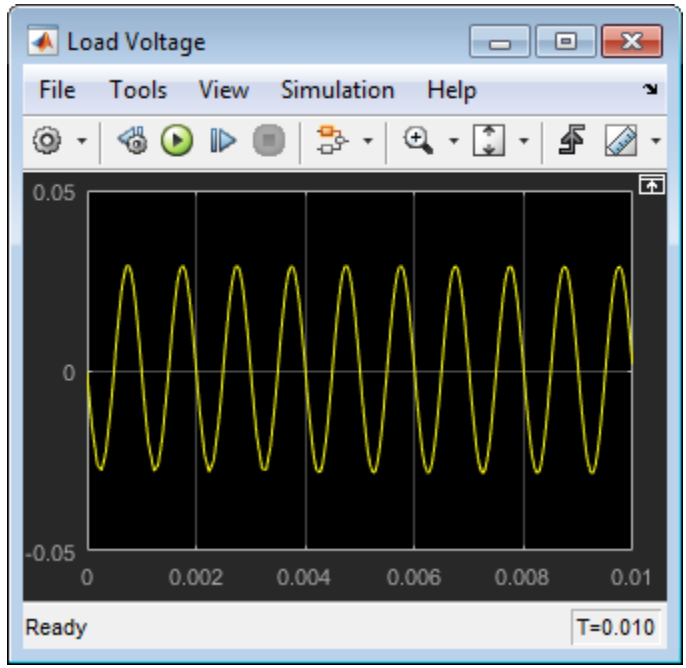
R1 and R2 set the nominal operating point, and the small signal gain is approximately set by the ratio R3/R4. The decoupling capacitors C1 and C2 have a capacitance of 1uF, to present negligible impedance at 1 kHz.

The model is configured for linearization. You can quickly generate and view the small-signal frequency response by clicking the `Linearize circuit` hyperlink in model annotation. To view the MATLAB script that generates the frequency response, click the next hyperlink in that annotation, `see code`. This documentation provides background information and alternative ways of linearization based on the software you have.

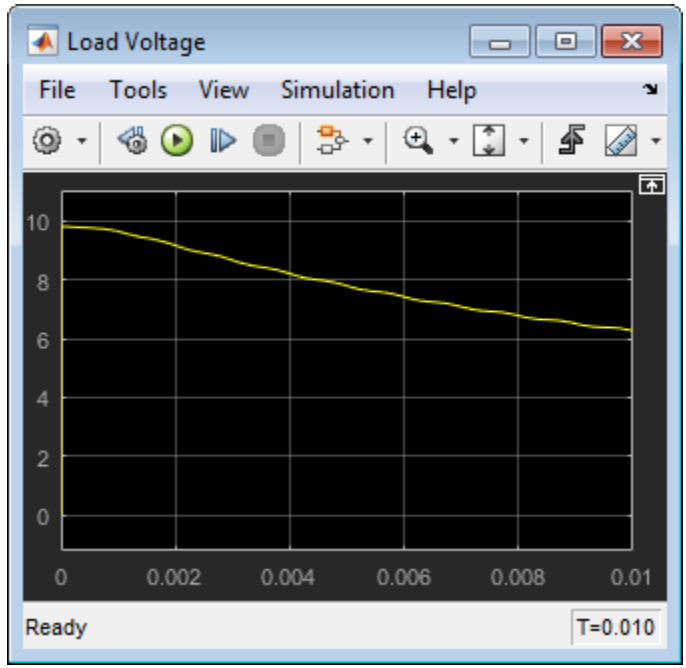
In general, to obtain a nontrivial linearized input-output model and generate a frequency response, you must specify model-level inputs and outputs. The Nonlinear Bipolar Transistor model meets this requirement in two ways, depending on how you linearize:

- Simulink requires top- or model-level input and output ports for linearization with `linmod`. The Nonlinear Bipolar Transistor model has such ports, marked `u` and `y`.
- Simulink Control Design software requires that you specify input and output signal lines with linearization points. The specified lines must be Simulink signal lines, not Simscape physical connection lines. The Nonlinear Bipolar Transistor model has such linearization points specified. For more information on using Simulink Control Design software for trimming and linearization, see documentation for that product.

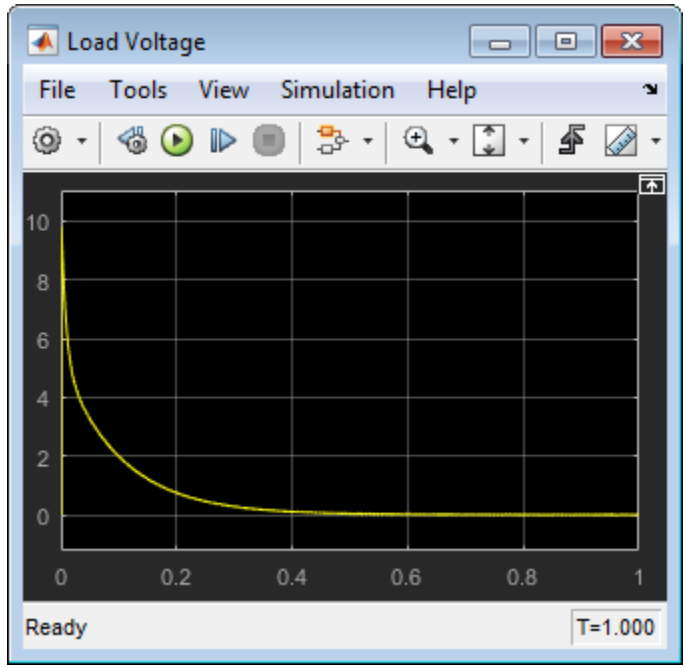
Open the Solver Configuration block and see that the **Start simulation from steady state** check box is selected. Then open the Load Voltage scope and run the simulation to see the basic circuit behavior. The transistor junction capacitance initial voltages are set to be consistent with the bias conditions defined by the resistors. The output is a steady sinusoid with zero average, its amplitude amplified by the transistor and bias circuit.



To see the circuit relax from a nonsteady initial state, in the Solver Configuration block, clear the **Start simulation from steady state** check box and click **OK**. With the Load Voltage scope open, simulate again. In this case, the output voltage starts at zero because the transistor junction capacitances start with zero charge.



You can get a more comprehensive understanding of the circuit behavior and how it approaches the steady state by long-time transient simulation. Increase the simulation time to 1 s and rerun the simulation. The circuit starts from its initial nonsteady state, and the transistor collector voltage approaches and eventually settles into steady sinusoidal oscillation.



Open the Solver Configuration block, select the **Start simulation from steady state** check box (as it was when you first opened the model), and click **OK**. Change the simulation time back to .01 s and rerun the simulation.

Linearize with Steady-State Solver and linmod Function

In this example, you:

1. Use the Simscape steady-state solver to find an operating point
2. Linearize the model using the Simulink `linmod` function
3. Generate the Bode plot using a series of MATLAB commands

Open the Solver Configuration block and make sure the **Start simulation from steady state** check box is selected. When you simulate the model with the Simscape steady-state solver enabled, the circuit is initialized at the state defined by the transistor bias resistors. This steady-state solution is an operating point suitable for linearization.

Note Also make sure that the **Use local solver** check box is cleared. Linearizing a model with the local solver enabled is not supported.

To linearize the model, type the following in the MATLAB Command Window:

```
[a,b,c,d]=linmod('ssc_bipolar_nonlinear');
```

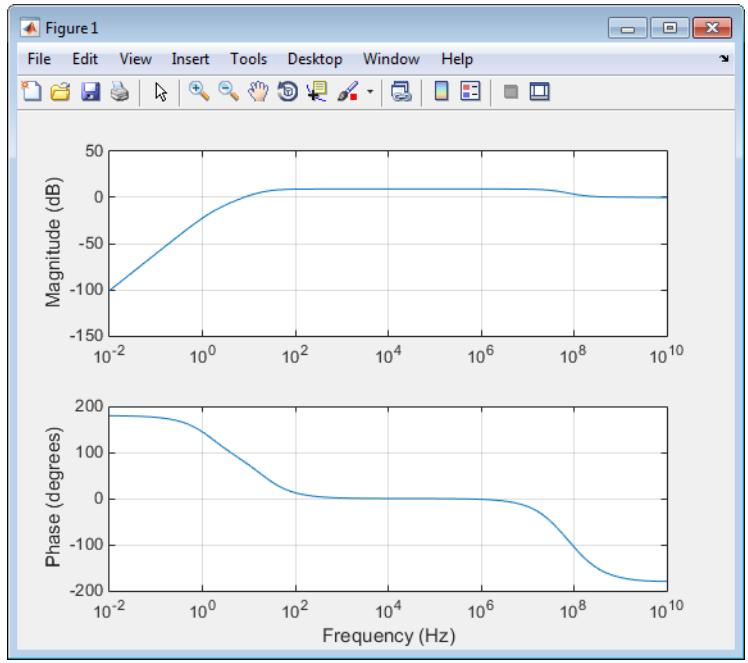
You can alternatively call the `linmod` function with a single output argument, in which case it generates a structure with states, inputs, and outputs, as well as the linear time-invariant (LTI) model.

The state vector of the Nonlinear Bipolar Transistor model contains 17 components. The full model has one input and one output. Thus, the LTI state-space model derived from linearization has the following matrix sizes:

- *a* is 17-by-17
- *b* is 17-by-1
- *c* is 1-by-17
- *d* is 1-by-1

To generate a Bode plot, type the following in the MATLAB Command Window:

```
npts = 100; f = logspace(-2,10,npts); G = zeros(1,npts);
for i=1:npts
    G(i) = c*(2*pi*1i*f(i)*eye(size(a))-a)^-1*b +d;
end
subplot(211), semilogx(f,20*log10(abs(G)))
grid
ylabel('Magnitude (dB)')
subplot(212), semilogx(f,180/pi*unwrap(angle(G)))
ylabel('Phase (degrees)')
xlabel('Frequency (Hz)')
grid
```

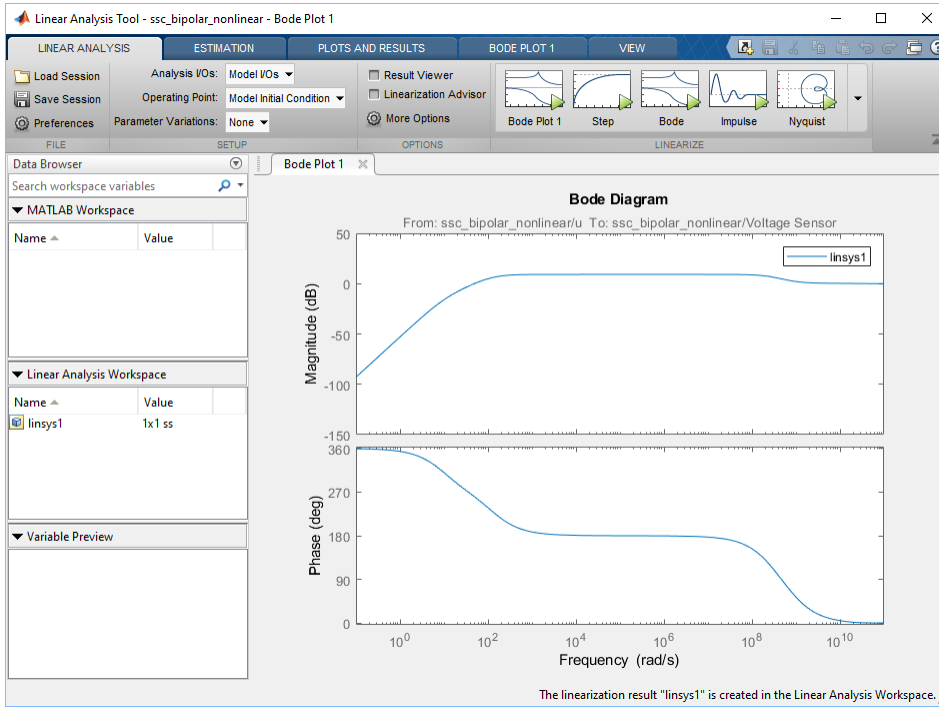


Linearize with Simulink Control Design Software

Note To work through this section, you must have a Simulink Control Design license.

Simulink Control Design software has tools that help you find operating points and returns a state-space model object that defines state names. This is the recommended way to linearize Simscape models.

1. In the 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

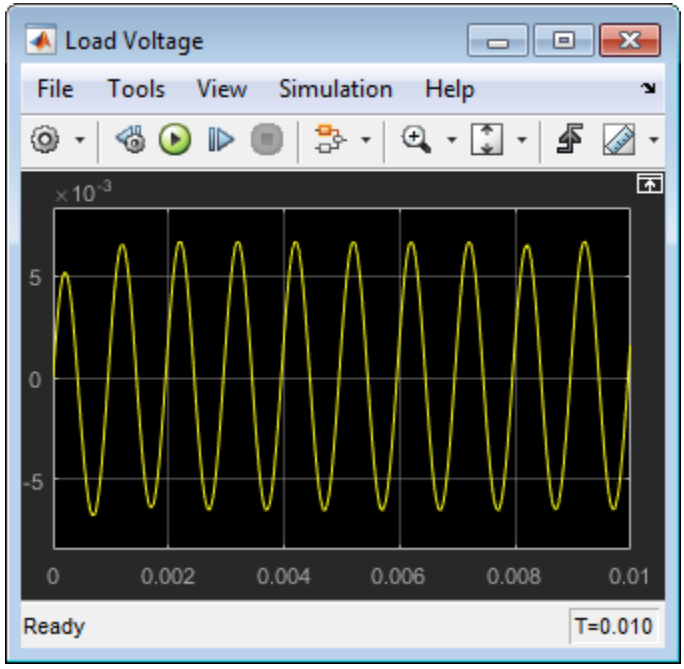
implies one or more zero eigenvalues. To analyze the LTI model, reduce the LTI matrices to a minimal realization. Obtain a minimal realization using the `minreal` function.

```
[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.

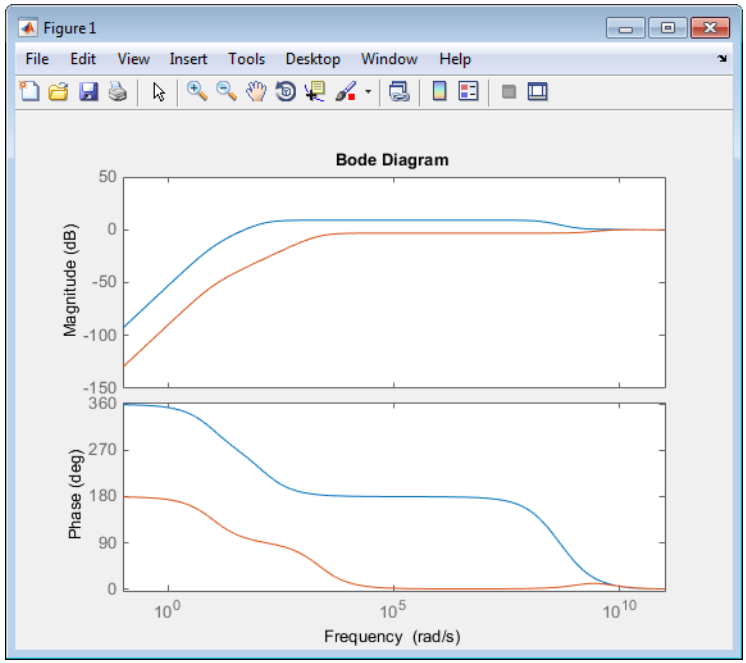
```
bode(a0,b0,c0,d0) % Creates first Bode plot
```

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.

See Also

Related Examples

- “Linearize a Plant Model for Use in Feedback Control Design” on page 8-22

More About

- “Finding Operating Points in Physical Models” on page 8-3
- “Linearizing a Physical Model” on page 8-8

Linearize a Plant Model for Use in Feedback Control Design

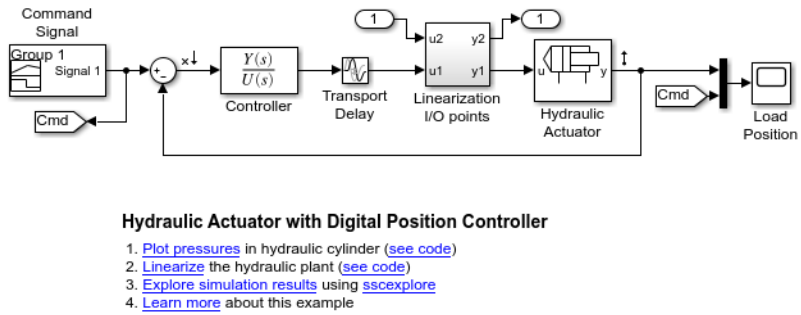
This example shows how you can linearize a hydraulic plant model to support control system stability analysis and design.

Depending on the software you have available, use the appropriate sections of this example to explore various linearization and analysis techniques.

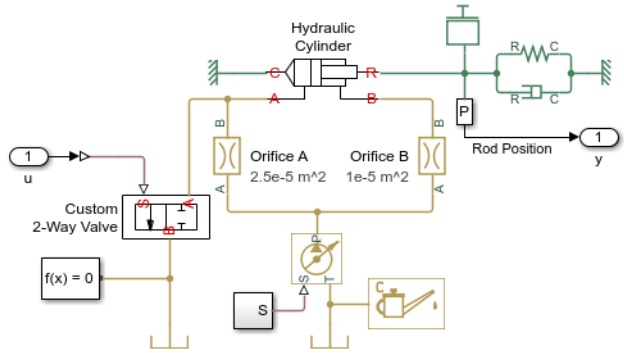
In this section...
“Explore the Model” on page 8-22 “Trim Using the Controller and Linearize with Simulink linmod Function” on page 8-25 “Linearize with Simulink Control Design Software” on page 8-26

Explore the Model

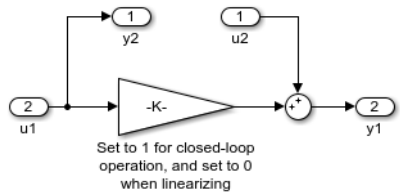
To open the Hydraulic Actuator with Digital Position Controller example model, type `ssc_hydraulic_actuator_digital_control` in the MATLAB Command Window. To display hidden block names for training purposes, select **Display** and clear the **Hide Automatic Names** check box.



The model represents a two-way valve acting in a closed-loop circuit together with a double-acting cylinder. Double-click the Hydraulic Actuator subsystem to see the model configuration.



The controller is represented as a continuous-time transfer function plus a transport delay that allows for computational time and a zero-order hold when implemented in discrete time. The Linearization I/O points subsystem lets you easily break and restore the feedback control loop by setting the base workspace variable ClosedLoop to 0 or 1, respectively.



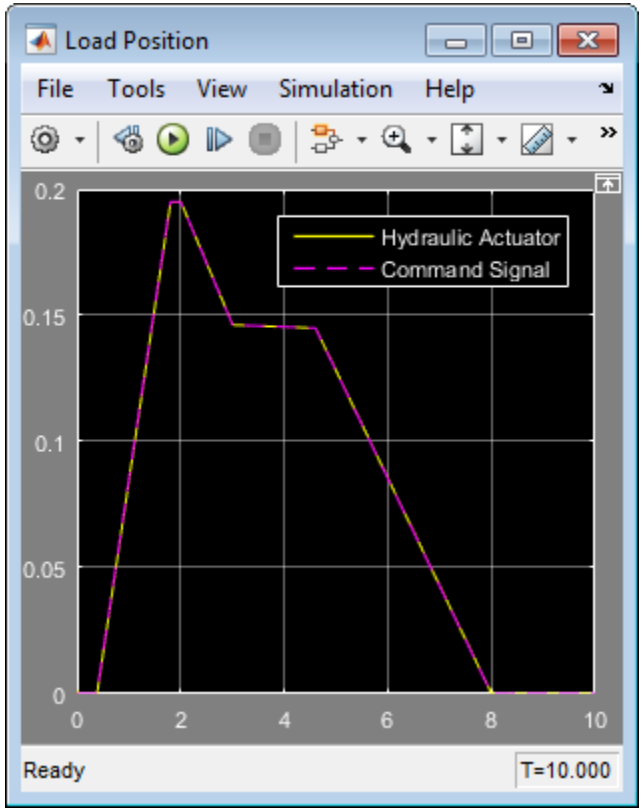
You can quickly generate and view the small-signal frequency response by clicking the `Linearize` hyperlink in model annotation. To view the MATLAB script that generates the frequency response, click the next hyperlink in that annotation, `see code`. This documentation provides background information and alternative ways of linearization based on the software you have.

In general, to obtain a nontrivial linearized input-output model and generate a frequency response, you must specify model-level inputs and outputs. The Hydraulic Actuator with Digital Position Controller model meets this requirement in two ways, depending on how you linearize:

- Simulink requires top- or model-level input and output ports for linearization with `linmod`. The model has such ports, marked `In1` and `Out1`.

- Simulink Control Design software requires that you specify input and output signal lines with linearization points. The specified lines must be Simulink signal lines, not Simscape physical connection lines. The model has such linearization points specified. For more information on using Simulink Control Design software for trimming and linearization, see documentation for that product.

Open the Load Position scope and simulate the model in a normal closed-loop controller configuration.



You can see that the model has a quasi-linear steady-state response between 2 and 3 seconds, when the two-way valve is open. Therefore, the state at 2.5 seconds is an operating point suitable for linearization.

Trim Using the Controller and Linearize with Simulink linmod Function

1. Set the controller parameters.

   To specify sample time for controller discrete-time implementation, type the following in the MATLAB Command Window:

   ```
   ts = 0.001;
   ```

   To specify continuous-time controller numerator and denominator, type:

   ```
   num = -0.5;
   den = [1e-3 1];
   ```

2. Find an operating point by running closed-loop and selecting the state at 2.5 seconds when the custom two-way valve is open.

   To close the feedback loop, type:

   ```
   assignin('base','ClosedLoop',1);
   ```

   To simulate the model and save the operating point information in the form of a state vector *X* and input vector *U*, type:

   ```
   [t,x,y] = sim('ssc_hydraulic_actuator_digital_control');
   idx = find(t>2.5,1);
   X = x(idx,:); U = y(idx);
   ```

3. Linearize the model using the Simulink `linmod` function.

   To break the feedback loop, type:

   ```
   assignin('base','ClosedLoop',0);
   ```

   To linearize the model, type:

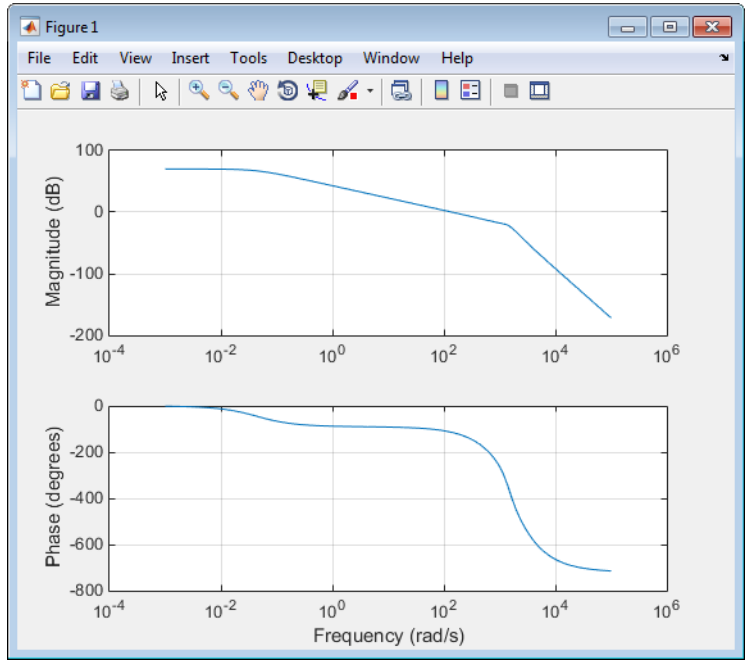
   ```
   [a,b,c,d] = linmod('ssc_hydraulic_actuator_digital_control',X,U);
   ```

   Close the feedback loop by typing:

   ```
   assignin('base','ClosedLoop',1);
   ```

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

```
c = -c; d = -d;
npts = 100; w = logspace(-3,5,npts); G = zeros(1,npts);
for i = 1:npts
    G(i) = c*(1i*w(i)*eye(size(a))-a)^-1*b +d;
end
subplot(211), semilogx(w,20*log10(abs(G)))
grid
ylabel('Magnitude (dB)')
subplot(212), semilogx(w,180/pi*unwrap(angle(G)))
ylabel('Phase (degrees)')
xlabel('Frequency (rad/s)')
grid
```

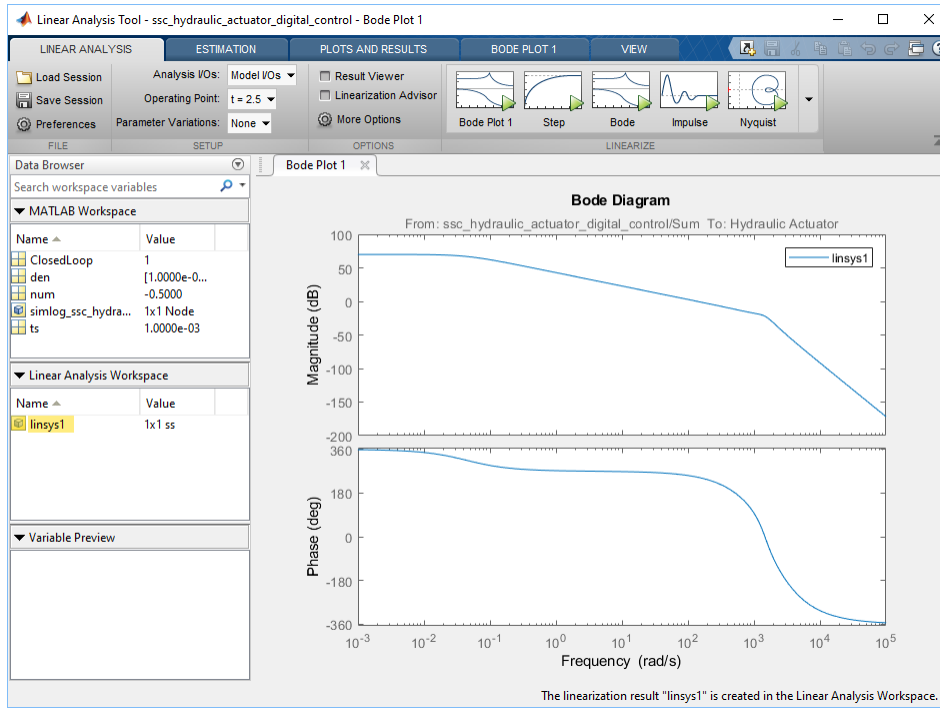


Linearize with Simulink Control Design Software

Note To work through this section, you must have a Simulink Control Design license.

Simulink Control Design software has tools that help you find operating points and returns a state-space model object that defines state names. This is the recommended way to linearize Simscape models.

1. In the 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.

See Also

Related Examples

- “Linearize an Electronic Circuit” on page 8-12

More About

- “Finding Operating Points in Physical Models” on page 8-3
- “Linearizing a Physical Model” on page 8-8

Simscape Run-Time Parameters

- “About Simscape Run-Time Parameters” on page 9-2
- “Show Simscape Run-Time Parameter Settings” on page 9-5
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Specify and Change a Simscape Run-Time Parameter” on page 9-10
- “Troubleshoot Simscape Run-Time Parameter Issues” on page 9-13
- “How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ” on page 9-15
- “Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters” on page 9-18
- “Decrease Computational Cost by Inlining Simscape Run-Time Parameters” on page 9-21

About Simscape Run-Time Parameters

Simscape run-time parameters are MATLAB variables or `Simulink.Parameter` objects that are run-time configurable. By default, run-time configurable parameters are noninlined during code generation. As such, they are coded not as constants, but as variables with values that you can change before or between simulations without recompiling the model.

Simscape run-time parameters help you work more efficiently than compile-time configurable parameters because they allow you to forgo model recompiling when you change parameter values:

- Between fast-restart, iterative simulations on a development computer
- In referenced models on a development computer
- In the generated code in a Rapid Accelerator (Rsim) mode or on real-time target hardware

For more information on using Simscape run-time parameters for these types of simulation, see “Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters” on page 9-18.

By default, all Simscape block parameters are compile-time configurable parameters. You cannot change the value of a compile-time configurable parameter in the model or in generated code without recompiling the model. Therefore, you can change the value of compile-time parameters only in the plant model on your development computer. To specify a Simscape block parameter as run-time configurable, change the run-time configuration setting, which appears next to the dialog box for the parameter, from `Compile-time` to `Run-time`. The figure shows a run-time configuration setting for the **Constant voltage** parameter of a Simscape voltage supply block.

For an example that shows how to specify and change Simscape run-time parameters on development and target computers, see “Specify and Change a Simscape Run-Time

Parameter” on page 9-10 and “Change Parameter Values on Target Hardware” on page 10-170.

Simscape supports run-time configurability for some, but not all, dialog box parameters. To determine if you can specify a particular parameter as a Simscape run-time parameter, examine the block dialog box run-time setting for the parameter. If an enabled run-time setting appears next to the parameter on the block dialog box, you can specify the parameter as run-time configurable. If there is no run-time setting, or if the setting is not enabled, then the parameter is strictly compile-time configurable. You cannot specify a strictly compile-time configurable parameter as a Simscape run-time parameter.

While Simscape run-time parameters can make iterative simulation more efficient, using them can decrease the efficiency of code that you generate. Code that contains compile-time or inlined run-time parameters is more computationally efficient because it does not have to store or retrieve parameter values. If you set the default parameter behavior for code generation to inlined, the generated code algorithm inlines the numeric values of all block parameters as constants.

For information that can help you decide when to inline Simscape run-time parameters, see “Decrease Computational Cost by Inlining Simscape Run-Time Parameters” on page 9-21. To learn how to inline Simscape run-time parameters, see “Manage Simscape Run-Time Parameters” on page 9-7.

Simscape run-time parameters are similar to, but different from Simulink tunable parameters. For example, you can forgo recompiling when you change the value of a tunable parameter while a simulation is either running or stopped. You can change the value of a run-time configurable parameter, without having to recompile the model, only if you do so while the simulation is stopped. For more information on differences between the two types of parameters, see “How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ” on page 9-15.

Limitations

These data types are not supported for run-time parameters:

- Structures
- Cell arrays

See Also

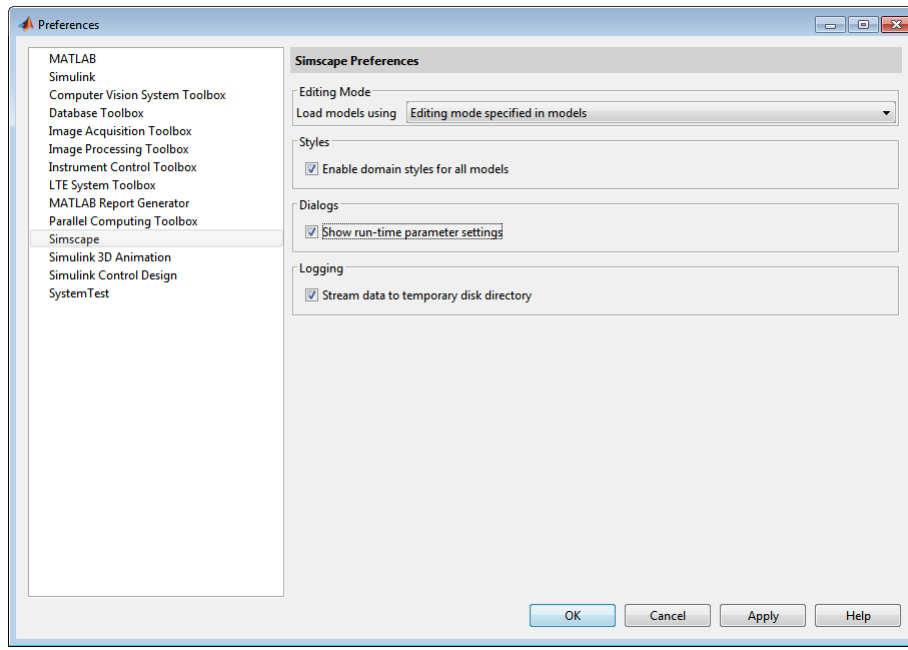
More About

- “Show Simscape Run-Time Parameter Settings” on page 9-5
- “Specify and Change a Simscape Run-Time Parameter” on page 9-10
- “Change Parameter Values on Target Hardware” on page 10-170
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Troubleshoot Simscape Run-Time Parameter Issues” on page 9-13
- “Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters” on page 9-18
- “Decrease Computational Cost by Inlining Simscape Run-Time Parameters” on page 9-21
- “How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ” on page 9-15

Show Simscape Run-Time Parameter Settings

You use block dialog box parameter settings to specify Simscape run-time parameters. However, Simscape block dialog box parameter settings do not show run-time settings unless you set the MATLAB preference to do so. To display run-time parameter settings for run-time configurable Simscape parameters by default:

1. Open the Preferences dialog box. On the MATLAB **Home** tab, in the **Environment** section, click the **Preferences** button.
2. In the left pane of the dialog box, select **Simscape**.
3. Select the **Show run-time parameter settings** check box.



Preferences take effect and remain persistent across MATLAB sessions.

See Also

More About

- “About Simscape Run-Time Parameters” on page 9-2
- “Specify and Change a Simscape Run-Time Parameter” on page 9-10
- “Manage Simscape Run-Time Parameters” on page 9-7

Manage Simscape Run-Time Parameters

By default, all Simscape parameters are compile-time configurable parameters. If you change the value of a compile-time parameter, recompile before simulating the modified model. If you want to change the value without recompiling between iterative simulations or in generated code, you must specify a Simscape dialog box parameter as a run-time configurable parameter.

Show Simscape Run-Time Parameter Settings

You use block dialog box parameter settings to configure Simscape run-time parameters. However, Simscape block dialog box parameter settings do not show run-time parameter settings unless you set the default behavior to do so in MATLAB preferences. For an example that shows you how to set the default behavior to show run-time settings for supported Simscape parameters, see “Show Simscape Run-Time Parameter Settings” on page 9-5.

Specify Simscape Run-Time Parameters

Some Simscape block parameters are strictly compile-time configurable parameters. You cannot specify strictly compile-time configurable parameters as run-time configurable parameters. To determine if a particular Simscape parameter is supported for run-time configurability, examine the block dialog box settings for the parameter. If an enabled run-time setting appears next to the parameter on the block dialog box, you can specify the parameter as run-time configurable. To do specify `Run-time` for the setting. You can change the run-time setting on the dialog box from `Compile-time` to `Run-time` at any time before you generate code from your Simscape model.

The figure shows an enabled run-time setting for the **Constant voltage** parameter of a Simscape block. If there is no run-time setting for a parameter, or if the setting is disabled, then the parameter is strictly compile-time configurable.

Simscape run-time parameters are changeable by variable, not by field. That is, to simulate without recompiling after a parameter change, you cannot simply update the value of the parameter in the dialog box setting where you specify the numeric value. You change the value of the variable that represents the parameter in the workspace or in the generated code. To do so, you first:

1. Specify the parameter as a variable in the dialog box setting.
2. Save the numeric value for the variable to the MATLAB workspace.

For an example that shows how to specify a Simscape run-time parameter using a variable, see “Specify and Change a Simscape Run-Time Parameter” on page 9-10.

Set the Default Parameter Behavior for Generated Code

By default, you can change the value of a Simscape run-time parameter while the simulation is stopped without having to recompile the model. If you change the default parameter behavior for code generation to inlined, the generated code algorithm inlines the numeric values of all the block parameters as constants. The code that you generate using inlined parameters is more computationally efficient because it does not have to store or retrieve parameter values.

To set the default behavior for Simscape run-time parameters, on the **Model Configuration Parameters > Optimization > Signal and Parameters** dialog box, select the **Default parameter behavior**. Options are:

- `Inline` — The Simulink Coder algorithm hard-codes the numeric values of all the block parameters as constants in the generated C code, rendering them non-modifiable.
- `Tunable` — Simulink Coder generates data structures that you can modify to change parameters without recompiling between simulation runs.

Selectively Override the Inline Default Behavior

While computational cost increases if you specify Simscape parameters in your model as run-time configurable, the cost increase is not proportional to the number of parameters that you so specify. If you declare a single parameter as run-time configurable, the computational cost increases sharply. The cost increases less per parameter for any additional Simscape run-time parameters that you specify. Therefore, even if the computational cost for your model decreases greatly with inlining, it decreases only slightly for each Simscape run-time parameter that you selectively inline. However if your

model is at risk of overrunning and it contains Simscape run-time parameters, you might decrease the computational cost enough to prevent overruns by selectively excluding only a few Simscape run-time parameters from inlining.

To inline with exceptions, set the **Default parameter behavior** to `Inline` and use the Model Parameter Configuration dialog box to remove individual parameters from inlining.

See Also

More About

- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)
- “Model Callbacks” (Simulink)
- “About Simscape Run-Time Parameters” on page 9-2
- “Show Simscape Run-Time Parameter Settings” on page 9-5
- “Specify and Change a Simscape Run-Time Parameter” on page 9-10
- “Change Parameter Values on Target Hardware” on page 10-170
- “Decrease Computational Cost by Inlining Simscape Run-Time Parameters” on page 9-21
- “How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ” on page 9-15

Specify and Change a Simscape Run-Time Parameter

Prerequisites

Show Simscape Run-Time Parameter Settings

You use block dialog box parameter settings to specify Simscape run-time parameters. However, Simscape block dialog box parameter settings do not show run-time settings unless you set the default behavior to do so using MATLAB preferences. For an example that shows you how to set the default behavior to show run-time parameter settings for run-time configurable Simscape parameters, see “Show Simscape Run-Time Parameter Settings” on page 9-5.

Set the Default Parameter Behavior for Code Generation

Simscape run-time parameters depend on the default parameter behavior setting for code generation. To enable run-time configurability for a Simscape run-time parameter so that you can change its value without recompiling your model, you can:

- Set the default to `tunable`
- Set the default `inline`, and override the default behavior for the parameter with the value that you want to change.

For information on setting and overriding the default behavior for Simscape run-time parameters, see “Manage Simscape Run-Time Parameters” on page 9-7.

Specify a Parameter as Run-Time Configurable

The PM DC Motor example model contains a DC Voltage block. You parameterize the block by specifying the *Constant voltage* for the source. The voltage is supported as a Simscape run-time parameter. To specify *Constant voltage* as a run-time configurable parameter:

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

   ```
   ssc_dcmotor
   ```

2. From the model window, access the parameter settings for the DC Voltage block.
3. To specify *Constant voltage* as a Simscape run-time parameter, for the run-time setting, select `Run-time`.

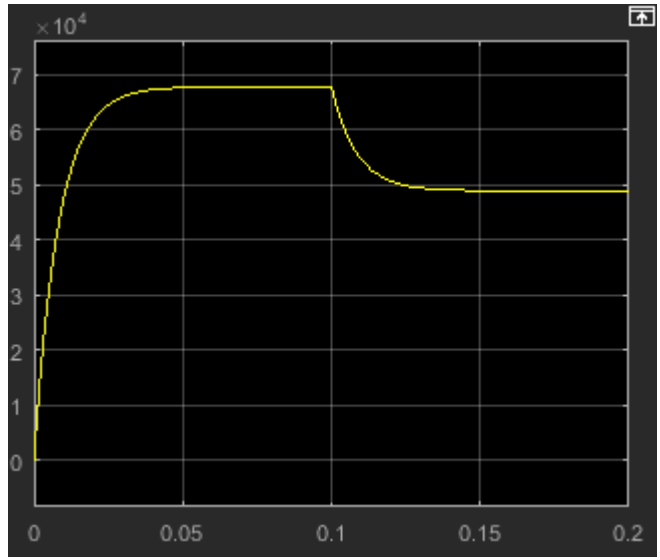
4. Specify the **Constant voltage** parameter value as the variable `vDC`.
5. Assign a numeric the value to the variable in the MATLAB workspace:

```
vDC = 5;
```

Change a Simscape Run-Time Parameter Using Fast Restart

To see how altering the value of a run-time configurable parameter can affect simulation results, change the value of the **Constant voltage** parameter between iterative fast-start simulations.

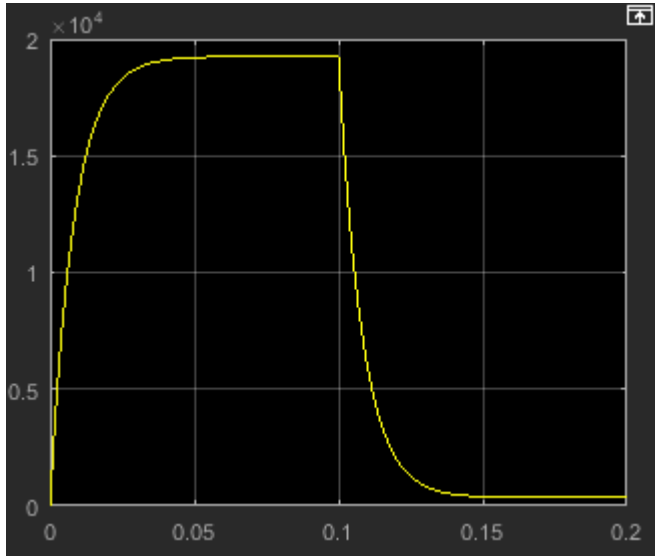
1. To enable fast restart, click the Fast restart icon on the Simulink Editor toolbar.
2. To simulate the model, click the Run button on the Simulink Editor toolbar.
3. Open the Motor RPM scope block. Autoscale to see the results better.



4. Assign a different value to the variable that represents voltage:

