

Simscape™ Driveline™

User's Guide



MATLAB® & SIMULINK®

R2019b



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simscape™ Driveline™ User's Guide

© COPYRIGHT 2004–2019 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

September 2010	Online only	New for Version 2.0 Beta (Release 2010b+)
April 2011	Online only	Revised for Version 2.0 (Release 2011a)
September 2011	Online only	Revised for Version 2.1 (Release 2011b)
March 2012	Online only	Revised for Version 2.2 (Release 2012a)
September 2012	Online only	Revised for Version 2.3 (Release 2012b)
March 2013	Online only	Revised for Version 2.4 (Release 2013a)
September 2013	Online only	Revised for Version 2.5 (Release 2013b)
March 2014	Online only	Revised for Version 2.6 (Release 2014a)
October 2014	Online only	Revised for Version 2.7 (Release 2014b)
March 2015	Online only	Revised for Version 2.8 (Release 2015a)
September 2015	Online only	Revised for Version 2.9 (Release 2015b)
March 2016	Online only	Revised for Version 2.10 (Release 2016a) (Renamed from <i>SimDriveline™ User's Guide</i>)
September 2016	Online only	Revised for Version 2.11 (Release 2016b)
March 2017	Online only	Revised for Version 2.12 (Release 2017a)
September 2017	Online only	Revised for Version 2.13(Release 2017b)
March 2018	Online only	Revised for Version 2.14 (Release 2018a)
September 2018	Online only	Revised for Version 2.15 (Release 2018b)
March 2019	Online only	Revised for Version 2.16 (Release 2019a)
September 2019	Online only	Revised for Version 3.0 (Release 2019b)

Getting Started

1	Introducing Simscape Driveline Software	
	Simscape Driveline Product Description	1-2
	Key Features	1-2
	Related Products	1-3
	Required Products	1-3
	Other Related Products	1-3
	Drivetrain Model	1-4
	What the Model Represents	1-4
	What the Model Illustrates	1-4
	Open CR-CR Transmission Example Model	1-6
	Run the Model	1-8
	Modify the Model	1-11
	Capabilities of Simscape Driveline Software	1-18
	What Simscape Driveline Software Contains	1-18
	Model Driveline Systems	1-19
	Model Inertias and Gears	1-20
	Model Dynamic Driveline Elements	1-21
	Model Custom Driveline Elements	1-21
	Actuate and Sense Motion	1-21
	Simulate and Analyze Motion	1-22

Modeling Driveline Systems

2

Start a New Simscape Driveline Model	2-2
Build a Drivetrain Model	2-3

Complete Vehicle Model

3

Complete Vehicle Model	3-2
Understanding the Global Structure of the Model	3-2
Model the Throttle/Brake Profile	3-3
Model the Engine	3-3
Model the Transmission	3-4
Couple the Engine to the Transmission	3-5
Model the Tires, Brakes, Wheels, and Road	3-6
Control the Clutches	3-7
Run the Model	3-9

Basic Motion, Torque, and Force Modeling

4

Couple Rotational Motion with Gears	4-2
Gear Coupling Rules	4-2
Couple Two Spinning Inertias with a Simple Gear	4-3
Modeling Two Spinning Inertias	4-3
Coupling Two Spinning Inertias with a Simple Gear	4-6
Torque-Actuating Two Coupled, Spinning Inertias	4-7
Sensing and Actuating Motion and Torque	4-9
Couple Two Spinning Inertias with a Variable Ratio Transmission	4-11
Couple Three Spinning Inertias with a Planetary Gear	4-14

Driveline Actuation

5

Best Practices for Modeling Torque-Force Actuation and Motion Actuation	5-2
Actuate a Driveline Using Torques and Forces	5-3
Actuate a Driveline Using Motions	5-4
Set Initial Conditions of Driveline Motion	5-5
Resolving Undetermined Motions in Complex Gears	5-5

Power Transmission Using Pulley Networks

6

Best Practices for Modeling Pulley Networks	6-2
Belt Direction	6-2
Inertia	6-3
Belt Tension	6-3
Power Window System Pulley Mechanism	6-3

Gear Coupling Control Using Clutches

7

How a Clutch Works	7-2
Model Friction Clutches at a Fundamental Level	7-3
Engage and Disengage Gears Using a Clutch	7-4
Simulate Gear Engagement and Disengagement	7-4
How the Clutch Mode Indicates Locking and Unlocking	7-8
Brake Motion Using Clutches	7-9
Braking with a Two-Clutch System	7-9

Model Transmissions Using Gear Ratios and Clutch Schedules

8

Transmission Design Principles and Best Practices	8-2
Model a Two-Speed Transmission with Braking	8-3
Setting Up the Gears, Clutches, and Brake	8-4
Controlling the Transmission State with a Clutch Schedule . . .	8-5
Adding Realistic Clutch Signals	8-6
Model a CR-CR 4-Speed Transmission Driveline with Braking	8-7
Replacing Programmed with Manually Controlled Clutch Pressures	8-8

Modeling Driveline Components

9

Specialized and Customized Driveline Components	9-2
Optimal Physical Modeling in the Simscape Environment	9-2
Reasons for Specialized Driveline Components	9-2
Greater Model Fidelity and Performance	9-3
Rotational-Translational Couplings	9-5
Convert Between Rotational and Translation Motion	9-5
Use Simscape and Simscape Driveline Elements to Couple Rotation and Translation	9-5
Modeling Transmissions	9-7
Transmission Templates	9-7
Transmission Ports	9-7
Gear Input Signal	9-7
Initial States	9-8
Clutch Control	9-9
Inertias and Friction Losses	9-9
Real-Time Simulation	9-10