```
vDC = 1.5;
```

5. Simulate the model.
6. Open the Motor RPM scope block.



The results reflect the change in value for the run-time parameter.

See Also

More About

- “About Simscape Run-Time Parameters” on page 9-2
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Change Parameter Values on Target Hardware” on page 10-170
- “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File” (Simulink Coder)
- “How Acceleration Modes Work” (Simulink)
- “Get Started with Fast Restart” (Simulink)
- “How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ” on page 9-15

Troubleshoot Simscape Run-Time Parameter Issues

Simscape Run-Time Parameter Setting Not Visible

If a Simscape parameter is potentially run-time configurable, a configuration setting appears on the tab that contains the settings for the parameter. If the setting is enabled, you can use it to specify the parameter as either a run-time or compile-time configurable parameter. If the configuration setting is not visible, then either the parameter is strictly compile-time configurable or the Simscape option for showing run-time parameter settings is not enabled. To see how to enable the option, see “Show Simscape Run-Time Parameter Settings” on page 9-5.

Simulation Does Not Respond to Simscape Run-Time Parameter Change

If you use an expression to define a variable for a Simscape run-time parameter, you might not be able to change the parameter without model recompilation. When an unsupported expression is encountered during code generation, a warning is issued and the equivalent numeric value is generated in the code. For information on the limitations for using expressions for run-time configurable parameters, see “Tunable Expression Limitations” (Simulink Coder).

If you change the value of a Simscape run-time parameter during a simulation, the results do not reflect the value change. For a run-time configurable parameter change to take effect, you make the change when the simulation is stopped. To see how to change the value of a Simscape run-time parameter, see “Specify and Change a Simscape Run-Time Parameter” on page 9-10 and “Change Parameter Values on Target Hardware” on page 10-170.

Unsupported Data Types

These data types are not supported for run-time parameters:

- Structures
- Cell arrays

See Also

More About

- “About Simscape Run-Time Parameters” on page 9-2
- “Show Simscape Run-Time Parameter Settings” on page 9-5
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Specify and Change a Simscape Run-Time Parameter” on page 9-10
- “Change Parameter Values on Target Hardware” on page 10-170
- “How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ” on page 9-15

How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ

Simscape run-time parameters and Simulink tunable parameters both allow you to change the values of parameters on your development or target computer without model recompilation. However, they differ in these important ways:

- You can change the value of a Simulink tunable parameter while a simulation is running. Simscape run-time parameters are run-time configurable. You cannot change the value of a run-time configurable parameter during simulation. You can only change the value of a run-time configurable parameter when a simulation is stopped.
- Simulink tunable parameters are tunable by default. Simscape block parameters are compile-time configurable by default. To make a Simscape block parameter run-time configurable, you specify it as such.
- Simulink tunable parameters are tunable-by-field. You can change a tunable parameter simply by changing the numeric value using the block dialog box parameter setting. Simscape run-time parameters are tunable-by-variable. To change the value of a Simscape run-time parameter, you:

  1. Specify a variable for the parameter in the block dialog box setting.
  2. Use input tools, such as the command prompt, callbacks, scripts, or MAT-files, to assign a value for the variable in the MATLAB workspace.
  3. Use input tools to change the value for the variable.
- For code generation, you specify the **Default parameter behavior** as `Tunable` or `Inlined`. You cannot modify inlined parameters in generated code because the compiler specifies them as constants. You can change the values of tunable parameters in generated code because the compiler specifies them as modifiable global variables or structure fields.

  If you set the default behavior to `Tunable`, the compiler specifies all Simscape run-time parameters and Simulink tunable parameters as modifiable entities in the generated code. However, if you set the default behavior to `Inlined`, the compiler inlines only the Simscape run-time parameters. The Simulink tunable parameters are still generated as modifiable entities in the code. To change the value of a particular Simscape run-time parameter in the generated code when the default behavior is inlined, you declare that parameter as an exception to inlining.

The table shows the state, mode, and code section in which you can change a run-time parameter or run-time configurable parameter.

Machine	Simulink Simulation Mode	Simulation Status	Section of generated code that you modify	Simscape run-time parameter is modifiable	Simulink tunable parameter is modifiable
Development	Normal	SimulationStatus is stopped	Not Applicable (NA)	Yes	Yes
Development	Normal	SimulationStatus is running	NA	No	Yes
Development or Target	Normal, Accelerator, Rapid Accelerator, SIL, PIL, or External	SimulationStatus is stopped	NA	Yes	Yes
Development or Target	Normal, Accelerator, Rapid Accelerator, SIL, PIL, or External	SimulationStatus is running	NA	No	Yes
Target	Normal, SIL, PIL, or External	SimulationStatus is stopped	Setup-function	Yes	Yes
Target	Normal, SIL, PIL, or External	SimulationStatus is running	• Step-function for a Simulink global variable • External code for a Simulink parameter object	No	Yes

Between normal mode simulations, as long as your changes do not affect the structure of the model, you can forgo recompiling by using fast restart when you change Simscape run-time and Simulink tunable parameters.

See Also

More About

- “Change Parameter Values on Target Hardware” on page 10-170
- “Tune and Experiment with Block Parameter Values” (Simulink)
- “Specify and Change a Simscape Run-Time Parameter” on page 9-10
- “Code Regeneration in Accelerated Models” (Simulink)
- “About Simscape Run-Time Parameters” on page 9-2
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Decrease Computational Cost by Inlining Simscape Run-Time Parameters” on page 9-21

Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters

Simscape run-time parameters are run-time configurable. They allow you to forgo recompiling if you change parameter values between iterative simulations during parameter surveys. You can only test a single value for a compile-time configurable parameter without recompiling your model.

You can benefit from this advantage when you perform parameter sweeps using fast restart, model referencing, or code generation. Code generation allows you to update Simscape run-time parameters between simulation runs using:

- Rapid Simulation (Rsim) on development or target hardware
- Real-time simulation on target hardware

Model Referencing with Run-Time Configurable Parameters

You can use Simulink Model blocks to represent one model within another. Each instance of a Model block represents a reference to another model, called a referenced model. For simulation and code generation, the referenced model effectively replaces the Model block that references it. To change the behavior of a referenced model without recompiling, specify a Simscape run-time parameter value in the referenced model using either global parameters or model arguments.

For information on using and parameterizing referenced models, see “Model Reference Basics” (Simulink) and “Parameterize Instances of a Reusable Referenced Model” (Simulink).

Code Generation with Run-Time Configurable Parameters

Simscape run-time parameters are run-time configurable. They allow you to test your design over a range of values for plant parameters without recompiling or redeploying code. You can:

- Change the value of a Simscape run-time parameter in both your plant model and in your generated code on your development computer for a rapid or real-time simulation.
- Update a run-time configurable parameter in your deployed code before you run your simulation as an executable on an external target machine.

For an example that uses Simscape run-time parameters for real-time simulation see “Change Parameter Values on Target Hardware” on page 10-170.

Note The Rapid Accelerator (Rsim) mode uses portions of the Simulink Coder product to create an executable. These modes replace the interpreted code normally used in Simulink simulations, shortening model run time. Although Rsim uses Simulink Coder code generation technology, you do not need Simulink Coder software on your development computer to accelerate your model with Rsim.

Fast Restart with Simscape Run-Time Parameters

Changing parameter values does not require model recompiling between simulation runs unless the changes alter the model structurally. However, when you use normal mode simulation without fast restart, each simulation compiles the model. The compiling occurs even if the new values do not change the structure of the model and each recompile increases the overall simulation time.

With Simulink fast restart, you can modify Simscape run-time parameters from the workspace variable, without recompiling. For an example that shows how to specify a Simscape run-time parameter and change the parameter value using fast restart, see “Specify and Change a Simscape Run-Time Parameter” on page 9-10.

See Also

More About

- “About Simscape Run-Time Parameters” on page 9-2
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Change Parameter Values on Target Hardware” on page 10-170
- “Get Started with Fast Restart” (Simulink)
- “Decrease Computational Cost by Inlining Simscape Run-Time Parameters” on page 9-21
- “Model Reference Basics” (Simulink)
- “How Fast Restart Improves Iterative Simulations” (Simulink)
- “Choosing a Simulation Mode” (Simulink)

- “Code Regeneration in Accelerated Models” (Simulink)
- “Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File” (Simulink Coder)

Decrease Computational Cost by Inlining Simscape Run-Time Parameters

Simscape run-time parameters tend to increase the complexity of your code and, therefore, the computational cost of simulating your model. While the additional cost is not typically problematic for desktop simulation, it could result in a CPU overload during real-time simulation.

Consider decreasing the complexity of your model by inlining Simscape run-time parameters in your model, if your simulation:

- Generates small steps during model preparation. For information, see “Real-Time Model Preparation Workflow” on page 10-6.
- Has a long execution time when you are configuring your model for real-time simulation. For information, see “Real-Time Simulation Workflow” on page 10-98.
- Overruns during real-time simulation on target hardware.

For information on inlining Simscape run-time parameters, see “Manage Simscape Run-Time Parameters” on page 9-7.

See Also

More About

- “Estimate Computation Costs” on page 10-118
- “Reduce Computation Costs” on page 10-34
- “Fixed-Cost Simulation for Real-Time Viability” on page 10-96
- “Improving Speed and Accuracy” on page 10-12
- “About Simscape Run-Time Parameters” on page 9-2
- “Manage Simscape Run-Time Parameters” on page 9-7
- “Change Parameter Values on Target Hardware” on page 10-170

Real-Time Simulation

- “Model Preparation Objectives” on page 10-3
- “Real-Time Model Preparation Workflow” on page 10-6
- “Improving Speed and Accuracy” on page 10-12
- “Determine Step Size” on page 10-17
- “Increase Simulation Speed Using the Partitioning Solver” on page 10-27
- “Reduce Computation Costs” on page 10-34
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Partition a Model” on page 10-86
- “Manage Model Variants” on page 10-95
- “Fixed-Cost Simulation for Real-Time Viability” on page 10-96
- “Real-Time Simulation Workflow” on page 10-98
- “Solvers for Real-Time Simulation” on page 10-104
- “Troubleshooting Real-Time Simulation Issues” on page 10-109
- “Determine System Stiffness” on page 10-111
- “Estimate Computation Costs” on page 10-118
- “Choose Step Size and Number of Iterations” on page 10-121
- “What Is Hardware-In-The-Loop Simulation?” on page 10-138
- “Hardware-In-The-Loop Simulation Workflow” on page 10-142
- “Code Generation Requirements” on page 10-149
- “Software and Hardware Configuration” on page 10-151
- “Signal and Parameter Visualization and Control” on page 10-153
- “Troubleshoot Hardware-in-the-Loop Simulation Issues” on page 10-155
- “Generate, Download, and Execute Code” on page 10-157
- “Check for Target Hardware Overruns” on page 10-160

- “Change Parameter Values on Target Hardware” on page 10-170
- “Requirements for Using Alternative Platforms” on page 10-179
- “Extending Embedded and Generic Real-Time System Target Files” on page 10-181
- “Precompiled Static Libraries” on page 10-183
- “Initialization Cost” on page 10-184
- “Generate HDL Code Using the Simscape HDL Workflow Advisor” on page 10-185

Model Preparation Objectives

In this section...
“Obtain Reference Results” on page 10-3
“Determine Step Size” on page 10-3
“Adjust Model Fidelity or Scope” on page 10-4

The main goal of model preparation is to ensure that your model is real-time capable. Your model is real-time capable if it is both:

- Accurate enough to generate simulation results that match your expectations, as based on theoretical models and empirical data
- Fast enough to run on your real-time target machine without overruns

During model preparation, you obtain reference results and determine step size to assess the likelihood that your model is real-time capable. If it is unlikely that your model is real-time capable, you adjust the model scope or fidelity to make real-time simulation with your model feasible.

Obtain Reference Results

Moving your model from desktop simulation to real-time simulation is an iterative process that can require extensive model reconfiguration. During model preparation, you obtain reference results from a variable-step simulation of your original model. These results provide a baseline against which you can judge the accuracy of your modified models.

Determine Step Size

In terms of speed, the only way to know if your model is real-time capable is to test for overruns while simulating on real-time hardware. You can, however, analyze solver execution speed using desktop simulation to determine if your model is probably fast enough for real-time simulation. You do so by analyzing the steps of a variable-step solver to find the maximum step size to use for sufficiently accurate real-time simulation results. If the required step size appears small enough to cause an overrun on your real-time hardware, you increase the step size by improving simulation speed.

Adjust Model Fidelity or Scope

You can adjust the fidelity or scope of your model to increase speed or accuracy. Adjustments include:

- Deleting or adding blocks or modifying block parameters to eliminate or reduce the effects of elements that introduce numerical stiffness or cause discontinuities. Simulations take small steps to calculate accurate solutions for these types of elements.
- Modifying elements or parameters to increase simulation efficiency. For example, simplify graphics that require excessive processing power or including lookup tables instead of utilizing processing power to calculate known values.
- Partitioning independent networks of the model to enable parallel processing.

You can also adjust solver settings to help to make your model real-time capable. For real-time simulation on target hardware, you use a fixed-step, fixed-cost solver that bounds the computation cost, that is, the time the solver takes to execute each time step. You configure the solver parameters before deploying it to a real-time target machine. The fixed-step solver settings that you adjust to improve the real-time viability of your model include step size, solver type, and number of iterations.

Due to the number of options, it is challenging to find the right combination of model size, model fidelity, and solver parameters to achieve real-time simulation. The relationship between speed and accuracy also makes it hard to find both system and solver configurations that help to make your model real-time capable. If you increase speed, you are likely to decrease accuracy. Conversely, increasing accuracy tends to decrease speed. It is especially difficult to achieve acceptable speed and accuracy if you try to adjust model fidelity and scope while you are changing fixed-step solver settings. A better approach to find the optimal configuration is to change only one or two related settings, analyze how those changes affect simulation speed and accuracy, and then make other adjustments.

The real-time model preparation and the real-time simulation workflows separate the configuration changes into two different step-wise processes. For the real-time model preparation workflow, you adjust only the size or fidelity of your model and use variable-step simulation to analyze the effects of your changes. For the real-time simulation workflow, you adjust only the solver parameters and you use fixed-step, fixed-cost simulation to analyze how the changes affect the speed and accuracy of your model.

See Also

Related Examples

- “Determine Step Size” on page 10-17
- “Estimate Computation Costs” on page 10-118
- “Reduce Computation Costs” on page 10-34
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Partition a Model” on page 10-86
- “Manage Model Variants” on page 10-95

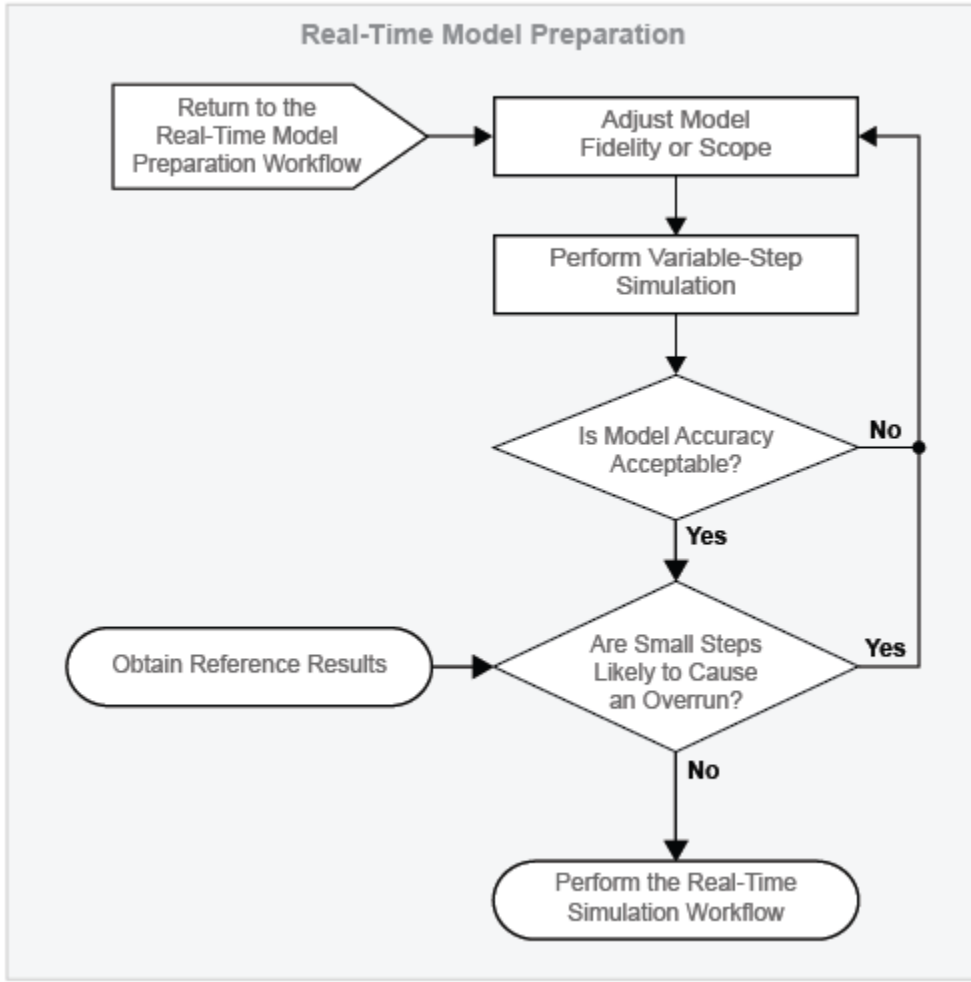
More About

- “Improving Speed and Accuracy” on page 10-12
- “Fixed-Cost Simulation for Real-Time Viability” on page 10-96
- “Real-Time Model Preparation Workflow” on page 10-6
- “Real-Time Simulation Workflow” on page 10-98

Real-Time Model Preparation Workflow

In this section...
“Prepare Your Model for Real-Time Simulation” on page 10-8 “Insufficient Computational Capability for Workflow Completion” on page 10-10

The figure shows the real-time model preparation workflow.



A real-time-capable model is both fast and accurate. When you simulate a real-time-capable model on your real-time target machine, it runs to completion and generates results that match your expectations, as based on theoretical models and empirical data. The only way to determine if your model is real-time capable is to run it on your real-time target machine. You can, however, use desktop simulation, that is, simulation on your development computer, to determine the likelihood that your model is real-time capable before you deploy it.

The real-time model preparation workflow is the first of two workflows that you perform on your development computer to make it more probable that your model is real-time capable. The workflow shows you how to adjust the size or fidelity of your model to improve speed without sacrificing and accuracy. When you finish this workflow, use the real-time simulation workflow to find the best fixed-step, fixed-cost solver configuration to use for simulating your model in real time.

Prepare Your Model for Real-Time Simulation

Obtain Reference Results

Use empirical or theoretical data to design and build your Simscape model. Use a Simulink global variable-step solver to simulate your model. Refine your model as needed to obtain simulation results that the underlying data supports. Reference results provide a baseline to assess model accuracy against throughout all stages of the model preparation and real-time simulation workflows.

Evaluate Overrun Risk

An overrun occurs when the step size is too small to allow the real-time computer to complete all the processing required for any one step. If your model requires a step size that is so small that it is likely to cause an overrun, then your model is not fast enough for real-time simulation. To determine if small steps are likely to cause an overrun, create a plot of the size of the steps that the variable-step step solver uses to execute the simulation of your model. The step size plot tells you the number and size of the small steps that the solver uses during the simulation.

There are no hard metrics for the size or number of small steps that are likely to cause a real-time-simulation overrun. Moving your model from desktop simulation to real-time simulation is an iterative process. The process, which involves modifying, simulating, and analyzing your model, helps you to determine if the small steps in your model are time limiting or numerous enough to force an overrun.

Experience that you gain by simulating different models on your real-time machine can also help you decide if the small steps in your model are likely to force an overrun. For example, consider a case with two models, M1 and M2, and two different real-time processors, RT1 and RT2. Processors RT1 and RT2 have the same nominal processing speed. Models M1, a mechanical model, and M2, an electrical model, both have a few steps that are 1e-15 seconds. It is possible for model M1 to simulate with sufficiently accurate results on real-time processor RT1, but to incur an overrun or simulate with insufficiently accurate results on real-time processor RT2. It is also possible that model

M1 runs to completion with accurate results on RT1 and RT2, whereas model M2 generates an overrun on both processors. These scenarios are possible because the distinct model topologies yield different dynamics and because nominal processing speed is not the only determinant for simulation execution time. Other factors such as the operating system and I/O configuration also affect how simulation execution proceeds on a real-time processor. Familiarity with system dynamics and the processing power of your real-time equipment can guide your decision making when you assess the impact of small step sizes on the real-time viability of a model.

Adjust Model Fidelity or Scope

Modify the model to increase speed or accuracy if your analysis indicates that real-time simulation with the model is likely to have an overrun or yield insufficiently accurate results.

When you evaluate overrun risk, if you find that the simulation uses too many small steps, use these approaches to improve simulation speed:

- Reduce computation costs.
- Reduce numerical stiffness.
- Reduce zero crossings.
- Reduce fast dynamics
- Partition the model for parallel processing.

When you evaluate model accuracy, if you find that the simulation results do not match the reference results, use these approaches to improve model accuracy:

- Simulink best practices for modeling dynamic systems
- Simscape essential modeling techniques

Obtain Results with a Variable-Step Solver

Using a Simulink global variable-step solver, obtain results for the modified version of your model.

The step size plot also helps you to:

- Estimate the maximum step size to use for the fixed-step solver to achieve accurate results during real-time simulation.
- Identify the exact times when discontinuities or fast dynamics slow down the simulation.

Evaluate Model Accuracy

Compare the results from simulating on the target computer to your reference results. Are the reference and modified model results the same? If not, are they similar enough that the empirical or theoretical data also supports the results from the simulation of the modified model? Is the modified model representing the phenomena that you want it to measure? Is it representing those phenomena correctly? If you plan on using your model to test your controller design, is the model accurate enough to produce results that you can rely on for system qualification? The answers to these questions help you to decide if your real-time results are accurate enough.

Perform Real-Time Simulation Workflow

When variable-simulation results indicate that your model has the speed and accuracy required for real-time processing, you can use the “Real-Time Simulation Workflow” on page 10-98 to configure your model for fixed-step, fixed-cost simulation.

Return to the Real-Time Model Preparation Workflow

The connector is an entry point for returning to the real-time model preparation workflow from another workflow (for example, the real-time simulation workflow or the hardware-in-the-loop simulation workflow).

Insufficient Computational Capability for Workflow Completion

It is possible that your real-time target machine lacks the computational capability for running your model in real time. If, after multiple iterations of the workflow, there is no level of model complexity that makes your model real-time capable, consider these options for increasing processing power:

- “Upgrading Target Hardware” on page 10-15
- “Simulating Parts of the System in Parallel” on page 10-15

See Also

Related Examples

- “Determine Step Size” on page 10-17

- “Estimate Computation Costs” on page 10-118
- “Reduce Computation Costs” on page 10-34
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Partition a Model” on page 10-86
- “Manage Model Variants” on page 10-95

More About

- “Essential Physical Modeling Techniques” on page 1-15
- “Hardware-In-The-Loop Simulation Workflow” on page 10-142
- “Improving Speed and Accuracy” on page 10-12
- “Model Preparation Objectives” on page 10-3
- “Modeling Dynamic Systems” (Simulink)
- “Real-Time Simulation Workflow” on page 10-98

Improving Speed and Accuracy

In this section...
“Why Speed and Accuracy Matter for Real-Time Simulation” on page 10-12
“Balancing Speed and Accuracy” on page 10-13
“Eliminating Effects That Require Intensive Computation” on page 10-14
“Optimizing Local and Global Solver Configurations” on page 10-15
“Upgrading Target Hardware” on page 10-15
“Simulating Parts of the System in Parallel” on page 10-15

Why Speed and Accuracy Matter for Real-Time Simulation

Speed and accuracy are the determining factors for making your model real-time capable. Your model is real-time capable if it satisfies both of these conditions when you simulate it on your particular target hardware:

- There are no overruns.
- The simulation results meet your criteria for accuracy.

Speed is objective. The real-time clock determines whether your model is fast enough for real-time simulation. For each step that your solver takes, your real-time hardware system tracks the time that it takes to complete these processing tasks:

- Execute the simulation.
- Process input and output.
- Perform general computer tasks.

An overrun occurs when, for any time step, the time that it takes your system to complete the processing tasks exceeds the real-time limit for the tasks. If your target machine reports any overruns when you use it to simulate your model, your model is not fast enough for real-time simulation.

Your Simscape model is accurate if it produces results that agree with the empirical and theoretical data that are the basis for your model. Accuracy is more subjective when the foundation and simulation data are similar, but are not in absolute agreement. To determine if your model is accurate enough for real-time simulation when the data do not match perfectly, consider these questions:

- Is the model representing the phenomena that you want it to measure?
- Is it representing those phenomena correctly?
- If you plan to use your model to test your controller design, is the model accurate enough to produce results that you can rely on for system qualification?

The only way to test whether your model is real-time capable is to run it on your actual real-time target hardware using fixed-step, fixed-cost solvers. You can, however, estimate whether the model is both fast and accurate enough for real-time simulation by analyzing the results from desktop simulation. To estimate whether your model is real-time capable, see “Determine Step Size” on page 10-17 and “Estimate Computation Costs” on page 10-118.

If the analysis from the desktop simulation indicates that your model likely is not real-time capable, increase model speed or accuracy before deploying your model to your real-time target machine. Increasing the speed of your simulation tends to decrease the accuracy, and conversely increasing accuracy decreases speed. To make your model real-time capable, maintain a balance between speed and accuracy.

Balancing Speed and Accuracy

Simulation speed and accuracy correlate to your choices for:

- Model fidelity and scope
- Real-time hardware computing power
- Solver sample time (step size) and number of iterations

To try to increase simulation speed, potentially at the expense of accuracy:

- Decrease model fidelity or scope.
- Increase sample time.
- Decrease the number of solver iterations.

To try to increase simulation accuracy, potentially at the expense of speed:

- Increase model fidelity or scope.
- Decrease sample time.
- Increase the number of solver iterations.

To try to increase both accuracy and speed, or either one without sacrificing the other, increase computing power. To increase computing power, use a faster real-time processor or compute in parallel.

The type of solver that you specify also affects simulation speed and accuracy. For fixed-step simulation, Simscape local solvers are faster and as accurate as Simulink global solvers. Implicit solvers are faster, but less accurate than explicit solvers. However, the numerical stiffness of the network is also a determinant for deciding whether to use an implicit solver or an explicit solver. Explicit solvers yield more accurate results for numerically stiff networks.

For more information on how model complexity affects speed and accuracy, see “Eliminating Effects That Require Intensive Computation” on page 10-14. For more information, on how solver configurations affect speed and accuracy, see “Optimizing Local and Global Solver Configurations” on page 10-15.

It is possible that there is no combination of model complexity and solver settings that can make your model real-time capable. If the simulation does not run in real time on the target machine, or if the accuracy is unacceptable, consider these options for increasing speed and accuracy:

- “Upgrading Target Hardware” on page 10-15
- “Simulating Parts of the System in Parallel” on page 10-15

Eliminating Effects That Require Intensive Computation

If your desktop simulation analysis indicates that your model likely is not fast enough for real-time simulation, eliminate effects that require intensive computation. Identify elements in your model that cause costly effects, such as discontinuities and rapid changes, that tend to slow down simulations.

Elements that cause discontinuities include:

- Hard stops or backlash
- Stick-slip friction
- Switches or clutches

Elements with small time constants that cause rapid changes include:

- Small masses attached to stiff springs with minimal damping

- Electrical circuits with low capacitance, inductance, and resistance
- Hydraulic circuits with small compressible volumes

To eliminate or modify the elements that are responsible for the effects that slow down your simulation, use these approaches:

- Replace nonlinear components with linearized versions.
- Replace complex equations with lookup tables for their solution.
- Replace complicated components with simplified models.
- Smooth discontinuous functions (step changes) with filters, delays, and other techniques.

Optimizing Local and Global Solver Configurations

You can also influence the speed and accuracy of your simulation by way of your solver specifications. The level of accuracy that your real-time target machine delivers does not necessarily correlate to a specific step size across all networks in a single model. A real-time target machine can give accurate results for a simple network in your model but inaccurate results for a more complex network. Take advantage of the ability to specify different solver configurations for each network in your Simscape model. To help make your model real-time capable, configure your fixed-step global solver and each local solver individually.

For information on solver options and determining the solvers that help to make your Simscape model real-time capable, see “Solvers for Real-Time Simulation” on page 10-104.

Upgrading Target Hardware

Different targets give varying levels of accuracy when using the same step size to simulate the same model. You can speed up or increase the accuracy of the real-time simulation by using a faster real-time target computer.

Simulating Parts of the System in Parallel

Another approach for increasing speed while maintaining accuracy is to configure your model to evaluate multiple physical networks in parallel. You can partition your model if the networks are not dependent upon one another. Work with and experiment with your model, the generated code, and the real-time target machine to use this approach.

See Also

Related Examples

- “Determine Step Size” on page 10-17
- “Estimate Computation Costs” on page 10-118
- “Reduce Computation Costs” on page 10-34
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Partition a Model” on page 10-86

More About

- “Model Preparation Objectives” on page 10-3
- “Real-Time Model Preparation Workflow” on page 10-6
- “Solvers for Real-Time Simulation” on page 10-104

Determine Step Size

For the first step in “Real-Time Model Preparation Workflow” on page 10-6, you obtain results from a variable-step simulation of the reference version of your Simscape model. The reference results provide a baseline against which you can assess the accuracy of your model as you modify it. This example shows how to analyze the reference results and the step size that the variable-step solver takes to:

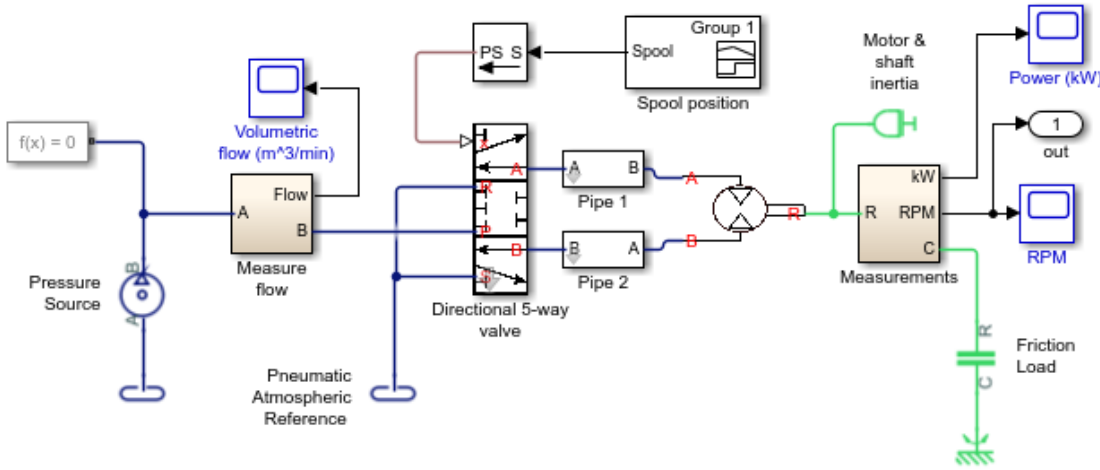
- Estimate the maximum step size that you can use for a fixed-step simulation
- Identify events that have the potential to limit the maximum step size

Discontinuities and rapid changes require small step sizes for accurately capturing these dynamics. The maximum step size that you can use for a fixed-step simulation must be small enough to ensure accurate results. If your model contains such dynamics, then it is possible that the required step size for accurate results, Ts_{max}, is too small. A step size that is too small does not allow your real-time computer to finish calculating the solution for any given step in the simulation.

The analysis in this example helps you to estimate the maximum step size that fixed-step solvers can use and still obtain accurate results. You can also use the analysis to determine which elements influence the maximum step size for accurate results. For more information on how obtaining reference results and performing a step-size analysis helps you to prepare your model for real-time simulation, see “Model Preparation Objectives” on page 10-3.

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

```
model = 'ssc_pneumatic_rts_reference';
open_system(model)
```

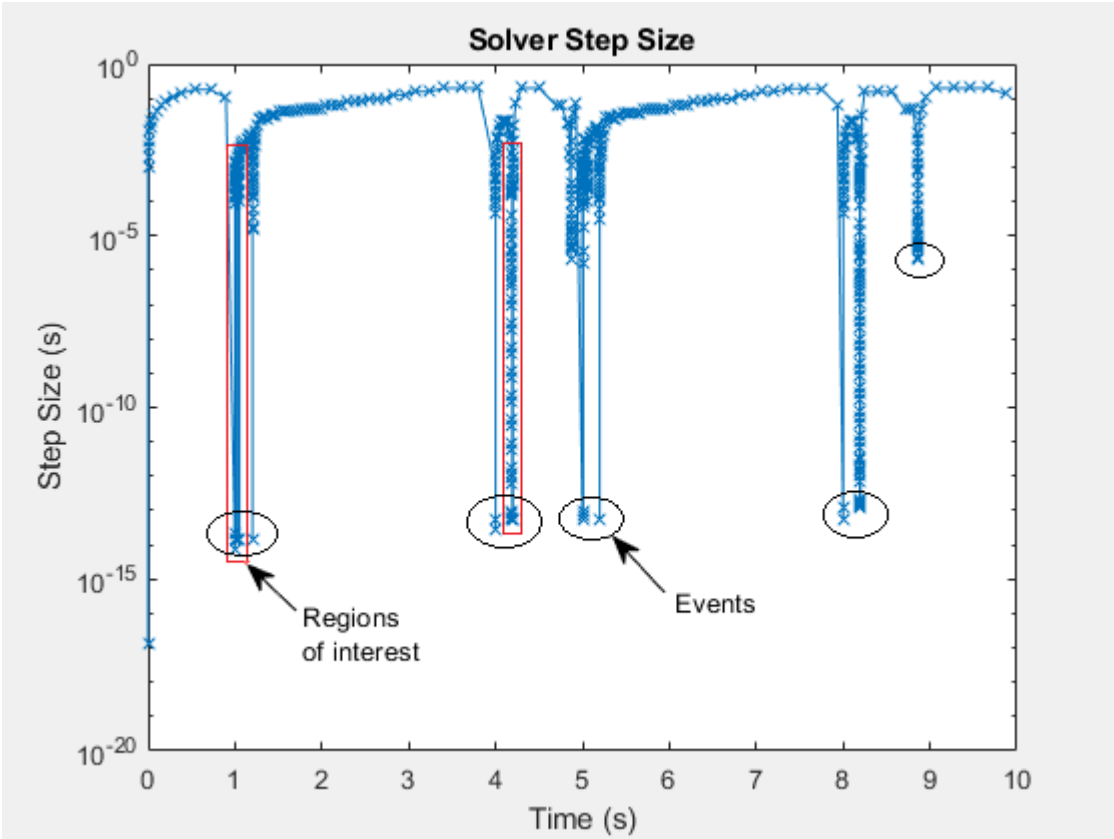


2 Simulate the model:

```
sim(model)
```

3 Create a semilogarithmic plot that shows how the step size for the solver varies during the simulation.

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



For much of the simulation, the step size is greater than the value of the Ts_{max} in the plot. The corresponding value, ~0.001 seconds, is an estimated maximum step size for achieving accurate results during fixed-step simulation with the model. To see how to configure the step size for fixed-step solvers for real-time simulation, see “Choose Step Size and Number of Iterations” on page 10-121.

The x markers in the plot indicate the time that the solver took to execute a single step at that moment in the simulation. The step-size data is discrete. The line that connects the discrete points exists only to help you see the order of the individual execution times over the course of the simulation.

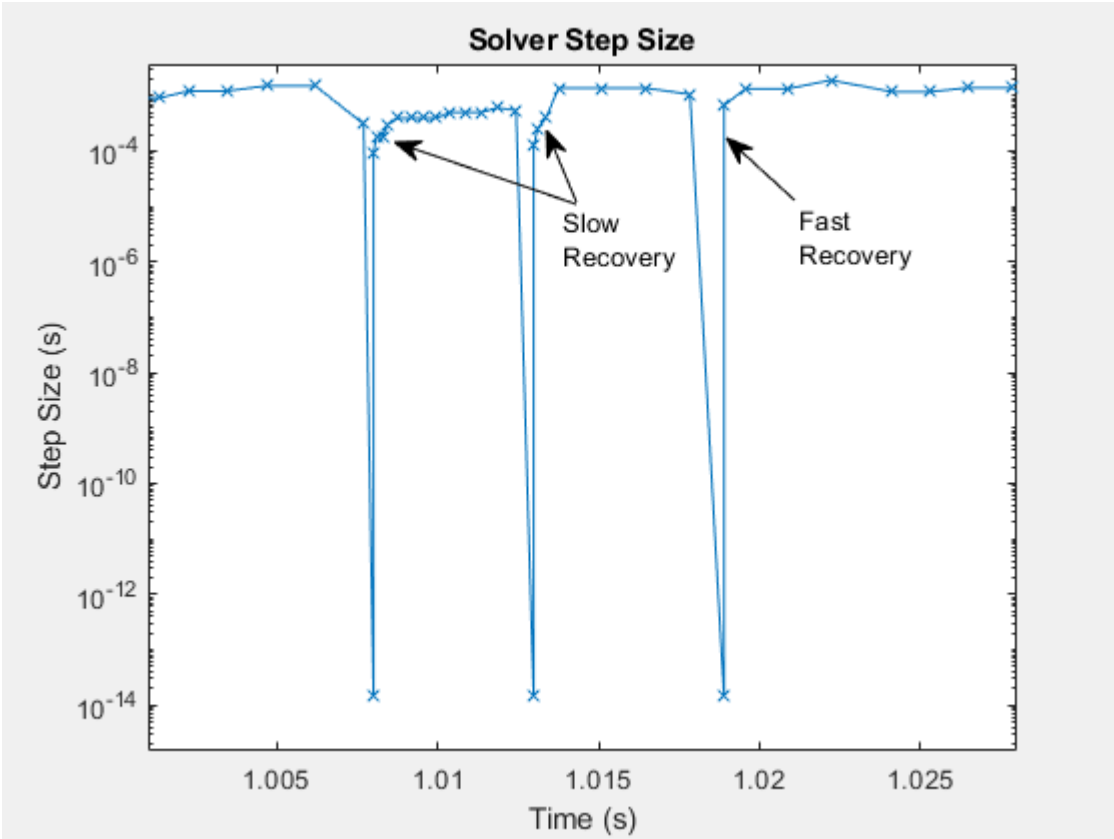
A large decrease in step size indicates that the solver detects a zero-crossing event. Zero-crossing detection can happen when the value of a signal changes sign or

crosses a threshold. The simulation reduces the step size to capture the dynamics for the zero-crossing event accurately. After the solver processes the dynamics for a zero-crossing event, the simulation step size can increase. It is possible for the solver to take several small steps before returning to the step size that precedes the zero-crossing event. The areas in the red boxes contain variations in recovery time for the variable step solver.

4 To see different post-zero-crossing behaviors, zoom to the region in the red box at time (t) = ~1 second.

Script for Zooming In

```
h1;
xStart = 0;
xEnd = 10;
yStart = 0;
yEnd = 10e0;
xZoomStart1 = 1.001;
xZoomEnd1 = 1.028;
yZoomStart1 = 1.5e-15;
yZoomEnd1 = 3.71e-3;
axis([xZoomStart1 xZoomEnd1 yZoomStart1 yZoomEnd1])
```



After t = 1.005 seconds, the step size decreases from ~10e-3 seconds to less than 10e-13 seconds to capture an event. The step size increases quickly to ~10e-5 seconds, and then slowly to ~10e-4 seconds. The step size decreases to capture a second event and recovers quickly, and then slowly to the step size from before the first event. The slow rates of recovery indicate that the simulation is using small steps to capture the dynamics of elements in your model. If the required step size limits the maximum fixed-step size to a small enough value, then an overrun might occur when you attempt simulation on your real-time computer.

The types of elements that require small step size are:

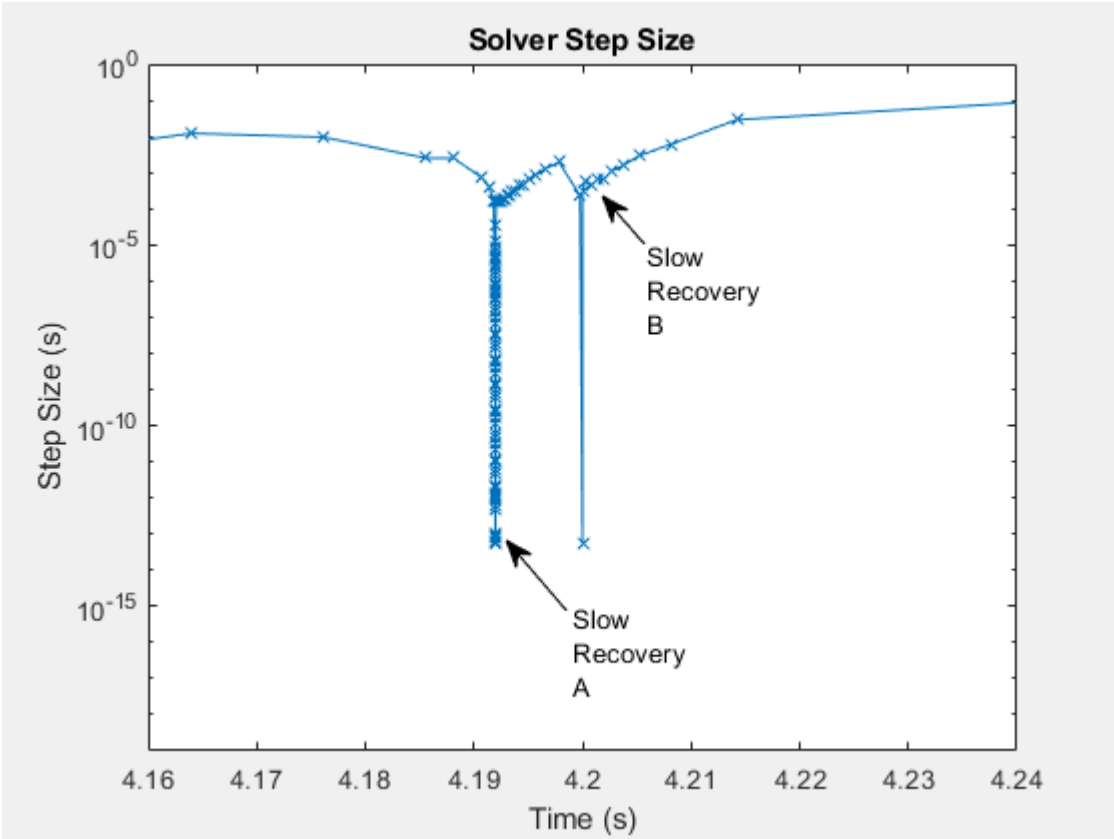
- Elements that cause discontinuities, such as hard-stops and stick-slip friction

- Elements that have small time constants, such as small masses with undamped, stiff springs and hydraulic circuits with small, compressible volumes

The step size recovers more quickly after it slows down to process the event that occurs before $t = 1.02$ seconds. This event is less likely to require small step sizes to achieve accurate results.

5 To see different types of slow solver recoveries, zoom to the region within the red box at $t = \sim 4.2$ seconds.

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



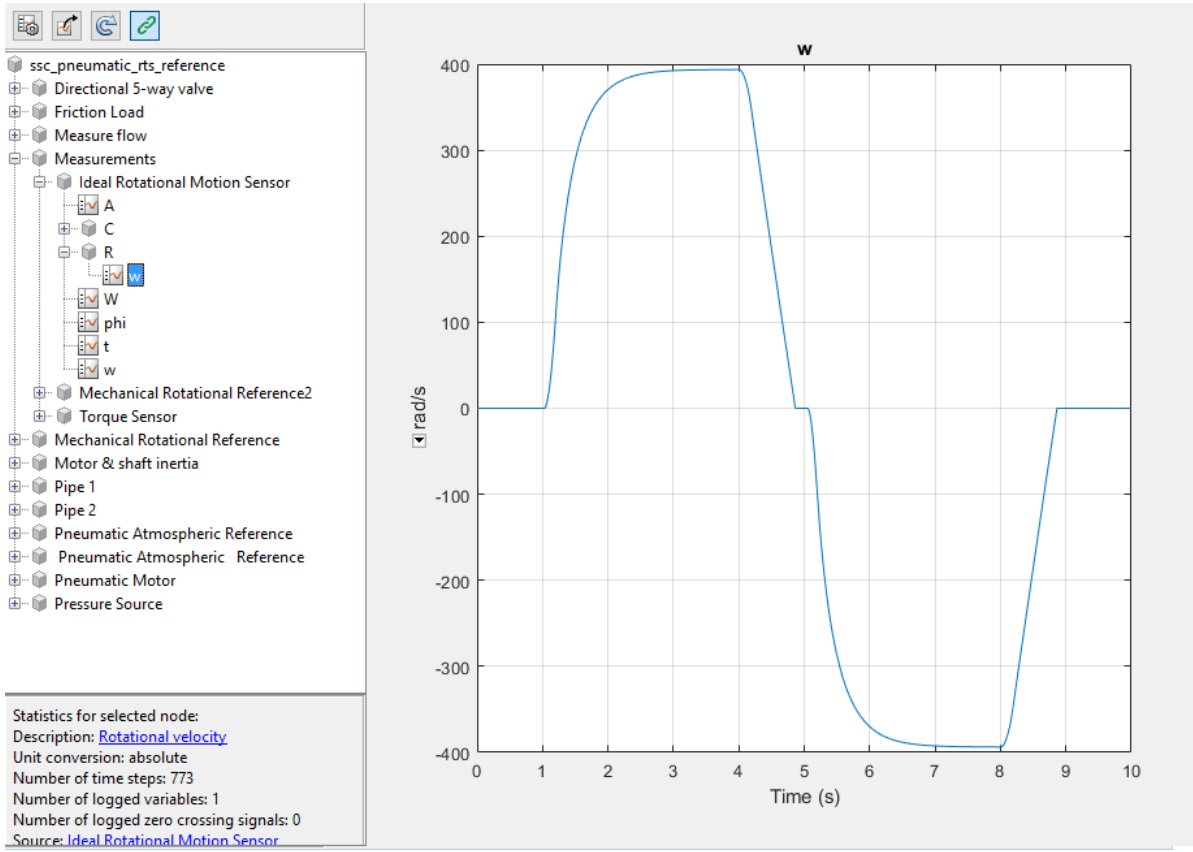
Just as there are different types of events that cause solvers to slow down, there are different types of slow solver recovery. The events that occur just before $t = 4.19$ and 4.2 seconds both involve zero-crossings. The solver takes a series of progressively larger steps as it reaches the step size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

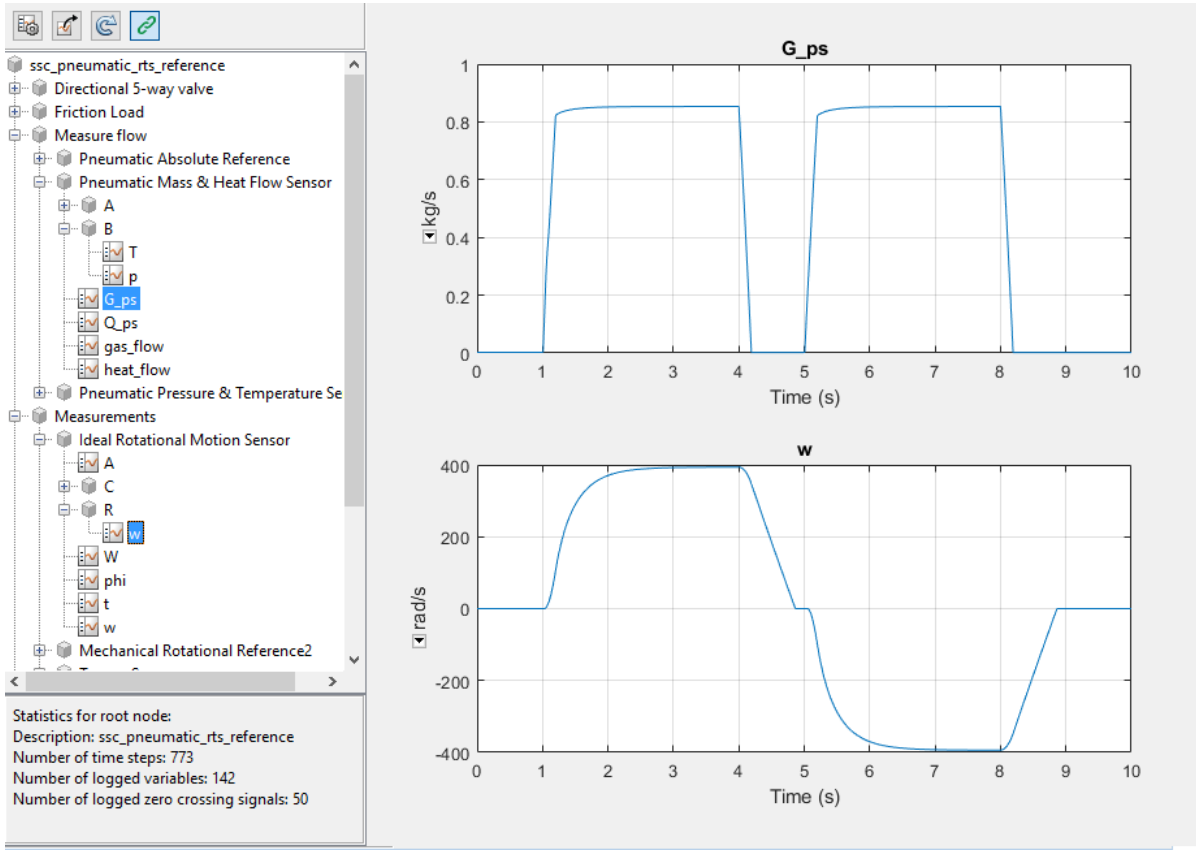
6 To see the results, open the Simscape Results Explorer.