Effective Inertias and Driveshafts

10

Model a Variable Inertia	10-2
Model Driveshafts with Loss	10-4

Specialized Gears

11

Custom Planetary Gear Model	11-2
Model Gears with Losses	11-4
Constant Efficiency	11-4
Load-Dependent Efficiency	11-5
Geometry-Dependent Efficiency	11-5
Viscous Friction	11-5
Constant and Load-Dependent Gear Efficiencies	11-6

Specialized Clutches

12

Clutches, Clutch-Like Elements, and Coulomb Friction	12-2
Model Clutches with Viscous Friction Loss	12-3
Creating a Torque Damping Subsystem	12-3
Connecting and Simulating the Damped Clutch System	12-4
Model Realistic Clutch Pressure Signals	12-8
Automatic Transmission with a Dual Clutch	12-9
Predefined Simulation Options	12-10

Control Vehicle Velocity

13

Control Vehicle Throttle Input Using a Powertrain Blockset

Driver	13-2
Open-Loop Simulation Using a Signal Builder Block	13-2
Closed-Loop Simulation Using a Longitudinal Driver Block	13-4
Simulation Comparison	13-10

Drivetrain Disturbances

14

Model Drivetrain Noise	14-2
Model and Detect Drivetrain Faults	14-13

Modeling Driveline Environments

15

Model a Road Profile with Varying Elevation and Friction ...	15-2
Updates to the Original Model	15-2
Run the Simulation	15-9

Analyzing Driveline Models and Simulations

16

Driveline Simulation Performance	16-2
About Simulation Performance	16-2
Adjust Model Fidelity	16-2
Improve Simulation Performance by Using the Partitioning Solver	16-3
Optimize Simulation of Stiff Drivelines	16-3

Optimize Simulation of Clutches	16-4
Resolve Partitioning Solver Simulation Issues for Simscape	
Driveline Models	16-7
Resolving Issues for Blocks with Stiffness or Friction	16-7
Using the Partitioning Solver	16-7
Resolve Initial Condition Errors and Warnings	16-8
Reduce Chatter Due to Friction	16-17
Resolve Chatter Due to Stiffness	16-8
Driveline Degrees of Freedom	16-38
About Driveline Degrees of Freedom and Constraints	16-38
Identify Degrees of Freedom	16-38
Define Fundamental Degrees of Freedom	16-39
Define Connected Degrees of Freedom	16-41
Define Constrained Degrees of Freedom	16-42
Actuate, Sense, and Terminate Degrees of Freedom	16-46
Count Independent Degrees of Freedom	16-47
Count Degrees of Freedom in a Simple Driveline with a Clutch	16-48
Driveline States — Effect of Clutches	16-52
Driveline States and Degrees of Freedom	16-52
Find and Use Driveline States	16-53
How Simscape Driveline Simulates a Drivetrain System ...	16-55
About Simscape Driveline and Simscape Simulation	16-55
Clutch State Determination	16-55
Model Thermal Losses in Driveline Components	16-56
Thermal Ports	16-56
Thermal-Modeling Parameters	16-57
Model Thermal Losses for a Simple Gear	16-57
Simscape Driveline Limitations	16-64
Simscape Driveline and Simulink Limitations	16-64
Additional Simscape Driveline Limitations	16-64

17

**Prepare Simscape Driveline Models for Real-Time Simulation
Using Simscape Checks 17-2**

Troubleshoot Driveline Simulation Issues

18

Troubleshoot Driveline Modeling and Simulation Issues . . . 18-2

**Troubleshoot Overconstrained and Conflicting Degrees of
Freedom 18-3**

 Checking the Number of DoFs **18-3**

 Checking the Consistency of DoFs **18-3**

Troubleshoot Clutch and Transmission Errors 18-4

Troubleshoot Inconsistent Initial Conditions 18-5

Troubleshoot Pulley Network Issues 18-6

Troubleshoot Engine Issues 18-7

Getting Started

Introducing Simscape Driveline Software

- “Simscape Driveline Product Description” on page 1-2
- “Related Products” on page 1-3
- “Drivetrain Model” on page 1-4
- “Capabilities of Simscape Driveline Software” on page 1-18

Simscape Driveline Product Description

Model and simulate rotational and translational mechanical systems

Simscape Driveline provides component libraries for modeling and simulating rotational and translational mechanical systems. It includes models of worm gears, lead screws, and vehicle components such as engines, tires, transmissions, and torque converters. You can use these components to model the transmission of mechanical power in helicopter drivetrains, industrial machinery, automotive powertrains, and other applications. You can integrate electrical, hydraulic, pneumatic, and other physical systems into your model using components from the Simscape family of products.

Simscape Driveline helps you develop control systems and test system-level performance. You can create custom component models with the MATLAB® based Simscape language, which enables text-based authoring of physical modeling components, domains, and libraries. You can parameterize your models using MATLAB variables and expressions, and design control systems for your physical system in Simulink®. To deploy your models to other simulation environments, including hardware-in-the-loop (HIL) systems, Simscape Driveline supports C-code generation.

Key Features

- Gear models, including planetary, differential, and worm gears with meshing losses, viscous losses, and thermal effects
- Clutch models, including cone, disk friction, synchronizer, unidirectional, and dog clutch
- Vehicle component models, including engine, tire, torque converter, and vehicle dynamics
- Models of translational elements, including leadscrew, rack and pinion, and translational friction
- MATLAB based Simscape language for creating custom component models
- Physical units for parameters and variables, with all unit conversions handled automatically
- Support for C-code generation (with Simulink Coder™)

Related Products

In this section...
“Required Products” on page 1-3
“Other Related Products” on page 1-3

Required Products

To use the Simscape Driveline product, you must have installed current versions of the following products:

- MATLAB
- Simulink
- Simscape

Other Related Products

On the MathWorks website, on the Simscape Driveline product page, the related products that are listed include toolboxes and blocksets that extend the capabilities of MATLAB and Simulink. These products can enhance Simscape Driveline modeling and simulation in various applications.

Physical Modeling Product Family

Use the Physical Modeling product family to model physical systems in Simulink. In addition to Simscape Driveline software, the product family includes:

- Simscape, the platform and unifying environment for Physical Modeling products.
- Simscape Electrical™, for modeling and simulating electronic, mechatronic, and electrical power systems.
- Simscape Fluids™, for modeling and simulating hydromechanical systems.
- Simscape Multibody™, for modeling and simulating mechanical systems.

For Information About MathWorks Products

- If you have the product installed, see the online documentation for that product.
- See the “Products” section at the MathWorks website at www.mathworks.com.

Drivetrain Model

In this section...

“What the Model Represents” on page 1-4

“What the Model Illustrates” on page 1-4

“Open CR-CR Transmission Example Model” on page 1-6

“Run the Model” on page 1-8

“Modify the Model” on page 1-11

What the Model Represents

The model `sd1_transmission_4spd_crcr` simulates a complete drivetrain. This example helps you understand how to model driveline components with Simscape Driveline blocks, connect them into a realistic model, use Simulink blocks and variant subsystems in driveline modeling, and simulate and modify a drivetrain model.

This driveline mechanism is part of a full vehicle, without the engine or engine-drivetrain coupling, and without the final differential and wheel assembly. The model includes an actuating torque, driver and driven shafts, a four-speed transmission, and a braking clutch.

For a complete vehicle model that uses this drivetrain, see the `sd1_car` example model and “Complete Vehicle Model” on page 3-2.

What the Model Illustrates

The `sd1_transmission_4spd_crcr` model contains a driveline that accepts a driving torque. The driveline system transfers this torque and the associated angular motion from the input or drive shaft to an output or driven shaft through a transmission. The model includes a CR-CR (carrier-ring-carrier-ring) four-speed transmission subsystem, based on two gears and four clutches. (The example does not use the reverse gear in the CR-CR transmission.) You can set the transmission to four different gear combinations, allowing four different effective torque and angular velocity ratios. A fifth clutch, outside the transmission, acts as a brake on the driven shaft.

The transmission subsystem illustrates a critical feature of transmission design, the *clutch schedule*. To be fully engaged, the transmission, with four clutches and two

planetary gears, requires two clutches to be locked and the other two unlocked at any time. (The transmission reverse clutch is not applicable here.) The choice of which two clutches to lock determines the effective gear ratio across the transmission. The clutch schedule is the relationship shown in the table of locked and free clutches corresponding to different gear settings. If all four clutches are unlocked, the transmission is in neutral. If the clutches are disengaged, no torque or motion at all is transferred across the transmission.

Clutch Schedule for the CR-CR 4-Speed Transmission

Gear Setting	Clutch A State	Clutch B State	Clutch C State	Clutch D State	Clutch R State	Drive Ratio
1	<i>L</i>	F	F	<i>L</i>	F	$1 + g_o$
2	<i>L</i>	F	<i>L</i>	F	F	$1 + g_o/(1 + g_i)$
3	<i>L</i>	<i>L</i>	F	F	F	1
4	F	<i>L</i>	<i>L</i>	F	F	$g_i/(1 + g_i)$
Reverse	F	F	F	F	<i>L</i>	$-g_i$

- *L* = locked
- F = free
- g_i = Input Planetary Gear ring-to-sun gear ratio
- g_o = Output Planetary Gear ring-to-sun gear ratio

Clutch Control Variant Subsystem

A Variant Subsystem block governs transmission gear changes. This block, named Clutch Control, contains two child subsystem blocks that provide different clutch control modes, or *variants*:

- Manual — Manually switch transmission clutches.
- Programmed — Automatically switch transmission clutches according to a programmed clutch schedule.

During simulation, one variant becomes active while the other does not. The choice of active variant determines which child subsystem controls the gear changes. By default, the Programmed variant is active and gear changes follow a programmed clutch schedule. To switch gears manually during simulation, change the active variant to Manual.

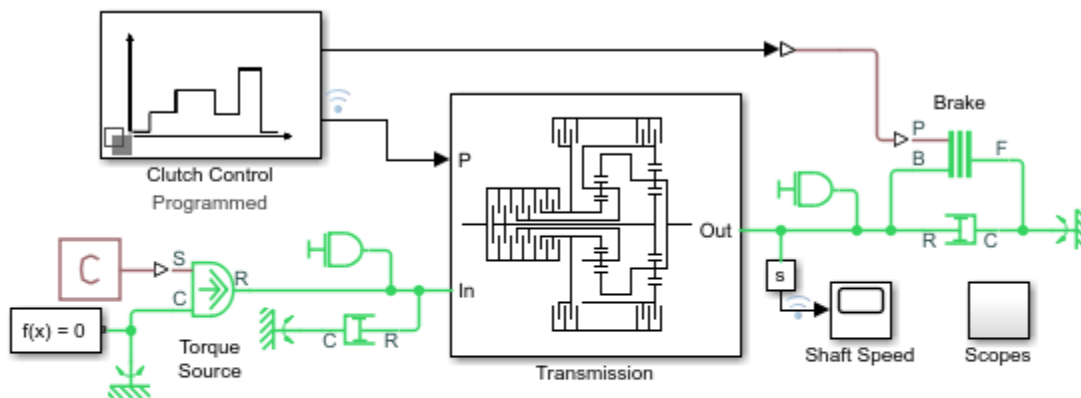
Open CR-CR Transmission Example Model

To open the CR-CR transmission example model, at the MATLAB command prompt, enter `sdl_transmission_4spd_crcr`

Block Diagram Model

Examine the model and its structure. The main model window contains the transmission subsystem, the input shaft assembly, and the output shaft assembly. Each assembly consists of a driveline axis with applied damping and inertia torques. Each driveshaft balances the torques applied across its ends with the damping and inertia forces. A net torque is transmitted along the driveline.