```
sscexplore(simlog)
```

7 Examine the angular speed. In the Simscape Results Explorer window, in the simulation log tree hierarchy, select **Measurements > Ideal Rotational Motion Sensor >** **w**.



8 To add a plot of the gas flow, select **Measure Flow > Pneumatic Mass & Heat Flow Sensor** and then, use Ctrl+click to select **G_ps**.



The slow recovery times occur when the simulation initializes, and approximately at t = 1, 4, 5, 8, and 9 seconds. These periods of small steps coincide with these times:

- The motor speed is near zero rpm (simulation time t = ~ 1, 5, and 9 seconds)
- The step change in motor speed is initiated from a steady-state speed to a new speed (time t = ~ 4 and 8 seconds)
- The step change in flow rate is initiated from a steady-state speed to a new flow rate (time t = ~ 4 and 8 seconds)
- The volumetric flow rate is near zero kg/s (t = ~ 1, 4, and 5 seconds)

These results indicate that the slow step-size recoveries are most likely due to elements in the model that involve friction or that have small, compressible volumes.

To see how to identify the problematic elements and modify them to increase simulation speed, see “Reduce Numerical Stiffness” on page 10-63 and “Reduce Zero Crossings” on page 10-73.

See Also

Related Examples

- “Estimate Computation Costs” on page 10-118
- “Reduce Computation Costs” on page 10-34
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Choose Step Size and Number of Iterations” on page 10-121

More About

- “About Simulation Data Logging” on page 12-2
- “Events and Zero Crossings” on page 6-3
- “Improving Speed and Accuracy” on page 10-12
- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Model Preparation Objectives” on page 10-3
- “Real-Time Model Preparation Workflow” on page 10-6
- “Real-Time Simulation Workflow” on page 10-98
- “Solvers for Real-Time Simulation” on page 10-104
- “Stiffness” on page 6-3
- “Zero-Crossing Detection” (Simulink)

Increase Simulation Speed Using the Partitioning Solver

The Partitioning solver is a Simscape fixed-step local solver that improves performance for certain models by reducing computational cost of simulation. Decreased computational cost yields faster simulation rates for desktop simulation and decreased task execution time (TET) for deployment. The solver converts the entire system of equations for the attached Simscape network into several smaller sets of switched linear equations that are connected through nonlinear functions. Computational cost is reduced because it is more efficient to calculate solutions for several smaller equation systems than it is to calculate the solution for one large system of equations.

The Partitioning solver does not partition models, that is it does not split a model into separate subsystems for multicore processing. To learn how to partition a Simscape model, see “Partition a Model” on page 10-86.

To use the Partitioning solver, open the Solver Configuration block dialog box and:

1. Select the **Use local solver** check box.
2. Set the **Solver type** parameter to `Partitioning`.
3. Clear the **Start simulation from steady state** check box.
4. Set the **Equation formulation** parameter to `Time`.

For real-time simulation, also select the **Use fixed-cost runtime consistency iterations** check box. For more information, see “Make Your Model Real-Time Viable” on page 10-101.

Limitations

Not all networks can simulate with the Partitioning solver. A simulation that uses the Partitioning solver results in an error if the Simscape network cannot be represented by switched linear equations connected through nonlinear functions. Simulating with the Partitioning solver also yields an error for networks that contain:

- A custom component that uses the Simscape language `delay` operator.
- A block that uses a discrete sample time for periodic events. Examples include the Counter, Random Number, Repeating Sequence, or Uniform Random Number blocks from the **Simscape > Physical Signals > Sources** library.

Certain Solver Configuration block settings are not compatible with the Partitioning solver. A simulation that uses a Partitioning solver results in an error if the model contains a Solver Configuration block with:

- **Start simulation from steady state** selected
- **Equation formulation** set to `Frequency and time`

Options

To further improve simulation performance, you can set the **Partition storage method** parameter to `Exhaustive` and specify a value for the **Partition memory budget [kB]** parameter, based on the **Total memory estimate** data in the Statistics Viewer. For more information, see Solver Configuration and “Model Statistics Available when Using the Partitioning Solver” on page 13-25.

Simulate a Simscape Model Using the Partitioning Solver

This example shows how to compare the speed and the accuracy of a simulation that uses the Partitioning solver to baseline results. It also shows how to compare the speeds of the Partitioning solver and the Backward Euler solver.

1 Open the model. At the MATLAB command prompt, enter the code.

See Code

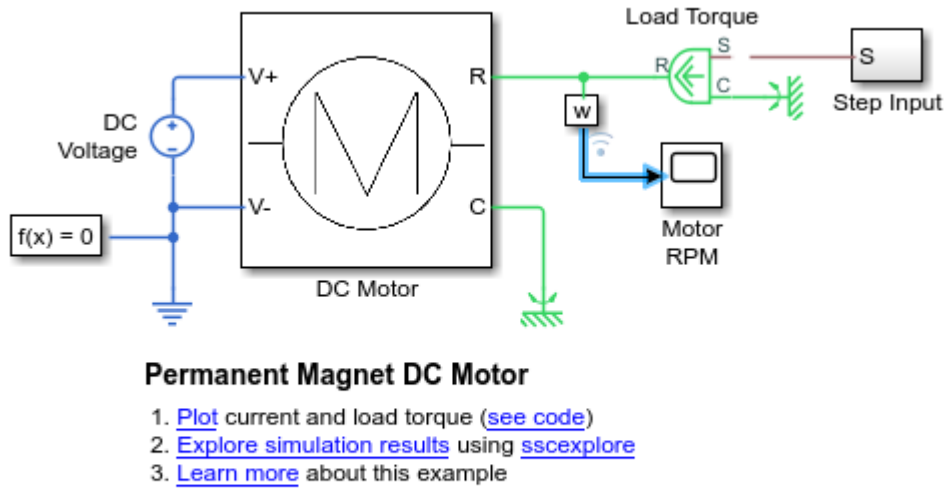
```
model = 'ssc_dcmotor';
open_system(model)
```

2 Enable the signal that goes to the **Motor RPM** scope block for Simulink data logging and viewing with the Simulation Data Inspector.

See Code

```
% Define the w sensor subsystem and the path
%    to it as variables
rpmSensor = 'Sensing';
rpmSensorPath = [model,'/',rpmSensor];

% Mark the output signal from the w sensor subsystem
%   for Simulink(R) data logging
phRpmSensor = get_param(rpmSensorPath,'PortHandles');
set_param(phRpmSensor.Outport(1),'DataLogging','on')
```

The logging badge marks the signal in the model.



3 Run timed simulations using the `sim`, `tic`, and `toc` functions with each of these solvers:

- Variable-step global solver, the original solver for the model
- Fixed-step local Backward Euler solver
- Fixed-step local Partitioning solver

See Code

```
% Define the Solver Configuration block and the path
%    to it as variables
solvConfig = 'Solver Configuration';
solvConfigPath = [model,'/',solvConfig];

% Run a timed simulation using the original solver
tic;
sim(model)
tBaseline = toc;
```

```
% Select option Solver Configuration >  Use local solver
set_param(solvConfigPath, 'UseLocalSolver', 'on')

% Set Solver Configuration >  Solver type > Partitioning
set_param(solvConfigPath, 'LocalSolverChoice',...
    'NE_PARTITIONING_ADVANCER')

% Run a timed simulation using the Partitioning solver
tic;
sim(model)
tPartition = toc;

% Set Solver Configuration >  Solver type > Backward Euler
set_param(solvConfigPath, 'UseLocalSolver', 'on',...
    'LocalSolverChoice','NE_BACKWARD_EULER_ADVANCER')

%%
% Run a timed simulation using the Backward Euler solver
tic;
sim(model)
tBackEuler = toc;

% Compute the percent difference for the simulation times
spdPartitionVsBaseline = 100*(tBaseline - tPartition)/tBaseline;
spdBackEulerVsBaseline = 100*(tBaseline - tBackEuler)/tBaseline;
spdPartitionVsBackEuler = 100*(tBackEuler - tPartition)/tBackEuler;

% Reset the solver to the original setting by deselecting Use local solver
set_param(solvConfigPath, 'UseLocalSolver', 'off')

%%
compTimeDiffTable = table({'Baseline';...
    'Partitioning';...
    'Backward Euler'},...
    {tBaseline;tPartition;tBackEuler},...
'VariableNames', {'Solver','Sim_Duration'});

display(compTimeDiffTable);

%%
compPctDiffTable = table({'Partitioning versus Baseline';...
    'Backward Euler versus Baseline';...
    'Partitioning versus Backward Euler'},...
    {spdPartitionVsBaseline;...
    spdBackEulerVsBaseline;...
    spdPartitionVsBackEuler},...
'VariableNames', {'Comparison','Percent_Difference'});

display(compPctDiffTable);
```

```
compTimeDiffTable =

  3×2 table

        Solver          Sim_Duration
    ________________    ____________

    'Baseline'            [5.1467]
    'Partitioning'        [4.6819]
```

```
    'Backward Euler'      [4.6942]


compPctDiffTable =

  3×2 table

                Comparison                 Percent_Difference
    ___________________________________    __________________

    'Partitioning versus Baseline'               [9.0308]
    'Backward Euler versus Baseline'             [8.7911]
    'Partitioning versus Backward Euler'         [0.2629]
```

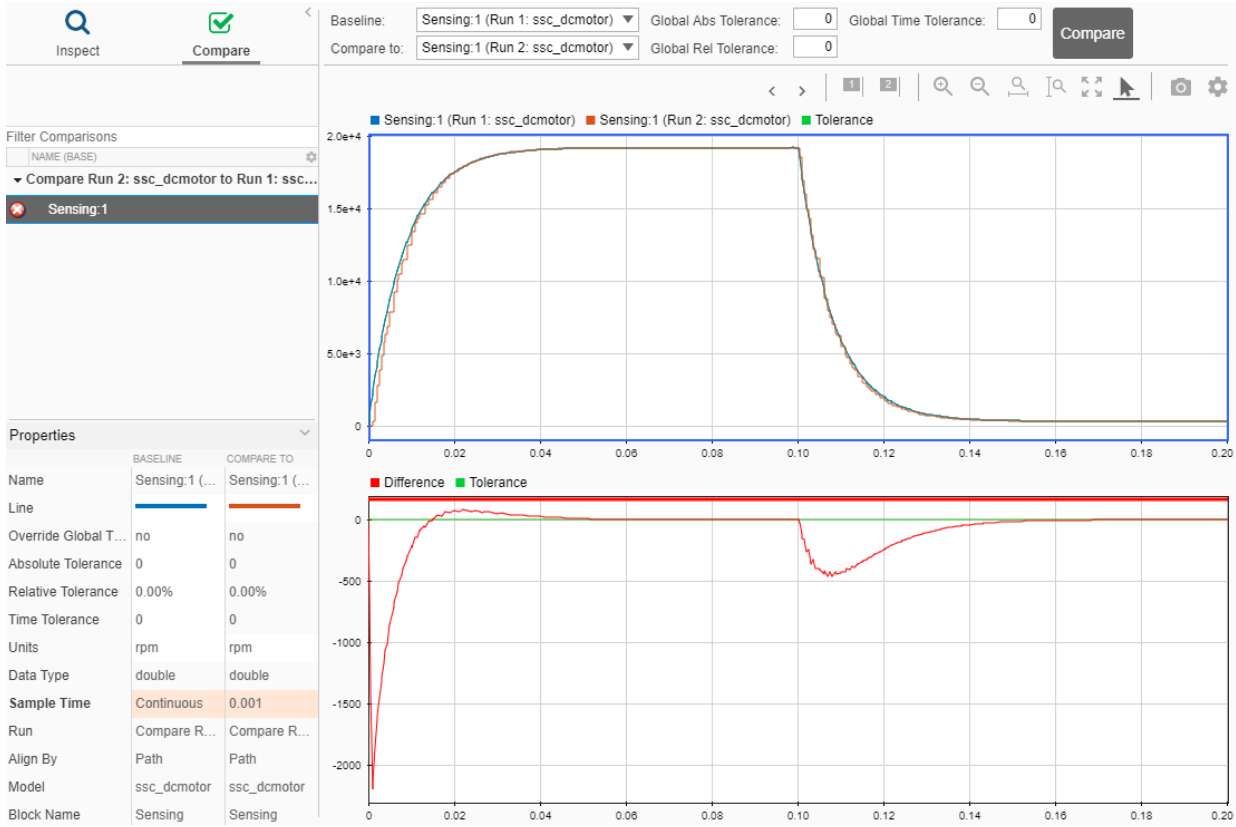
Simulation time on your machine may differ because simulation speed depends on machine processing power and the computational cost of concurrent processes.

The local fixed-step Partitioning and Backward Euler solvers are faster than the variable-step baseline solver. The Partitioning solver is faster than the Backward Euler solver.

4 Compare the results using the Simulation Data Inspector.

See Code

```
% Get Simulink Data Inspector run IDs for
%    the last three runs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 2);
runPartition = runIDs(end - 1);
runBackEuler = runIDs(end);

% Open the Simulink Data Inspector
Simulink.sdi.view

compBaselinePartition = Simulink.sdi.compareRuns(runBaseline,...
    runPartition);
```



The first plot shows the overlay of the baseline and Partitioning solver simulation results. The second plot shows how they differ. The default tolerance for differences is `0`. To determine if the accuracy of the results meet your requirements, you can adjust the relative, absolute, and time tolerances. For more information, see “Compare Simulation Data” (Simulink).

See Also

Related Examples

- “Determine Step Size” on page 10-17

- “Estimate Computation Costs” on page 10-118
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Partition a Model” on page 10-86

More About

- “Model Statistics Available when Using the Partitioning Solver” on page 13-25
- “About Simulation Data Logging” on page 12-2
- “Limitations” on page 12-3
- “Improving Speed and Accuracy” on page 10-12
- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Model Preparation Objectives” on page 10-3
- “Real-Time Model Preparation Workflow” on page 10-6
- “Real-Time Simulation Workflow” on page 10-98

Reduce Computation Costs

In this section...
“Data Logging and Monitoring Guidelines” on page 10-34 “Improve Data Logging and Monitoring Efficiency” on page 10-35 “Additional Methods for Reducing Computational Cost” on page 10-38

In this section...
“Data Logging and Monitoring Guidelines” on page 10-34 “Improve Data Logging and Monitoring Efficiency” on page 10-35 “Additional Methods for Reducing Computational Cost” on page 10-38

Computational cost is a measure of the number and the complexity of tasks that a processor performs per time step during a simulation. Lowering the computational cost of your model increases simulation execution speed and helps you to avoid overruns when you simulate in real time on target hardware.

Data Logging and Monitoring Guidelines

Data logging and monitoring are interactive procedures that consume memory and processing power. One way to reduce computational cost is to reduce the amount of interactive processing that occurs during simulation. Best practices for limiting computational costs while logging and monitoring data are:

- Use an outport block only if you need to log data for your analysis via the Simulink model on your development computer.
- Use a scope block only if you need to monitor data during real-time simulation via the Simulink model on your development computer.
- If you need to log data or monitor a variable, limit the number or the decimation of data points that you collect whenever your analysis requirements permit you to do so.
- Log data only once.
- If you use Simscape data logging, use local settings to log only the blocks that contain variables that you need for your analysis.

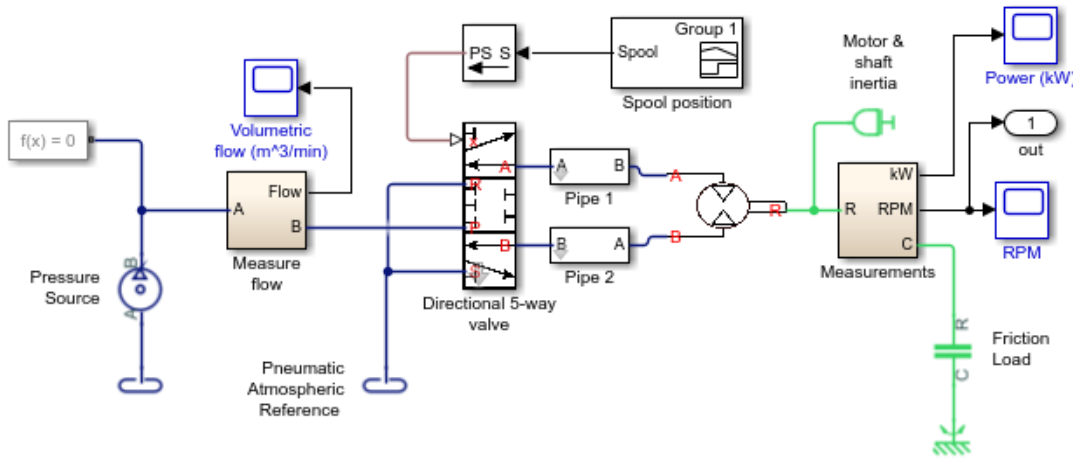
Note Simscape simulation data logging is not supported for generated code.

Improve Data Logging and Monitoring Efficiency

Examine the configuration of the model and the simulation results to determine if the model is logging and monitoring data efficiently.

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

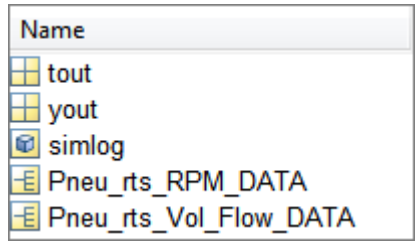
```
model = 'ssc_pneumatic_rts_zc_redux';
open_system(model)
```



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

2 Simulate the model:

```
sim(model)
```



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

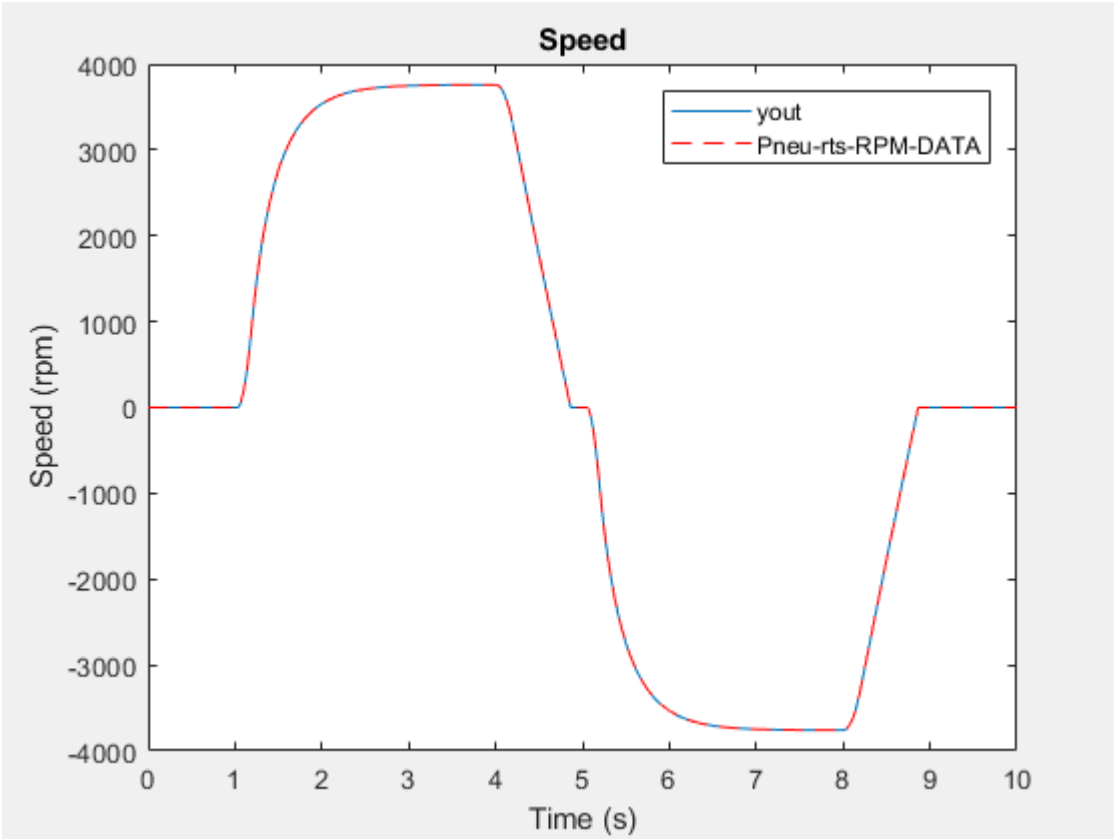
3. To determine the source for the `Pneu_rts_RPM_DATA`, in the MATLAB workspace, open the structure. Alternately, at the command line, enter:

```
Pneu_rts_RPM_DATA.blockName

ans =

    'ssc_pneumatic_rts_zc_redux/RPM'
```

The `blockName` variable shows that the RPM scope logs the data. In the model, the outport that logs data to `yout` connects to the signal between the Measurements subsystem and the RPM scope block.

4. To compare the data that `Pneu_rts_RPM_DATA` and `yout` log, plot both data sets to a single figure.

```
h1 = figure;
plot(tout,yout)
h1;
hold on
plot(Pneu_rts_RPM_DATA.time,Pneu_rts_RPM_DATA.signals.values,'r--')
title('Speed')
xlabel('Time (s)')
ylabel('Speed (rpm)')
h1Leg = legend({'yout','Pneu-rts-RPM-DATA'});
```



The data is the same, which means that you are logging the same data twice.

To reduce the computational cost for logging or monitoring the speed data via the Simulink model on your development computer during real-time simulation:

- If you only need to log the speed data, delete the RPM scope block.
- If you need to log and monitor the speed data, delete the outport block.
- If you only need to monitor the speed data, delete the outport block and disable data logging for the RPM scope.

If you do not need to log or monitor the speed data via the Simulink model on your development computer during real-time simulation with target hardware, delete both the RPM scope block and the outport block.

If you want to reduce costs by deleting the scope and outport blocks, but you want to log data while you prepare your model for real-time simulation, configure the model to log only the data that you need. To do so, use a `simlog` node in the MATLAB workspace. For information, see “Log Data for Selected Blocks Only” on page 12-6.

Additional Methods for Reducing Computational Cost

In addition to reducing the number of logged and monitored signals, you can use these methods for decreasing the number and complexity of tasks that the processor performs per time step during simulation:

- Avoid using large images and complex graphics.
- Disable unnecessary error and warning diagnostics.
- Reconfigure tolerances.
- Simplify complex subsystems or replace them with lookup tables.
- Linearize nonlinear effects.
- Eliminate redundant calculations, for example, multiplication by one.
- Reduce the number of differential algebraic equations (DAEs).

See Also

Related Examples

- “Determine Step Size” on page 10-17
- “Estimate Computation Costs” on page 10-118
- “Reduce Fast Dynamics” on page 10-40
- “Reduce Numerical Stiffness” on page 10-63
- “Reduce Zero Crossings” on page 10-73
- “Partition a Model” on page 10-86

More About

- “About Simulation Data Logging” on page 12-2
- “Limitations” on page 12-3

- “Improving Speed and Accuracy” on page 10-12
- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Model Preparation Objectives” on page 10-3
- “Real-Time Model Preparation Workflow” on page 10-6
- “Real-Time Simulation Workflow” on page 10-98

Reduce Fast Dynamics

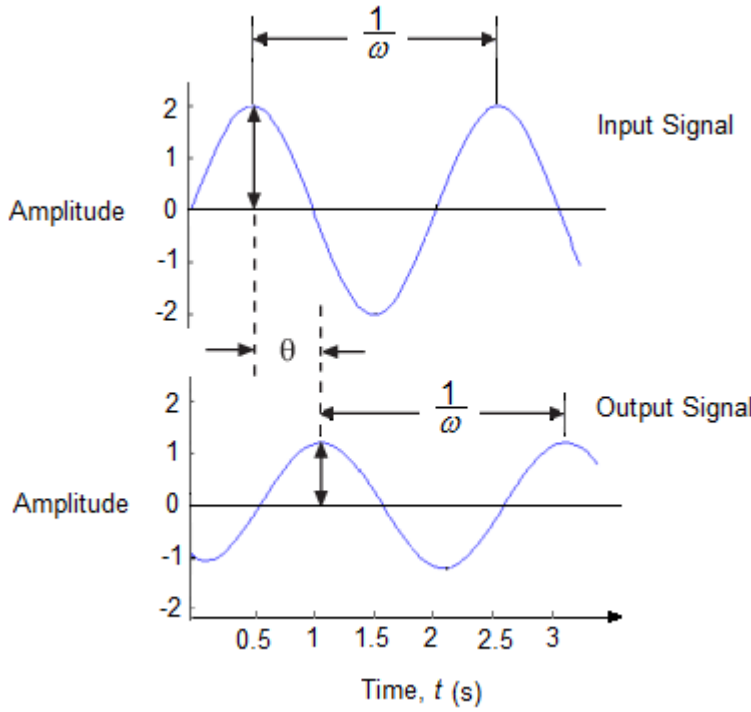
You can make your model real-time capable by identifying and reducing sources of fast or high-frequency dynamics. A real-time capable model is one that produces acceptable results on a real-time machine without generating overruns.The example shows you how to identify fast dynamics by examining the frequency response and pole locations of a linearized model. It also shows how to identify and remove sources of the fast dynamics.

Why Reduce Fast Dynamics

Models with fast dynamics typically have a high computational cost. Removing fast dynamics decreases the computational cost and increases the minimum step size that you can specify for a fixed-step, fixed-cost simulation. Using a larger step size increases the likelihood that your model is real-time capable.

Frequency-Response Analysis

Frequency response describes the steady-state response of a system to sinusoidal inputs. For a linear system, a sinusoidal input results in an output that is a sinusoid with the same frequency, ω, but with a different amplitude and phase, θ.



Frequency analysis shows how amplitude and phase change over a given range of frequencies. For a small change in frequency, a large magnitude or phase change indicates that a system has fast dynamics. This example uses Bode plots, which allow you to see how the amplitude, in terms of magnitude in dB, and phase vary as a function of frequency.

Pole Analysis

Fast poles are also indicative of fast dynamics. Fast poles are poles that respond or oscillate rapidly. Poles that have real components that are far to the left of the imaginary axis on the complex plane have a fast response speed. Complex pole pairs that have imaginary components that are far from the real axis oscillate rapidly. For example, the real pole at -1500 has a faster response speed than the real pole at -1000 and the complex pole pair at -500 ± 1500i has a faster oscillation speed than the complex pole pair at -500 ± 500i.

b Add a Scope block to the Simscape model.

c Connect the scope to the output signal from the PS-Simulink Converter block.

2 Configure Simulink Real-Time Scope block as a host scope.

a Specify the **Scope type** parameter as `Host`.

b Set the **Number of samples** to `100000`.

c Click **OK**.

3 Configure the Simscape model configuration parameters for TET analysis using Simulink Real-Time.

a Open the model configuration parameters.

b In the left pane of the Model Parameter Configuration window, select **Code Generation**.

c Click **Browse...**

d In the System Target File Browser, for the **System Target File**, select `slrt.tlc`.

e Click **OK**.

f In the left pane of the Model Parameter Configuration window, select **Code Generation > Verification**.

g Select the **Measure task execution time** check box.

h Click **OK**.

Evaluate Task-Execution Time

Connect your development computer to a real-time target machine before you can complete these steps. For information on configuring and connecting your development computer to target hardware, see Simulink Real-Time Setup and Configuration (Simulink Real-Time).

You can use a Simulink Real-Time target object to check for overruns and determine TET after you simulate in real time on target hardware. The target object, which communicates with the real-time target machine, reports the status of the real-time application to the development computer.

1 Run your real-time application on the target hardware.

a To generate code from your model, click the Build button .

b To set the simulation mode for executing on a target machine, in the Simulink window, on the toolbar, select `External`.

c To connect your development computer and target machine and to transfer your model parameters to the target machine, click the Connect to Target button .

d To execute your real-time application on the target machine, click the Run button .

2 To use the target object to print the simulation status, on your development computer, at the MATLAB command prompt, enter:

```
tg = slrt
```

```
Target: SLRTLABTGT2
   Connected            = Yes
   Application          = ssc_resistive_ac_circuit
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = detected
```

```
ExecTime              = 0.0000
SessionTime           = 11.8761
StopTime              = 1.000000
SampleTime            = 0.000010
AvgTET                = 0.000021
MinTET                = 0.000021
MaxTET                = 0.000021
ViewMode              = 0

TimeLog               = Vector(2)
StateLog              = Off
OutputLog             = Off
TETLog                = Vector(2)
MaxLogSamples         = 50000
NumLogWraps           = 0
LogMode               = Normal

Scopes                = 1
NumSignals            = 3
ShowSignals           = off

NumParameters         = 2
ShowParameters        = off
```

In this case, when the task execution time (**maxTET = 0.000021**) exceeds the step size (**SampleTime = 0.000010**), the Simulink Real-Time kernel detects an overrun (**CPUOverload = detected**) and stops the simulation.

3. The simulation stops at the very beginning of the execution even though it has a specified stop time of 1 second (**ExecTime = 0**, **StopTime = 1.000000**). In order to determine the maximum TET for the entire simulation time, use the Simulink Real-Time `TLCOptions` Properties function to configure the real-time application to continue to run after it detects an overrun. The function allows you to specify target computer overload values for the `Number of acceptable overloads property` programmatically. The application executes through the number of overloads that you specify, but stops executing if it detects an additional overload.

   ```
   set_param('ssc_resistive_ac_circuit',...
     'TLCOptions', '-axPCMaxOverloads=200000')
   ```

4. To regenerate the code with the new property setting, click the Build button.
5. To reconnect to the target machine, click the Connect To Target button.

6 Run the simulation.

7 Obtain the statistics.

```
tg = slrt

Target: SLRTLABTGT2
   Connected            = Yes
   Application          = ssc_resistive_ac_circuit
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none

   ExecTime             = 1.0000
   SessionTime          = 6.3129
   StopTime             = 1.000000
   SampleTime           = 0.000010
   AvgTET               = 0.000016
   MinTET               = 0.000014
   MaxTET               = 0.000032
   ViewMode             = 0

   TimeLog              = Vector(50000)
   StateLog             = Off
   OutputLog            = Off
   TETLog               = Vector(50000)
   MaxLogSamples        = 50000
   NumLogWraps          = 2
   LogMode              = Normal

   Scopes               = 1
   NumSignals           = 3
   ShowSignals          = off

   NumParameters        = 2
   ShowParameters       = off
```

The simulation executes to completion with a maximum TET of 0.000032.

Adjust Step Size Based on Maximum Task-Execution Time

Adjust the sample time, rerun the simulation, check for overruns and for the accuracy of the results.

1. To avoid simulation overruns, specify a step size that is at least 20% larger than the TET_{max} from the simulation on your real-time target machine. At the MATLAB command prompt, enter:

   ```
   ts = 4e-5;
   ```

2. Remove the overload override.

   ```
   set_param('ssc_resistive_ac_circuit', 'TLCOptions', '')
   ```

3. Regenerate the code.
4. Connect to the target machine.
5. Run the simulation.
6. Obtain the statistics.

   ```
   tg = slrt
   ```

   ```
   Target: SLRTLABTGT2
      Connected            = Yes
      Application          = ssc_resistive_ac_circuit
      Mode                 = Real-Time Single-Tasking
      Status               = stopped
      CPUOverload          = none

      ExecTime             = 1.0000
      SessionTime          = 8.7715
      StopTime             = 1.000000
      SampleTime           = 0.000040
      AvgTET               = 0.000016
      MinTET               = 0.000015
      MaxTET               = 0.000028
      ViewMode             = 0

      TimeLog              = Vector(25001)
      StateLog             = Off
      OutputLog            = Off
      TETLog               = Vector(25001)
      MaxLogSamples        = 50000
      NumLogWraps          = 0
      LogMode              = Normal

      Scopes               = 1
      NumSignals           = 3
      ShowSignals          = off
   ```

```
NumParameters        = 2
ShowParameters       = off
```

The simulation executes to completion with a maximum TET of 0.000028.

7 To see the results, open the Development Scope block.

The current through the resistor is 0.3 A. The results match the reference results and the simulation now runs without yielding an overrun.

See Also

Get Overload Counter | Scope | Set Overload Counter | `SimulinkRealTime.target` | TLCOptions | Task Execution Time | `slrt` | `xPCIsOverloaded`

Related Examples

- “Set Up and Configure Simulink Real-Time” (Simulink Real-Time)
- “Configure Real-Time Host Scope Blocks” (Simulink Real-Time)
- “Generate, Download, and Execute Code” on page 10-157
- “Configure and Control a Real-Time Application” (Simulink Real-Time)

More About

- “Real-Time Model Preparation Workflow” on page 10-6
- Simulink Real-Time Setup and Configuration (Simulink Real-Time)
- “CPU Overload Options” (Simulink Real-Time)
- “Real-Time Application Objects” (Simulink Real-Time)
- “Host Scope Usage” (Simulink Real-Time)

Change Parameter Values on Target Hardware

In this section...
“Prerequisites” on page 10-170 “Configure the Simscape Model for Deployment” on page 10-170 “Deploy the Model to the Real-Time Target Machine” on page 10-172 “Change Parameters and See Results Using Simulink Real-Time Explorer” on page 10-173

This example shows how to:

- Configure a Simscape plant model to generate code that supports signal visualization and changes to Simscape run-time parameters.
- Use Simulink Real-Time and Simulink Coder to deploy an executable version of the plant model to a real-time target machine.
- Use Simulink Real-Time Explorer on your development computer to change the value of a Simscape run-time parameter on the target machine and to see the effects of the parameter change.

Prerequisites

This example requires an active connection between your development computer and a real-time target machine. For information on configuring and connecting your development computer to target hardware, see Simulink Real-Time Setup and Configuration (Simulink Real-Time).

Configure the Simscape Model for Deployment

To prepare to change parameter values on a real-time target machine, configure the Simscape run-time, code generation, and scope parameters for your Simscape model on your development computer.

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

```
ssc_resistive_ac_circuit
```

The model opens and the PreLoadFcn loads parameters for the model to the MATLAB workspace. The peak voltage, *A_peak_voltage_src*, is 3 V, the resistance, *R_resistor*, is 10 Ohms, and the step size is 1e-5.

2. To allot enough time to see the effects of parameter-tuning on the target machine, configure the application to run until you stop the simulation, set the simulation stop time to `inf`.
3. Adjust the step size for real-time simulation. At the MATLAB command prompt, enter:

   ```
   ts = 4e-5;
   ```

4. Configure the model for code generation using Simulink Coder and Simulink Real-Time.
   a. Open the model configuration parameters.
   b. To set the System Target File, on the **Code Generation** pane, click **Browse** and select `slrt.tlc`.
   c. In the System Target File Browser window, click **OK**.
   d. Open the **Report** pane.
   e. To display a code generation report, on the **Report** pane, select the **Create Code Generation report** check box.
   f. Click **OK**.
5. Add a Simulink Real-Time Scope block to the Simscape plant model.
   a. In the Simulink library browser, navigate to the **Simulink-Real Time** > **Displays and Logging** library.
   b. Add a Scope block to the Simscape model.
   c. Connect the scope to the output signal from the PS-Simulink Converter block.

6 Configure Simulink Real-Time Scope block as a host scope.

   a Specify the **Scope type** parameter as `Host`.

   b Set the **Number of samples** to `100000`.

   c Click **OK**.

Deploy the Model to the Real-Time Target Machine

Build an executable application and download it to real-time hardware.

1 Check that you are connected to the real-time target machine, in this case, the target name is SLRTLABTG2:

```
tg = slrt
```

```
Target: SLRTLABTGT2
Connected            = Yes
Application          = loader
```

2 To build and download the code to the target machine, in the model window, click the Build button .

The code report opens after the code download.

3 Verify that the generated code represents the Simscape run-time variables in a data structure.

a. In the Code Generation Report, in the left pane, in the **Data files** node, open `ssc_resistive_ac_circuit_data.c`.

b. Search for the section of the code that contains the parameter variables. In the **Find** box, enter `Block parameters (auto storage)`.

c. Verify that the *A_peak_voltage_src* and the *R_resistor* variables are represented in the *P_ssc_resistive_ac_circuit_T ssc_resistive_ac_circuit_P* data structure.

Change Parameters and See Results Using Simulink Real-Time Explorer

Use Simulink Real-Time Explorer to change Simscape run-time parameters between runs of your real-time application on target hardware. Visualize the simulation results on a scope in the Explorer window.

1. To open Simulink Real-Time Explorer, on your development computer, at the MATLAB command prompt, enter:

   ```
   slrtexplr
   ```

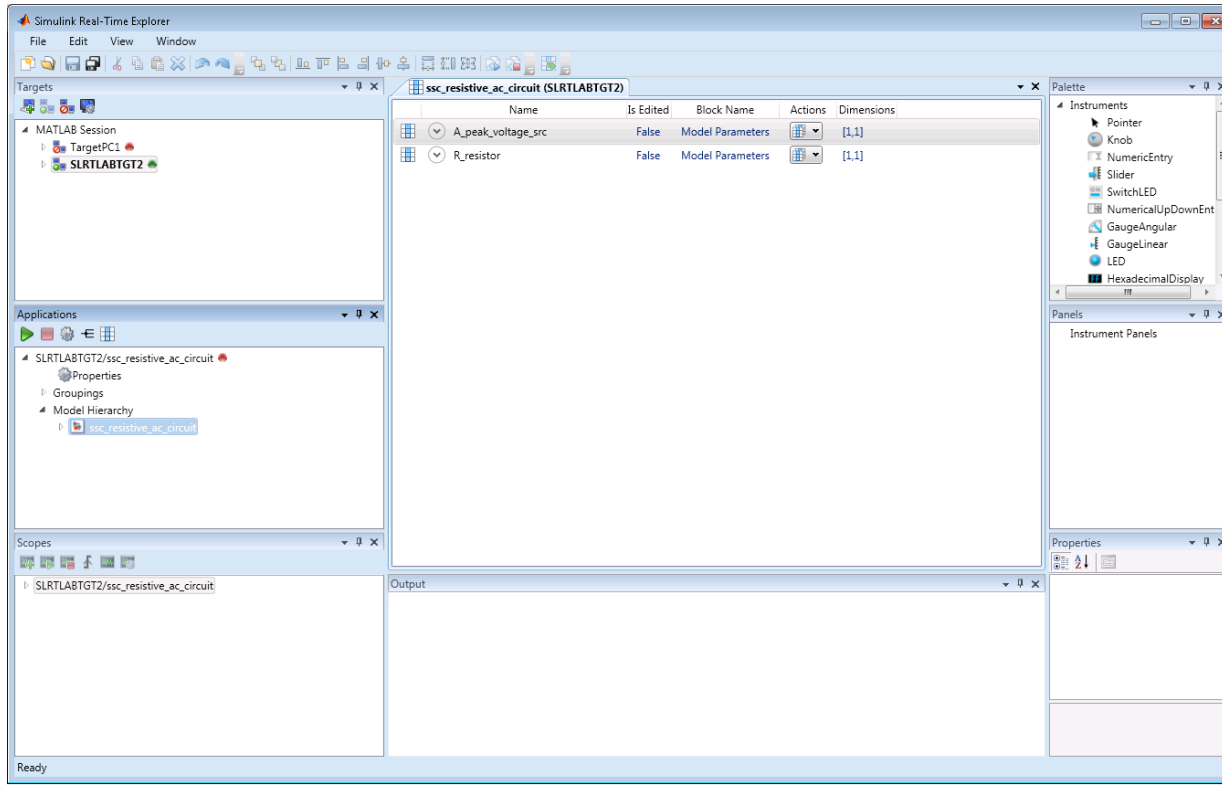
2. View the Simscape run-time parameters in Simulink Real-Time Explorer.

   a. In the **Applications** pane, expand the real-time application node, in this case, SLRTLABTGT2/ssc_resistive_ac_circuit.

   b. Under the real-time application node, expand the **Model Hierarchy** node.

   c. Select **ssc_resistive_ac_circuit** and click the View Parameters button .

      The Parameters workspace opens, showing a table of parameters with properties and actions.



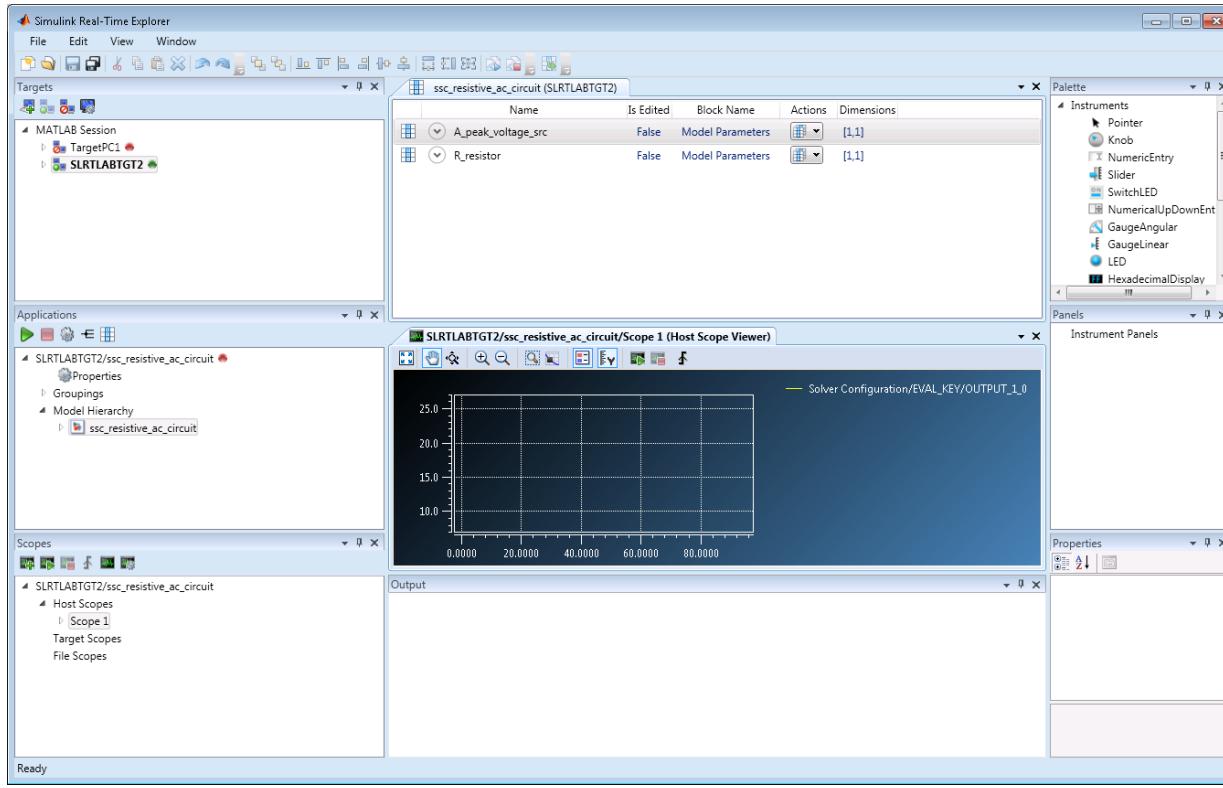
3 View the host scope in Simulink Real-Time Explorer.

   a In the **Scopes** pane, expand the real-time application node (SLRTLABTGT2/ssc_resistive_ac_circuit).

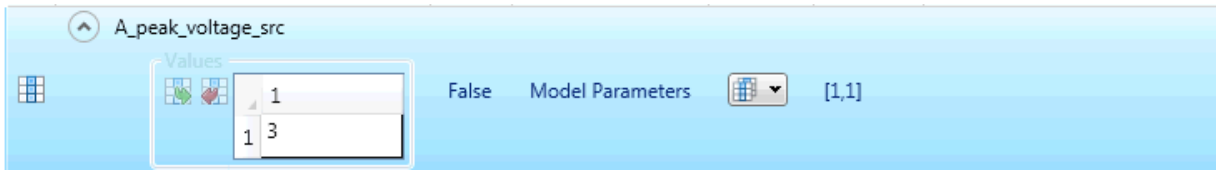
   b Under the real-time application node, expand the **Host Scopes** node.

   c To open the Scopes workspace, select **Scope 1** and click the View Host Scope button .

   d To make both the parameter and scope workspaces visible at the same time, click the scope workspace tab and drag the tab down until the Pane Move button appears in the middle of the dialog box. Release the tab when the cursor reaches the lower quadrant of the Pane Move button.

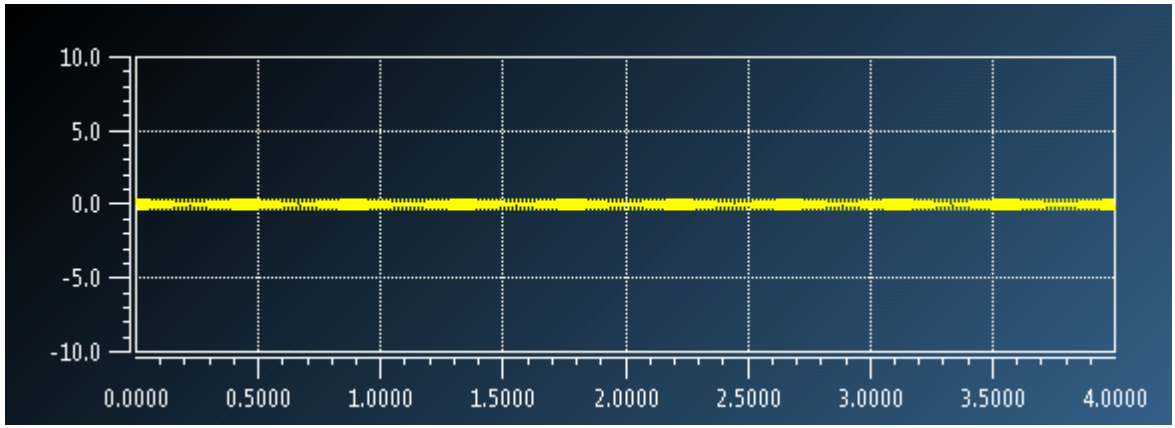


4 To see the value for the peak amplitude of the voltage source, click the arrow next to the **A_peak_voltage_src** parameter.



The **Values** text box opens, containing the original value of 3 V for the peak amplitude.

5 Simulate with the original peak amplitude value. To start execution, click the real-time application (SLRTLABTGT2/ssc_resistive_ac_circuit) and then click the Start button on the Applications toolbar.



The scope shows that the current is approximately 0.3 A. The defining equation for the circuit in the model, is $I = V/R$. The results are correct for the given voltage (10 V) and resistance (3 Ohms).

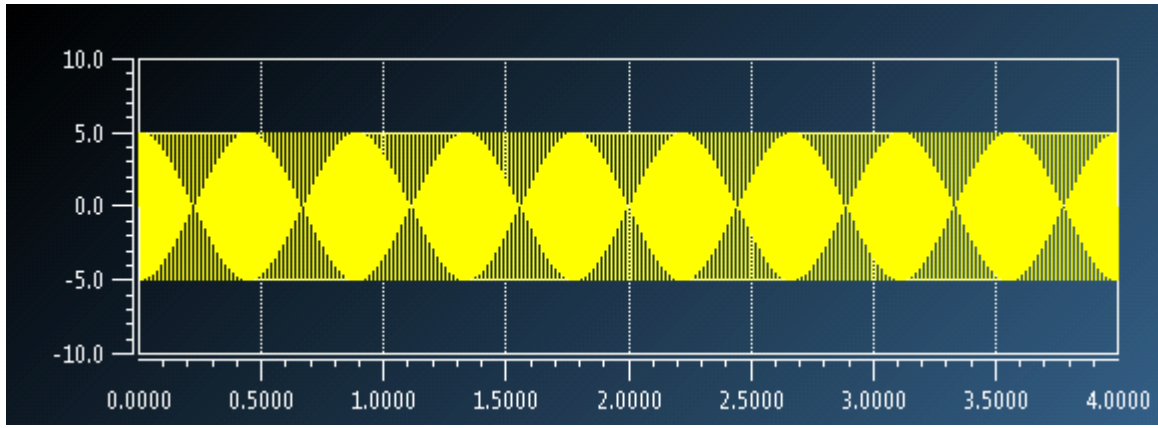
6 Change the parameter that represents the peak amplitude for the Simscape Voltage Source block. Because Simscape run-time parameters are run-time configurable, cannot change the parameter value during simulation. Instead, you stop the simulation, change the value of the parameter, and apply the parameter change. Then, you restart the simulation to see how changing the parameter affects the results.

a To stop execution, click the real-time application in the Applications workspace, and then click the Stop button .

b Specify `50` in the **Values** text box for the peak amplitude parameter and press **Enter**.

c Click the Apply parameter value changes button .

d Click the Start button to simulate with the modified peak amplitude value.

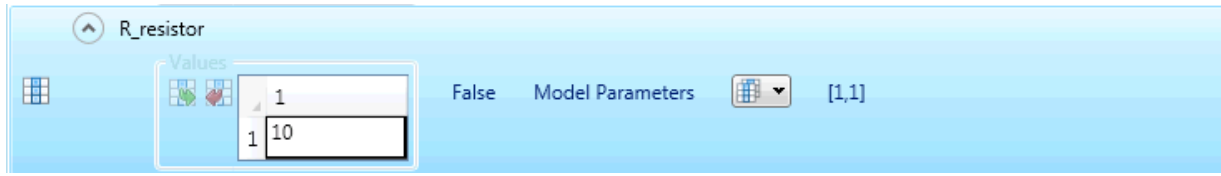


The scope shows that the current is approximately 5 A when the peak amplitude is 50 V. The results reflect the change in the value for the voltage given that the resistance is 10 Ohms.

7 Modify and revert the value for the resistance of the Resistor block.

a Stop the execution.

b Click the arrow next to the **R_resistor** parameter.



The **Values** text box opens, containing the original value of 10 Ohms for the resistance.

c Specify 2 in the text box, and then press Enter.

d To revert the resistance to its previous value, click the Revert button .

See Also

Scope | `SimulinkRealTime.target` | `slrt` | `slrtexplr`

Related Examples

- “Generate, Download, and Execute Code” on page 10-157
- “Set Up and Configure Simulink Real-Time” (Simulink Real-Time)
- “Configure and Control a Real-Time Application” (Simulink Real-Time)
- “Tune and Monitor Models Running on Target Hardware” (Simulink)

More About

- “Manage Simscape Run-Time Parameters” on page 9-7
- Simulink Real-Time Setup and Configuration (Simulink Real-Time)
- “Tunable Block Parameters and Tunable Global Parameters” (Simulink Real-Time)
- “Signal Monitoring Basics” (Simulink Real-Time)
- “Simulink Real-Time Scope Usage” (Simulink Real-Time)
- “Real-Time Application Objects” (Simulink Real-Time)
- “Model Callbacks” (Simulink)

Requirements for Using Alternative Platforms

In this section...
“Hardware Requirements” on page 10-179 “Software Requirements” on page 10-179

Simulink Real-Time creates standalone, real-time applications for performing hardware-in-the-loop (HIL) simulation on dedicated hardware. Creating and executing a standalone, real-time application using an alternative platform requires specific hardware and software.

Hardware Requirements

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

- Development computer with a network, serial, or USB interface for communicating with the real-time processor
- A real-time capable target CPU or computer that supports 64-bit precision floating-point arithmetic and 32-bit integer size
- I/O board supported by the real-time target machine
- Controller preconfigured with code from your controller model
- Peripheral for transferring code to the real-time target machine
- Wiring harness to connect the real-time target machine to the controller

Note Your real-time target machine may also require a real-time operating system (RTOS).

Software Requirements

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

- Embedded Coder and the Embedded Coder Software Requirements.
- Simulink Coder and the Simulink Coder Software Requirements.
- Template example main function that you can manually or automatically combine with generated code. For information, see “Incorporate Generated Code Using an Example Main Function” (MATLAB Coder).

- I/O driver. Options are:
  - C code I/O drivers for the code generation build
  - Precompiled static or dynamic library with the necessary documentation
- Compiler requirements
  - C compiler.
  - Cross compiler that supports 64-bit precision floating-point arithmetic and 32-bit integer size.

  For details on supported compiler versions, see Supported and Compatible Compilers.

  By default, Simulink Coder uses ISO®/IEC 9899:1990 (C89/C90 [ANSI]) library to produce C code. Not all compilers support this library. To learn how to enable the code generator to use a different math extensions library in a model, see “Configure Standard Math Library for Target System” (Simulink Coder).

See Also

More About

- “About Code Generation from Simscape Models” on page 11-2
- “How Simscape Code Generation Differs from Simulink” on page 11-5
- “Hardware-In-The-Loop Simulation Workflow” on page 10-142
- “Limitations” on page 12-3
- “What Is Hardware-In-The-Loop Simulation?” on page 10-138
- “Configure Run-Time Environment Options” (Simulink Coder)

Extending Embedded and Generic Real-Time System Target Files

Simulink Coder and Embedded Coder use system target files (STFs) to generate code for interfacing with specific real-time operating systems. The Target Language Compiler (TLC) uses STFs and various other target files to convert a model into generated code. In addition to including STFs for ready-to-run configurations, Simulink Coder and Embedded Coder allow you to extend STFs to support third-party and custom target hardware. For more information on TLC files and STFs, including a list of available STFs, see “Configure a System Target File” (Simulink Coder) and “Target Language Compiler Basics” (Simulink Coder).

Generic real-time (GRT) is a Simulink Coder STF that you can use when you generate code from a Simscape model for hardware-in-the-loop (HIL) simulation. To generate code for HIL simulation, you must configure your Simscape model to use a fixed step, local solver. To learn about Simscape solver configurations that support HIL simulation, see “Solvers for Real-Time Simulation” on page 10-104.

Embedded real-time (ERT) is an Embedded Coder STF for deploying production-quality code for real-time execution of the algorithm for your Simulink controller. Do not deploy code that you generate from a Simscape model to production platforms. Simscape models contain constructs that are not compatible with performance-related Embedded Coder Model Advisor checks, such as “Check for blocks not recommended for C/C++ production code deployment”. For more information, see “Embedded Coder Model Advisor Checks for Standards, Guidelines, and Code Efficiency” (Embedded Coder).

To extend ERT or GRT system target files and create hardware-specific, standalone applications, use the toolchain build process approach. The toolchain approach generates optimized makefiles and supports custom toolchains. For information, see “Customize System Target Files” (Simulink Coder) and “Support Toolchain Approach with Custom Target” (Simulink Coder). To get started extending system target files, see “Sample Custom Targets” (Simulink Coder).

Third-party vendors supply additional system target files for the Simulink Coder product. For more information about third-party products, see the MathWorks Connections Program Web page: `https://www.mathworks.com/products/connections`.

See Also

`tlc`

More About

- “Compare System Target File Support Across Products” (Simulink Coder)
- “Template Makefiles and Make Options” (Simulink Coder)
- “Customize System Target Files” (Simulink Coder)
- “Custom Target Optional Features” (Simulink Coder)
- “Configure Production and Test Hardware” (Simulink Coder)
- “Target Language Compiler Basics” (Simulink Coder)
- “Configure Generated Code with TLC” (Simulink Coder)
- “Target Language Compiler Library Functions Overview” (Simulink Coder)

Precompiled Static Libraries

Simscape and its add-on products provide static libraries precompiled for compilers supported by Simulink Coder software. For details on supported compilers, see Supported and Compatible Compilers. For all other compilers, the static libraries needed by code generated from Simscape models are compiled once per model during the code generation build process.

To save time during the build process, precompile new or updated S-function libraries (MEX-files) for a model by using the MATLAB language function `rtw_precompile_libs`. You can also use the `rtw_precompile_libs` function to recompile a precompiled S-function library. Recompiling a precompiled library allows you to customize compiler settings for various platforms or environments. For details on using `rtw_precompile_libs`, see “Precompile S-Function Libraries” (Simulink Coder).

Typically, a target machine places cross-compiled versions of the precompiled libraries in a default location as specified in an rtwmakecfg.m file. The default file suffix and file extension used by the Simulink Coder code generator to name precompiled libraries during the build process are:

- On Windows® systems, *`model`*`_rtwlib.lib`
- On UNIX® or Linux® systems, *`model`*`_rtwlib.a`

You can control the file destination, location, suffix, and extension by customizing the system target file (STF) for your target hardware. For more information, see “Control Library Location and Naming During Build” (Simulink Coder) and “Use rtwmakecfg.m API to Customize Generated Makefiles” (Simulink Coder).

See Also

`rtw_precompile_libs`

Related Examples

- “Use rtwmakecfg.m API to Customize Generated Makefiles” (Simulink Coder)
- “Control Library Location and Naming During Build” (Simulink Coder)
- “Customize Template Makefiles” (Simulink Coder)
- “Precompile S-Function Libraries” (Simulink Coder)

Initialization Cost

Initialization occurs at the beginning of simulation when simulation time, *t*, is at the model Start Time. During the first call to ModelOutputs, Simscape performs model initialization. The solver determines the simulation starting point by iteratively computing the initial values for all system variables. Finding a solution that satisfies the model equations for a nonlinear system can take more time than the time step allows. If the computation time exceeds the time step, a central processing unit (CPU) overload occurs. Real-time processors typically respond to CPU overloads by terminating model execution.

If your real-time application terminates when *t* is at the model Start Time, disable the automatic shutdown response on your real-time hardware to CPU overloads for that time period. To determine how to disable such processes, check with your hardware vendor.

See Also

More About

- “How Simscape Simulation Works” on page 6-7

Generate HDL Code Using the Simscape HDL Workflow Advisor

If you have a license for HDL Coder, you can generate HDL code from your Simscape model for deployment onto FPGA platforms using the Simscape HDL Workflow Advisor. The Simscape HDL Workflow Advisor first helps you to convert your Simscape model to a Simulink implementation. It then converts the Simulink model to HDL code using HDL Coder. Converting your Simscape model to HDL code allows you to:

- Accelerate the simulation of physical systems by using an optimized implementation of Simscape models
- Use the reconfigurability and parallelism capabilities of the FPGA for rapid prototyping
- Simulate the HDL implementation in real time using hardware-in-the-loop (HIL) simulation

Simscape HDL Workflow Advisor Steps

The general workflow for converting a Simscape model to HDL code using the Simscape HDL Workflow Advisor consists of these steps.

1. Generate baseline results for your Simscape model.
2. Ensure that the model contains only linear or switched linear blocks by using the `simscape.findNonLinearBlocks` function. This function detects the nonlinear blocks in your Simscape model. The function returns the number and type of networks, that is, linear, switched linear, or nonlinear, based on the blocks that the network contains. The function also returns the names of any blocks that yield nonlinear equations. Update or replace any blocks that yield nonlinear equations.
3. Ensure that simulation results of the model, which now contains no blocks that yield nonlinear equations, match the baseline results.
4. Configure the Simscape network for real-time simulation and HDL code generation:
    a. Add blocks that allow you to monitor the progress of the HDL workflow in terms of simulation time.
    b. Display sample time information.
    c. Configure the Simscape network for fixed-step, fixed cost simulation.

5. Ensure that simulation results of the discrete model match the baseline results.
6. Run the Simscape HDL Workflow Advisor tasks by using the `sschdladvisor` function. The `sschdladvisor` function:
   - Checks for HDL code generation compatibility by ensuring that the model is switched linear and is configured for real-time simulation.
   - Extracts state-space coefficients for the Simscape network.
   - Generates an HDL code generation compatible implementation of the Simscape network.
7. Ensure that simulation results from the HDL code generation compatible implementation match the baseline results.
8. Generate HDL code:
   a. Configure the Simulink model for HDL code generation by running the `hdlsetup` function. The `hdlsetup` function configures the fixed-step solver for HDL code generation compatibility and specifies the simulation start and stop times.
   b. Save the model parameters and validation model generation settings.
   c. Generate code using the `makehdl` function.

Limitations

The Simscape HDL Workflow Advisor does not work for Simscape networks that contain:

- Events.
- Mode charts.
- Delays.
- Enabled runtime parameters.
- Periodic sources.
- Nonlinearities that result from network connectivity. If your model does contain a nonlinearity of this sort, the `sschdladvisor` function may run all tasks to completion, but generates a zero-value output.

Generate HDL Code for a Simscape Model Using the Simscape HDL Workflow Advisor

This example shows how to convert your Simscape model to HDL code using the Simscape HDL Workflow Advisor. To learn how to configure your Simscape network and Simulink model for real-time simulation and HDL code generation, see “Model Preparation” on page 10-187. To open a version of the model that is already prepared for using the Simscape HDL Workflow Advisor, see “Generate HDL Code by Using the Simscape HDL Workflow Advisor” on page 10-195.

Model Preparation

To prepare your Simscape model for conversion to HDL code for FPGA deployment:

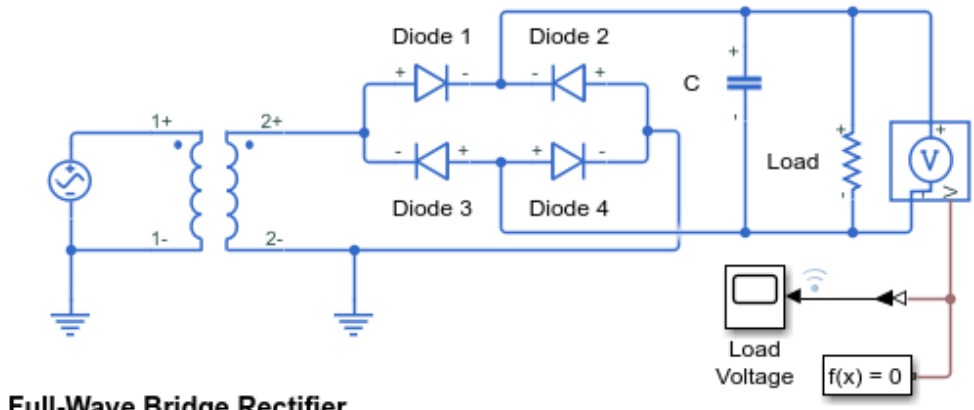
1. Open the model and show hidden block names. At the MATLAB command prompt, enter

   ```
   baselineModel = 'ssc_bridge_rectifier';
   load_system(baselineModel)
   set_param(baselineModel,'HideAutomaticNames','off')
   open_system(baselineModel)
   ```

2. To compare the baseline simulation results to results from modified versions of the model later, remove the data point limitation on the **Load Voltage** scope block and enable data logging to the Simulation Data Inspector for the signal that inputs data to the scope.

   a. In the configuration parameters for the scope, for the **Logging** parameters, clear the option to limit data points.

   b. Right-click the connection line to the **Load Voltage** scope block and select `Log selected signals`.

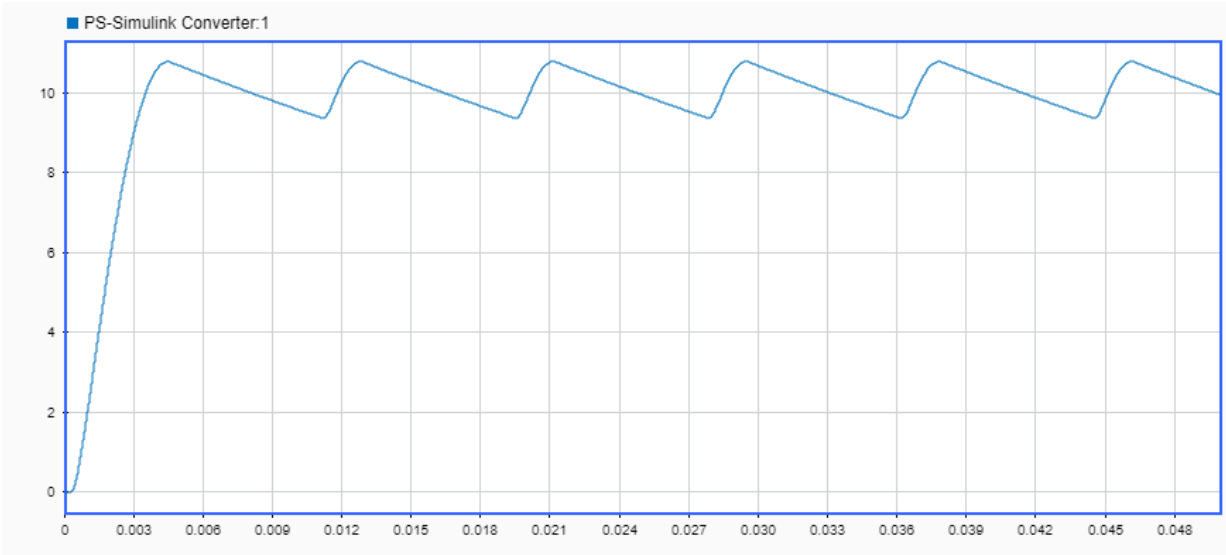


The logging badge marks the signal in the model.

3 Simulate the model and view the results in the Simulation Data Inspector.

```
%% Simulate baseline model
sim(baselineModel)

%% Get Simulation Data Inspector run IDs for
runIDs = Simulink.sdi.getAllRunIDs;
runID = runIDs(end);
run = Simulink.sdi.getRun(runID);
signal1 = run.getSignalByIndex(1);
% run.signalCount
signal1.checked = true;
Simulink.sdi.view
```

As needed, press the spacebar on your keyboard to fit the Simulation Data Inspector plot to view.



The baseline simulation results are as expected for the full-wave bridge rectifier load voltage.

4 The Simscape HDL Workflow Advisor cannot convert nonlinear networks to HDL Code. Before running the advisor, identify and replace blocks that cause your network to be nonlinear. To identify the blocks, use the `simcape.findNonlearBlocks` function.

```
simscape.findNonlinearBlocks(baselineModel)
```

```
Found network that contains nonlinear equations in the following blocks:
    'ssc_bridge_rectifier/AC Voltage Source'

The number of linear or switched linear networks in the model is 0.
The number of nonlinear networks in the model is 1.

ans =

  1×1 cell array

    {'ssc_bridge_rectifier/AC Voltage Source'}
```

The model contains an AC Voltage Source block, a periodic source that yields nonlinear equations.

5 You can replace the Simscape periodic source by adding a Simscape Controlled Voltage Source block in the Simscape network with a Simulink Sine Wave block

outside the network. An added benefit is that you can configure the frequency and amplitude for the Sine Wave block at run time during real-time simulation.

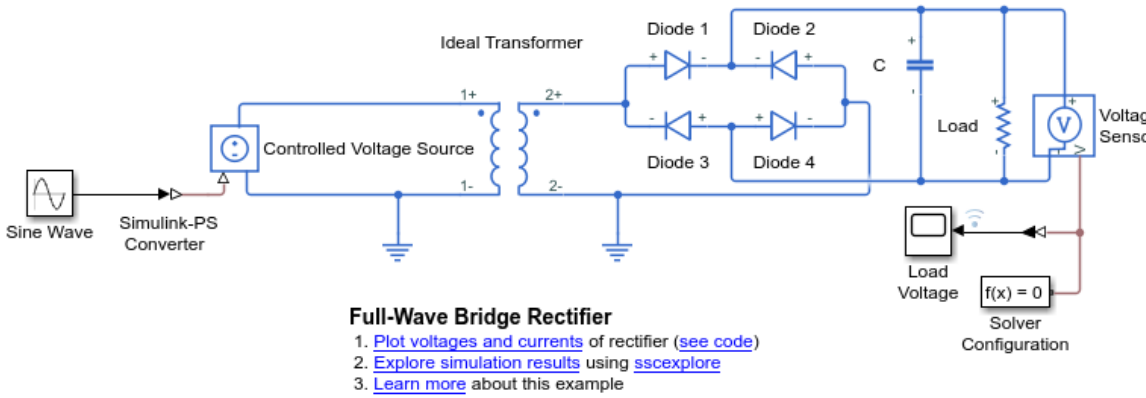
a Delete the AC Voltage Source block.

b Add a Sine Wave block from the **Simulink/Sources** library.

c Add a Simulink-PS Converter block from the Simscape **Utilities** library.

d Add a Controlled Voltage Source block from the Simscape **Foundation Library/Electrical/Electrical Sources** library.

e Connect the Sine Wave block to the Simulink-PS Converter block and the Simulink-PS Converter block to the Controlled Voltage Source block.



6 The Simulink model is configured for variable-step simulation. If you specify the Sine Wave block sample time as `-1`, for inheriting the sample time, the simulation generates a warning. Instead, specify the sample time for the Sine Wave as `0` by using a workspace variable that you can later adjust for improving simulation speed or accuracy. The removed AC Voltage Source block has a **Peak amplitude** of `sqrt(2)*120 V` and a **Frequency** of `60 Hz`.

Configure the Sine Wave block.

a Define the sample time in the workspace.

```
Ts = 1e-5;
```

b Set the **Amplitude** parameter to `sqrt(2)*120`.

c Set the **Frequency (rad/sec)** parameter to `60*2*pi`.

d Set the **Sample time** parameter to `Ts`.

7 Ensure that there are no blocks that cause your network to be nonlinear.

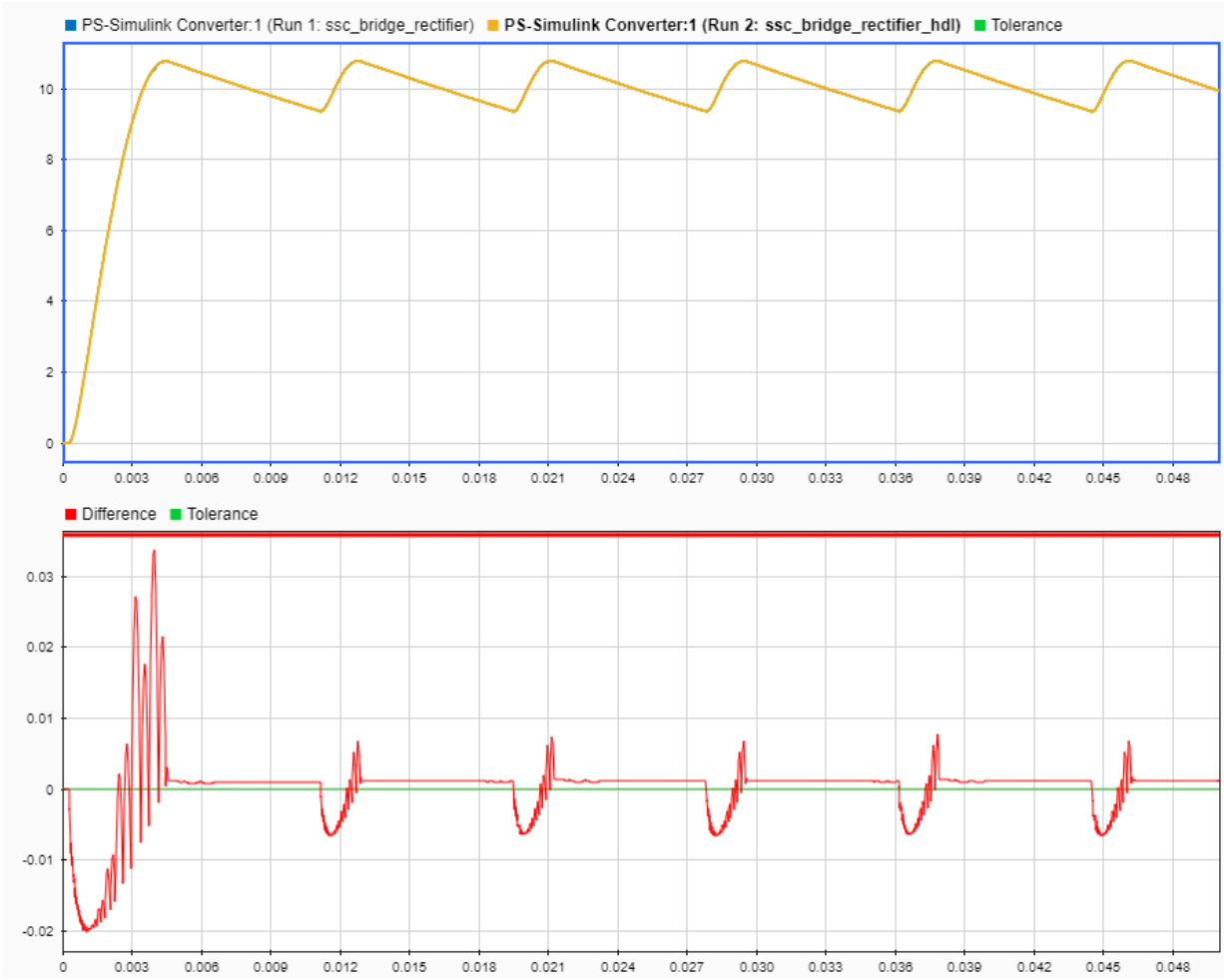
```
% Simulate
sim(baselineModel)

% Check for nonlinear blocks
simscape.findNonlinearBlocks(baselineModel)
```

```
The number of linear or switched linear networks in the model is 1.

ans =

  0×0 empty cell array
```

The model contains only blocks that yield linear or switched linear equations.

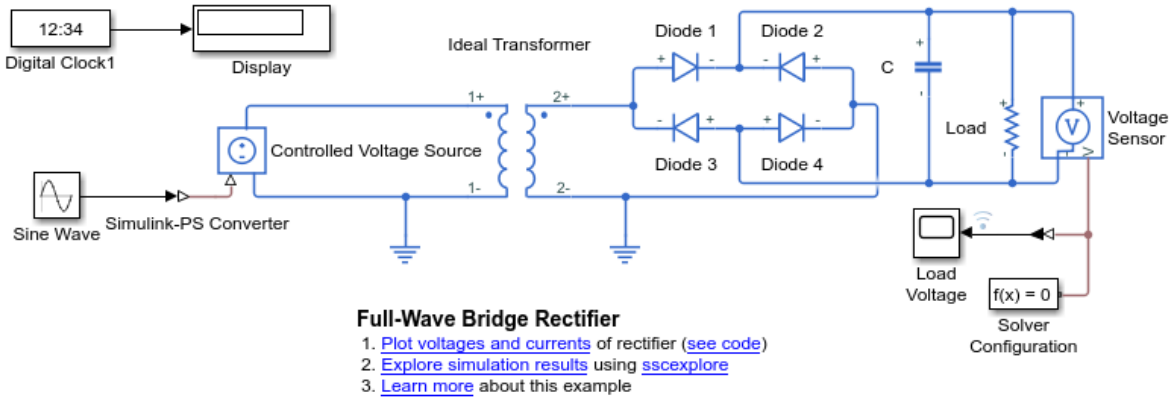
8 Simulate the model and compare the results to the baseline results in the Simulation Data Inspector.

```
% Get Simulation Data Inspector run IDs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 1);
runSwitchedLinear = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaseline1 = Simulink.sdi.compareRuns(runBaseline,...
    runSwitchedLinear);
```



The results are similar to the baseline results.

9 To examine the progress of the Simscape HDL Workflow Advisor later, add and connect a Digital Clock block from the **Simulink/Sources** library and a Display block from the **Simulink/Sinks** library, as shown in the figure. For the Digital Clock, specify `Ts` for **Sample time** parameter.



10 The model is currently simulating in continuous time using a variable-step solver. For real time-simulation, a fixed-step solver is required for discrete-time simulation. Sample time colors and annotations help you to determine if your model contains any continuous settings. To turn on sample time colors and annotations, in the Simulink model window, select **Display** > **Sample Time** > **All**.

The model diagram updates and the Sample Time Legend displays.

11 Configure the model for real-time simulation.

a Configure the Simulink model for fixed-step, fixed-cost simulation. In the Model Configuration Parameters, for the **Solver** parameters, set:

- Set **Type** to `Fixed-step`.
- Set **Solver** to `discrete (no continuous states)`.

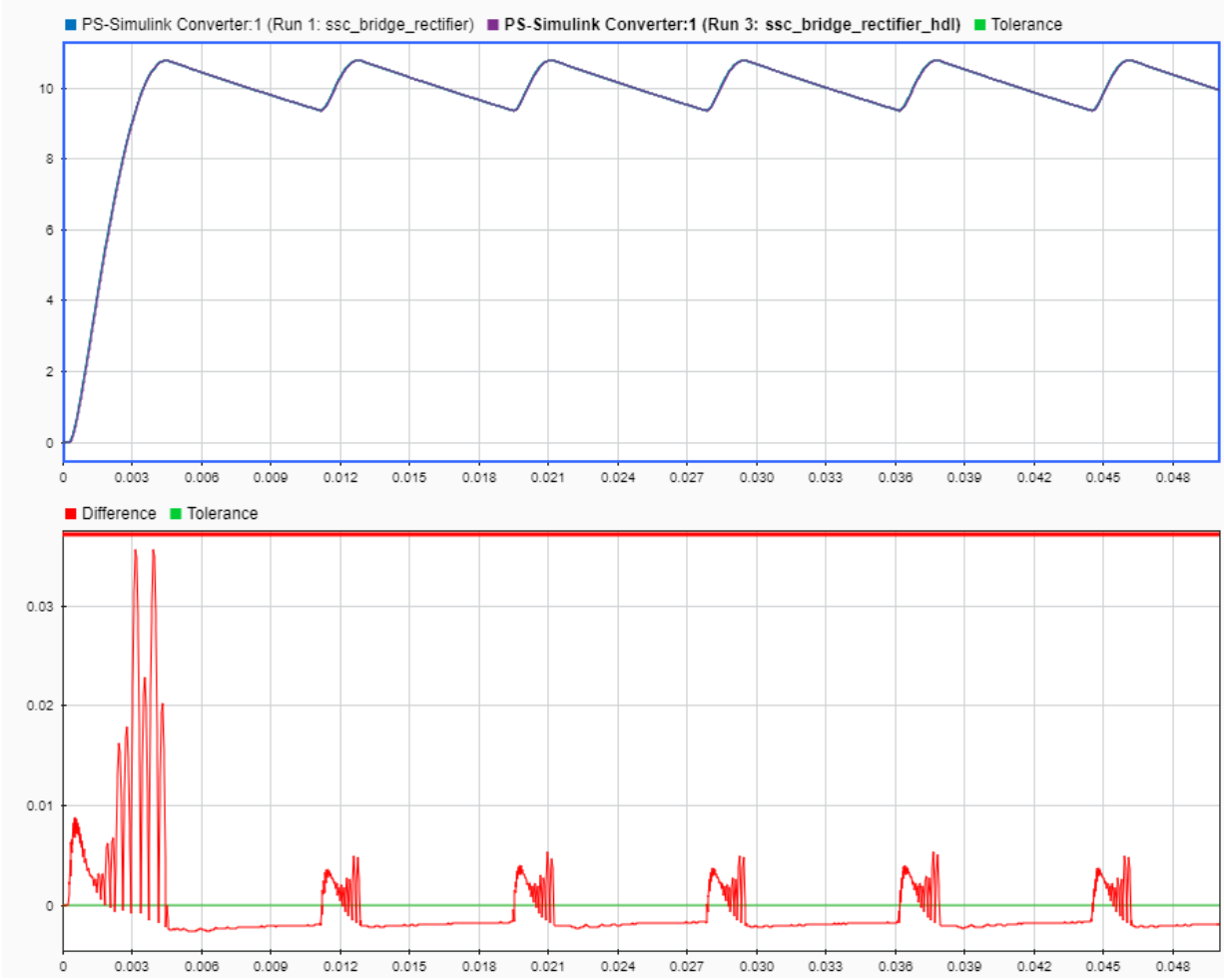
b Configure the Simscape network for fixed-step, fixed-cost simulation. For the Solver Configuration block:

- Select **Use local solver**.
- Ensure that **Solver type** is set to `Backward Euler`.
- Specify `Ts` for the **Sample time**.

12 Simulate the model and compare the results to the baseline results in the Simulation Data Inspector.

```
% Simulate
sim(baselineModel)
```

```
% Get Simulation Data Inspector run IDs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 2);
runRealTime = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaseline1 = Simulink.sdi.compareRuns(runBaseline,...
    runRealTime);
```



The results are similar to the baseline results.

Generate HDL Code by Using the Simscape HDL Workflow Advisor

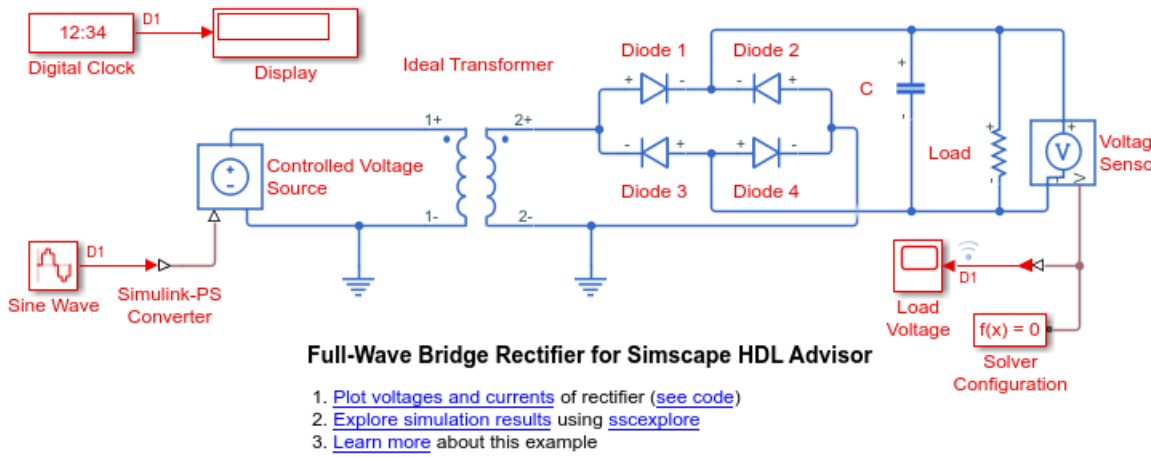
Generate HDL code by running the Simscape HDL Workflow Advisor either on the Simscape model that you prepared by stepping through “Model Preparation” on page 10-187 or on a Simscape model that is already prepared for code generation.

1 Rename the model.

- To continue working with the model that you prepared for HDL code generation, rename the model `ssc_model`.
- To open and use a model that is already prepared for HDL code generation, at the MATLAB command prompt, enter

  ```
  open_system('ssc_bridge_rectifier_hdl')
  ```

  Save the model to a local directory as `ssc_model`.



2 Run the Simscape HDL Workflow Advisor.

```
sschdladvisor('ssc_model')
```

The Simscape HDL Workflow Advisor opens.

3 Run the code generation compatibility checks.

a Select **Code generation compatibility checks > Check solver configuration** and then click **Run this task**.

b Select **^Check switched linear** and then click **Run this task**.

The advisor reports when your model has passed each of these checks.

4 Extract the state-space coefficients. Select **State-space conversion > Get state-space parameters** and then click **Run this task**. The conversion can take some time. The Display block in the model window shows the elapsed simulation time.

After running the task, the advisor displays a summary of the state-space representation and a table of parameters.

- Number of states: 5
- Number of inputs: 1
- Number of outputs: 1
- Number of modes: 7
- Discrete sample time: 1e-05

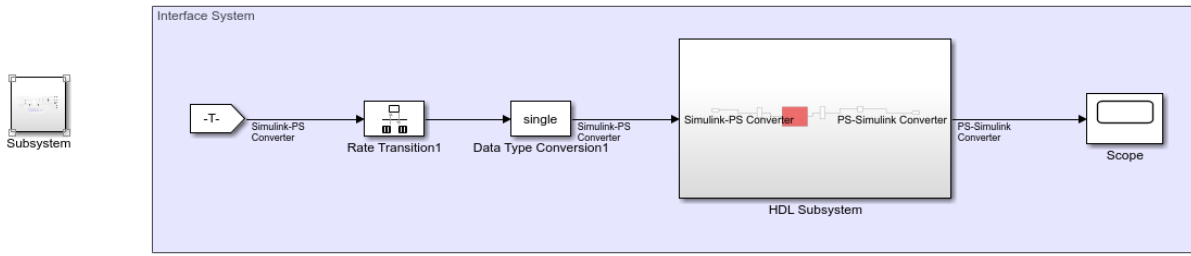
Parameter	Parameter size
A	5 x 5 x 7
B	5 x 1 x 7
F0	5 x 1 x 7
C	1 x 5 x 7
D	1 x 1 x 7
Y0	1 x 1 x 7

The size of the state, mode, and parameter data helps you estimate how much of the FPGA resources are required to deploy the model. The higher the values, the more FPGA resources are required. The input and output data indicate the number and type of I/O connections needed for real-time deployment and visualization.

5 Generate an HDL implementation of your model. Select **Implementation model generation > Generate implementation model** and then click **Run this task**.

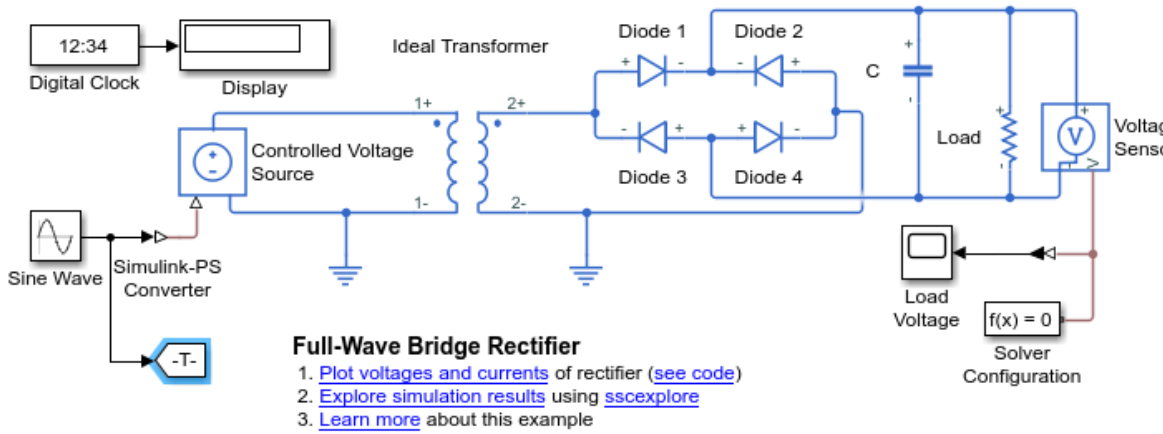
When the implementation model is generated, the advisor reports that the task is passed and displays a link to the generated implementation model, which is named **gmStateSpaceHDL_ssc_model**.

6 Open the generated implementation model by clicking the provided link.



The model contains blocks labeled:

- **Subsystem** — Simulink subsystem that contains the prepared model and any signal routing that the Simscape HDL Workflow Advisor adds. For this model, the advisor adds a Goto block that routes the **Sine Wave** block input to the **HDL Subsystem**.



- **T** — From block that routes the **Sine Wave** block input from the **Subsystem** block to the **HDL Subsystem** block.
- **Rate Transition1** — Handles the transfer of data between blocks operating at different rates.

- **Data Type Conversion1** — Converts double data types to single data types, as is required for HDL code generation.
- **HDL Subsystem** —Simulink subsystem that contains an HDL code generation compatible version of your Simscape network. For this model, the advisor adds a Goto block that routes the **Sine Wave** block input to the **HDL Subsystem** block.
- **Scope** — Displays the load voltage.

7 Prepare the **HDL Subsystem** for a simulation comparison to the baseline results:

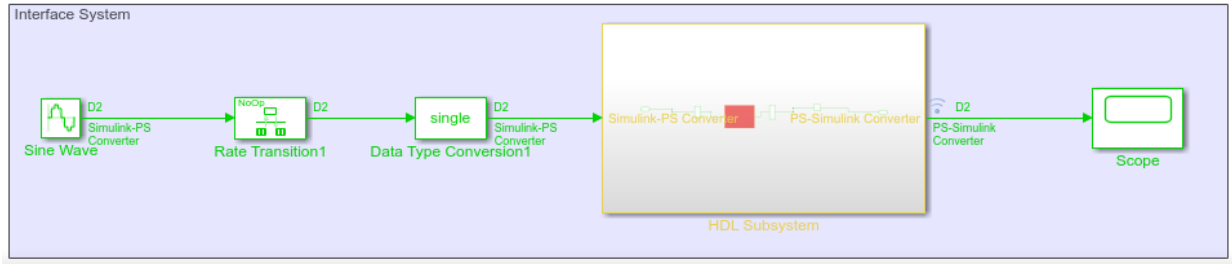
a Delete the **T** From block.

b Copy and paste the **Sine Wave** block from the **Subsystem** block into the top-level model.

c Connect the **Sine Wave** block to the **Rate Transition1** block.

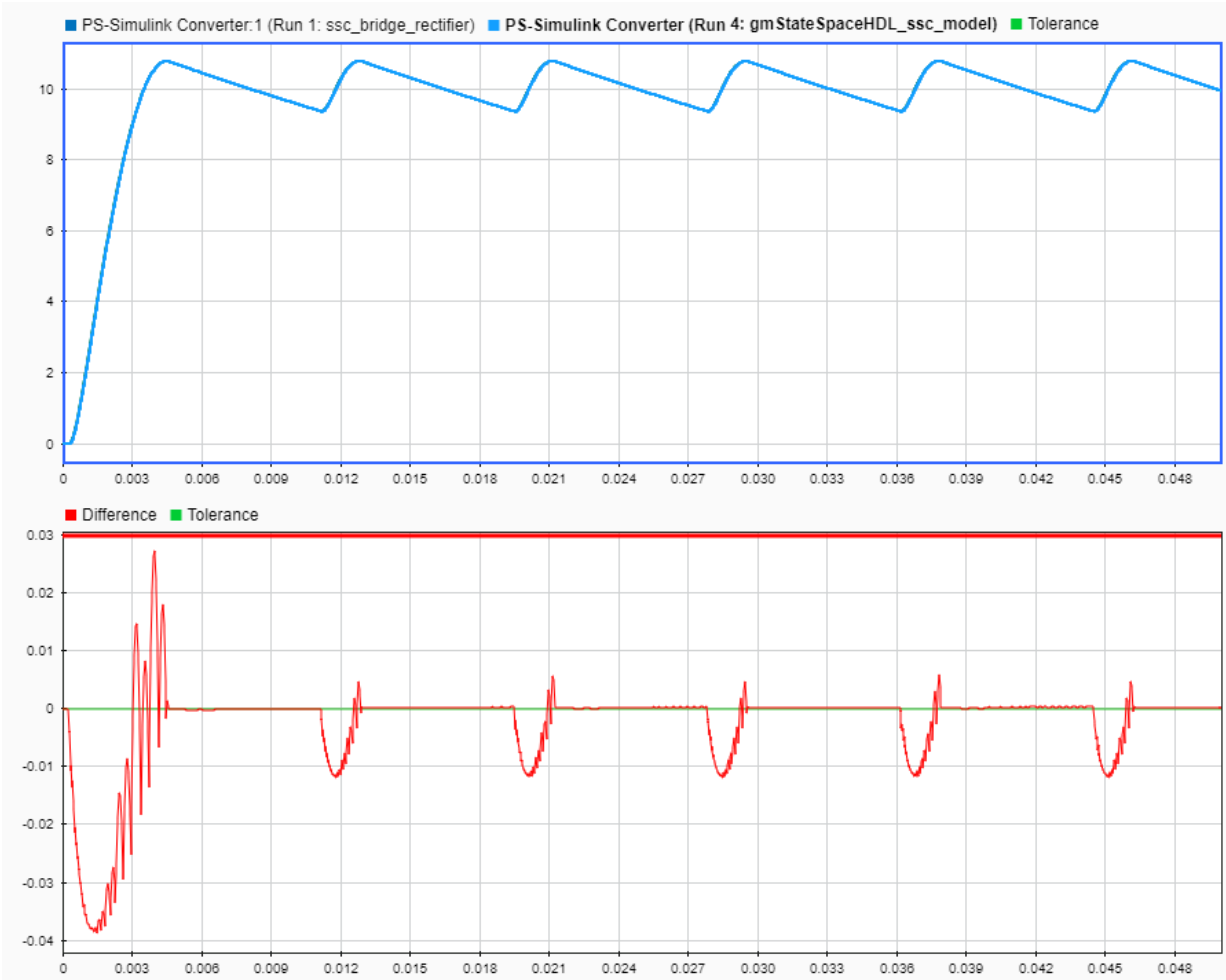
d Delete the **Subsystem** block, which contains the baseline model.

e Enable data logging to the Simulation Data Inspector for the signal that goes to the **Scope** block.



8 To ensure that the HDL subsystem corresponds to your original Simscape model, simulate the model and compare the results to the baseline simulation results.

```
% Simulate
sim('gmStateSpaceHDL_ssc_model')

% Get Simulation Data Inspector run IDs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 3);
runHDLImplementation = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaseline1 = Simulink.sdi.compareRuns(runBaseline,...
    runHDLImplementation);
```



The results are similar to the baseline results. Your Simscape model is now converted to an HDL code generation compatible implementation.

9 Generate HDL code from the implementation:

a In the model configuration parameters, for **HDL Code Generation Report**, select the **Generate traceability report** and the **Generate resource utilization report** options.

b Run the `hdlsetup` function.

```
hdlsetup('gmStateSpaceHDL_ssc_model')
```

c Save the model and subsystem parameter settings.

```
hdlsaveparams('gmStateSpaceHDL_ssc_model');
```

```
%% Set Model 'gmStateSpaceHDL_ssc_model' HDL parameters
hdlset_param('gmStateSpaceHDL_ssc_model', 'FloatingPointTargetConfiguration', ...
   hdlcoder.createFloatingPointTargetConfig('NativeFloatingPoint'));
hdlset_param('gmStateSpaceHDL_ssc_model', 'MaskParameterAsGeneric', 'on');
hdlset_param('gmStateSpaceHDL_ssc_model', 'Oversampling', 100);

% Set SubSystem HDL parameters
hdlset_param('gmStateSpaceHDL_ssc_model/HDL Subsystem', 'FlattenHierarchy', 'on');
```

d Save validation model generation settings.

```
HDLmodelname = 'gmStateSpaceHDL_ssc_model';
hdlset_param(HDLmodelname, 'GenerateValidationModel', 'on');
```

e Generate HDL code.

```
makehdl('gmStateSpaceHDL_ssc_model/HDL Subsystem')
```

```
### Generating HDL for 'gmStateSpaceHDL_ssc_model/HDL Subsystem'.
### Using the config set for model gmStateSpaceHDL_ssc_model ...
    for HDL code generation parameters.
### Starting HDL check.
### The code generation and optimization options you ...
    have chosen have introduced additional pipeline delays.
### The delay balancing feature has automatically inserted ...
    matching delays for compensation.
### The DUT requires an initial pipeline setup latency. ...
    Each output port experiences these additional delays.
### Output port 0: 1 cycles.
### Clock-rate pipelining results can be diagnosed by running this script:...
    C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\highlightClockRatePipelining.m
### One or more feedback loops in the model are inhibiting optimizations. ...
    To highlight these loops in your model, click the following MATLAB ...
    script: C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\highlightFeedbackLoop.m
### To clear highlighting, click the following MATLAB ...
    script: C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\clearhighlighting.m
### Generating new validation model: gm_gmStateSpaceHDL_ssc_model_vnl.
### Validation model generation complete.
### Begin VHDL Code Generation for 'gmStateSpaceHDL_ssc_model'.
### MESSAGE: The design requires 100 times faster clock ...
    with respect to the base rate = 2e-07.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/HDL Algorithm/Mode ...
    Selection/Mode Vector To Index/Subsystem1 ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\Subsystem1.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_mul_comp ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\nfp_mul_comp.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_add_comp ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\nfp_add_comp.vhd.
### Working on MultiplyAndAdd_2_entries ...
```

```
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\MultiplyAndAdd_2_entries.vhd.
### Working on MultiplyAndAdd_1_entries ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\MultiplyAndAdd_1_entries.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_uminus_comp ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\nfp_uminus_comp.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_relop_comp ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\nfp_relop_comp.vhd.
### Working on HDL Subsystem_tc ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\HDL_Subsystem_tc.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem ...
    as C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\HDL_Subsystem.vhd.
### Generating package file ...
    C:\Temp\hdlsrc\gmStateSpaceHDL_ssc_model\HDL_Subsystem_pkg.vhd.
### Generating HTML files for code generation report ...
    at gmStateSpaceHDL_ssc_model_codegen_rpt.html
### Creating HDL Code Generation Check Report HDL_Subsystem_report.html
### HDL check for 'gmStateSpaceHDL_ssc_model' complete ...
    with 0 errors, 0 warnings, and 3 messages.
### HDL code generation complete.
```

The HDL code generation report opens and includes any generated errors or warnings. The report includes a link to the resource utilization report, which describes the resource requirements for FPGA deployment.

The generated HDL code and validation model are saved in the **hdlsrc \gmStateSpaceHDL_ssc_model\html** directory. The generated code is saved as **HDL_Subsystem_tc.vhd**.

To generate HDL code for deployment to a specified target, use the HDL Workflow Advisor.

See Also

Blocks
Digital Clock | Display | Sine Wave

Simscape Blocks
Controlled Voltage Source | Simulink-PS Converter

Functions
`hdladvisor` | `hdlsaveparams` | `hdlset_param` | `hdlsetup` | `makehdl` | `simscape.findNonlinearBlocks` | `sschdladvisor`

Related Examples

- “View Sample Time Information” (Simulink)
- “Generate HDL Code from Simscape Models” (HDL Coder)
- “Deploy Simscape™ Plant Models to Speedgoat FPGA I/O Modules” (HDL Coder)
- “Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model” (HDL Coder)
- “Troubleshoot Conversion of Simscape™ Permanent Magnet Synchronous Motor to HDL-Compatible Simulink Model” (HDL Coder)
- “View Data with the Simulation Data Inspector” (Simulink)

More About

- “Getting Started with the HDL Workflow Advisor” (HDL Coder)
- “Create and Use Code Generation Reports” (HDL Coder)
- “Hardware-In-The-Loop Simulation Workflow” on page 10-142
- “Real-Time Model Preparation Workflow” on page 10-6
- “Real-Time Simulation Workflow” on page 10-98

Code Generation

- “About Code Generation from Simscape Models” on page 11-2
- “Reasons for Generating Code” on page 11-3
- “Using Code-Related Products and Features” on page 11-4
- “How Simscape Code Generation Differs from Simulink” on page 11-5

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 11-5, as well as “Limitations” on page 6-52.

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 11-5
“Compiler and Processor Architecture Requirements” on page 11-5
“Precompiled Libraries Provided for Selected Compilers” on page 11-5
“Simscape Code Reuse Not Supported” on page 11-6
“Tunable Parameters Not Supported” on page 11-6
“Simscape Run-Time Parameter Inlining Override of Global Exceptions” on page 11-6

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

https://www.mathworks.com/support/compilers/current_release

Precompiled Libraries Provided for Selected Compilers

Simscape software and its add-on products provide static runtime libraries precompiled for compilers supported by Simulink Coder software. For details, see

https://www.mathworks.com/support/compilers/current_release

For all other compilers, the static runtime libraries needed by code generated from Simscape models are compiled once per model during the code generation build process.

Simscape Code Reuse Not Supported

Reusable subsystems in Simulink reuse code that is generated once from the subsystem. You cannot generate reusable code from subsystems containing Simscape blocks.

Tunable Parameters Not Supported

A tunable parameter is a Simulink run-time parameter that you can change while the simulation is running. Simscape blocks do not support tunable parameters in either simulations or generated code.

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 12-2
- “Enable Data Logging for the Whole Model” on page 12-5
- “Log Data for Selected Blocks Only” on page 12-6
- “Data Logging Options” on page 12-7
- “Log and Plot Simulation Data” on page 12-9
- “Log Simulation Statistics” on page 12-15
- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “About the Simscape Results Explorer” on page 12-30
- “Plot Simulation Data in Different Units” on page 12-39
- “Use Custom Units to Plot Simulation Data” on page 12-45
- “View Sparkline Plots of Simulation Data” on page 12-49
- “Stream Logging Data to Disk” on page 12-56
- “Saving and Retrieving Logged Simulation Data” on page 12-60

About Simulation Data Logging

In this section...
“Suggested Workflows” on page 12-2 “Limitations” on page 12-3

Suggested Workflows

You can log simulation data to the workspace, or to a temporary file on disk, for debugging and verification. Data logging lets you analyze how internal block variables change with time during simulation. For example, you might want to see that the pressure in a hydraulic cylinder is above some minimum value or compare it against the pump pressure. If you log simulation data, you can later query, plot, and analyze it without rerunning the simulation.

There are two methods of simulation data logging: you can store the data directly in a workspace variable, or you can stream data to a temporary file on disk and have the workspace variable point to that temporary file. For more information on the second method, see “Stream Logging Data to Disk” on page 12-56. In either case, you interact with the logged simulation data through the simulation log variable.

Simulation data logging can replace connecting sensors and scopes to track simulation data. These blocks increase model complexity and slow down simulation. “Log and Plot Simulation Data” on page 12-9 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 12-6.

After running the simulation, you can use the Simscape Results Explorer tool to navigate and plot the data logging results.

For additional information on how you can query, plot, and analyze data by accessing the simulation log variable, see the reference pages for the 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. This way, you can view and analyze the data while the simulation is running. Set up your model to log simulation data, either for the whole model or on a block-by-block basis. Enable data streaming by selecting the **Record data in Simulation Data Inspector** check box on the **Simscape** pane of the Configuration Parameters dialog box. When you simulate the model, as soon as the streamed data becomes available, the Simulation Data Inspector button in the model toolbar highlights. Open the Simulation Data Inspector to view the data during simulation and to compare data for different simulation runs. For detailed information on how to configure and use the Simulation Data Inspector, see “Inspect and Analyze Simulation Results” (Simulink).

To make your model simulation and data logging compatible with the `parfor` command, select the **Single simulation output** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box. In this case, Simscape log data will be part of the single output object instead of being stored as a separate workspace variable. For more information, see “Single simulation output” (Simulink).

Limitations

Simulation data logging is not supported for:

- Model reference
- Generated code
- Accelerator mode
- Rapid accelerator mode
- Partitioning local solver

If you use the `sim` command with a `'StopTime'` name-value pair, the Simscape logging results are not updated.

See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Log and Plot Simulation Data” on page 12-9
- “Log Simulation Statistics” on page 12-15

More About

- “Data Logging Options” on page 12-7
- “Stream Logging Data to Disk” on page 12-56

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 12-7.

See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Log and Plot Simulation Data” on page 12-9
- “Log Simulation Statistics” on page 12-15

More About

- “Data Logging Options” on page 12-7

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 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 it and select **Simscape** > **Log simulation data** again to remove the check mark.

If you set the **Log simulation data** parameter to `All`, the simulation log will contain data from the whole model, regardless of the block selections. Setting the **Log simulation data** parameter to `None` disables data logging for the whole model.

See Also

Related Examples

- “Log and View Simulation Data for Selected Blocks” on page 12-20

More About

- “Data Logging Options” on page 12-7

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 12-15.
- **Record data in Simulation Data Inspector** — Use this check box if you want to stream the logged simulation data to Simulation Data Inspector, to be able to view it during simulation. By default, this check box is not selected. For more information on how to configure and use the Simulation Data Inspector, see “Inspect and Analyze Simulation Results” (Simulink).
- **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 12-30.

  If the **Record data in Simulation Data Inspector** check box is also selected, then the Simulation Data Inspector opens instead of the Simscape Results Explorer.
- **Workspace variable name** — Specifies the name of the workspace variable that stores the simulation data. Subsequent simulations overwrite the data in the simulation log variable. If you want to compare data from two models or two simulation runs, use different names for the respective log variables. The default variable name is `simlog`.
- **Decimation** — Use this parameter to limit the number of data points saved, by outputting data points for every *n*th time step, where *n* is the decimation factor. The default is 1, which means that all points are logged. Specifying a different value results in the first step, and every *n*th step thereafter, being logged. For example, specifying `2` logs data points for every other time step, while specifying `10` logs data points for just one in ten steps.
- **Limit data points** — Use this check box in conjunction with the **Data history (last N steps)** parameter to limit the number of data points saved. The check box is selected by default. If you clear it, the simulation log variable contains the data points for the whole simulation, at the price of slower simulation speed and heavier memory consumption.
- **Data history (last N steps)** — Specify the number of simulation steps to limit the number of data points output to the workspace. The simulation log variable contains

the data points corresponding to the last N steps of the simulation, where N is the value that you specify for the **Data history (last N steps)** parameter. You have to select the **Limit data points** check box to make this parameter available. The default value logs simulation data for the last 5000 steps. You can specify any other positive integer number. If the simulation contains fewer steps than the number specified, the simulation log variable contains the data points for the whole simulation.

Saving data to the workspace can slow down the simulation and consume memory. To avoid this, you can use either the **Decimation** parameter, or **Limit data points** in conjunction with **Data history (last N steps)**, or both methods, to limit the number of data points saved. The two methods work independently from each other and can be used separately or together. For example, if you specify a decimation factor of `2` and keep the default value of `5000` for the **Data history (last N steps)** parameter, your workspace variable will contain downsampled data from the last 10,000 time steps in the simulation.

Another way to reduce memory consumption is to enable data streaming to disk, as described in “Stream Logging Data to Disk” on page 12-56.

Note The **Output options** parameter, under **Additional parameters** on the **Data Import/Export** pane of the Configuration Parameters dialog box, also affects which data points are logged. For more information, see “Model Configuration Parameters: Data Import/Export” (Simulink).

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

See Also

Related Examples

- “Enable Data Logging for the Whole Model” on page 12-5
- “Log Data for Selected Blocks Only” on page 12-6

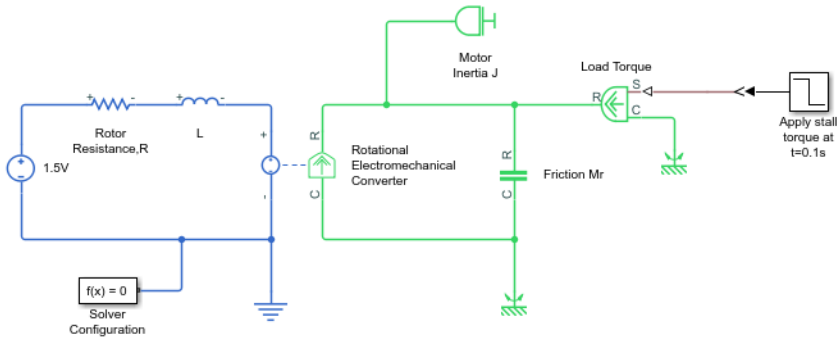
More About

- “About Simulation Data Logging” on page 12-2

Log and Plot Simulation Data

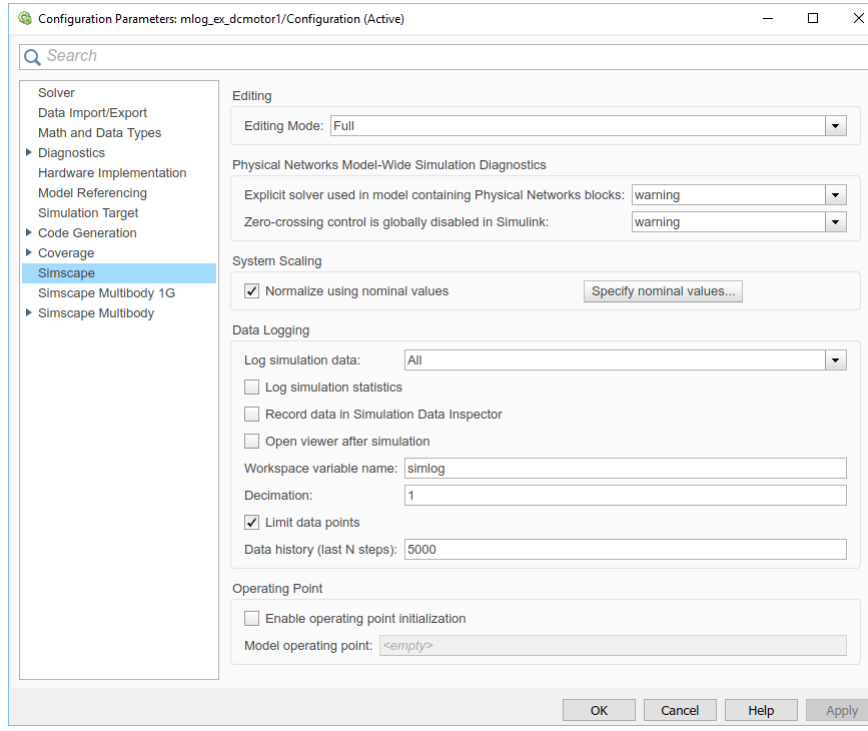
This example shows how you can log and plot simulation data instead of adding sensors to your model.

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. To display hidden block names, select **Display** and clear the **Hide Automatic Names** check box.
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
+-Friction_Mr
| +-C
| | +-w
| +-R
| | +-w
```

```
| +-t
| +-w
+-L
| +-i
| +-i_L
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-Load_Torque
| +-C
| | +-w
| +-R
| | +-w
| +-S
| +-t
| +-w
+-Mechanical_Rotational_Reference
| +-W
|   +-w
+-Mechanical_Rotational_Reference1
| +-W
|   +-w
+-Motor_Inertia_J
| +-I
| | +-w
| +-t
| +-w
+-Rotational_Electromechanical_Converter
| +-C
| | +-w
| +-R
| | +-w
| +-i
| +-n
| | +-v
| +-p
| | +-v
| +-t
| +-v
| +-w
+-Rotor_ResistanceR
| +-i
```