The main model also includes a brake clutch. When this clutch is locked, the shaft slows, but doesn't necessarily stop. The transmission can be engaged at the same time as the brake. If the transmission is engaged, the clutch remains unlocked.



CR-CR Four-Speed Transmission

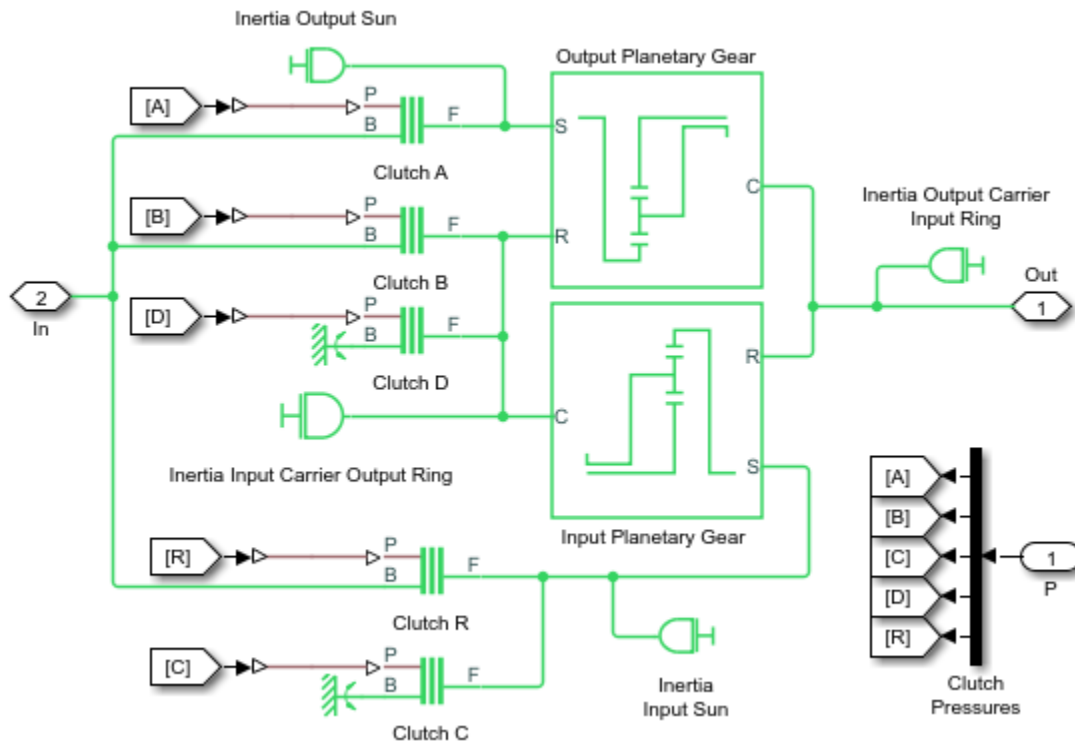
1. [Plot speeds](#) of shafts ([see code](#))
2. Configure Clutch Control: [Programmed](#), [Manual](#)
3. [Explore simulation results](#) using [sscexplore](#)
4. [Learn more](#) about this example

Main Model Window

What the Model Contains—Opening the Subsystems

Open each subsystem.

The transmission subsystem contains four clutches, two planetary gears, and four inertias (rotating bodies). Ignoring the reverse gear and its clutch, this transmission has four possible (forward) gear settings. Exactly two clutches must be locked at any one time for the transmission to engage and to avoid conflicting constraints on the gear motions.

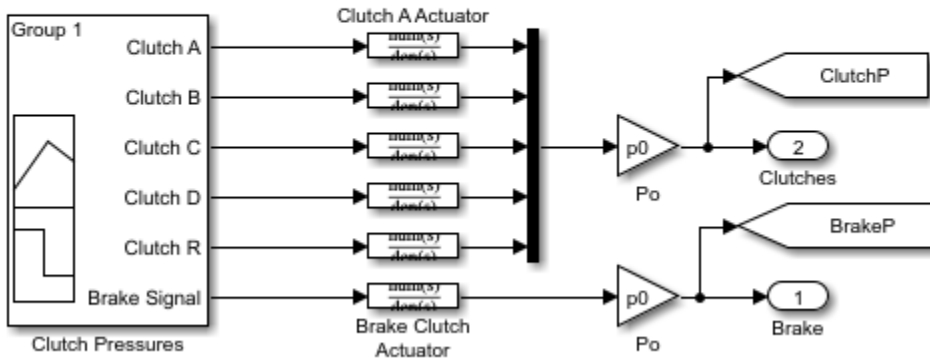


CR-CR 4-Speed Transmission Subsystem

The Clutch Control variant subsystem provides the pressures that lock the necessary clutches. In its default state, the clutch controller is programmed to move the transmission through a fixed sequence of gears, then unlock all the transmission clutches. This control program allows the driven shaft to “coast” for a time, and then engage and lock the brake clutch to stop the driven shaft.

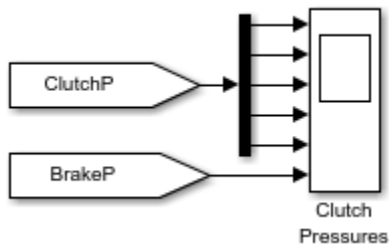
Programmed Clutch Control for the CR-CR Transmission

Use this subsystem to control the CR-CR transmission with a preprogrammed schedule of clutch pressures.



Clutch Control Subsystem

The Scopes subsystem provides Scope blocks to display the clutch pressure and the input and output shaft velocity signals.