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

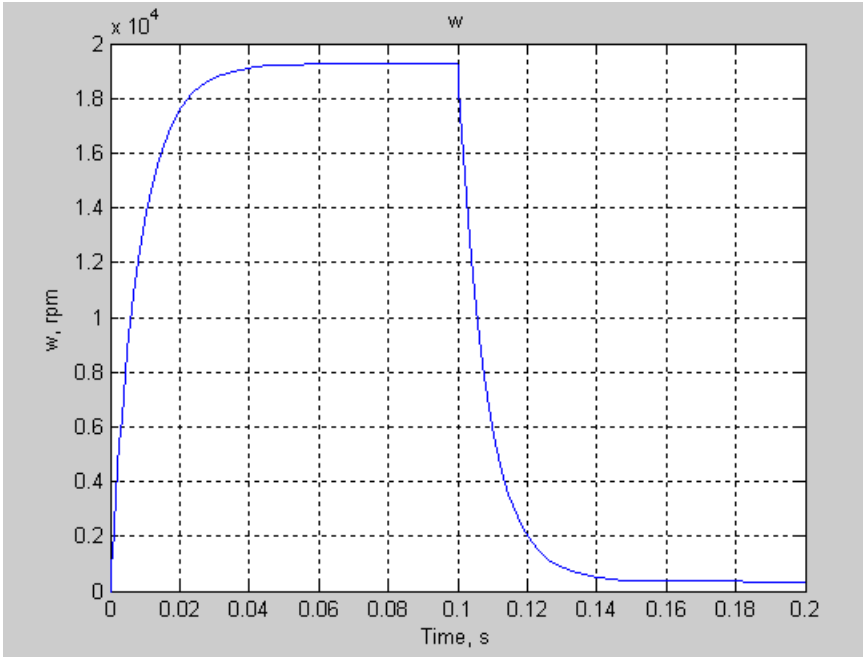
5. Every node that represents an Across, Through, or internal block variable contains series data. To get to the series, you have to specify the complete path to it through the tree, starting with the top-level variable name. For example, to get a handle on the series representing the angular velocity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

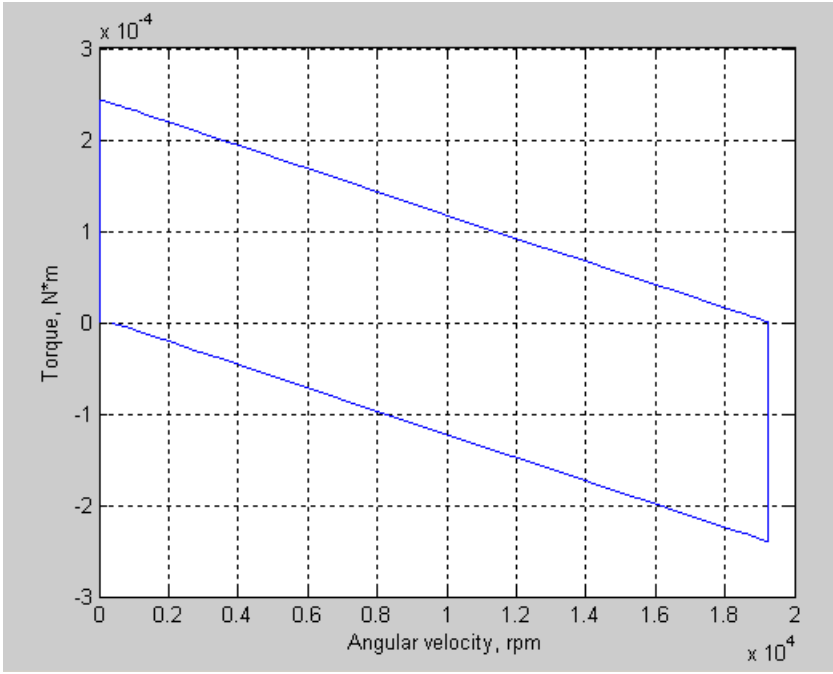
   ```
   plot(simlog.Rotational_Electromechanical_Converter.R.w,'units','rpm')
   ```



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')
```

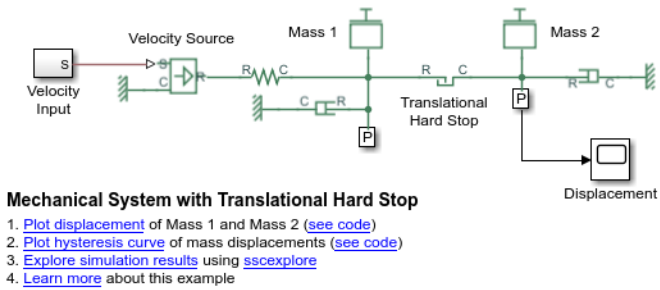


For more information on plotting logged simulation data, see the `simscape.logging.plot` and `simscape.logging.plotxy` reference pages.

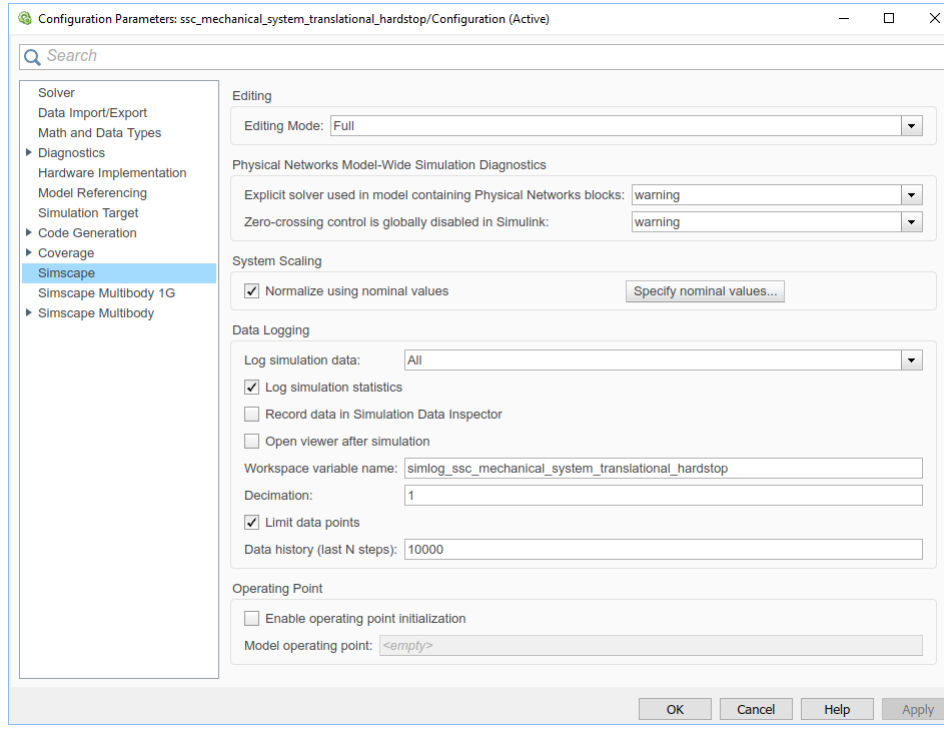
Log Simulation Statistics

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.

1. Open the Mechanical System with Translational Hard Stop example model by typing `ssc_mechanical_system_translational_hardstop` 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_mechanical_system_translational_hardstop`.



3 Simulate the model. This creates a workspace variable named `simlog_ssc_mechanical_system_translational_hardstop` (as specified by the **Workspace variable name** parameter), which contains the simulation data. Because you selected the **Log simulation statistics** check box, the workspace variable contains additional nodes that represent zero-crossing data.

4 The `simlog` variable has the same hierarchy as the model. To see the whole variable structure, at the command prompt, type:

```
simlog_ssc_mechanical_system_translational_hardstop.print
```

This command prints the whole data tree.

```
ssc_mechanical_system_translational_hardstop
+-Damper_M1
| +-C
| | +-v
| +-R
| | +-v
```

```
| +-f
| +-v
+-Damper_M2
| +-C
| | +-v
| +-R
| | +-v
| +-f
| +-v
+-MTRef_DM1
| +-V
|   +-v
+-MTRef_DM2
| +-V
|   +-v
+-MTRef_VS
| +-V
|   +-v
+-Mass_1
| +-M
| | +-v
| +-f
| +-v
+-Mass_2
| +-M
| | +-v
| +-f
| +-v
+-Sensor_M1
| +-Ideal_Translational_Motion_Sensor
| | +-C
| | | +-v
| | +-P
| | +-R
| | | +-v
| | +-V
| | +-f
| | +-v
| | +-x
| +-MTRef
| | +-V
| |   +-v
| +-PS_Terminator
| | +-I
```

```
| +-PS_Terminator1
|   +-I
+-Sensor_M2
| +-Ideal_Translational_Motion_Sensor
| | +-C
| | | +-v
| | +-P
| | +-R
| | | +-v
| | +-V
| | +-f
| | +-v
| | +-x
| +-MTRef
|   +-V
|     +-v
+-Spring_M1
| +-C
| | +-v
| +-R
| | +-v
| +-f
| +-v
| +-x
+-Translational_Hard_Stop
| +-C
| | +-v
| +-R
| | +-v
| +-SimulationStatistics
| | +-zc_1
| | | +-crossings
| | | +-values
| | +-zc_2
| |   +-crossings
| |   +-values
| +-f
| +-v
| +-x
+-Velocity_Source
  +-C
  | +-v
  +-R
  | +-v
```

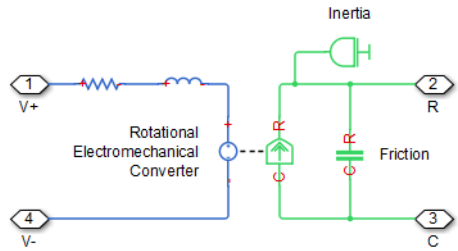
```
+-S
+-f
+-v
```

5. Under the `Translational_Hard_Stop` node, there is a node called `SimulationStatistics`, which contains zero-crossing information. This means that Translational Hard Stop is the only block in the model that can generate zero-crossings during simulation.
6. You can access and analyze this data similar to other data that is logged to workspace during simulation. For more information, see `simscape.logging.Node` and `simscape.logging.Series` reference pages.

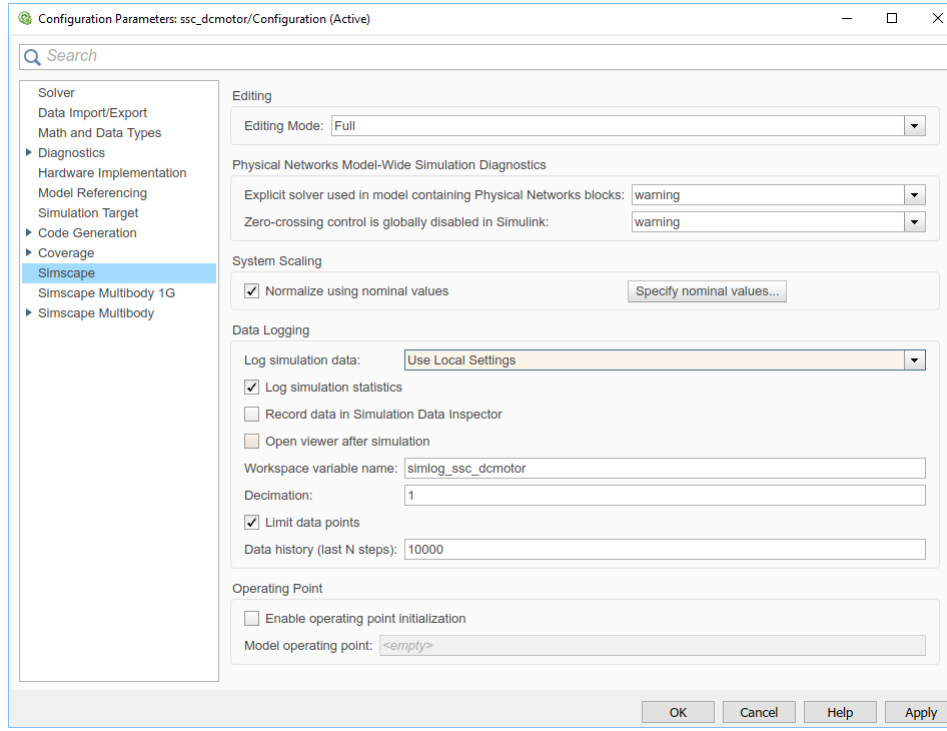
Log and View Simulation Data for Selected Blocks

This example shows how you can set your model to log simulation data for selected blocks only and how to view simulation data using Simscape Results Explorer.

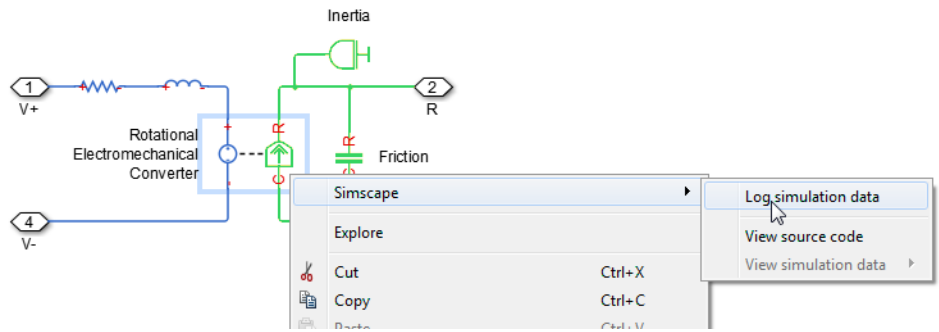
1. Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. Double-click the DC Motor subsystem to open it.



2. Open the Configuration Parameters dialog box and then, in the left pane, select **Simscape**. This example model has data logging for the whole model enabled. To enable data logging on a block-by-block basis, set the **Log simulation data** parameter to `Use local settings` and click **OK**.

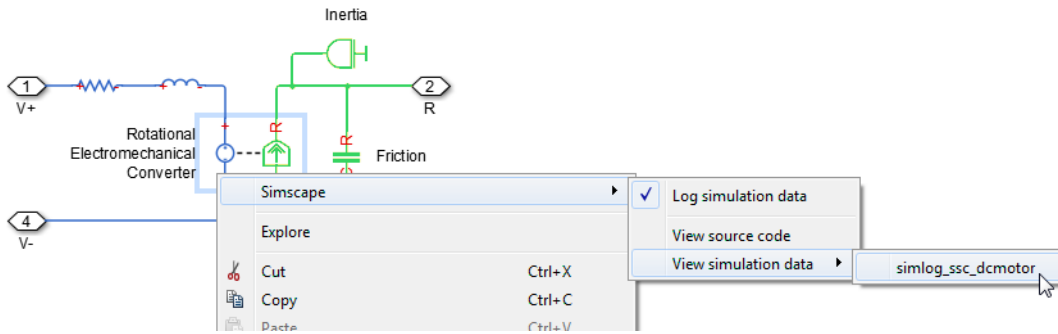


3 Select the blocks for data logging. Right-click the Rotational Electromechanical Converter block. From the context menu, select **Simscape** > **Log simulation data** .



After you select a block for data logging, a check mark appears in front of the **Log simulation data** option in the context menu for that block.

4. Right-click the Inertia block and select it for data logging, as described in the previous step.
5. Simulate the model. This creates a workspace variable named `simlog_ssc_dcmotor` (as specified by the **Workspace variable name** parameter), which contains the simulation data for selected blocks only.
6. To open the Simscape Results Explorer, right-click one of the blocks previously selected for data logging, for example, the Rotational Electromechanical Converter block. From the context menu, select **Simscape > View simulation data > simlog_ssc_dcmotor**.

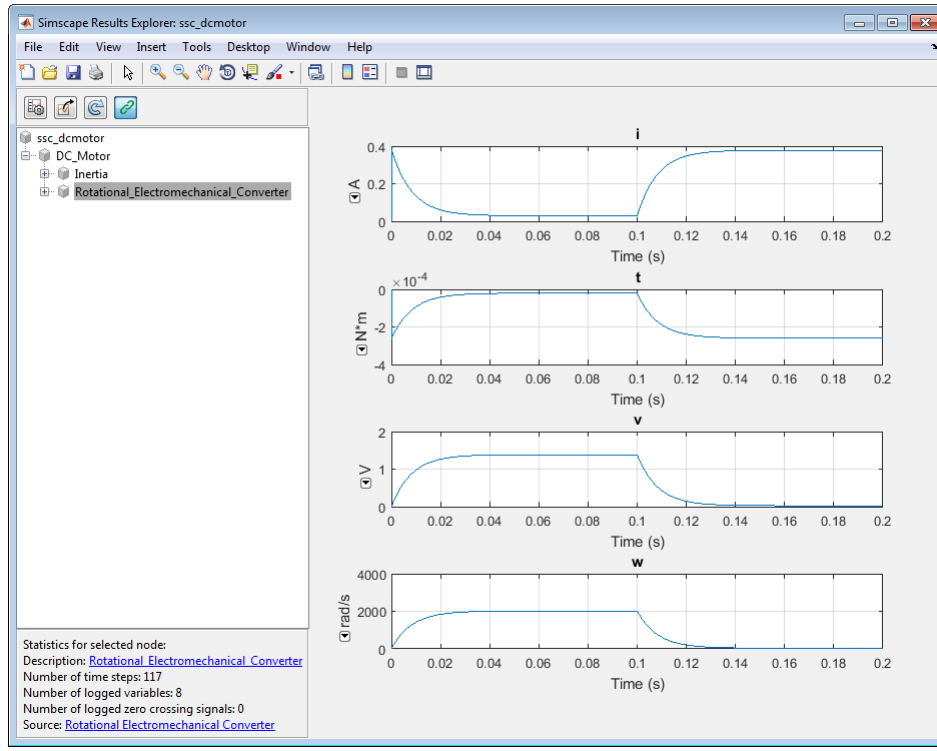


Note If you right-click a block that has not been selected for simulation data logging, for example, the Load Torque block, the **View simulation data** option is not available.

If you change the name of the log variable between simulation runs, the context menu lists the names of all the log variables associated with the block. For example, to compare data from two simulation runs, you can use different variable names (such as `simlog1` and `simlog2`). Open a Simscape Results Explorer window with `simlog1` results, then unlink it from the session and open another window with `simlog2` results. For more information, see “About the Simscape Results Explorer” on page 12-30.

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 Inertia.



See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Plot Simulation Data in Different Units” on page 12-39

More About

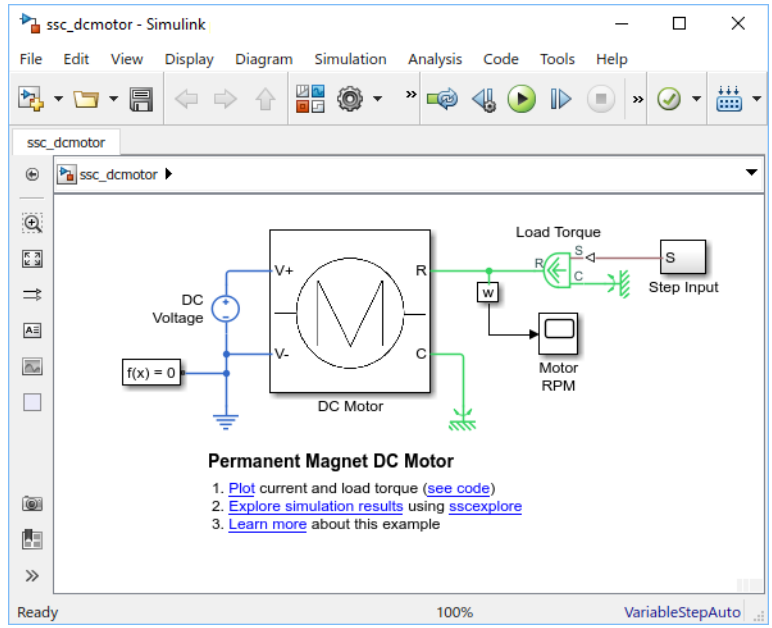
- “About Simulation Data Logging” on page 12-2

- “About the Simscape Results Explorer” on page 12-30

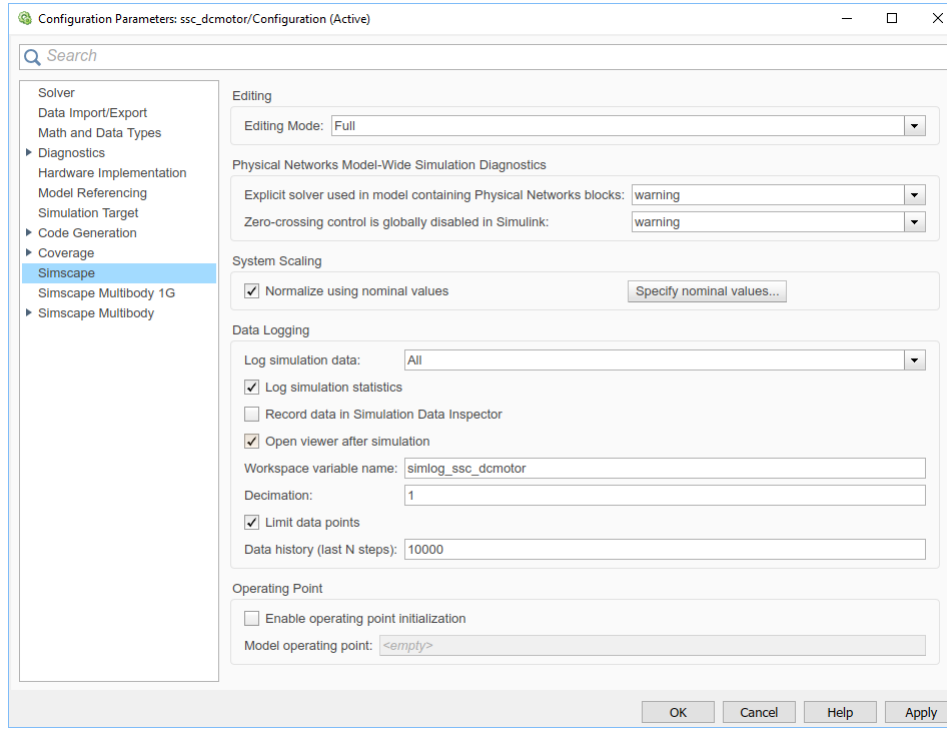
Log, Navigate, and Plot Simulation Data

This example shows the basic workflow for logging simulation data for the whole model and then navigating and plotting the logged data using Simscape Results Explorer.

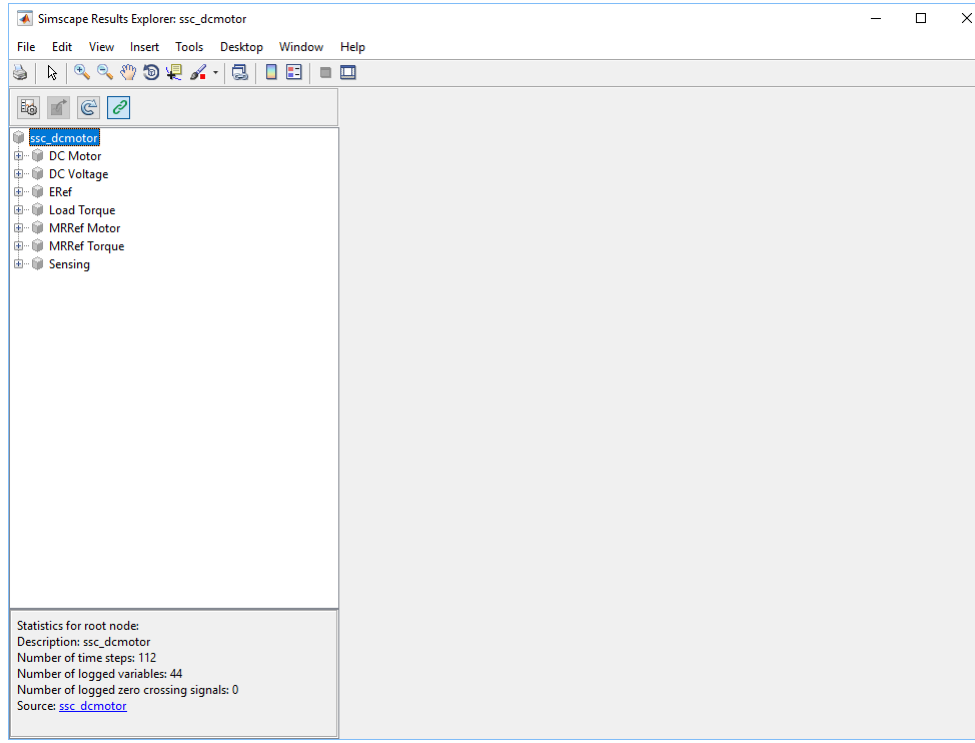
1 Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window.



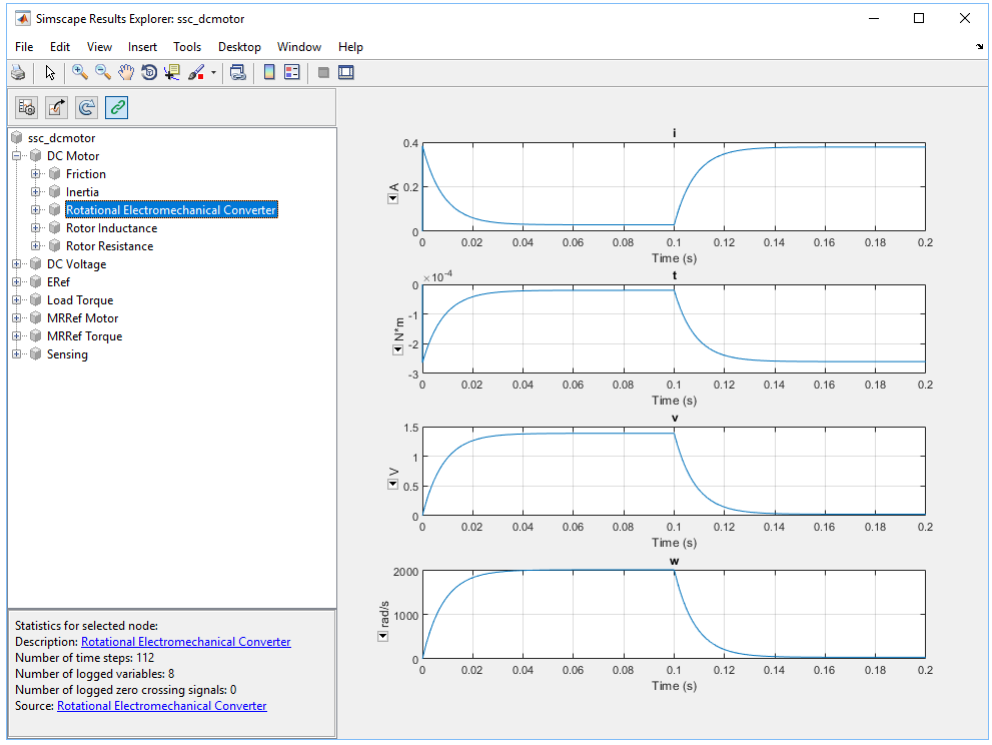
2 Open the Configuration Parameters dialog box and then, in the left pane, select **Simscape**. You can see that this example model already has data logging for the whole model enabled, as well as simulation statistics, and that the workspace variable name is `simlog_ssc_dcmotor`. Select the **Open viewer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



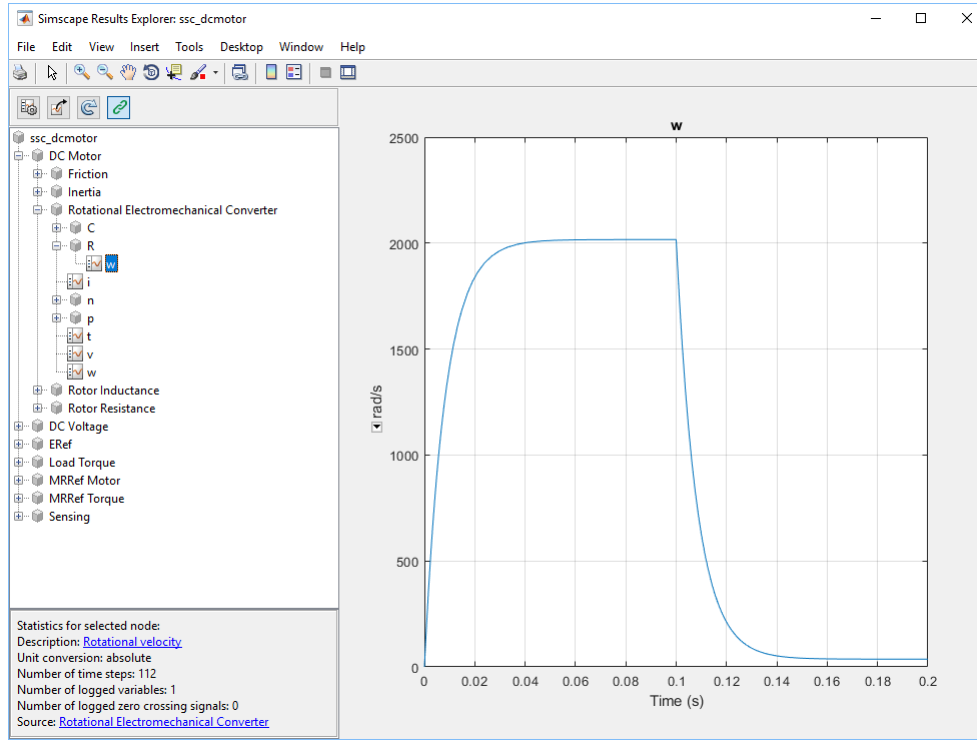
3 Simulate the model. When the simulation is done, the Simscape Results Explorer window opens. In the left pane, it contains the simulation log tree hierarchy, which corresponds to the model hierarchy.



4 When you click a node in the left pane, the corresponding plots appear in the right pane. Expand the `DC_Motor` node, and then click the `Rotational_Electromechanical_Converter` node to see all the node plots for this block.



5 To isolate the plot of the rotor angular velocity series against time, keep expanding the nodes in the left pane until you get to the series data.



See Also

Related Examples

- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Plot Simulation Data in Different Units” on page 12-39

More About

- “About Simulation Data Logging” on page 12-2
- “About the Simscape Results Explorer” on page 12-30

About the Simscape Results Explorer

In this section...
“Selecting Nodes to Plot Data” on page 12-30 “Link to MATLAB Session” on page 12-34 “Link to Block Diagram” on page 12-35

Simscape Results Explorer is an interactive tool that lets you navigate and plot the simulation data logging results.

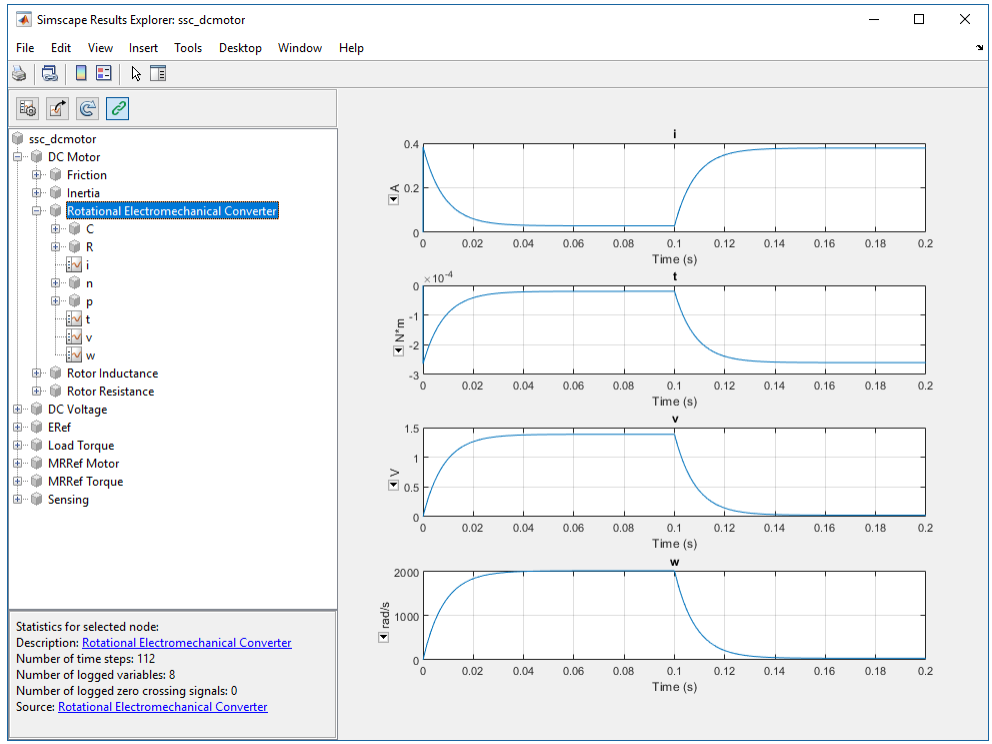
When you configure the model to log simulation data (for the whole model or just the selected blocks), you can make the Simscape Results Explorer window open automatically upon completing a simulation run by selecting the **Open viewer after simulation** check box in the Configuration Parameters dialog box. For more information on this workflow, see “Log, Navigate, and Plot Simulation Data” on page 12-25.

Another way to open the Simscape Results Explorer window is to right-click a block and, from the context menu, select **Simscape > View simulation data**. For more information, see “Log and View Simulation Data for Selected Blocks” on page 12-20.

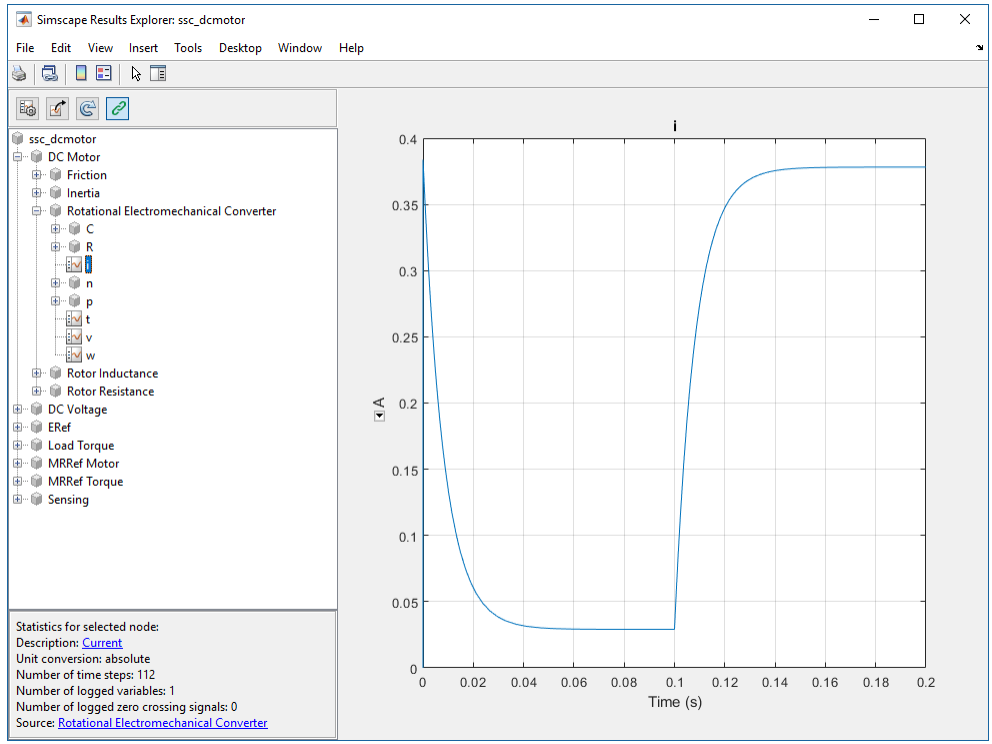
Selecting Nodes to Plot Data

When you click a node in the left pane of the Simscape Results Explorer, the corresponding plots appear in the right pane:

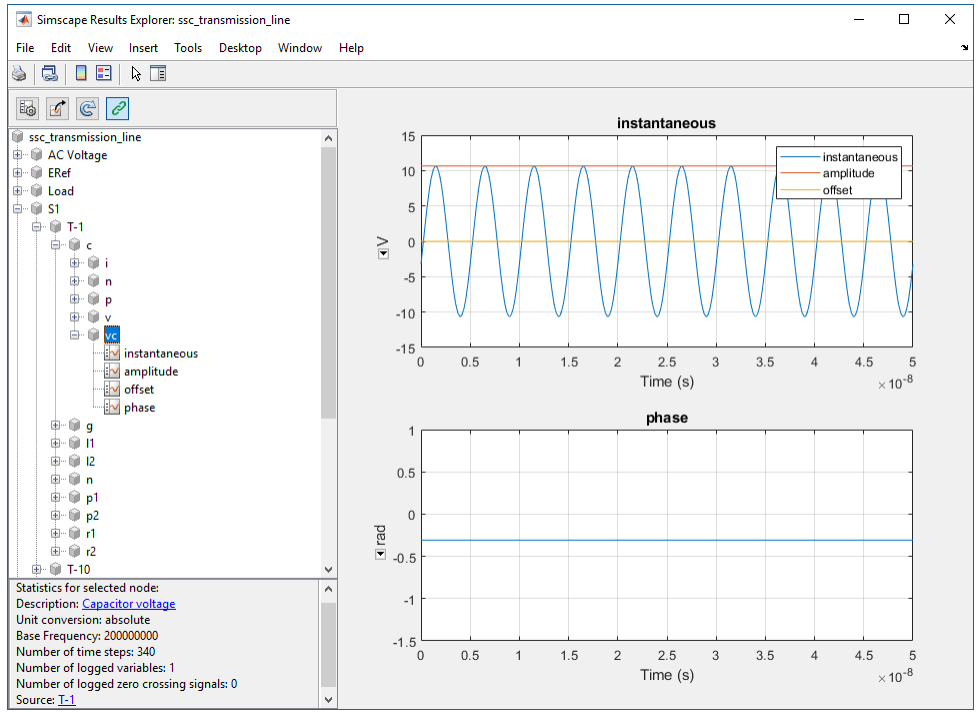
- Clicking a node that represents a block displays plots of all the variables in this block.



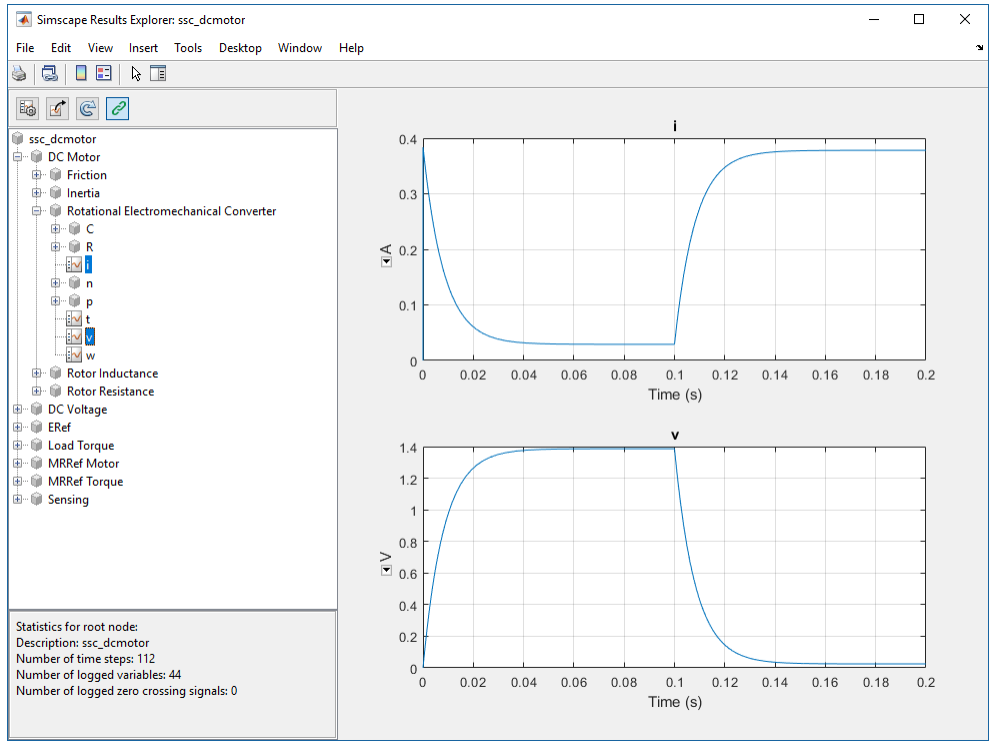
- Clicking a node that represents a variable displays the plot for this variable only.



- In frequency-and-time simulation mode, clicking a node that represents a frequency variable displays the plots for this variable's instantaneous value, amplitude, offset, and phase. You can click each of the subnodes to see that data separately. For more information, see “Frequency and Time Simulation Mode” on page 6-37.



- To select several variables for side-by-side plot comparison, press CTRL and click multiple nodes.



Link to MATLAB Session

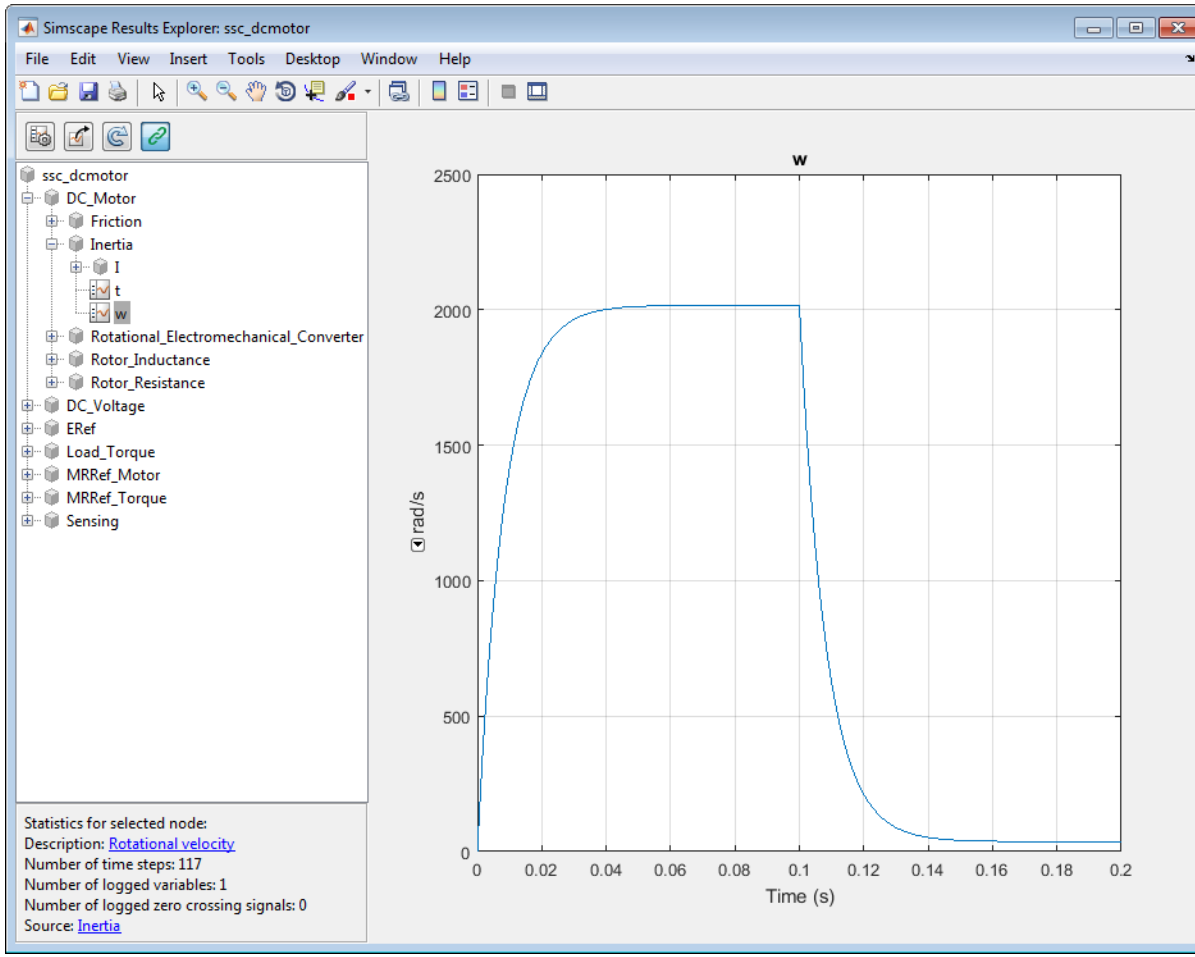
You can control whether the Simscape Results Explorer window is reused when you rerun the simulation, or a new window is opened after the next simulation run, by linking and unlinking the window.

When you first open the Simscape Results Explorer window, it is linked to the current MATLAB session. This means that when you run a new simulation, the results in the window will be overwritten. To retain the current results and open a new window after the next simulation, click the 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 its button) unlinks the previous one.

Link to Block Diagram

The Simscape Results Explorer tool provides direct linking to the block diagram. These links let you highlight the appropriate block or open the block dialog box, to easily go from a variable listed in the Simscape Results Explorer tree to the **Variables** tab in the corresponding block dialog box.

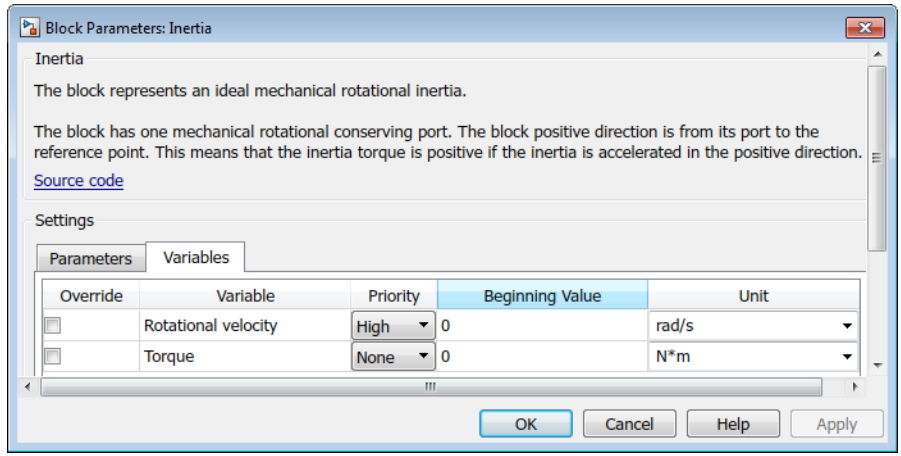


When you select a node in the Simscape Results Explorer tree, the status panel in the bottom-left corner of the window contains the following links:

- **Description** — If the node represents a block or subsystem, displays the block or subsystem name. If the node represents a variable, displays the descriptive variable name, which is the same name that appears on the **Variables** tab in the block dialog box. Clicking the link opens the corresponding block dialog box.

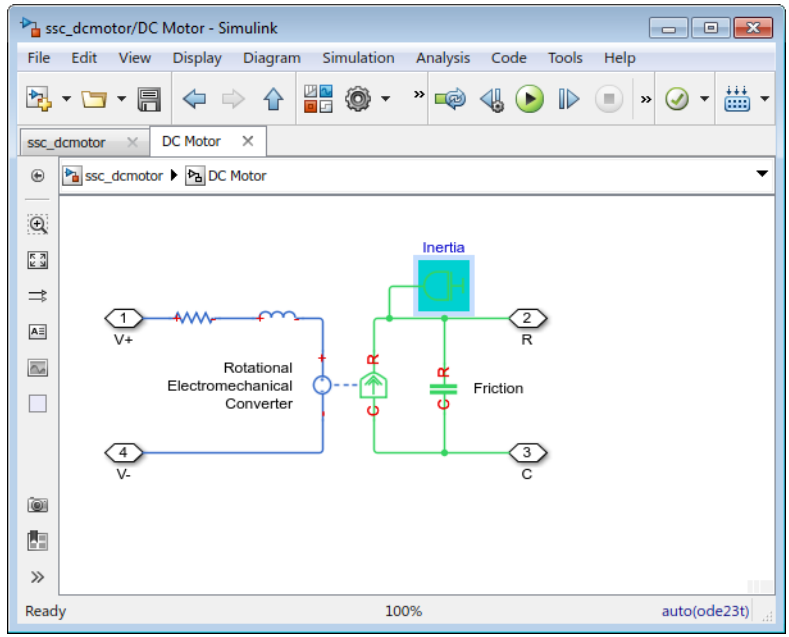
  For example, in the illustration above, the selected node `w` represents a variable called `Rotational velocity`. Clicking the **Description** link opens the Inertia block dialog

box, which is the parent block for this variable. In the block dialog box, click the **Variables** tab to see the **Rotational velocity** variable.



- **Source** — If the node represents a variable, displays the name of the parent block for this variable. Clicking the link highlights the corresponding block in the block diagram, opening the appropriate subsystem if needed.

  In the same example, clicking the **Source** link opens the DC Motor subsystem and highlights the Inertia block, which is the parent block for the selected node `w`.



Tip If the descriptive name of a variable or block is too long to fit into the status panel, it is truncated with an ellipsis (…). If you hover over the truncated name, the tooltip for the status panel displays the entire descriptive name.

See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Plot Simulation Data in Different Units” on page 12-39
- “Use Custom Units to Plot Simulation Data” on page 12-45

Plot Simulation Data in Different Units

When you display logged simulation data in Simscape Results Explorer, the data along the *x*-axis is always time, in seconds. However, you can change the *y*-axis units directly on the plot.

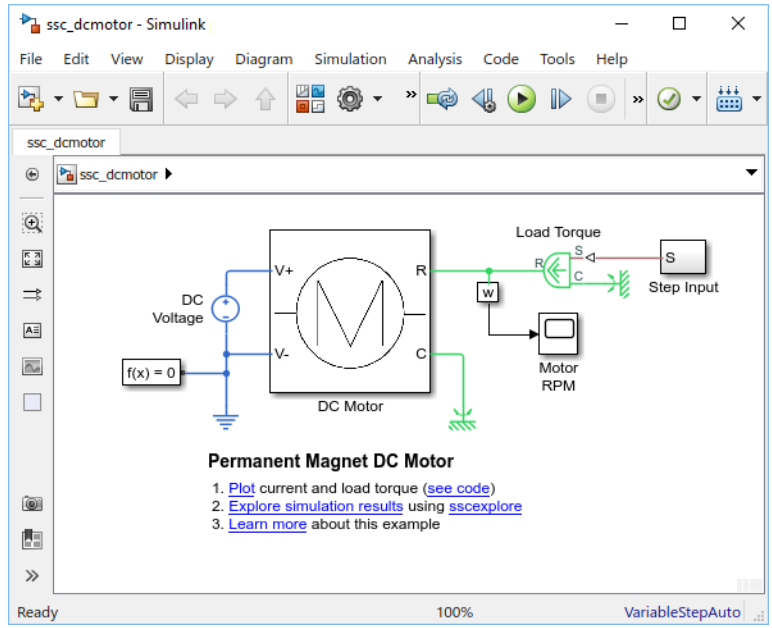
Each of the plots has a drop-down arrow next to the unit name for the *y*-axis. When you click this arrow, a context menu appears containing names of all the units in the unit registry that are commensurate with the current plot unit, as well as two other options:

- `Default` — Use the default unit.
- `Specify` — Type the unit name or expression in a pop-up window and click **OK**. The specified unit name or expression must be commensurate with the current plot unit.

Once you select the option you want, the drop-down menu collapses and the plot is redrawn in specified units.

This example shows how you can plot the data in different units by selecting the unit interactively in the plot pane.

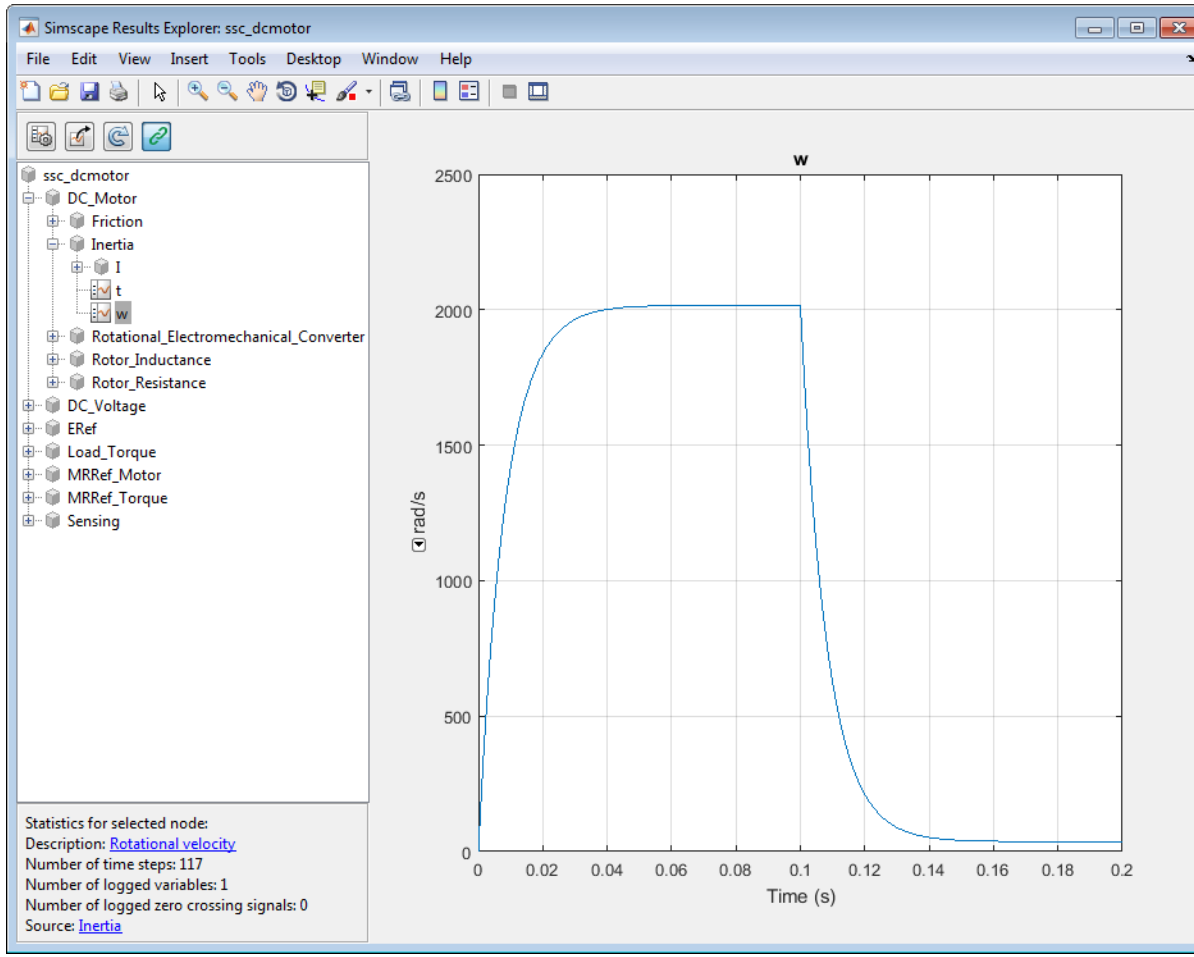
1 Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This example model has data logging enabled for the whole model, with the **Workspace variable name** parameter set to `simlog_ssc_dcmotor`.



2 Simulate the model to log the simulation data.

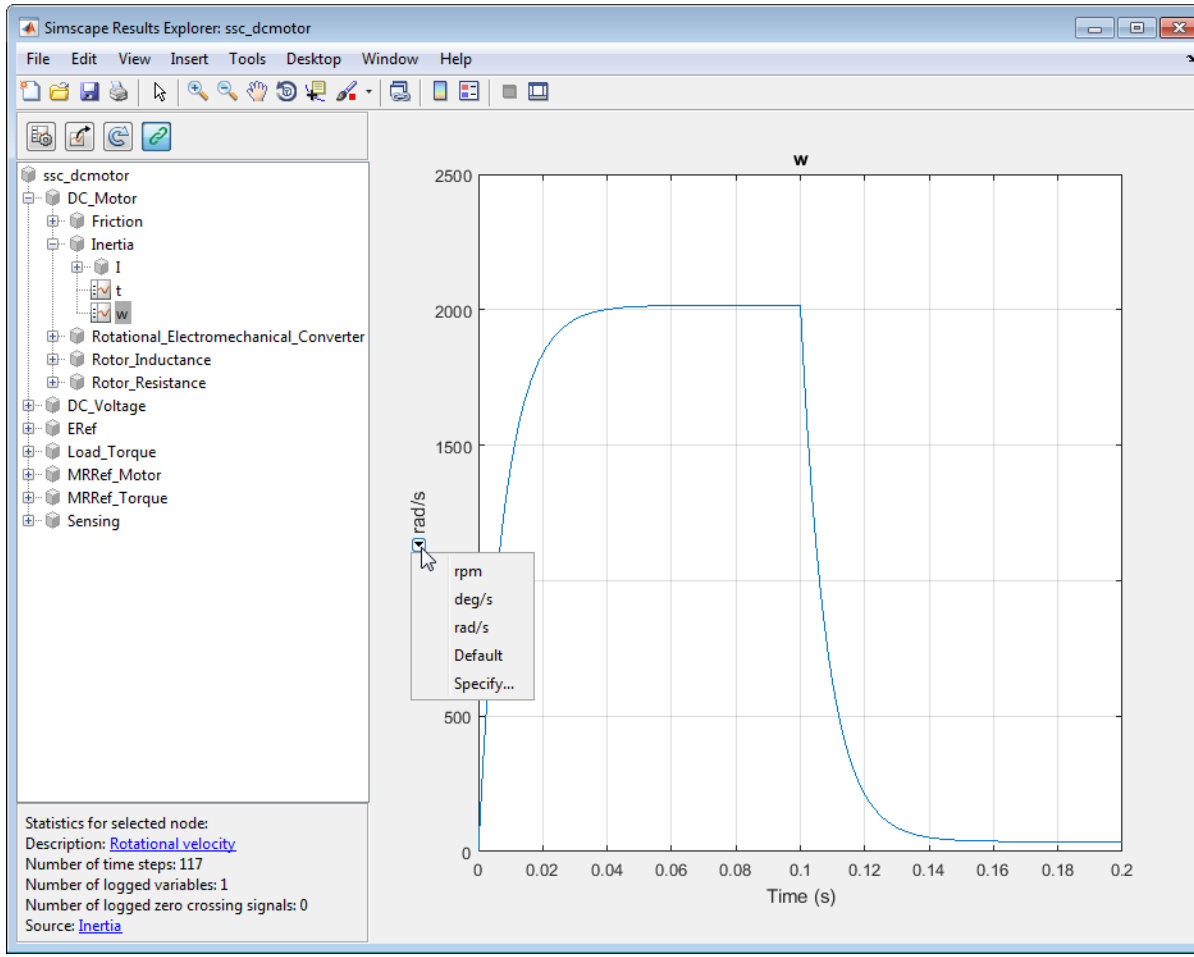
3 Open the Simscape Results Explorer window and plot the rotational velocity of the Inertia block:

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

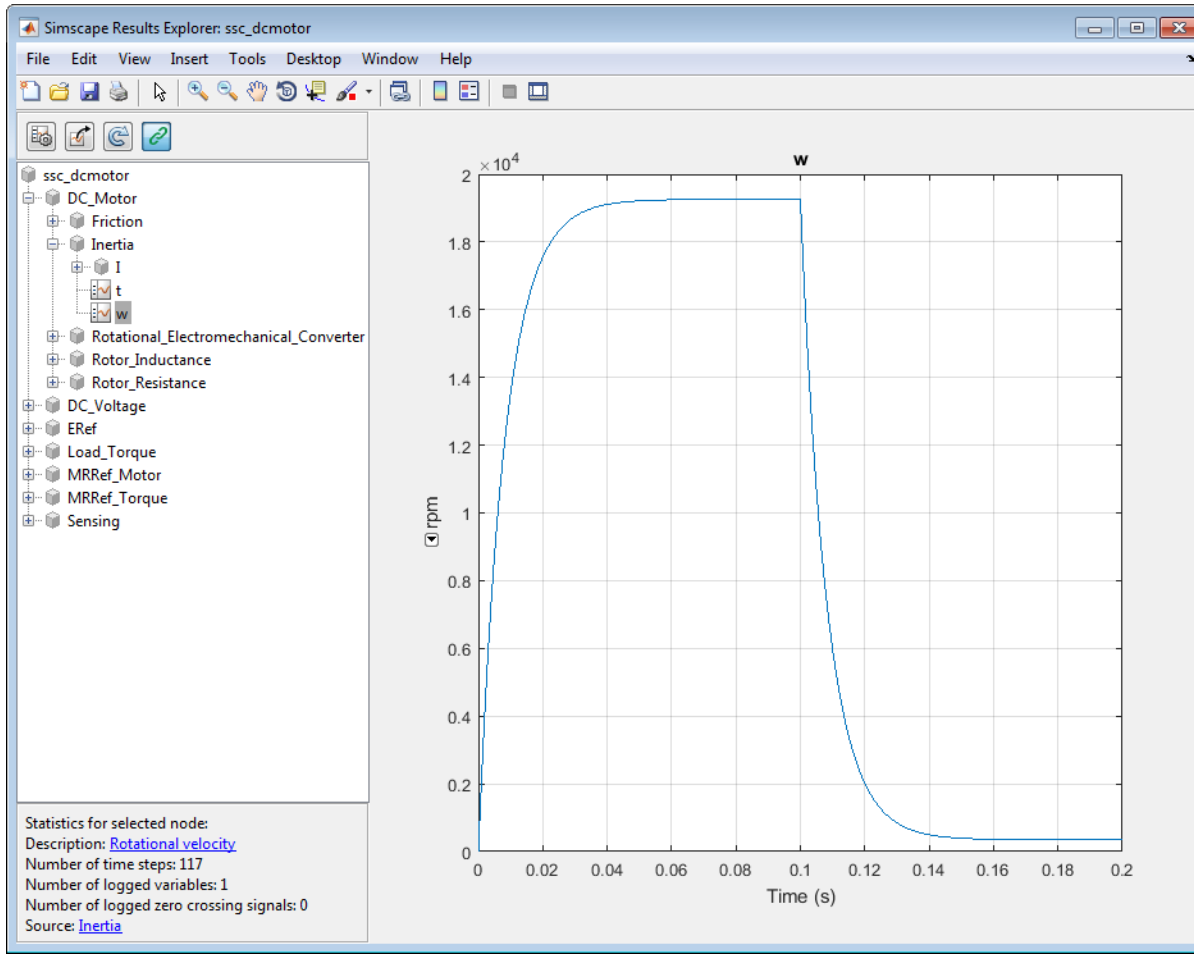


By default, Simscape Results Explorer plots rotational velocity in rad/s.

4 To plot data in different units, click the arrow under the unit name and then, from the context menu, select `rpm`.



The rotational velocity plot is redrawn in rpm.



See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Use Custom Units to Plot Simulation Data” on page 12-45

More About

- “About Simulation Data Logging” on page 12-2
- “About the Simscape Results Explorer” on page 12-30

Use Custom Units to Plot Simulation Data

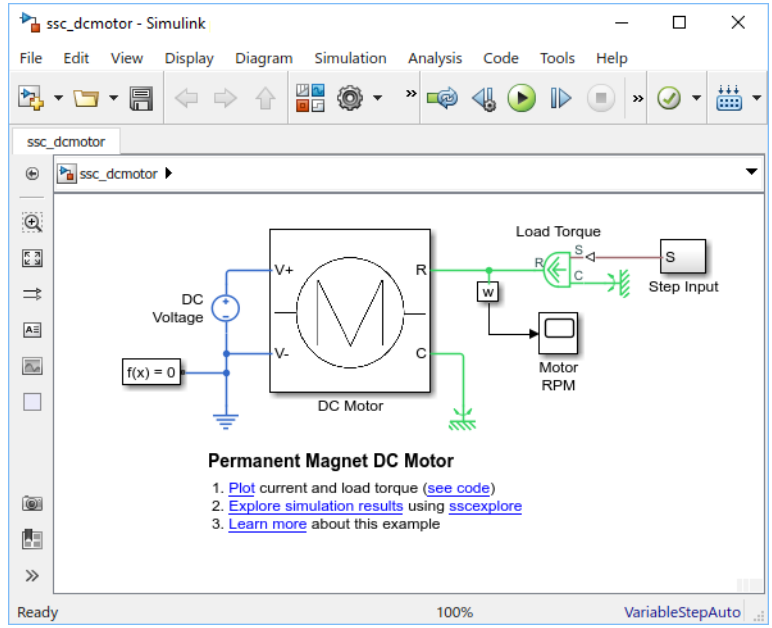
Simscape Results Explorer has a set of default units for plotting the logged data. This example shows how you can change to a custom unit; for example, to plot rotations in degrees rather than radians.

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
   ```