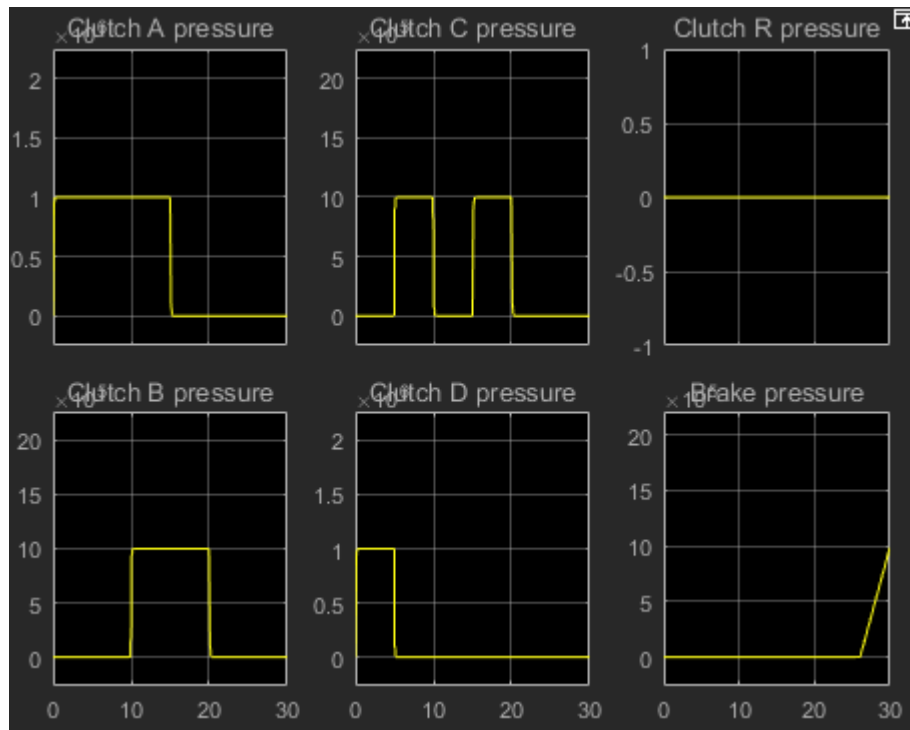


Scopes Subsystem

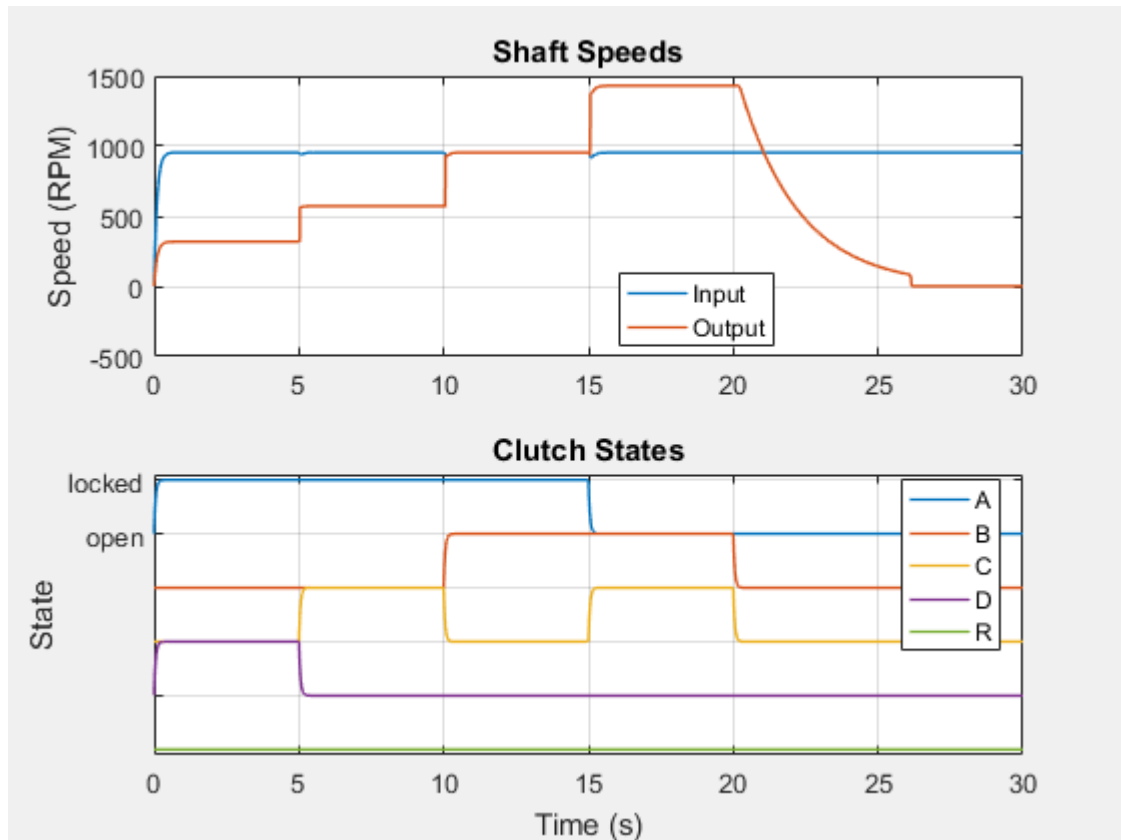
Run the Model

To display the CR-CR driveline model behavior:

- 1 Open the Scopes subsystem and then the Scope block. Close the Scopes subsystem.
- 2 Click **Start**. The model steps through the gears and then brakes.
- 3 Observe how the clutch pressure signals move the transmission into one gear after another, at 0, 5, 10, and 15 seconds of simulation time. To determine which gear settings the model is implementing, compare these clutch pressure signals to the clutch schedule in the CR-CR transmission subsystem. The model steps through gears 1, 2, 3, and 4, before coasting and then braking.

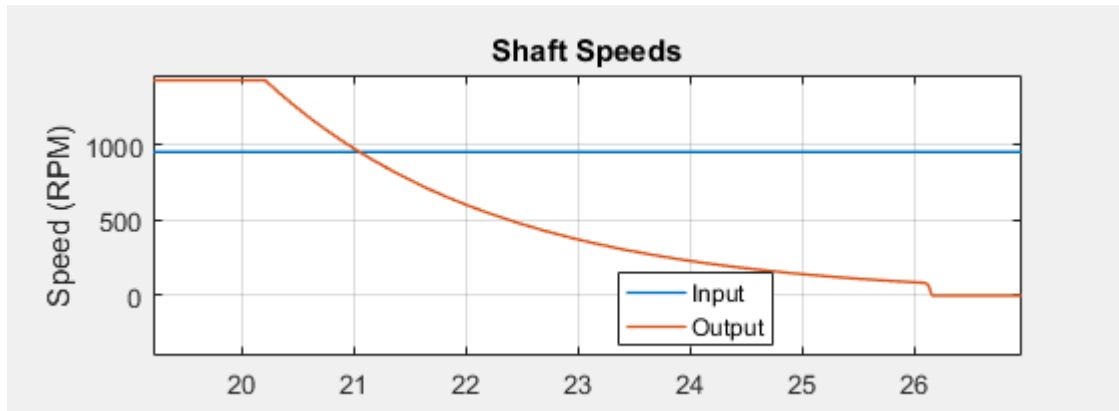


- 4 To compare the angular velocities of the input and output shafts, click **Plot speeds**. A figure that contains the Shaft Speeds and Clutch States opens.



In the transmission, the two planetary gears are coupled in different ways in the different gear settings, producing different relationships between the driven and driver shaft velocities. The effective *drive ratio* of output to input shafts is the *reciprocal* of the ratio of output to input angular velocities.

- 5 Zoom to observe the results for the shaft speeds 20-26 seconds.



The transmission clutch pressures drop to zero, and the transmission disengages. The transmission ceases to transfer angular motion and torque from the driver to the driven shaft, and the driven shaft continues to spin from inertia alone. A small kinetic friction damping gradually slows the driven shaft over the next six seconds.

At 26 seconds of simulation time, the brake clutch pressure begins to rise from zero, and the brake clutch engages. The driven shaft decelerates more drastically now. 26.0–26.2 seconds, the brake clutch locks, and the driven shaft stops rotating completely.

Modify the Model

You can modify this example model to explore other Simscape Driveline features. Here you modify and rerun the model to investigate two aspects of its motion.

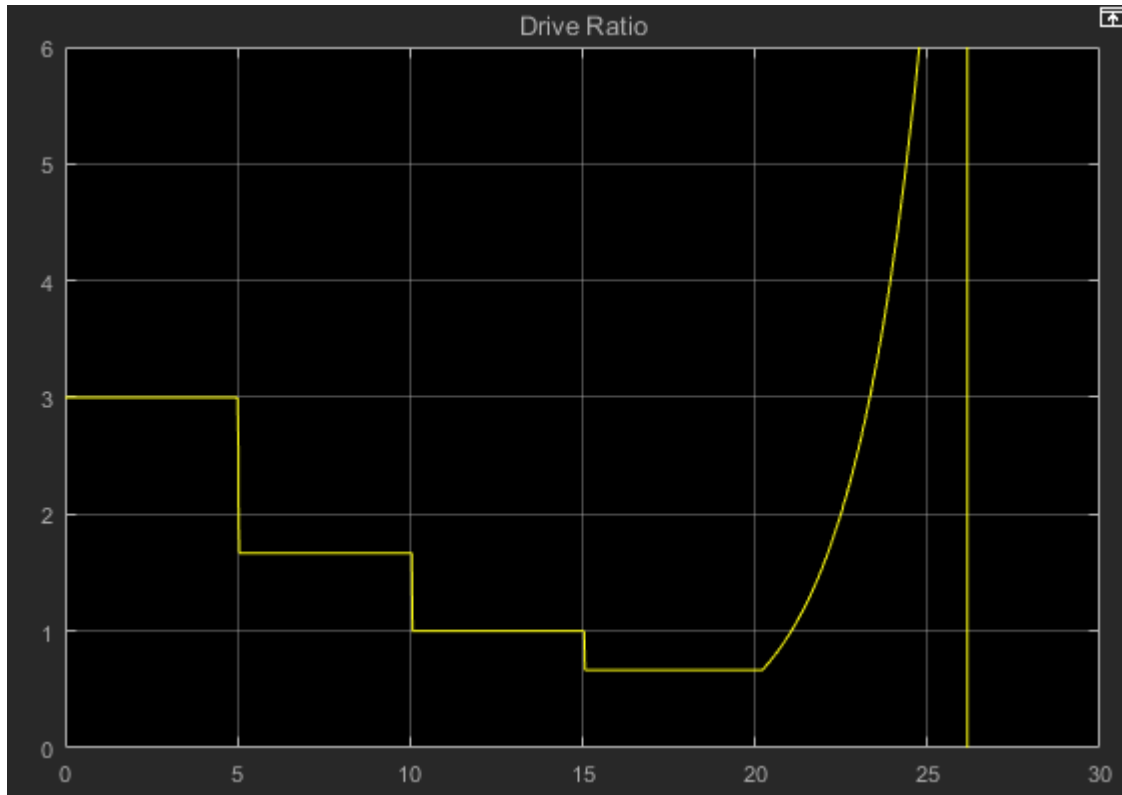
- Measure the effective drive ratio of the CR-CR transmission in each gear setting that it steps through.
- Change the gear sequence.

Measuring the Drive Ratio of the CR-CR Transmission States

A transmission is a set of coupled gears. For a particular transmission gear setting, the ratio of driven (output) shaft velocity to the driver (input) is fixed. The reciprocal ratio, the *drive ratio*, is like a gear ratio of an individual gear coupling, but for the whole transmission.