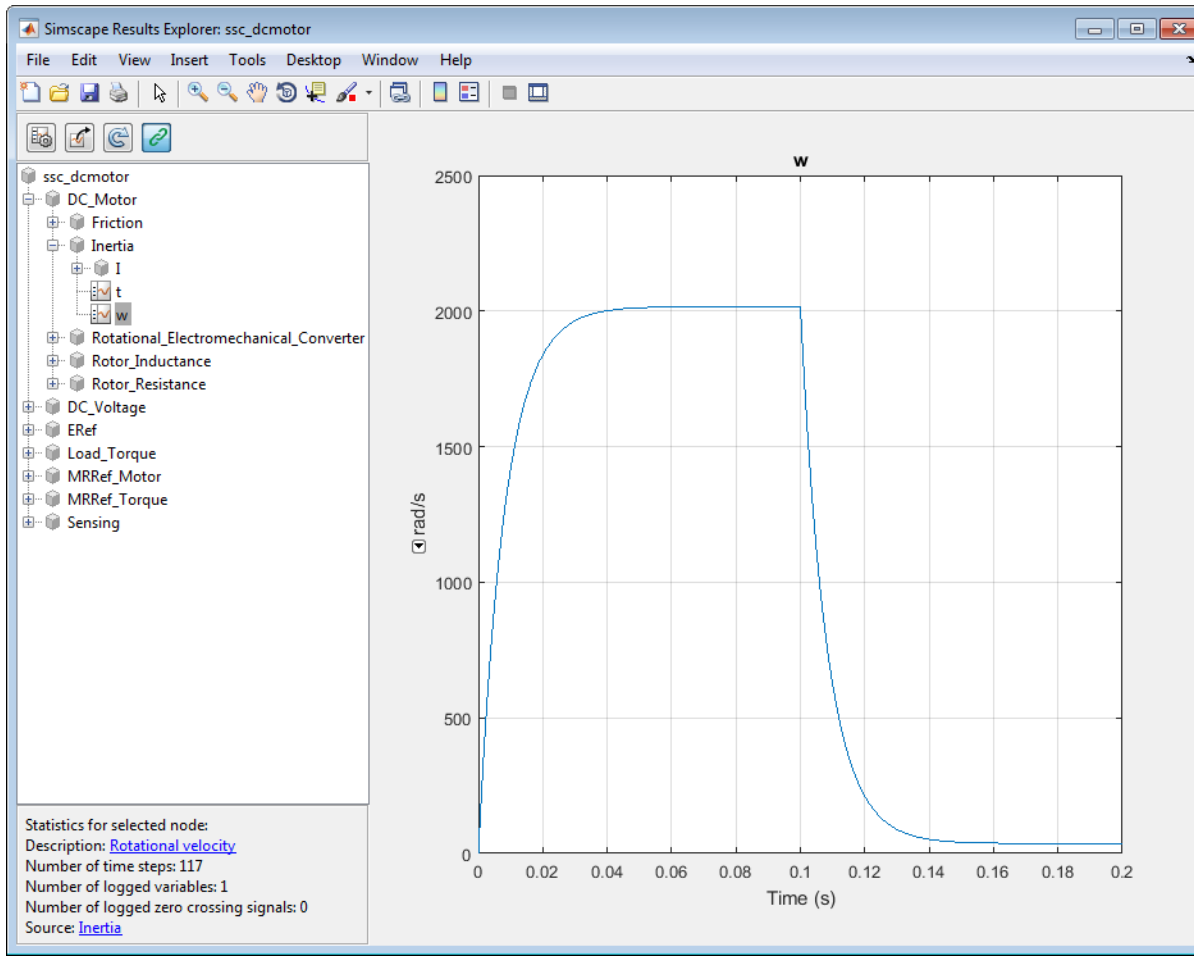
   Include only the units you want to customize. For everything else, Simscape Results Explorer will use the default units.

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`.



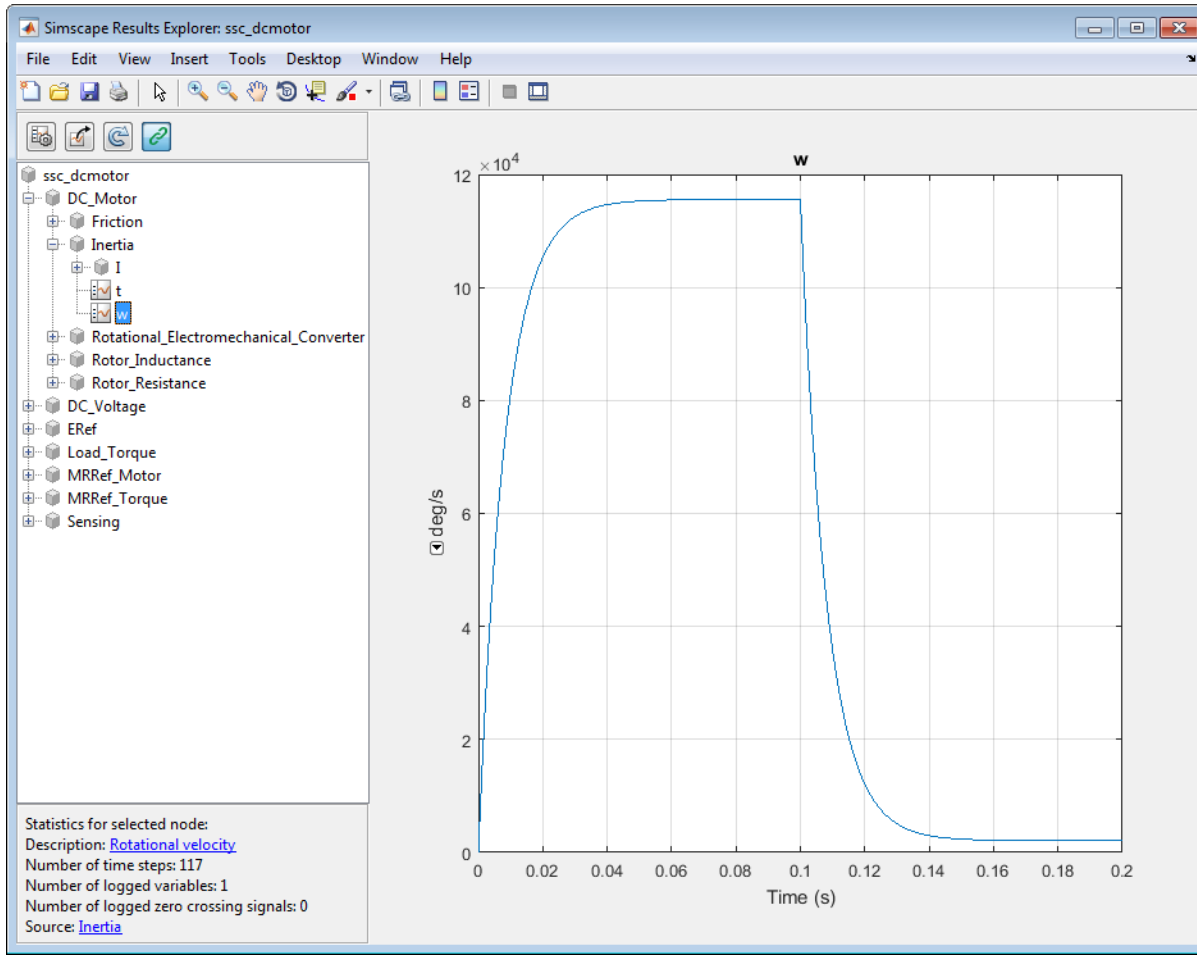
3. Simulate the model to log the simulation data.
4. Open the Simscape Results Explorer window and plot the rotational velocity of the Inertia block:

```
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.

See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Log and View Simulation Data for Selected Blocks” on page 12-20
- “Plot Simulation Data in Different Units” on page 12-39

More About

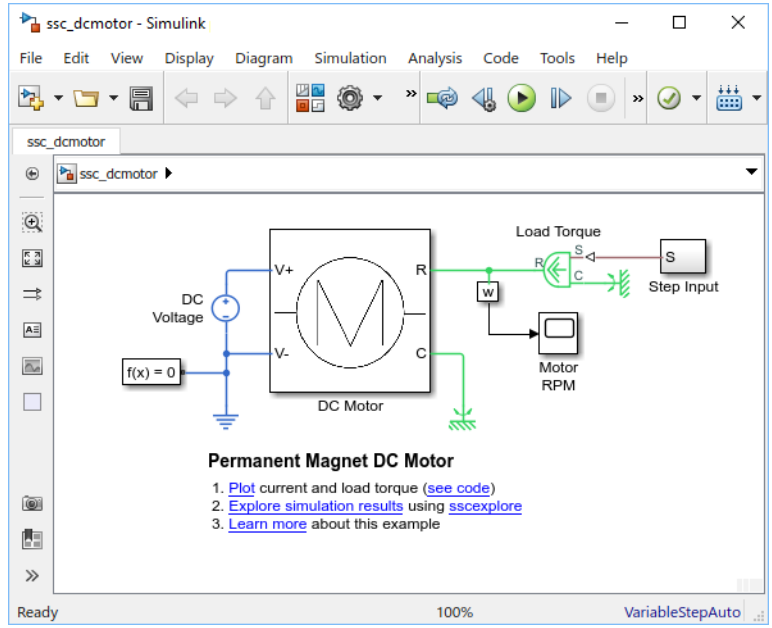
- “About Simulation Data Logging” on page 12-2
- “About the Simscape Results Explorer” on page 12-30

View Sparkline Plots of Simulation Data

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

Note If you log Simscape data as part of the single simulation output, the sparkline plots functionality is not available. For more information, see “Single simulation output” (Simulink).

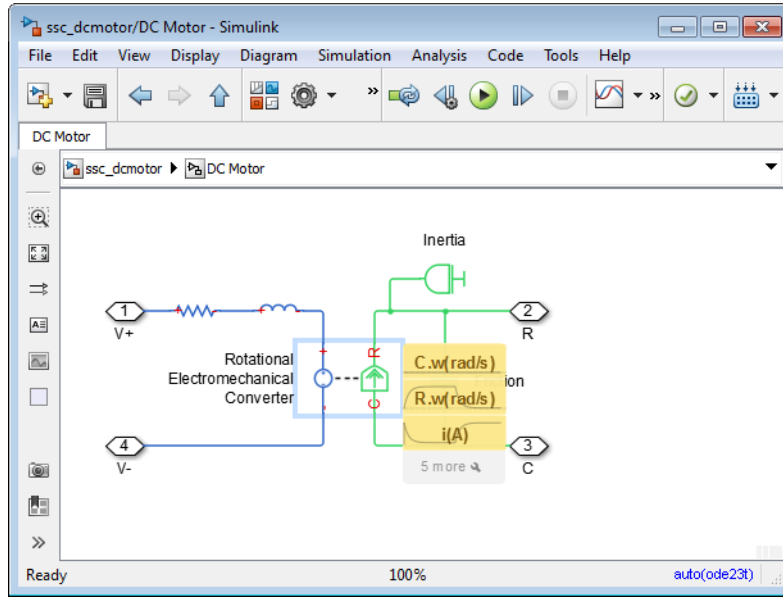
1. Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This example model has data logging for the whole model enabled.



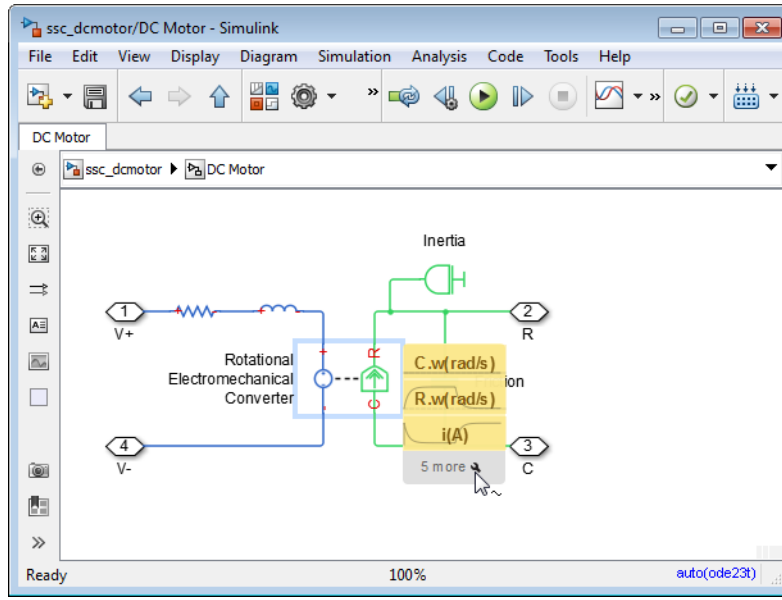
2. Double-click the DC Motor subsystem to open it.
3. Simulate the model.

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.

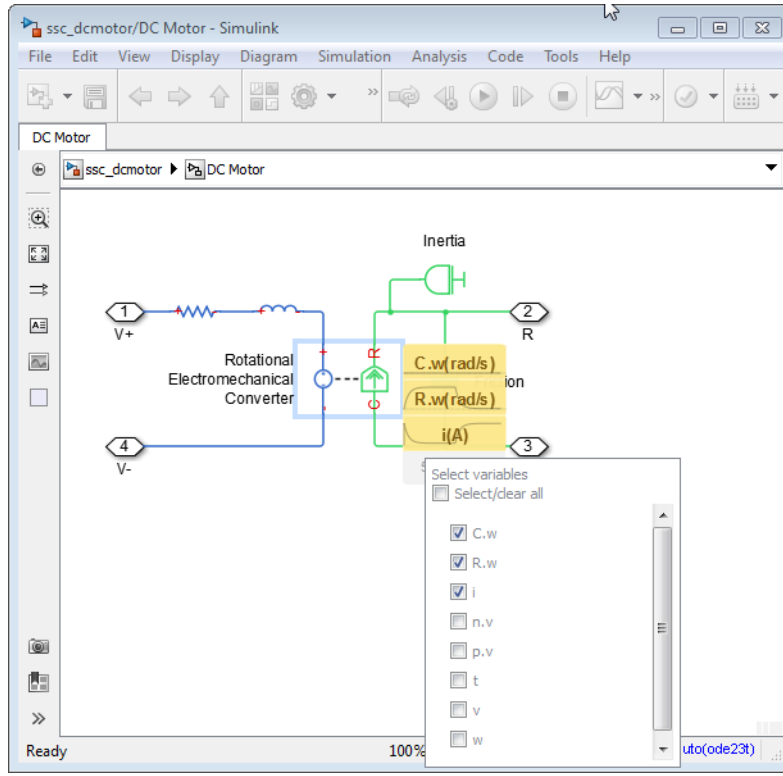
5 Select the Rotational Electromechanical Converter block. Sparkline plots of the first three variables available for this block are displayed on the canvas, and the field below the plots shows that 5 more variables are available.



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

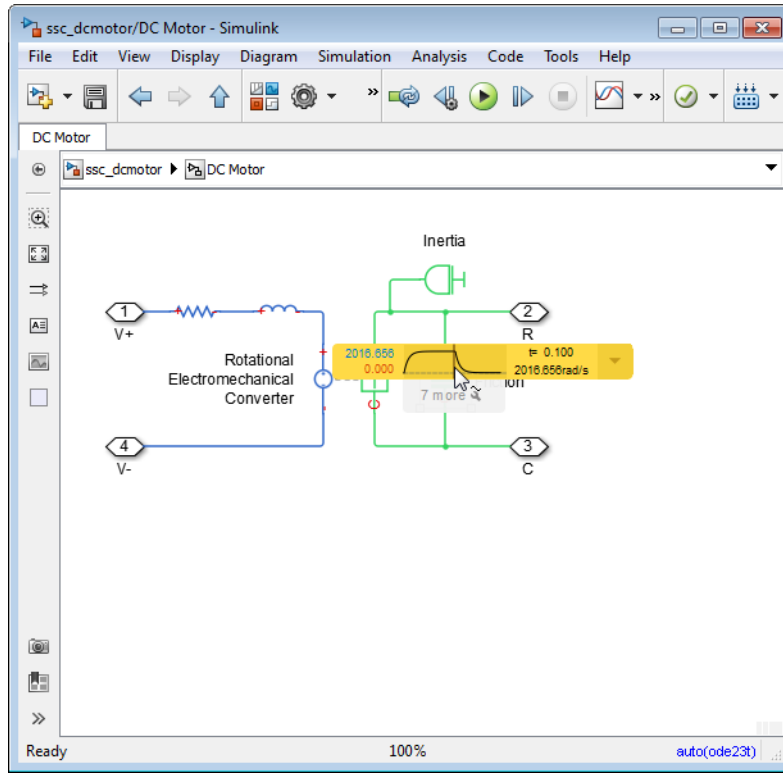


This action displays a list of all the block variables available, with check marks next to the one currently plotted.



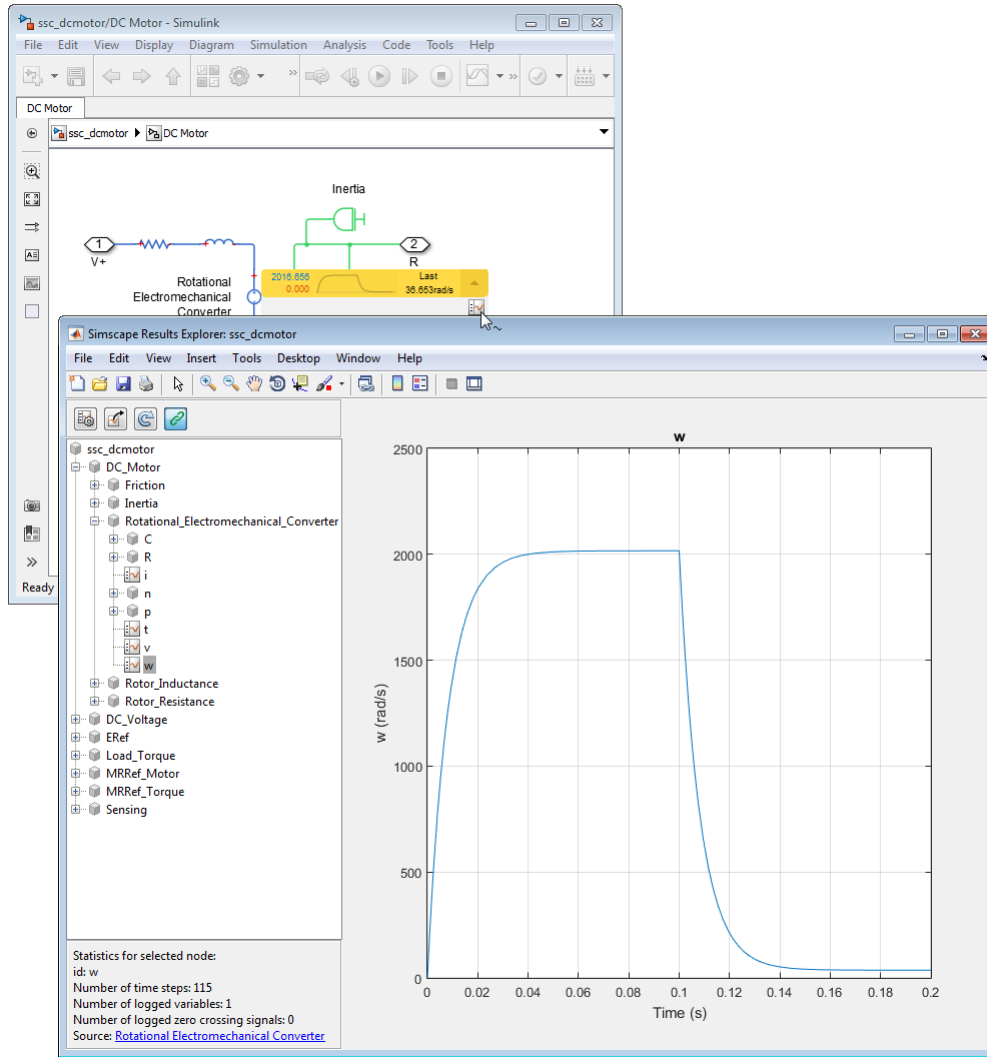
7 Clear all the check marks and select the last variable `w` instead. Then click anywhere on the model canvas to close the variable selection box.

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.

See Also

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 12-25
- “Log and View Simulation Data for Selected Blocks” on page 12-20

More About

- “About Simulation Data Logging” on page 12-2
- “About the Simscape Results Explorer” on page 12-30

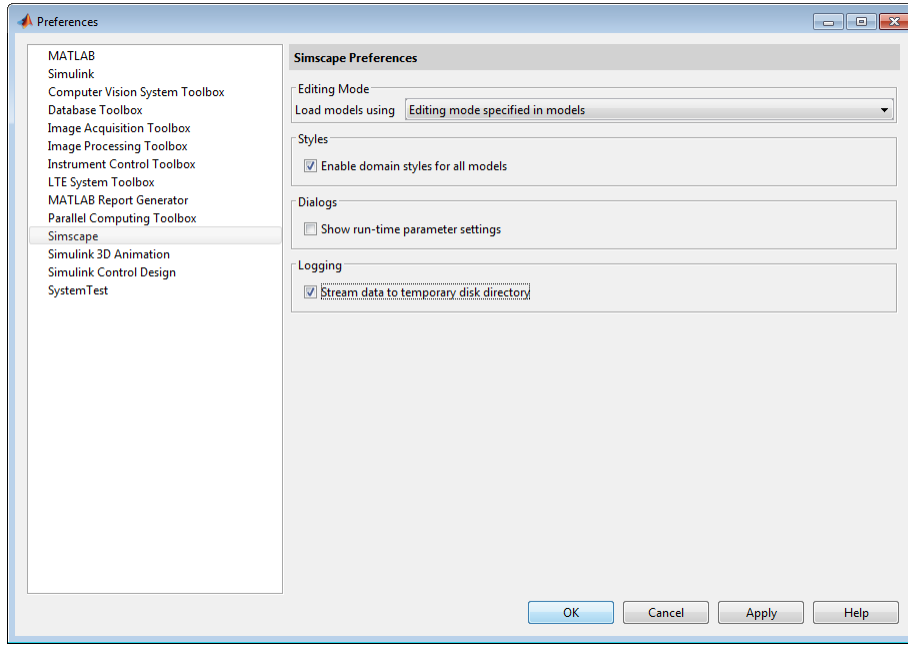
Stream Logging Data to Disk

In this section...
“Streaming to Disk and parfor Loop” on page 12-57 “Streaming to Disk with parsim” on page 12-58

When you log simulation data, you can either store the data in a workspace variable, or stream the data to a temporary file on disk and have the workspace variable point to that temporary file. In either case, you interact with the logged simulation data through the simulation log variable.

Saving data to the workspace consumes memory. Streaming logged data to disk significantly increases the data logging capacity, because you are no longer limited by the system memory.

To enable streaming data to disk for all models, on the MATLAB Toolstrip, click **Preferences**. In the left pane of the Preferences dialog box, select **Simscape**, then select the **Stream data to temporary disk directory** check box.



When this preference is turned on, the simulation data, in the form of a `simlog` object generated during simulation, is stored in an HDF5 file in a temporary folder under your user name. The workspace variable of type `simscape.logging.Node`, named as specified by the **Workspace variable name** configuration parameter, gets created, but instead of storing all the simulation data it references the `simlog` object in the temporary file. The temporary file persists as long as there is a logging variable name in the workspace that references it.

You view and analyze logged simulation data by accessing the simulation log variable, exactly the same way as if the simulation data was stored inside it. All the interaction between the workspace variable and the stored object happens behind the scenes. Therefore, you can use the Simscape Results Explorer, as well as all the methods associated with the `simscape.logging.Node` and `simscape.logging.Series` classes to query, plot, and analyze logged simulation data.

The following limitations apply when streaming data to disk:

- The **Limit data points** and **Data history (last N steps)** configuration parameters are ignored. However, you can use the **Decimation** parameter to limit the number of logged data points. For more information, see “Data Logging Options” on page 12-7.
- When you pause model simulation and step back and then forward, all the time points are logged on disk. This is different from storing the data directly in the workspace variable, where the log data is rolled back in this case.

Streaming to Disk and parfor Loop

When you simulate a model inside the `parfor` loop, the temporary HDF5 file is generated within the address space of the worker threads. To access the simulation data outside the `parfor` loop, export the data and then import the exported file outside the `parfor` loop.

```
parfor i=1:2
   model = 'ssc_dcmotor'
   load_system(model);
   set_param(model, 'SimulationMode', 'normal');
   set_param(model, 'SimscapeLogType', 'all', 'SimscapeLogName', 'simlog');
   simOut = sim(model, 'ReturnWorkspaceOutputs', 'on');

   % save to a different file by appending the index
   file = ['fileName_' num2str(i) '.h5'];
   simscape.logging.export(simOut.get('simlog'), file);
end

% import the exported files
var = simscape.logging.import('fileName_1.h5');
...
```

Streaming to Disk with parsim

Simulating a model with the `parsim` command provides additional functionality, compared to using the `parfor` loop. The following example shows how you can use the `parsim` command when streaming logged simulation data to disk.

```
model = 'ssc_dcmotor';
% Create array of inputs to run multiple simulations
num = 10;
in(1:num) = Simulink.SimulationInput(model);

% Specify any directory where the H5 files will be created for every run
currDir = '/tmp'; % any directory

for i = 1:num
    % This will only work with local pools
    in(i).PostSimFcn = @(x) locHandleSimscapeLTF(model, x, currDir, i);
end
out = parsim(in);

for idx = 1:numel(out)
    simlog = simscape.logging.import(out(idx).SimscapeFileName);
    sscexplore(name);
end

function newOut = locHandleSimscapeLTF(model, out, dirName, runId)
    % All the logged variables along with simlog should be part of 'newOut' object
    loggedVars = out.who;
    newOut = struct;
    for i = 1 : numel(loggedVars)
        loggedData = out.(loggedVars{i});
        if isa(loggedData, 'simscape.logging.Node')
            % Specify any file name with .h5 extension
            filename = [model '_simlog_file_' num2str(runId) '.h5'];
            simscapeFileName = fullfile(dirName, filename);

            % Export simlog to H5 file
            simscape.logging.export(loggedData, simscapeFileName);
            newOut.SimscapeFileName = simscapeFileName;
        else
            newOut.(loggedVars{i}) = out.(loggedVars{i});
        end
    end
end
```

See Also

More About

- “About Simulation Data Logging” on page 12-2
- “Data Logging Options” on page 12-7

- “About the Simscape Results Explorer” on page 12-30

Saving and Retrieving Logged Simulation Data

In this section...
“Data Logged to Memory” on page 12-61
“Data Logged to Temporary File on Disk” on page 12-61
“Data Recorded in Simulation Data Inspector” on page 12-61

Ways of saving and retrieving logged simulation data depend on the logging method. Each simulation log has two properties, `savable` and `exportable`, that indicate how the data was logged and therefore, how to save and retrieve it.

Logging Method	Enabled By	Savable	Exportable	Use
Memory	**Stream data to temporary disk directory** preference is off, **Record data in Simulation Data Inspector** option is off	`1`	`0`	MATLAB `save` and `load` functions, “Workspace Variables and MAT-Files” (MATLAB)
Temporary File on Disk	**Stream data to temporary disk directory** preference is on, **Record data in Simulation Data Inspector** option is off	`0`	`1`	`simscape.logging.export` and `simscape.logging.import`
Simulation Data Inspector	**Record data in Simulation Data Inspector** option is on	`0`	`0`	Simulation Data Inspector data management functions, “Inspect and Analyze Simulation Results” (Simulink)

Data Logged to Memory

When you log simulation data to workspace (**Stream data to temporary disk directory** check box on the **Simscape** pane of the Preferences dialog box is off), all the data is stored in the workspace variable. You can use the regular MATLAB interface to save the workspace variable as a MAT-file and load a MAT-file into a variable. For more information, see “Workspace Variables and MAT-Files” (MATLAB).

Data Logged to Temporary File on Disk

When you stream simulation data to disk (**Stream data to temporary disk directory** check box on the **Simscape** pane of the Preferences dialog box is on), the data is stored as a `simlog` object in a temporary file, and the workspace variable references the `simlog` object. The temporary file persists as long as there is a logging variable name in the workspace that references it. The `simscape.logging.export` function saves the logged simulation data to an HDF5 file, other than the temporary file, for use in a later session. To load the HDF5 file containing the stored `simlog` object back into workspace and associate it with a workspace variable, use the `simscape.logging.import` function.

The following restrictions apply:

- When streaming data to disk, you can export and import only the entire `simlog` object. This is different from logging data to workspace, where you can save part of the workspace variable (such as a node in the `simlog` tree) as a separate MAT-file.
- `simscape.logging.export` function does not support cross-locale characters in file name.

Data Recorded in Simulation Data Inspector

If you select the **Record data in Simulation Data Inspector** check box on the **Simscape** pane of the Configuration Parameters dialog box for a particular model, this choice overrides the logging preference. The `simlog` object has both `savable` and `exportable` properties set to `0`. You must use the Simulation Data Inspector data management functions to save and retrieve this data. For more information, see “Inspect and Analyze Simulation Results” (Simulink).

See Also

More About

- “About Simulation Data Logging” on page 12-2
- “Data Logging Options” on page 12-7
- “Stream Logging Data to Disk” on page 12-56

Model Statistics

- “Simscape Model Statistics” on page 13-2
- “1-D Physical System Statistics” on page 13-5
- “3-D Multibody System Statistics” on page 13-9
- “1-D/3-D Interface Statistics” on page 13-12
- “View Model Statistics” on page 13-14
- “Access Block Variables Using Statistics Viewer” on page 13-20
- “Model Statistics Available when Using the Partitioning Solver” on page 13-25

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 6-9.

Use model statistics as part of the iterative model building process. For example, after you make a change to the model, you can view model statistics to answer the following questions:

- Did the change increase the number of variables?
- Does the model have redundant constraints or have I resolved them?
- How many potential zero-crossing signals does the model have?
- Is the circuit high-index, and therefore hard to solve? Did my change have any effect on the index?

The Statistics Viewer analysis tool is available for models containing Simscape blocks and blocks from add-on products. Depending on the types of blocks in the model, the analysis can produce any or all of the following statistics categories:

- **1-D Physical System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, Simscape Driveline, Simscape Fluids, and Simscape Electrical (except the Specialized Power Systems) libraries.
- **3-D Multibody System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape Multibody 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 Simscape Multibody 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.

Updating the Statistics Viewer

Opening the Statistics Viewer does not trigger an automatic block diagram update. For complex models, a diagram update can last several minutes, and unnecessary diagram updates could lead to loss of productivity.

When you open the Statistics Viewer, it gets populated with the data from the last simulation or diagram update. You have to update the data explicitly by clicking the **Refresh** button ( ).

The status at the bottom of the viewer window displays the timestamp of its last update. If you have modified the model since the viewer has last been updated, the **Refresh** button displays a warning symbol ( ), and the timestamp at the bottom of the viewer window turns red to indicate that the data in the viewer might not reflect the latest model changes.

If you open a model, and then open the Statistics Viewer before simulating the model, then the viewer does not contain any data. The **Refresh** button displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.

See Also

Related Examples

- “View Model Statistics” on page 13-14
- “Access Block Variables Using Statistics Viewer” on page 13-20
- “Model Statistics Available when Using the Partitioning Solver” on page 13-25

More About

- “1-D Physical System Statistics” on page 13-5
- “3-D Multibody System Statistics” on page 13-9
- “1-D/3-D Interface Statistics” on page 13-12

1-D Physical System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, Simscape Driveline, Simscape Fluids, and Simscape Electrical libraries, with the exception of Specialized Power Systems blocks.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic.

The individual statistics are:

- **Number of variables** — This statistic represents the number of variables associated with all 1-D physical systems in the model. Variables are categorized further as continuous, eliminated, 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.

If your model uses a Partitioning local solver, the Statistics Viewer contains additional statistics specific to this solver type. For more information, see “Model Statistics Available when Using the Partitioning Solver” on page 13-25.

See Also

Related Examples

- “View Model Statistics” on page 13-14
- “Access Block Variables Using Statistics Viewer” on page 13-20

More About

- “Simscape Model Statistics” on page 13-2

3-D Multibody System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape Multibody 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. Simscape Multibody 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. Simscape Multibody attempts to remove redundant equations to improve simulation performance.

- **Number of mechanism degrees of freedom (minimum)** — This statistic provides a lower bound on the number of degrees of freedom in a mechanical system. It equals the difference between the 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 Simscape Multibody fails to detect a position constraint equation.

  Some position constraint equations become redundant only in certain configurations. If an equation becomes redundant during simulation, the actual number of degrees of freedom in a model can change. However, that number must still equal or exceed the lower bound that this statistic provides.
- **State vector size** — This statistic provides the number of scalar values in the state vector of a mechanical system.
- **Average 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)**.

See Also

More About

- “Simscape Model Statistics” on page 13-2

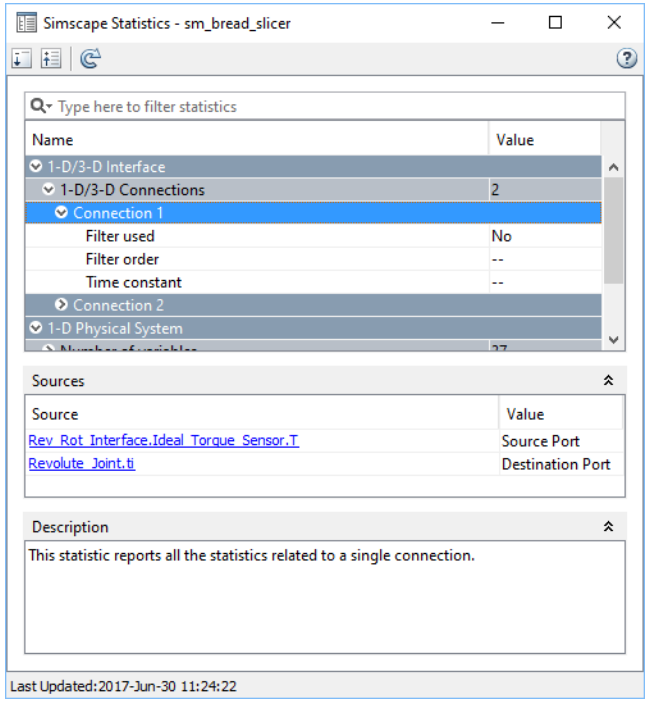
1-D/3-D Interface Statistics

This node lists statistics related to the interface between all 1-D physical and 3-D multibody systems present in the model. It appears only for models that connect blocks from Simscape Multibody Second Generation library to blocks from Simscape, Simscape Driveline, Simscape Fluids, and Simscape Electrical libraries, except the Specialized Power Systems blocks.

All connections are listed individually as `Connection 1`, `Connection 2`, and so on. If you select an individual connection, the **Sources** section lists the source and destination ports for this connection:

- The **Source** column contains the full path to the interface port, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram.
- The **Value** column specifies whether the port is the source or destination.

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



See Also

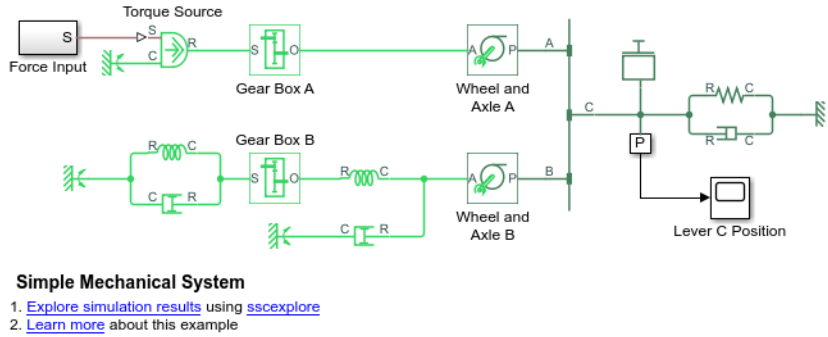
More About

- “Simscape Model Statistics” on page 13-2

View Model Statistics

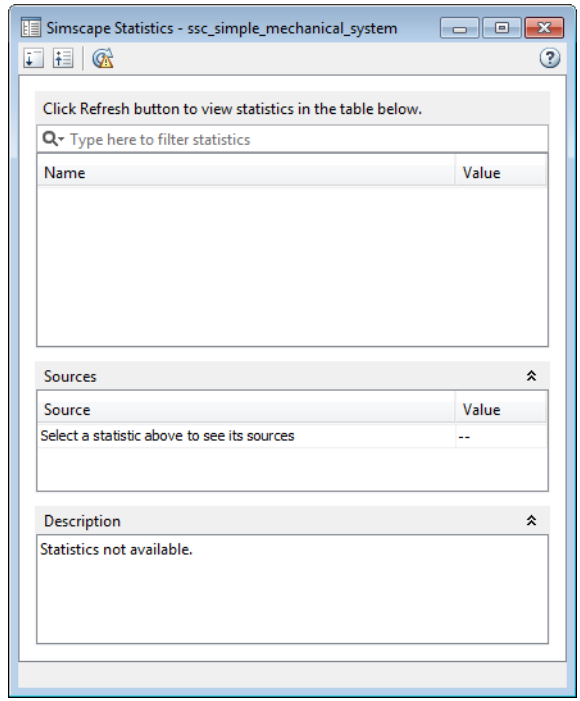
This example shows how you can use model statistics to determine the effect of a change on model complexity.

1 Open the Simple Mechanical System example model. To display hidden block names for training purposes, select **Display** and clear the **Hide Automatic Names** check box.



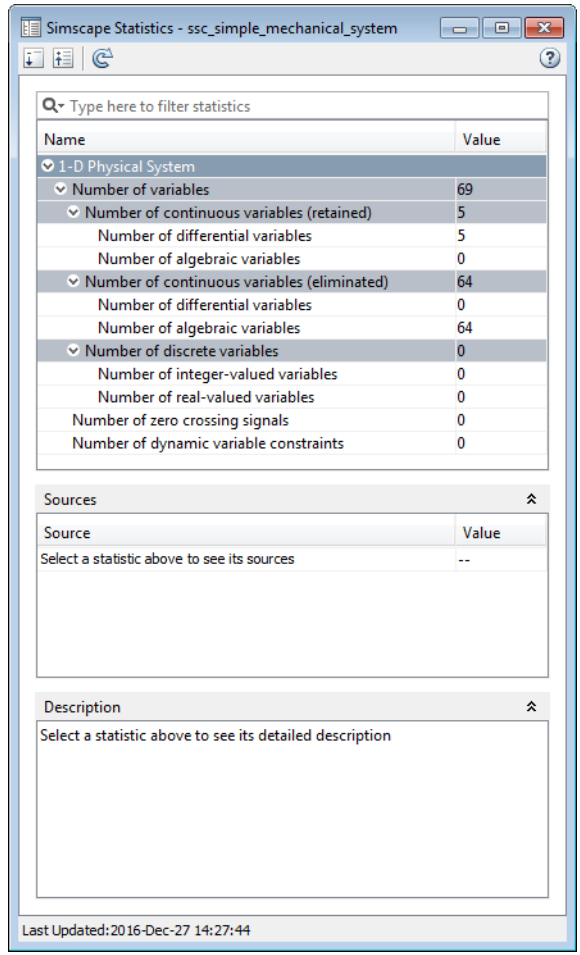
2 To view model statistics, in the top menu bar of the model window, select **Analysis** > **Simscape** > **Statistics Viewer**.

The Simscape Statistics window opens, but it does not contain any data. If you open a model, and then open the Statistics Viewer before running the simulation, the statistics data is not available. The **Refresh** button in the toolbar of the viewer window displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.



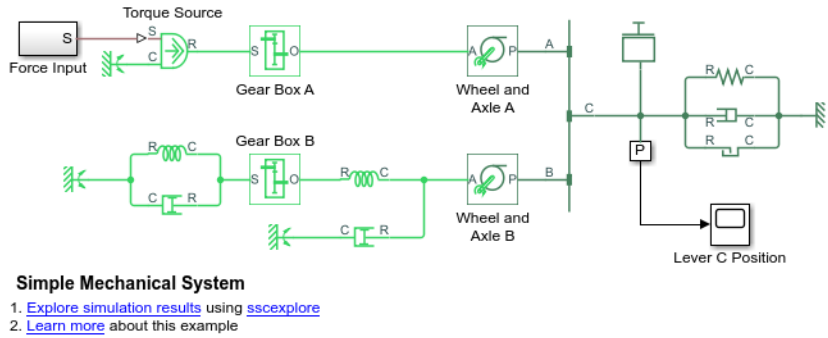
3 Click the **Refresh** button. The Simscape Statistics window now displays an overview of the models statistics in a collapsed state.

4 Click to expand all nodes.

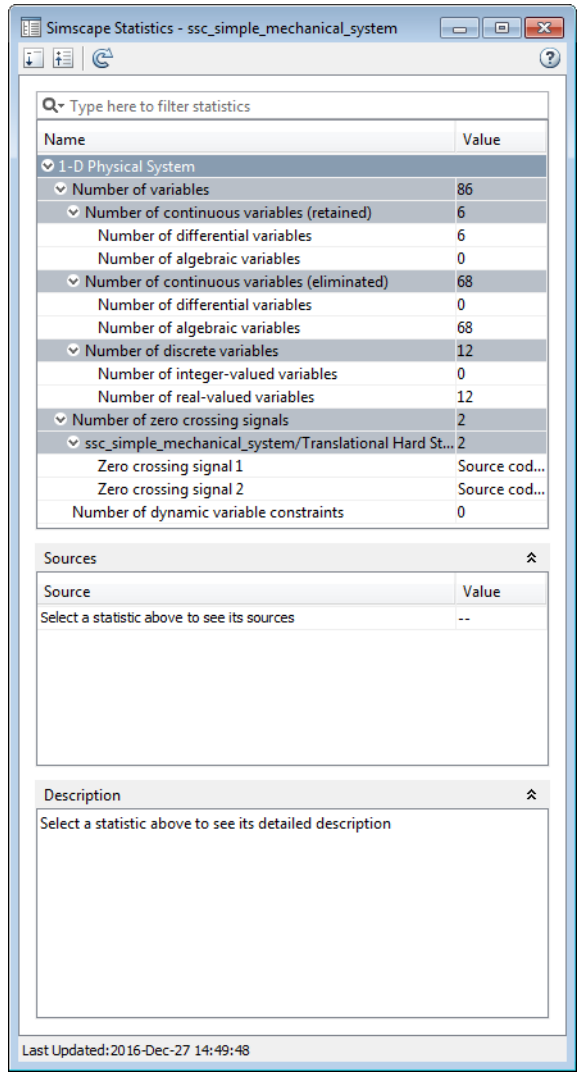


You can see that, after variable elimination, the model contains five continuous differential variables, no algebraic variables, no discrete variables, and no zero-crossing signals.

5 To limit the range of motion, add a Translational Hard Stop block to the model diagram, in parallel with the Translational Inertia and the Translational Damper blocks, as shown in the following figure.



6 Refresh the model statistics. Again, expand all the nodes.



The revised model contains six differential variables, twelve discrete real-values variables, and two zero-crossing signals. This happened because you added a nonlinear block (Translational Hard Stop). Therefore the linear optimization that the solver initially performed on the model no longer applies.

See Also

Related Examples

- “Access Block Variables Using Statistics Viewer” on page 13-20

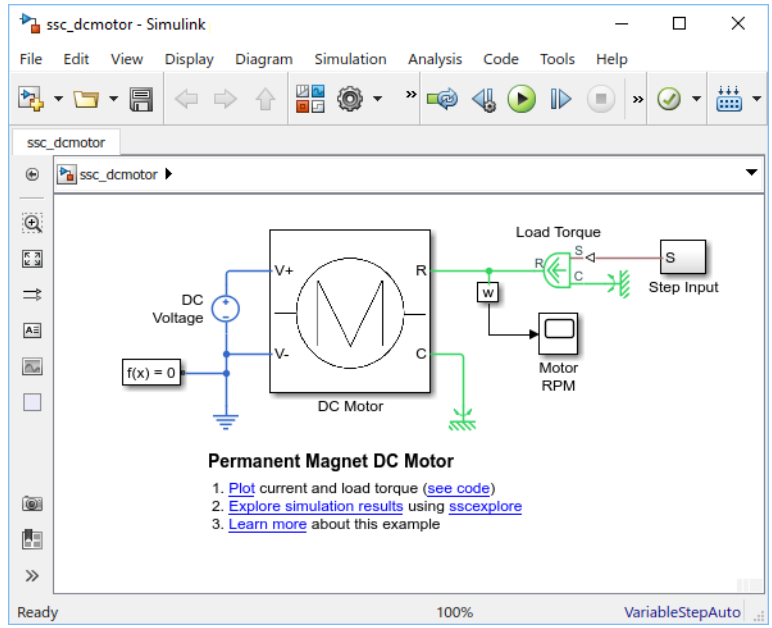
More About

- “1-D Physical System Statistics” on page 13-5

Access Block Variables Using Statistics Viewer

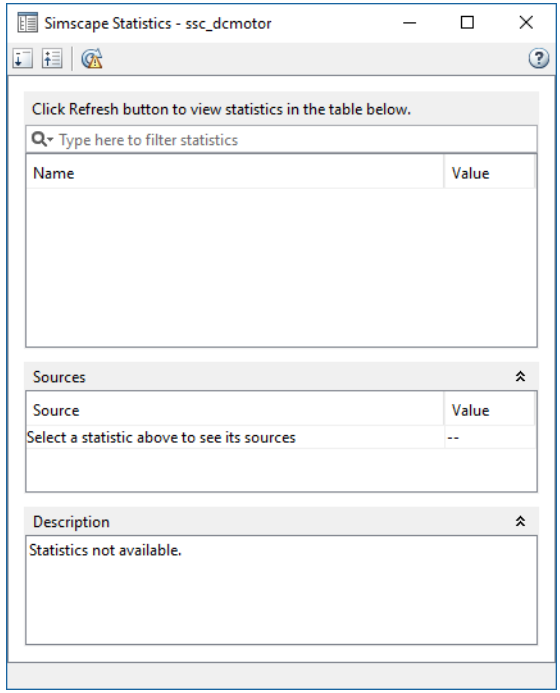
This example shows how you can use the **Sources** section of the Statistics Viewer to access a block variable of interest, to verify (or change) its initialization priority and target value.

1 Open the Permanent Magnet DC Motor example model.

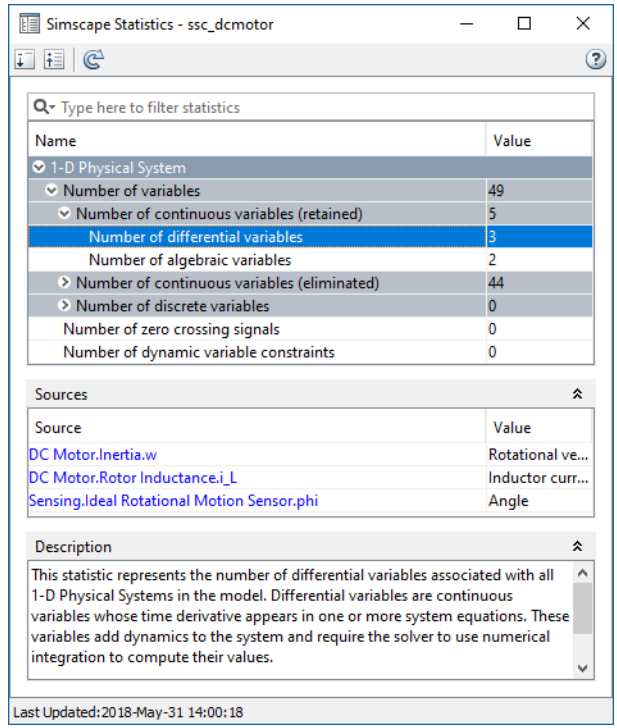


2 To view model statistics, in the top menu bar of the model window, select **Analysis** > **Simscape** > **Statistics Viewer**.

The Simscape Statistics window opens, but it does not contain any data. If you open a model, and then open the Statistics Viewer before running the simulation, the statistics data is not available. The **Refresh** button in the toolbar of the viewer window displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.



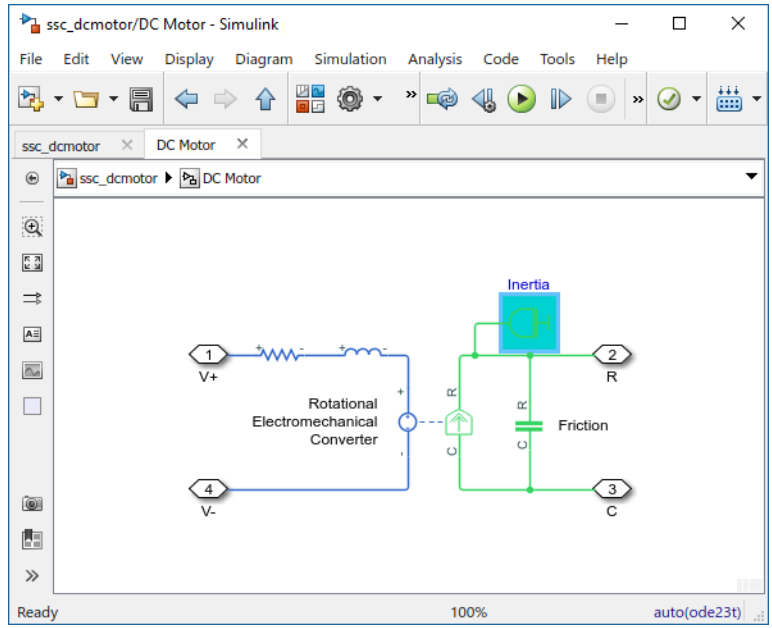
3. Click the **Refresh** button. The Simscape Statistics window now displays an overview of the models statistics in a collapsed state. The status at the bottom of the window displays the timestamp of its last update.
4. Expand the **Number of variables** node, then **Number of continuous variables (retained)**, and then click **Number of differential variables**.



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

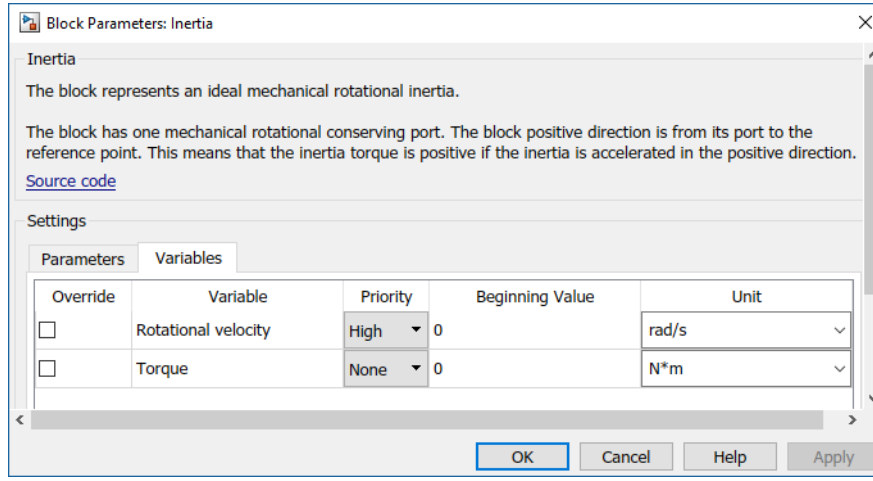
- The **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block.
- The **Value** column contains the name of the variable, as it would appear in the **Variables** tab of the block dialog box.

5 Click the first link in the **Source** column. The full path indicates that the source variable `w` belongs to the Inertia block in the DC Motor subsystem of the top-level example model, therefore the DC Motor subsystem opens and the corresponding block is highlighted in the block diagram, as shown in the following figure.



6 Double-click the highlighted block.

7 In the block dialog box, 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.

See Also

Related Examples

- “Set Priority and Initial Target for Block Variables” on page 7-5

More About

- “1-D Physical System Statistics” on page 13-5
- “Block-Level Variable Initialization” on page 7-2
- “Variable Viewer” on page 7-25

Model Statistics Available when Using the Partitioning Solver

If your model uses a Partitioning local solver, the Statistics Viewer contains additional statistics specific to this solver type:

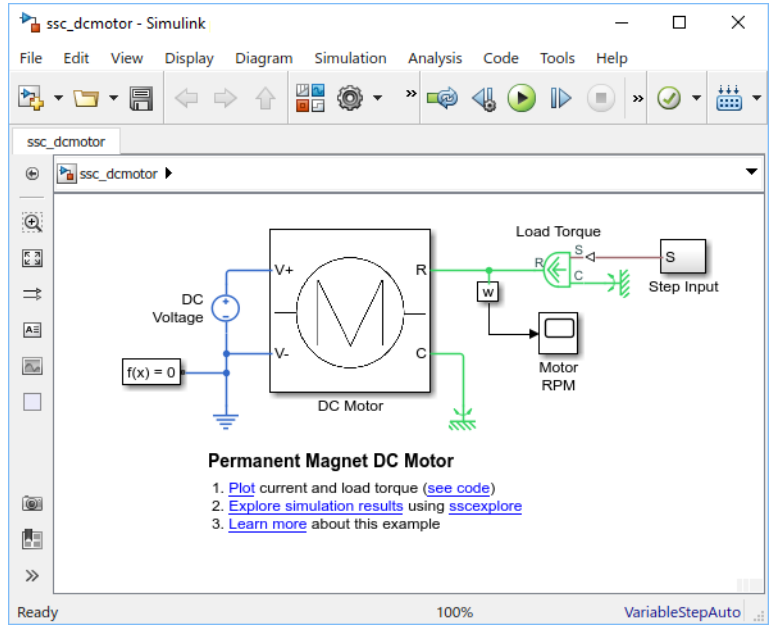
- **Number of partitions** — This statistic represents the total number of partitions in the model. When you expand this node, you can view additional statistics, listed below.
- **Total memory estimate** — This statistic represents the estimate for memory usage, in kB, for this model when using the exhaustive partition storage method.
- Additional nodes, named **Partition1**, **Partition2**, and so on, up to the total number of partitions in the model. For each of these nodes, the **Value** column lists the integration method applied to this partition. Possible methods are forward Euler (explicit) and backward Euler (implicit). For each partition node, you can also see:
  - **Equation type** — The **Value** column for this statistic lists the type of the equations in the partition. Possible types are linear time-invariant, switched linear, and linear time-varying.
  - **Number of variables** — This statistic represents the number of scalar variables in the partition. When you select this node, the **Sources** section of the Statistics Viewer lists all the variables that fall under this statistic. For each variable, the **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram. The **Value** column contains the name of the variable, as it would appear in the **Variables** tab of the block dialog box.
  - **Number of equations** — This statistic represents the number of scalar equations in the partition. Sources of the equations are provided, if they are available. When you select this node, the **Sources** section of the Statistics Viewer lists all the blocks that provide the equations. If the block source code is available (that is, not protected), clicking the link in the **Source** column opens the Simscape source file for this block in the MATLAB Editor, pointing to the appropriate equation.
  - **Number of modes** — If the **Equation type** is linear time-invariant or switched linear, this statistic represents the number of modes in the partition. Every `if` and `elseif` statement in Simscape source code corresponds to a mode. In general, when an implicit integration method is applied to a partition, the more modes there are, the more iterations are possibly needed to solve the equations for this partition.

- **Number of configurations** — This statistic represents the total number of different systems of linear equations that need to be solved when simulating the partition. In the linear time-invariant or switched linear cases, this is 2^*n*, where *n* is the number of modes in the partition. To accelerate computation, decompositions of some systems are cached for each set of modes (the **Partition storage method** parameter in the Solver Configuration block defines how the decompositions are cached). If this number exceeds the greatest possible supported unsigned integer value, the **Value** column for this statistic displays `Overflow`.
- **Memory estimate** — This statistic represents the estimate for memory usage, in kB, for this partition when using the exhaustive partition storage method.

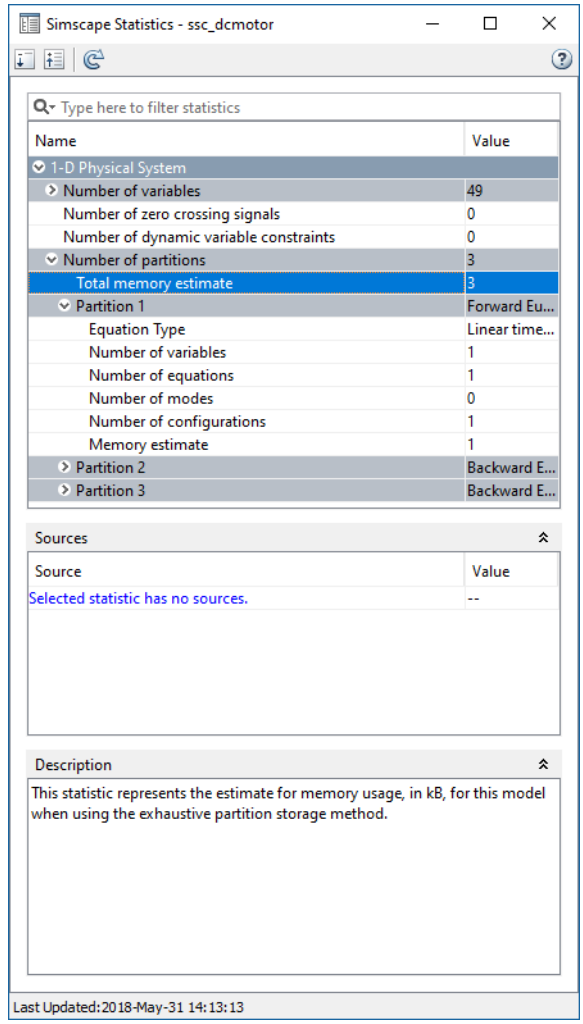
Estimate the Memory Budget for Exhaustive Partitioning Storage

This example shows how you can use the Statistics Viewer to estimate the memory budget needed for simulating a model that uses the Partitioning solver.

1 Open the Permanent Magnet DC Motor example model.



2. Double-click the Solver Configuration block, select the **Use local solver** check box, and then set the **Solver type** to `Partitioning`.
3. To view model statistics, in the top menu bar of the model window, select **Analysis** > **Simscape** > **Statistics Viewer**. Click the **Refresh** button in the toolbar of the viewer window, if necessary, to populate the viewer with data.
4. Expand the **Number of partitions** node.



The **Total memory estimate** statistic indicates that the estimate for memory usage for this model is 3 kB. When you use the exhaustive partition storage method, the default memory budget allocated for partition storage is 1024 kB. Therefore, the default memory budget value is sufficient for simulating this model, and you can even reduce it, if necessary.

If the memory estimate is unexpectedly large, you can investigate further by expanding each of the individual partition nodes and checking the **Memory estimate** value for that partition.

See Also

Solver Configuration

More About

- “Increase Simulation Speed Using the Partitioning Solver” on page 10-27
- “1-D Physical System Statistics” on page 13-5
- “Access Block Variables Using Statistics Viewer” on page 13-20

Physical Units

- “How to Work with Physical Units” on page 14-2
- “Unit Definitions” on page 14-4
- “How to Specify Units in Block Dialogs” on page 14-11
- “Thermal Unit Conversions” on page 14-13
- “How to Apply Affine Conversion” on page 14-15
- “Angular Units” on page 14-17
- “Units for Angular Velocity and Frequency” on page 14-18
- “Working with Simulink Units” on page 14-20

How to Work with Physical Units

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 14-4). 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 14-11.

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. If you use a block from the Physical Signals library after a Simulink-PS Converter block or before a PS-Simulink Converter block, the unit specification on the converter block is ignored. For more information and examples, see the Simulink-PS Converter and PS-Simulink Converter block reference pages.

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
Current	`A`	Ampere
	`pA`	Picoampere
	`nA`	Nanoampere
	`uA`	Microampere
	`mA`	Milliampere
	`kA`	Kiloampere

Quantity	Abbreviation	Unit
Energy	`J`	Joule
	`Btu_IT`	British thermal unit
	`eV`	Electronvolt
Flow rate	`lpm`	Liter/minute
	`gpm`	Gallon/minute
Force	`N`	Newton
	`dyn`	Dyne
	`lbf`	Pound-force
	`mN`	Millinewton
Frequency	`Hz`	Hertz
	`kHz`	Kilohertz
	`MHz`	Megahertz
	`GHz`	Gigahertz
Inductance	`H`	Henry
	`uH`	Microhenry
	`mH`	Millihenry

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

Quantity	Abbreviation	Unit
Pressure	`Pa`	Pascal
	`kPa`	Kilopascal
	`MPa`	Megapascal
	`GPa`	Gigapascal
	`bar`	Bar
	`kbar`	Kilobar
	`atm`	Atmosphere
	`psi`	Pound/inch^2
Power	`W`	Watt
	`uW`	Microwatt
	`mW`	Milliwatt
	`kW`	Kilowatt
	`MW`	Megawatt
	`HP_DIN`	Horsepower
Resistance	`Ohm`	Ohm
	`kOhm`	Kiloohm
	`MOhm`	Megaohm
	`GOhm`	Gigaohm

Quantity	Abbreviation	Unit
Temperature	`K`	Kelvin
	`degC`	Celsius
	`degF`	Fahrenheit
	`degR`	Rankine
	`deltaK`, `deltadegC`, `deltadegF`, `deltadegR`	Relative temperature units (see “Thermal Unit Conversions” on page 14-13)
Time	`s`	Second
	`min`	Minute
	`hr`	Hour
	`ms`	Millisecond
	`us`	Microsecond
	`ns`	Nanosecond
Velocity	`mph`	Miles/hour
	`fpm`	Feet/minute
	`fps`	Feet/second
Viscosity absolute	`P`	Poise
	`cP`	Centipoise
	`reyn`	Reyn
Viscosity kinematic	`St`	Stokes
	`cSt`	Centistokes
	`newt`	Newt

Quantity	Abbreviation	Unit
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 14-11.

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`, `in^3`, `ft^3`, `mm^3`, `cm^3`, `m^3`, and `km^3`, 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 14-13.

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 6-9 for details.

Thermal Unit Conversions

In this section...
“About Affine Units” on page 14-13 “When to Apply Affine Conversion” on page 14-13

About Affine Units

Thermal units often require an affine conversion, that is, a conversion that performs both multiplication and addition. To convert from the old value T_{old} to the new value T_{new}, we need a linear conversion coefficient L and an offset O:

$$T_{new} = L * T_{old} + O$$

For example, to convert a temperature reading from degrees Celsius into degrees Fahrenheit, the linear term equals 9/5, and the offset equals 32:

$$T_{Fahr} = 9 / 5 * T_{Cels} + 32$$

Simscape unit manager defines kelvin (`K`) as the fundamental temperature unit. This makes Celsius (`degC`) and Fahrenheit (`degF`) affine units because they are both related to kelvin with an affine conversion. Rankine (`degR`) is defined in terms of kelvin with a zero linear offset and, therefore, is not an affine unit.

The following are the default Simscape unit registry definitions for temperature units:

```
pm_adddimension('temperature', 'K');        % defines kelvin as fundamental temperature unit
pm_addunit('degC', [1 273.15], 'K');        % defines Celsius in terms of kelvin
pm_addunit('degF', [5/9 -32*5/9], 'degC');  % defines Fahrenheit in terms of Celsius
pm_addunit('degR', [5/9 0], 'K');           % defines rankine in terms of kelvin
```

When to Apply Affine Conversion

In dealing with affine units, sometimes you need to convert them using just the linear term. Usually, this happens when the value you convert represents relative, rather than absolute, temperature, $\Delta T = T_1 - T_2$.

$$\Delta T_{new} = L * \Delta T_{old}$$

In this case, adding the affine offset would yield incorrect conversion results.

For example, the outdoor temperature rose by 18 degrees Fahrenheit, and you need to input this value into your model. When converting this value into kelvin, use linear conversion

$$\Delta T_{kelvin} = 5 / 9 * \Delta T_{Fahr}$$

and you get 10 K, that is, the outdoor temperature changed by 10 kelvin. If you apply affine conversion, you will get a temperature change of approximately 265 kelvin, which is incorrect.

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

- If the outdoor temperature *changed* by 10 degrees Celsius (relative temperature value), then it changed by 10 kelvin (do not apply affine conversion).
- If the outdoor temperature *is* 10 degrees Celsius (absolute temperature value), then it is 283 kelvin (apply affine conversion).

For relative temperatures, you can also use the relative temperature units: `deltaK`, `deltadegC`, `deltadegF`, `deltadegR`. These units are consistent with the Simulink unit database (see “Units in Simulink” (Simulink)). If you use these units, affine conversion does not apply.

See Also

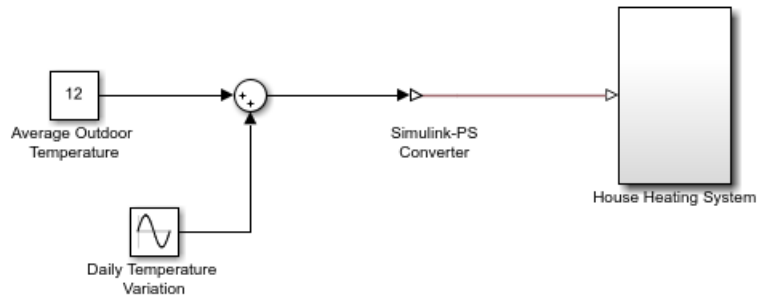
Related Examples

- “How to Apply Affine Conversion” on page 14-15

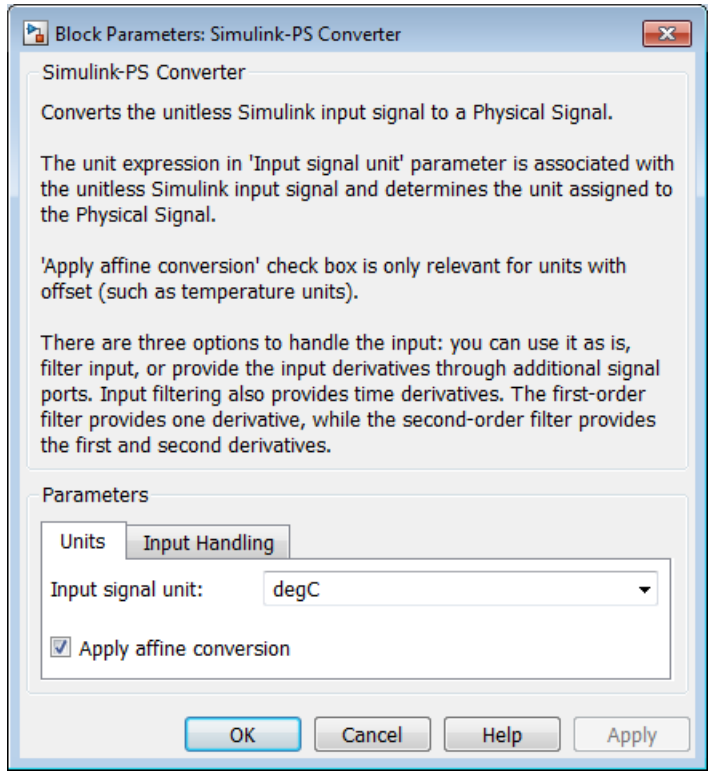
How to Apply Affine Conversion

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 14-13).

For example, you model a house-heating system, and you need to input the outdoor temperature. In the following diagram, the Constant source block represents the average outdoor temperature, in degrees Celsius, and the Sine source block adds the daily temperature variation. The average outdoor temperature, in this case, is 12 degrees Celsius. Daily variation with an amplitude of 8 makes the input outdoor temperature vary between 4 and 20 degrees Celsius.



This signal is an absolute temperature reading. Therefore, when the signal converts into kelvin for further computations, you need to specify that it should use affine conversion. Double-click the Simulink-PS Converter block, type `degC` in the **Input signal unit** field, and select the **Apply affine conversion** check box.



As a result, the Simulink-PS Converter block outputs a value varying between 277 K and 293 K.

See Also

More About

- “Thermal Unit Conversions” on page 14-13

Angular Units

Simscape implementation of angular units relies on the concept of angular units, specifically radians, being a unit but dimensionless. The notion of angular units being dimensionless is widely held in the metrology community. The fundamental angular unit, radian, is defined in the Simscape unit registry as:

```
pm_addunit('rad', 1, 'm/m');
```

which corresponds to the SI and NIST definition [1 on page 14-17]. In other words, Simscape unit manager does not introduce a separate dimension, `'angle'`, with a fundamental unit of `'rad'` (similar to dimensions for length or mass), but rather defines the fundamental angular unit in terms of meter over meter or, in effect, `1`.

The additional angular units, degree and revolution, are defined respectively as:

```
pm_addunit('deg', pi/180, 'rad');
pm_addunit('rev', 2*pi, 'rad');
```

As a result, forward trigonometric functions, such as `sin`, `cos`, and `tan`, work directly with arguments expressed in angular units. For example, cosine of 90 degrees equals the cosine of (pi/2) radians and equals the cosine of (pi/2). Expansion of forward trigonometric functions works in a similar manner.

Another effect of dimensionless implementation of angular units is the convenience of the work-energy conversion. For example, torque (in N*m) multiplied by angle (in rad) can be added directly to energy (in J, or N*m). If you specify other commensurate units for the components of this equation, Simscape unit manager performs the necessary unit conversion operations and the result is the same.

References

[1] *The NIST Reference on Constants, Units, and Uncertainty*, `https://physics.nist.gov/cuu/Units/units.html`

Units for Angular Velocity and Frequency

Angular velocity units, such as `rad/s`, `deg/s`, and `rpm`, can also be used to measure frequency for cyclical processes. This is consistent with frequency defined as revolutions per second in a mechanical context, or cycles per second in an electrical context, and lets you write frequency-dependent equations without requiring the `2*pi` conversion factor. In the SI unit system, however, the unit of frequency is hertz (Hz), defined as `1/s`.

Simscape software defines the unit hertz (`Hz`) as `1/s`, in compliance with the SI unit system. This definition works well when frequency refers to a nonrotational periodic signal such as the frequency of a PWM source. For cyclical processes, however, the block equations have to contain the `2*pi` conversion factor, to convert the numerical value specified in `Hz`, or s^{-1}, to angular frequency.

As a result, frequency units (based on `Hz`) and angular velocity units (based on `rpm`) are not directly convertible, and using one instead of the other may result in unexpected conversion factors applied to the numerical values by the block equations. For example, the AC Voltage Source block explicitly multiplies the value you specify for its **Frequency** parameter by `2*pi`, to convert it to angular frequency before calculating the sine function.

Drop-down lists of suggested units in block dialogs reflect this distinction. For example, if a block has a **Frequency** parameter with the default unit of `Hz`, the drop-down list for this parameter contains only units directly convertible to Hz (such as `kHz`, `MHz`, and `GHz`) and does not contain the angular velocity units. Conversely, if you define a custom block where the **Frequency** parameter has the default unit of `rpm`, its drop-down list of suggested units will include `deg/s` and `rad/s`, but will not contain Hz, `kHz`, `MHz`, or `GHz`.

When you type a unit expression in the parameter units combo box (instead of selecting a value from the drop-down list), the Simscape unit manager considers the units of frequency and angular velocity to be commensurate. For example, when the default parameter unit is `Hz`, you are able to type not only `1/s`, but also expressions such as `deg/s` and `rad/s`. This behavior is consistent with the Simscape implementation of angular units (see “Angular Units” on page 14-17). It is your responsibility to verify that the unit expression you typed works correctly with the block equations and reflects your design intent.

Note Prior to Release R2013a, the unit definition for Hz was `rev/s`. For information on how to update legacy models and custom Simscape libraries written in R2012b or earlier,

see Compatibility Considerations under “Unit definition of Hz now consistent with SI”, in the R2013a Release Notes.

Working with Simulink Units

You can specify physical units on Simulink signals. For details, see “Units in Simulink” (Simulink).

Interface blocks, such as Simulink-PS Converter and PS-Simulink Converter, handle the boundary between the Simscape physical network and the Simulink blocks connected to it. These converter blocks also handle the physical signal units:

- On a Simulink-PS Converter block, you specify the unit using the **Input signal unit** parameter. This parameter defines the unit of the physical signal at the PS output port of the block, which serves as the input signal for the Simscape physical network.
- On a PS-Simulink Converter block, you specify the unit using the **Output signal unit** parameter. This unit must be commensurate with the unit of the input physical signal coming into the block. The block applies a gain equal to the conversion factor before outputting the Simulink signal.

If you specify a physical unit on a Simulink signal connected to a Simulink-PS Converter or a PS-Simulink Converter block, the software compares this unit with the unit specified inside the block. If the parameter value does not match the physical unit of the Simulink signal connected to the block, you get a warning.

Simulink unit database is fixed: you cannot add units or change unit definitions. When you add a new unit to your Simscape unit registry, by using the `pm_addunit` function, and use this unit inside a Simulink-PS Converter or PS-Simulink Converter block:

- If your unit definition conflicts with the one in the Simulink database, you get a warning about incompatible unit.
- If you add a unit that does not exist in the Simulink database, you get a warning about undefined unit.

Note that these warnings apply only to the Simulink database; the Simscape physical network works as expected.

For example, you want to view the motor speed in revolutions per second, rather than revolutions per minute (`rpm`):

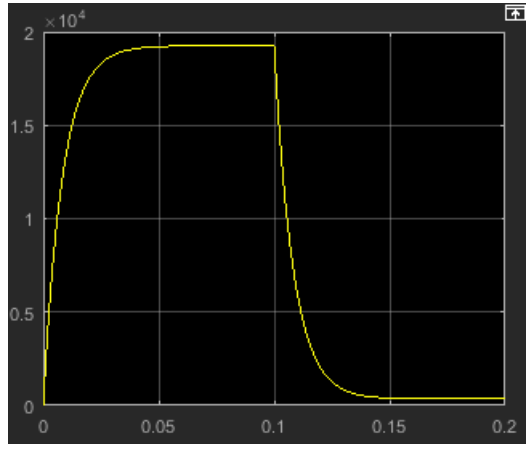
1. Add a new unit `rps`, defined in terms or `rpm`:

   ```
   pm_addunit('rps', 1/60, 'rpm');
   ```

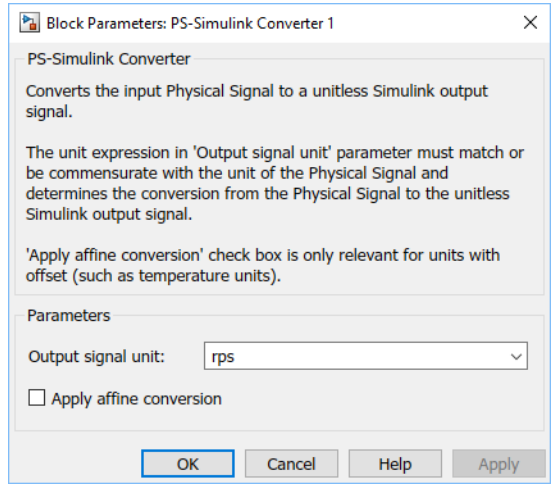
2. To open the Permanent Magnet DC Motor example model, in the MATLAB Command Window, type:

   ```
   ssc_dcmotor
   ```

3. Simulate the model. Examine the simulation results in the Motor RPM scope window.

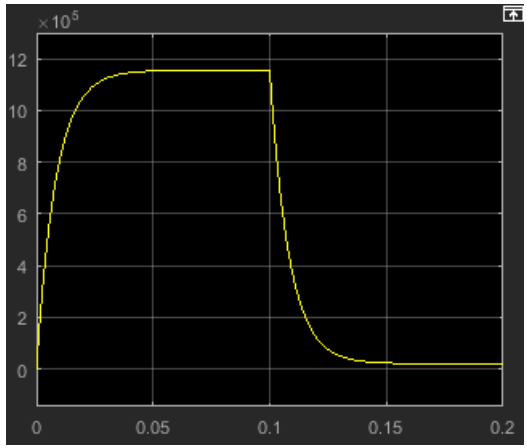


4. Open the Sensing subsystem (designated as `w` in the block diagram), double-click the PS-Simulink Converter block and type `rps` as the **Output signal unit** parameter value.

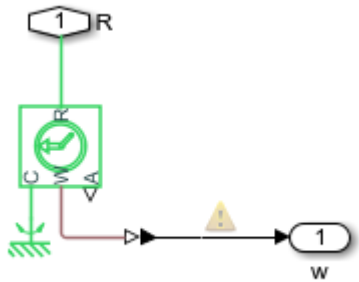


5 Rerun the simulation.

The model works correctly, with the scope displaying the results in revolutions per second.



However, the output Simulink signal of the PS-Simulink Converter block now displays a warning badge, with a message `The units 'rps' are undefined`. The detailed message explains that the units are not defined in the Simulink unit database.



If you issue a `pm_getunits` command, you can see `rps` in the return unit list, which means that the unit is successfully defined in the Simscape unit registry. In other words, the warning applies only to Simulink unit checking.

6 To turn off the unit inconsistency warnings, in the MATLAB Command Window, type:

```
set_param('ssc_dcmotor','UnitsInconsistencyMsg','none');
```

Add-On Product License Management

- “About the Simscape Editing Mode” on page 15-2
- “Set the Model Loading Preference” on page 15-9
- “Save a Model in Restricted Mode” on page 15-11
- “Work with a Model in Restricted Mode” on page 15-15
- “Switch from Restricted to Full Mode” on page 15-29
- “Get Editing Mode Information” on page 15-31

About the Simscape Editing Mode

In this section...
“Suggested Workflows” on page 15-2
“What You Can Do in Restricted Mode” on page 15-3
“What You Can Do in Full Mode” on page 15-4
“Switching Between Modes” on page 15-4
“Working with Block Libraries” on page 15-7

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: Simscape Driveline, Simscape Electrical, Simscape Fluids, and Simscape Multibody. It allows you to open, simulate, and save models that contain blocks from add-on products in Restricted mode, without checking out add-on product licenses, as long as the products are installed on your machine. It is intended to provide an economical way to distribute simulation models throughout a team or organization.

Note Unless your organization uses concurrent licenses, see the Simscape product page on the MathWorks Web site for specific information on how to install add-on products on your machine, to be able to work in Restricted mode.

The Editing Mode functionality supports widespread use of Physical Modeling products throughout an engineering organization by making it economical for one user to develop a model and provide it to many other users.

Specifically, this feature allows a user, model developer, to build a model that uses Simscape platform and one or more add-on products and share that model with other users, model users. When building the model in Full mode, the model developer must have a Simscape license and the add-on product licenses for all the blocks in the model. For example, if a model combines Simscape, Simscape Fluids, and Simscape Driveline blocks, the model developer needs to check out licenses for all three products to work with it in Full mode. Once the model is built, model users need only to check out a Simscape license to simulate the model and fine-tune its parameters in Restricted mode.

As long as no structural changes are made to the model, model users can work in Restricted mode and do not need to check out add-on product licenses.

Another workflow, available with concurrent licenses only, lets multiple users, who all have Simscape licenses, share a small number of add-on product licenses by working mostly in Restricted mode, and temporarily switching models to Full mode only when they need to perform a specific design task that requires being in Full mode.

Note 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 on page 15-4, your model has to be in Full mode. Some of these disallowed changes are impossible to make in Restricted mode (for example, Restricted parameters are grayed out in block dialog boxes). Other changes, like changing the physical topology of a model, are not explicitly disallowed, but if you make these changes in Restricted mode, the software will issue an error message when you try to run, compile, or save such a model.

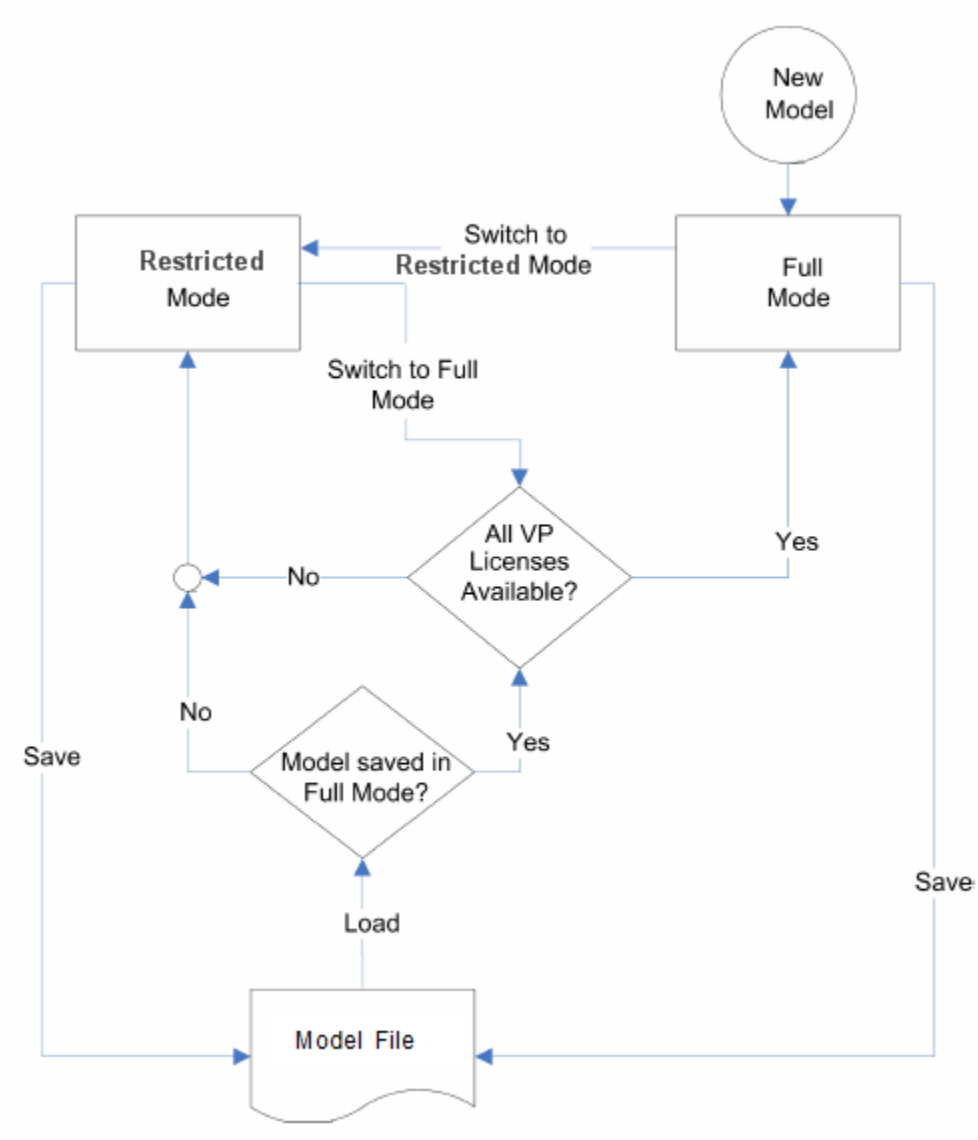
What You Can Do in Full Mode

You need to open a model in Full mode if you need to do any of the following:

- Add or delete Physical Modeling blocks (that is, Simscape blocks or blocks from the add-on product libraries).
- Make or break Physical connections (between Conserving or Physical Signal ports).
- Change the types of signals going into actuators or out of sensors (for example, from velocity to torque).
- Change configuration parameters.
- Change block parameterization options and other restricted parameters.
- Change physical units of parameters.
- Protect a referenced model containing Physical Modeling blocks.

Switching Between Modes

The following flow chart shows what happens when you switch between modes.



New models are always created in Full mode. You can then either save the model in Full mode, or switch to Restricted mode and save the model in Restricted mode.

When you load an existing model, the license manager checks whether it has been saved in Full or Restricted mode.

- If the model has been saved in Restricted mode, it opens in Restricted mode.
- If the model has been saved in Full mode, the license manager checks whether all the add-on product licenses for this model are available and, if so, opens it in Full mode. If a add-on product license is not available, the license manager issues an error message and opens the model in Restricted mode. See also “Example with Multiple Add-On Products” on page 15-6.

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 15-6.
- 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 Simscape Fluids and Simscape Driveline libraries, but the user who tries to open it has only the Simscape Driveline license available. It may happen that the license manager checks out a Simscape Driveline license first, and then tries to check out a Simscape Fluids license, which is not available. The license manager then issues an error message and opens the model in Restricted mode, but the Simscape Driveline license stays checked out until the end of the MATLAB session.

Working with Block Libraries

This section describes the specifics of working with block libraries while using the Editing Mode functionality. These rules are applicable to any physical modeling blocks, that is, blocks from all Simscape libraries, including the add-on products. In general, you need to work in Full mode when you modify a library block. However, when you open a model that references the modified block, you may work in Restricted mode, under certain conditions. The following summary details the Editing Mode rules for modifying and using library blocks:

- To add 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.

See Also

Related Examples

- “Set the Model Loading Preference” on page 15-9
- “Save a Model in Restricted Mode” on page 15-11
- “Work with a Model in Restricted Mode” on page 15-15
- “Switch from Restricted to Full Mode” on page 15-29

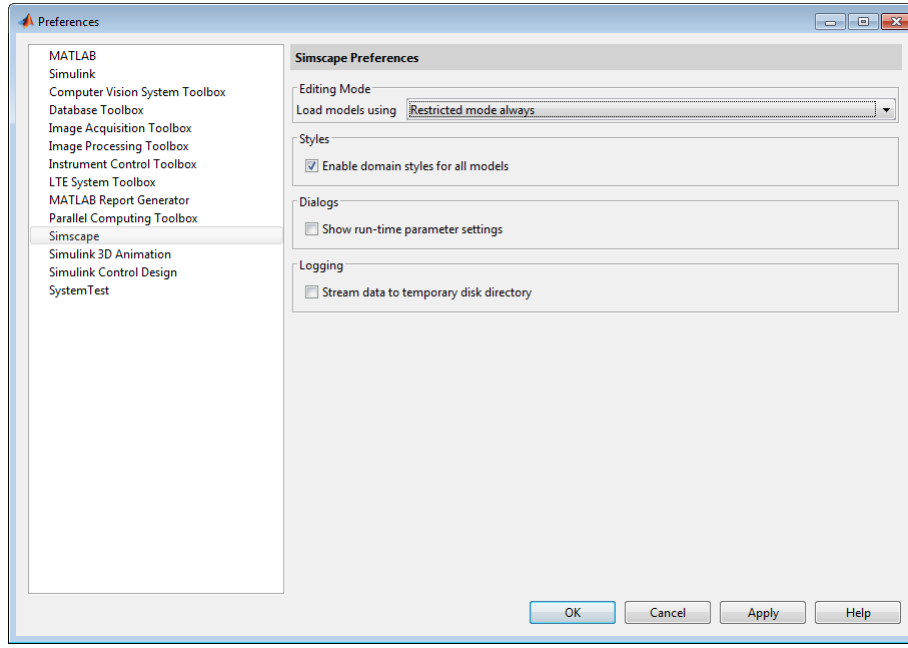
More About

- “Get Editing Mode Information” on page 15-31

Set the Model Loading Preference

By default, when you load an existing model, the license manager checks whether it has been saved in Full or Restricted mode and tries to open it in this mode. However, you can set your preferences so that the models are always open in Restricted mode, regardless of the way they have been saved.

1. On the MATLAB Toolstrip, click **Preferences**. The Preferences dialog box opens.
2. In the left pane of the Preferences dialog box, select **Simscape**. The right pane displays the **Editing Mode** group box. By default, the **Load models using** option is set to `Editing mode specified in models`.
3. Select `Restricted mode always` from the drop-down list, as shown, and click **OK**.



Now, when you open a model, the license manager does not attempt to check out add-on product licenses and always opens the model in Restricted mode.

See Also

Related Examples

- “Save a Model in Restricted Mode” on page 15-11
- “Work with a Model in Restricted Mode” on page 15-15
- “Switch from Restricted to Full Mode” on page 15-29

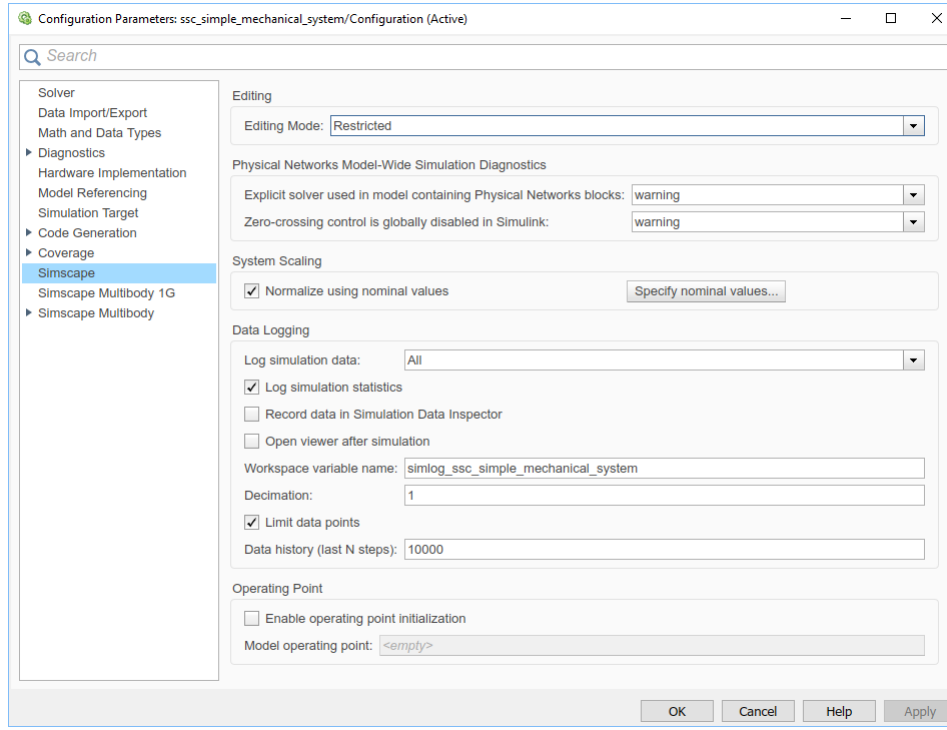
More About

- “About the Simscape Editing Mode” on page 15-2
- “Get Editing Mode Information” on page 15-31
- “Domain-Specific Line Styles” on page 1-42
- “About Simscape Run-Time Parameters” on page 9-2
- “Stream Logging Data to Disk” on page 12-56

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.

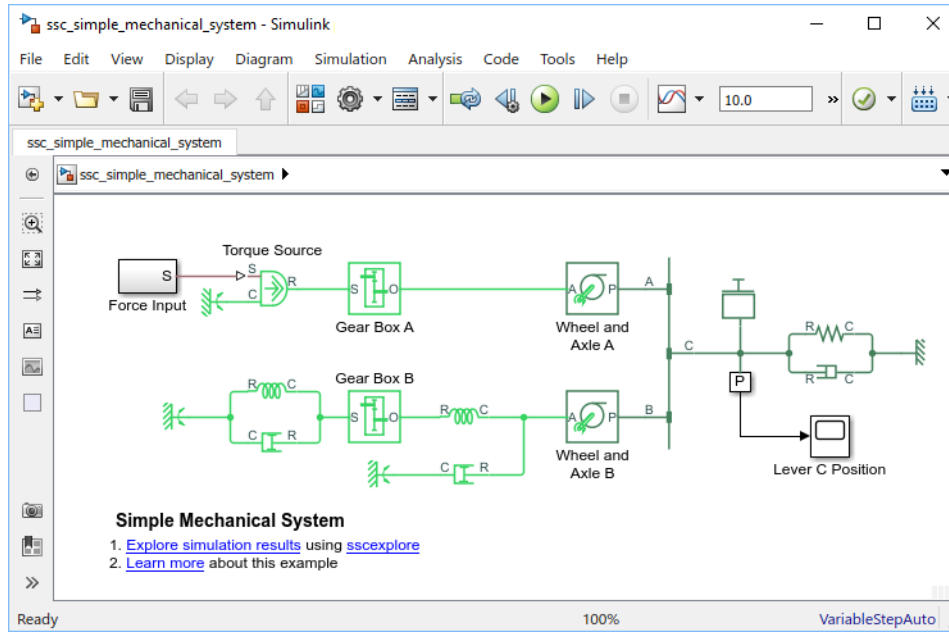
Once you have switched a model to Restricted mode, working with it follows the rules described in “Work with a Model in Restricted Mode” on page 15-15. Note, however, that the add-on product licenses for this model stay checked out until you quit the MATLAB session.

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`). To display hidden block names for training purposes, select **Display** and clear the **Hide Automatic Names** check box.



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`.

See Also

Related Examples

- “Set the Model Loading Preference” on page 15-9
- “Work with a Model in Restricted Mode” on page 15-15
- “Switch from Restricted to Full Mode” on page 15-29

More About

- “About the Simscape Editing Mode” on page 15-2
- “Get Editing Mode Information” on page 15-31

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 15-3.

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.

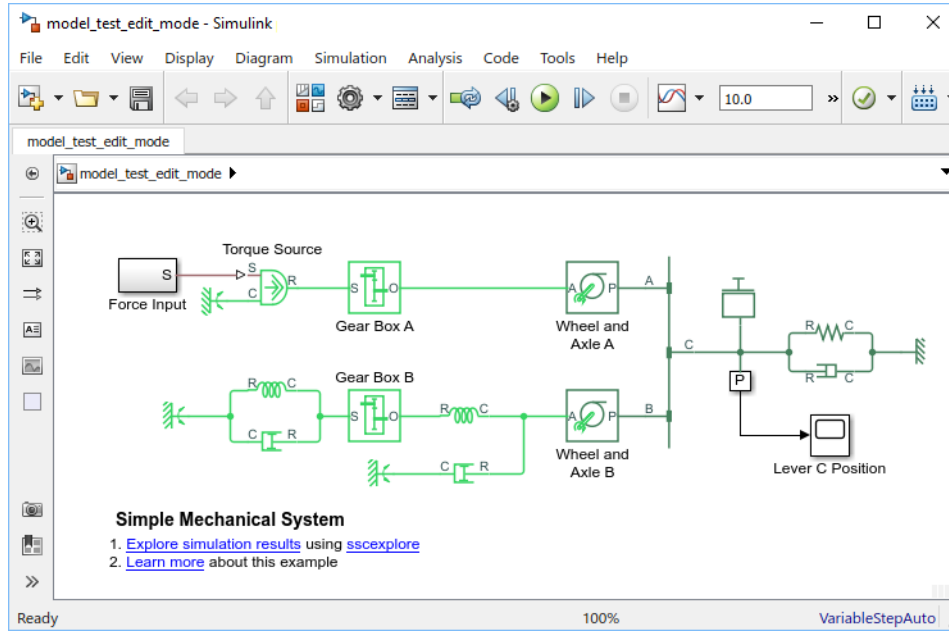
The following examples illustrate operations allowed and disallowed in Restricted mode:

In this section...
“How to Simulate and Fine-Tune a Model in Restricted Mode” on page 15-15
“How to Add and Delete Simulink Blocks in Restricted Mode” on page 15-20
“Performing an Operation Disallowed in Restricted Mode” on page 15-25

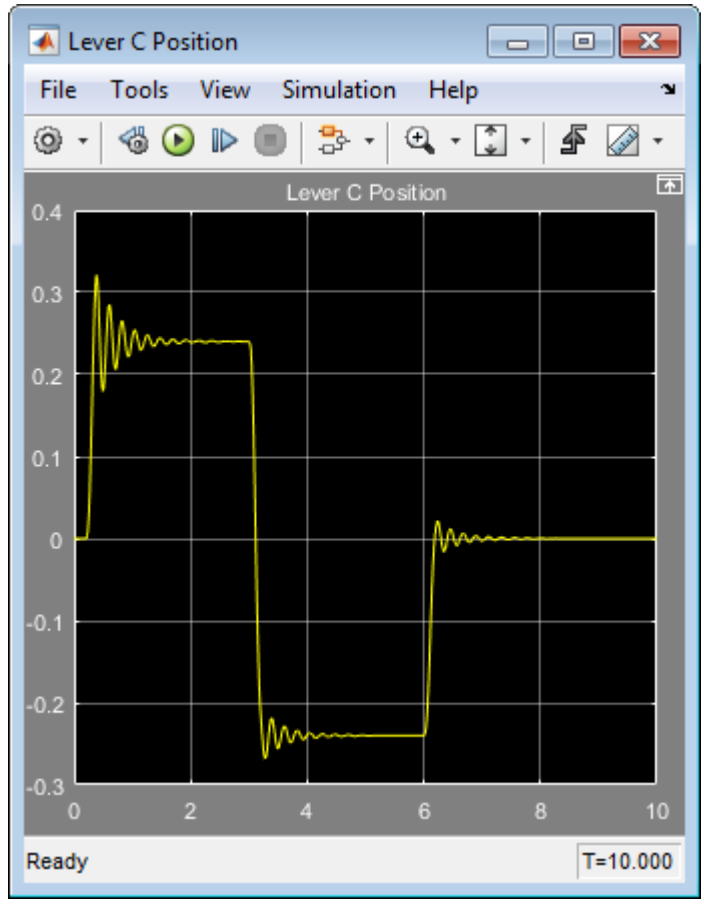
How to Simulate and Fine-Tune a Model in Restricted Mode

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

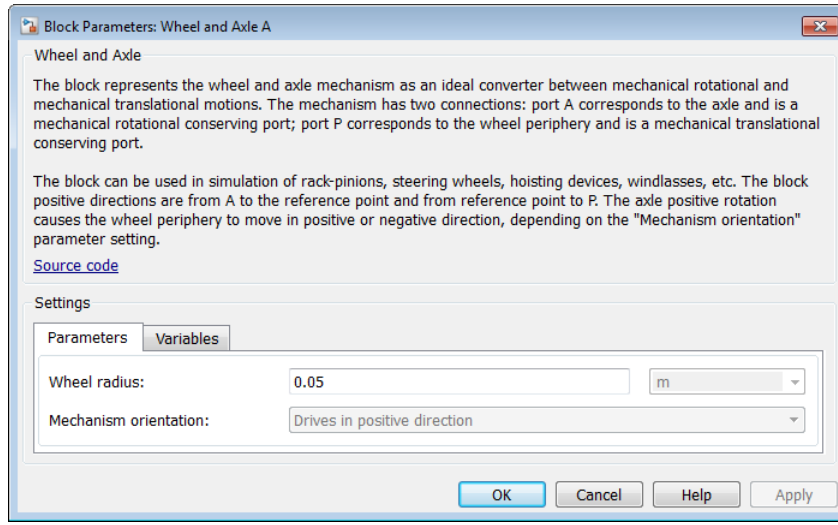
1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 15-12. The model opens in Restricted mode.



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

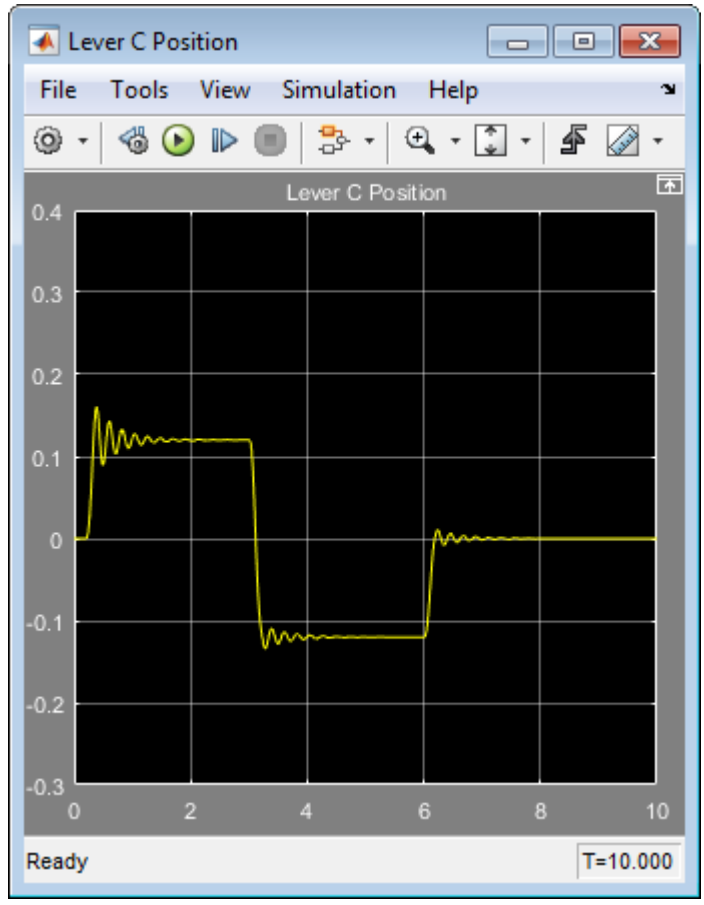


3 Double-click the Wheel and Axle A block to open its dialog box. Notice that the **Mechanism orientation** parameter is grayed out, because you cannot modify the block driving direction in Restricted mode.



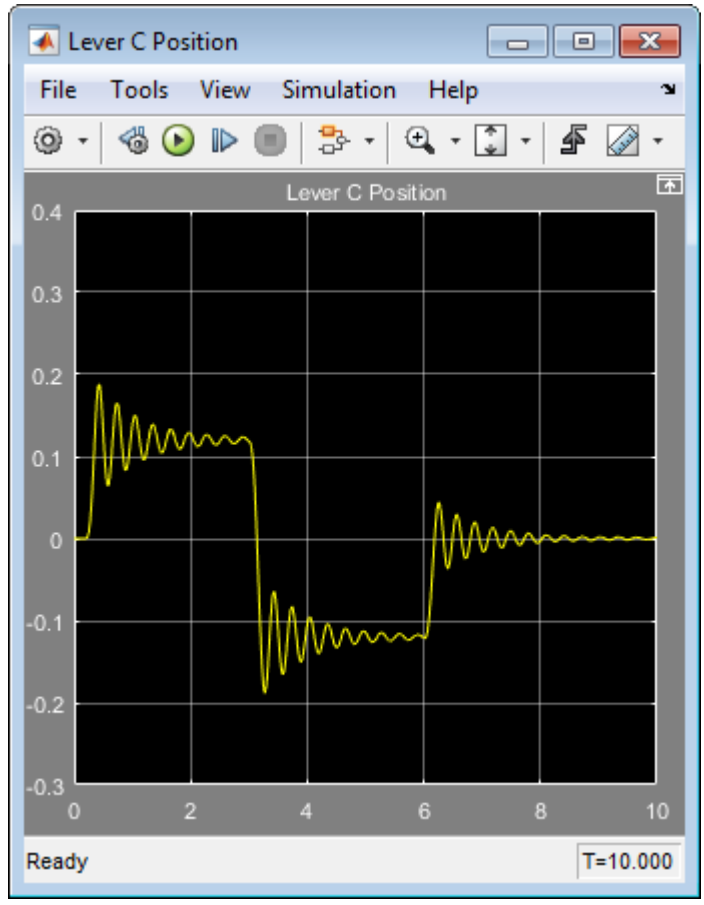
4 Change the **Wheel radius** parameter value to `0.1`.

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.

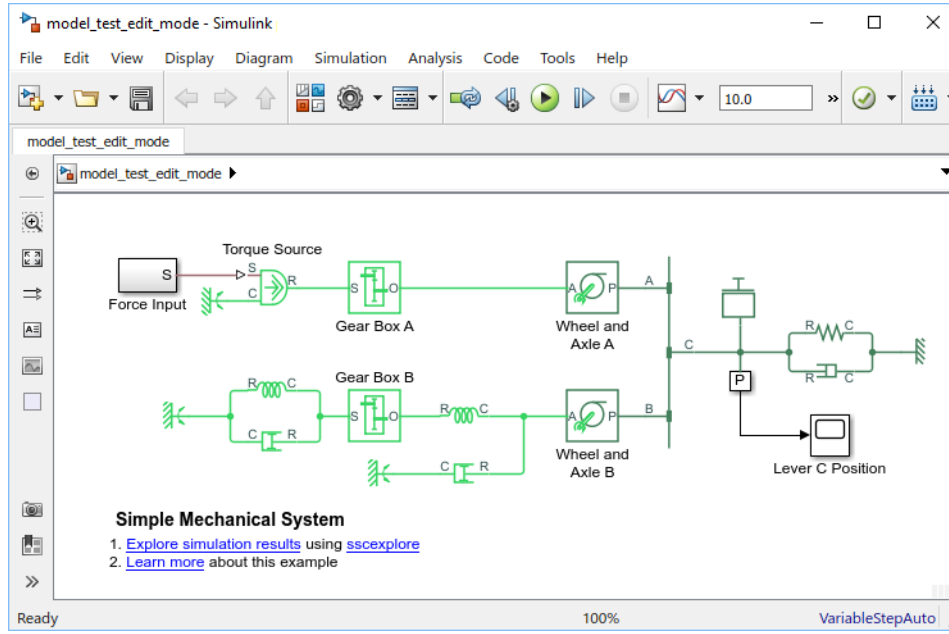
7 Simulate the model. Notice that doubling the mass resulted in increased vibrations.



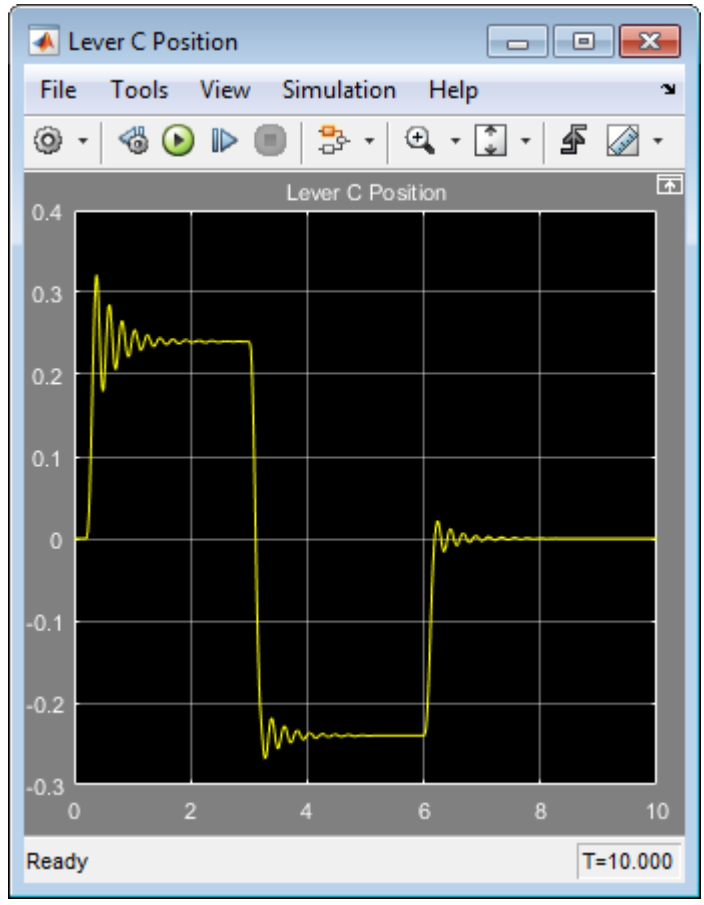
How to Add and Delete Simulink Blocks in Restricted Mode

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

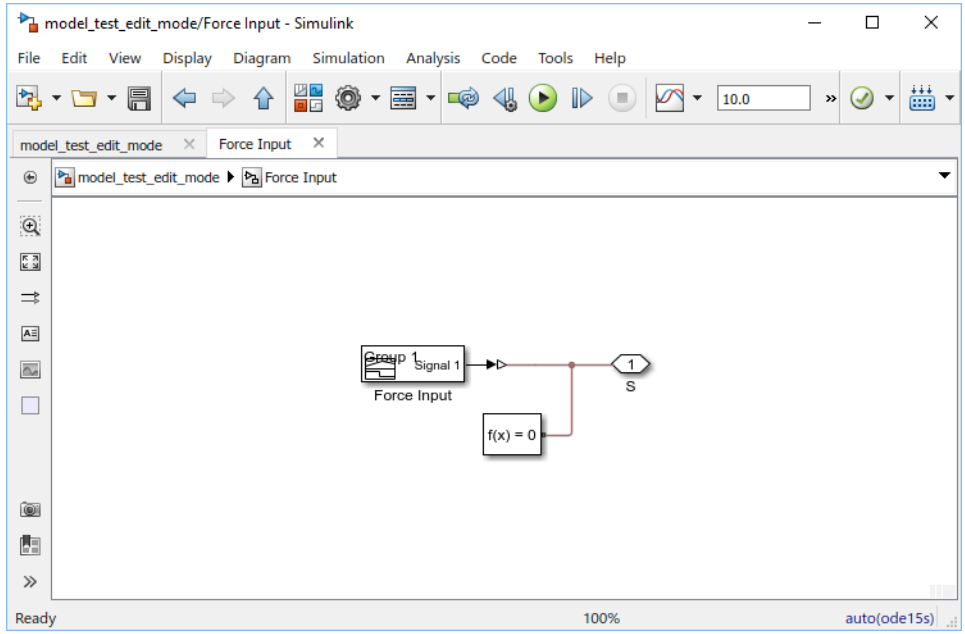
1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 15-12. The model opens in Restricted mode.



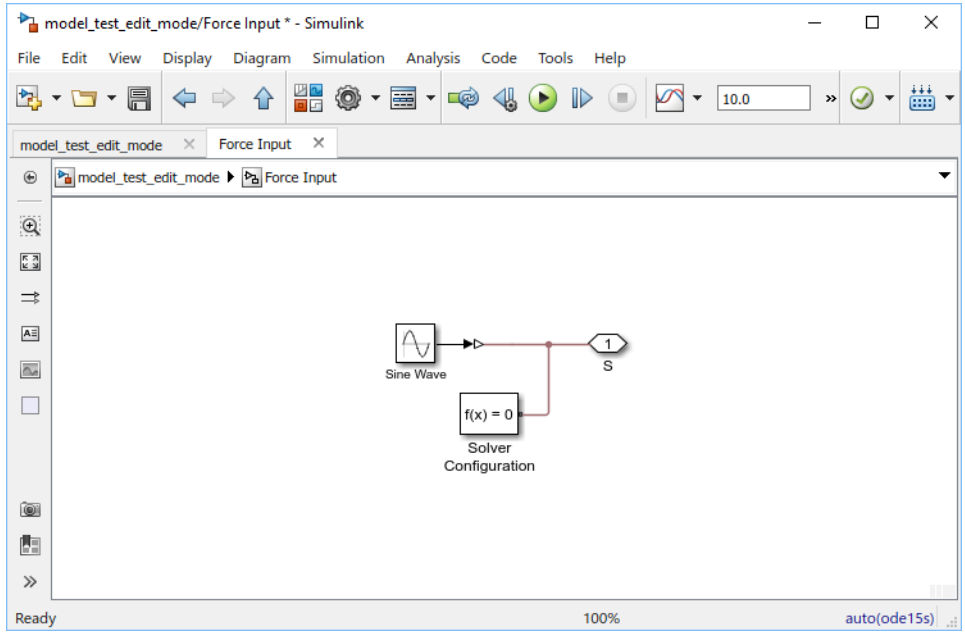
2 Open the Lever C Position scope and simulate the model.



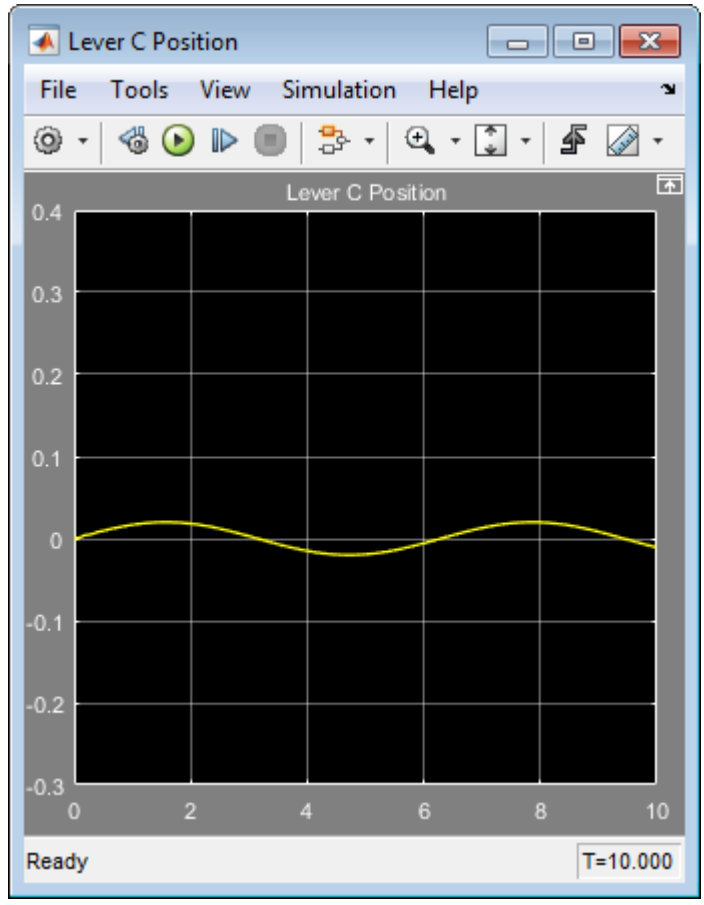
3 Double-click the Force Input subsystem to open it.



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.



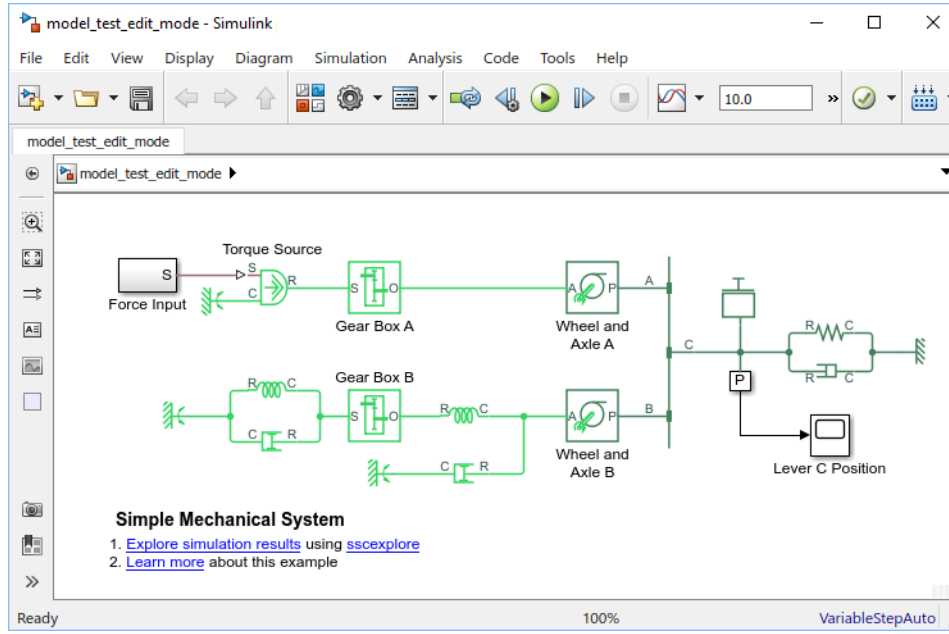
5 Simulate the model again. The model successfully compiles and simulates in Restricted mode.



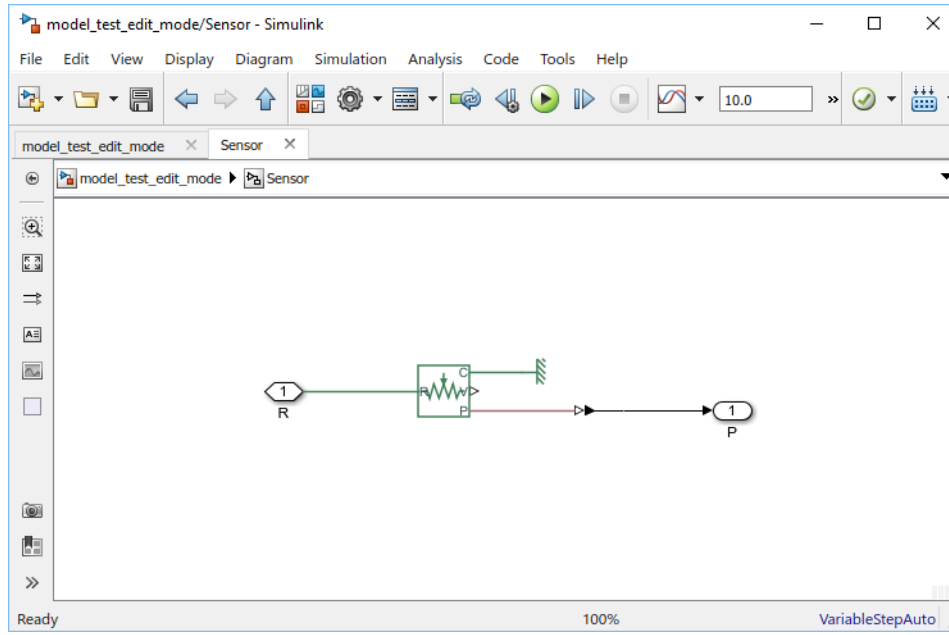
Performing an Operation Disallowed in Restricted Mode

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

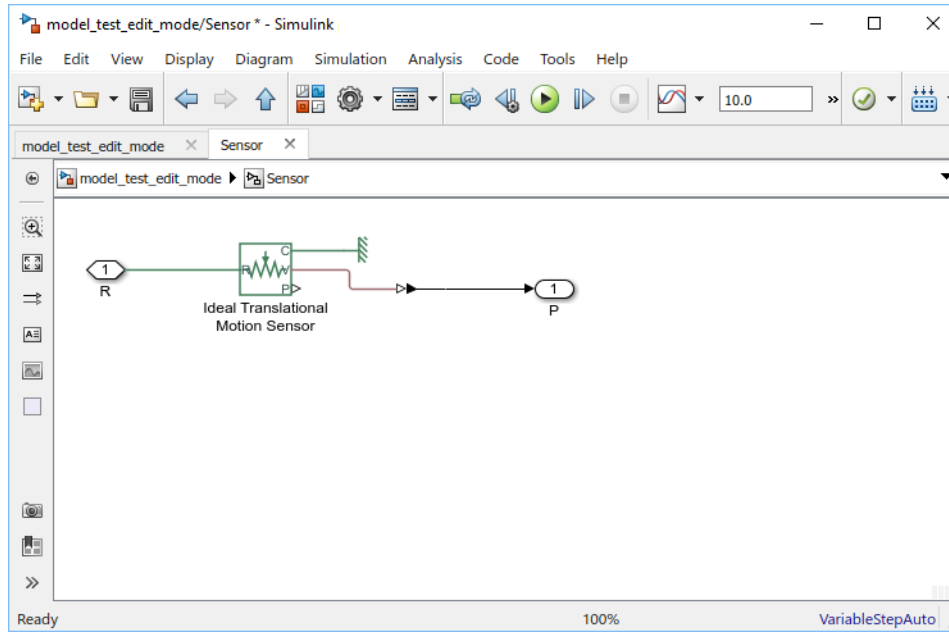
1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 15-12. The model opens in Restricted mode.



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 15-29.

See Also

Related Examples

- “Set the Model Loading Preference” on page 15-9
- “Save a Model in Restricted Mode” on page 15-11
- “Switch from Restricted to Full Mode” on page 15-29

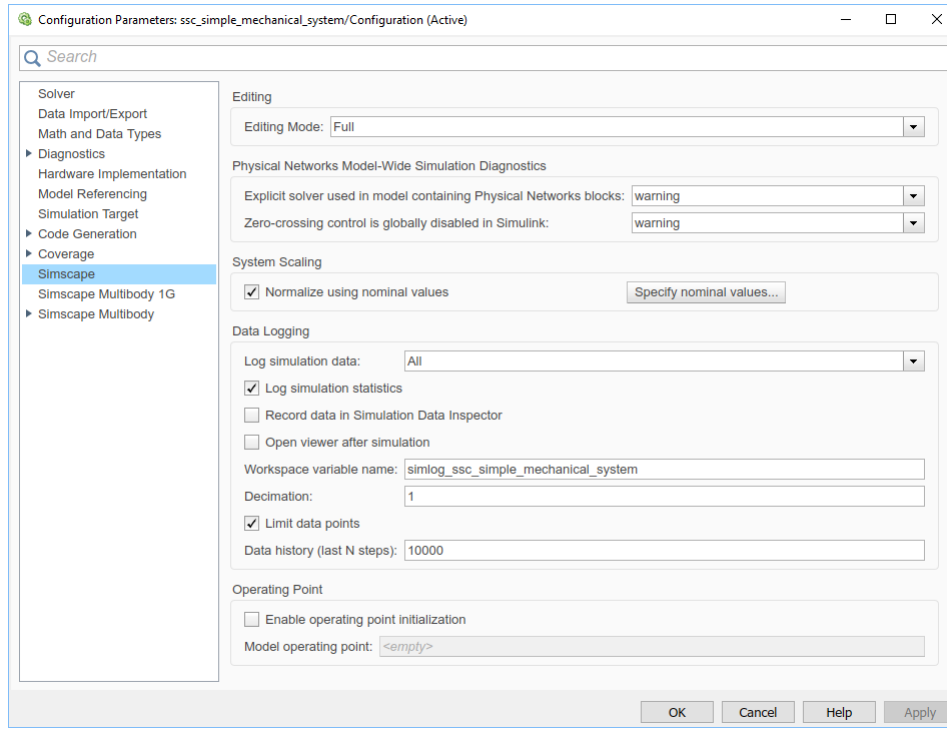
More About

- “About the Simscape Editing Mode” on page 15-2
- “Get Editing Mode Information” on page 15-31

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 15-6.

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.

See Also

Related Examples

- “Set the Model Loading Preference” on page 15-9
- “Save a Model in Restricted Mode” on page 15-11
- “Work with a Model in Restricted Mode” on page 15-15

More About

- “About the Simscape Editing Mode” on page 15-2
- “Get Editing Mode Information” on page 15-31

Get Editing Mode Information

In this section...
“What Is the Current Mode?” on page 15-31
“Which Licenses Are Checked Out?” on page 15-31

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.

See Also

Related Examples

- “Set the Model Loading Preference” on page 15-9
- “Save a Model in Restricted Mode” on page 15-11
- “Work with a Model in Restricted Mode” on page 15-15

- “Switch from Restricted to Full Mode” on page 15-29

More About

- “About the Simscape Editing Mode” on page 15-2