The drive ratio is the ratio of input to output shaft velocities. Add and connect the necessary Simulink blocks to measure the drive ratio for the CR-CR 4-speed transmission.

- 1** Collect data for the angular velocity for the driver shaft:
 - a** Make a copy of the S sensor subsystem that is connected to the Out port of the transmission subsystem. The output sensor captures the angular velocity of the driven shaft.
 - b** Connect the new sensor subsystem to the connector between the input shaft assembly and the In port of the transmission subsystem.
- 2** To calculate the drive ratio, from the Simulink Library Browser, from the **Simulink > Math Operations** library, add a Divide block.
- 3** To visualize the drive ratio, add and configure a Scope block:
 - a** Make a copy of the Shaft Speed scope block.
 - b** Change the name the new scope block to Drive Ratio.
 - c** Open the Drive Ratio block.
 - d** Open the configuration parameters for the scope.
 - e** On the **Display** tab, set the Y-limits (Minimum) to 0 and the Y-limits (Maximum) to 6.
 - f** Connect the block as shown in the figure.
 - g** Label the input signal to the Drive Shaft block as Drive Ratio.



- 5 Consult the table, Clutch Schedule for the CR-CR 4-Speed Transmission. Check the drive ratios for each gear, 1, 2, 3, and 4, in terms of the gear ratios of the two Planetary Gears in the transmission. Determine the numerical values for these drive ratios for gear settings 1, 2, 3, and 4. Then check them against the values displayed in the Drive Ratio scope.

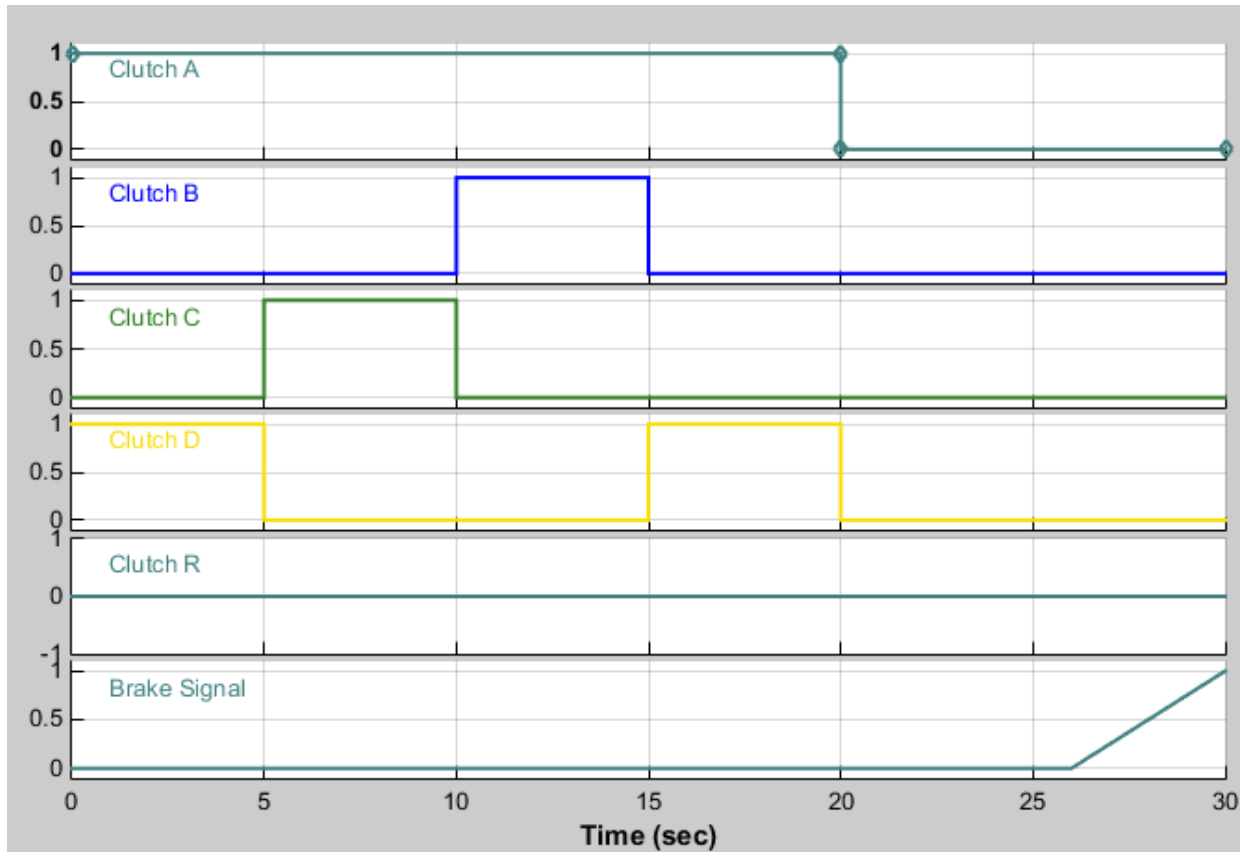
The drive ratio sequence is 3, 5/3, 1, and 2/3, respectively, for the first, second, third, and fourth intervals of five seconds each.

Changing the Transmission Gear Sequence

When you first open the `sdl_transmission_4spd_crcr` example, the Clutch Control variant subsystem is programmed to step through CR-CR gear settings 1, 2, 3, and 4, before disengaging. Modify it to step through settings 1, 2, 3, and 1, then disengage. The fourth gear requires that A is free, B is locked, C is locked, and D is free. Modify the

clutch pressure signal sequence from 15 to 20 seconds so that the transmission is set in first, not fourth, gear. The first gear requires clutches that A and D are locked and clutches B and C are free.

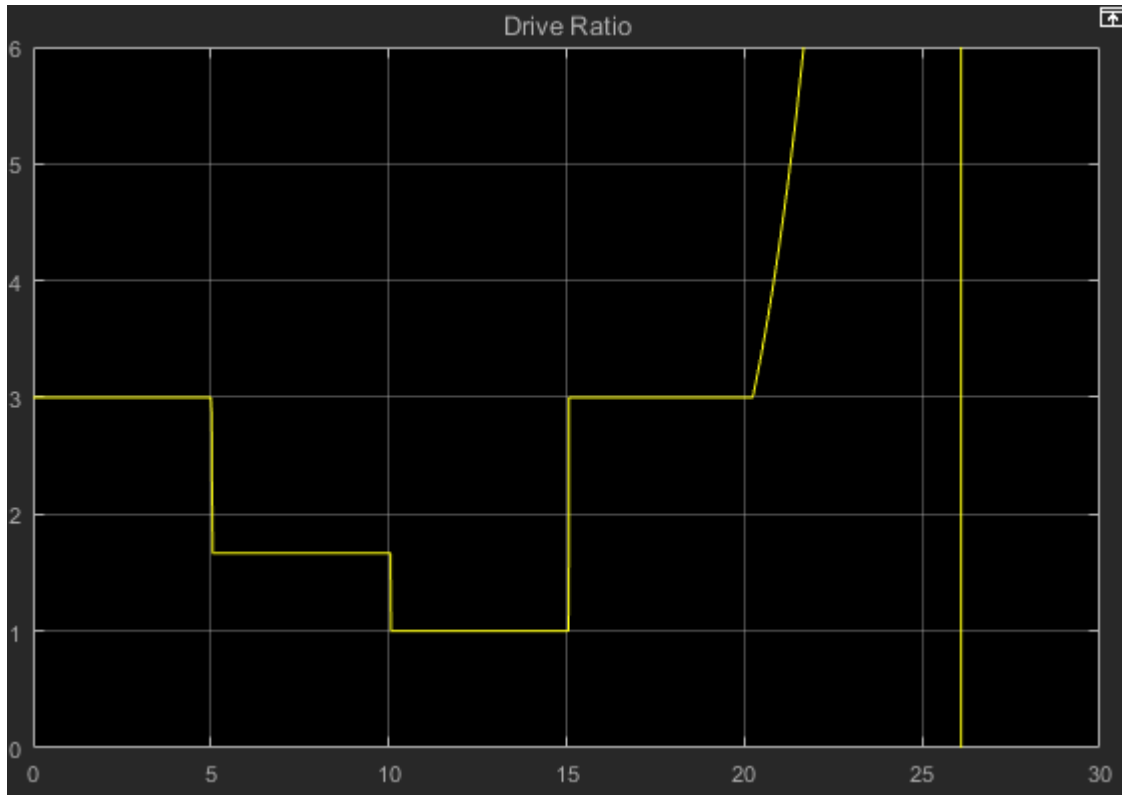
- 1** Determine the clutch states that correspond to first gear. Refer to table Clutch Schedule for the CR-CR 4-Speed Transmission.
- 2** Double-click the Clutch Control subsystem.
- 3** In the Clutch Control subsystem, double-click Programmed.
- 4** In the Programmed subsystem, double-click Clutch Pressures. The signal builder window opens with the clutch pressure signals.
- 5** In the time interval 15-20 seconds, update clutch signals A through D to match first gear. Clutches A and D must lock, while clutches B and C must remain free. Specify a signal value of one to lock a clutch, zero to unlock it.



Modified CR-CR 4-Speed Transmission Clutch Pressures

- 6 Run simulation.

Clutch pressures, clutch modes, and driven shaft velocities in the time interval 15-20 seconds now correspond to first gear. Refer to the Drive Ratio plot for the updated model. The ratio has changed from 2/3 (fourth gear) to 3 (first gear) accordingly.



Capabilities of Simscape Driveline Software

In this section...

“What Simscape Driveline Software Contains” on page 1-18

“Model Driveline Systems” on page 1-19

“Model Inertias and Gears” on page 1-20

“Model Dynamic Driveline Elements” on page 1-21

“Model Custom Driveline Elements” on page 1-21

“Actuate and Sense Motion” on page 1-21

“Simulate and Analyze Motion” on page 1-22

What Simscape Driveline Software Contains

Simscape Driveline software is a set of block libraries in the Simulink environment and based on Simscape software. You connect Simscape Driveline blocks to normal Simulink blocks through Simscape physical signal blocks that define physical units.

The blocks in the Simscape Driveline library and the related mechanical blocks in the Simscape Foundation library are the elements to model driveline systems. These systems consist of one or more inertias and masses, rotating about or translating along one or more axes, constrained to rotate or translate together by gears, which transfer torque and forces to different parts of the driveline. You can represent drivelines with components organized into hierarchical subsystems, as in any Simulink model. You can:

- Constrain motion with gears.
- Add complex dynamic elements such as clutches, clutch-like elements, and other couplings.
- Represent such vehicle components as bodies and tires.
- Actuate bodies with torques, forces, and motions.
- Integrate the Newtonian rotational and translational dynamics, then measure the resulting motions.

Relation to Simscape Software


To model and simulate physical systems, Simscape Driveline models use such Simscape technologies as nondirectional physical connections and conserving ports, physical


signals carrying physical units, custom component modeling, specialized solvers, and data logging.

The Simscape mechanical rotational and translational domains form the basis of the Simscape Driveline block libraries and models. The Simscape Foundation library (Simscape) includes physical signal blocks (Simscape); and basic mechanical blocks (Simscape) representing inertia, mass, and simple mechanical couplings. It also includes motion, torque, and force sources and sensors.

For more about modeling and simulating in the Simscape environment, see the “Simscape” documentation.

Physical Connections, Mechanical Conserving Ports, and Physical Signals

On Simscape Driveline blocks, the mechanical conserving ports  anchor physical connection lines that, in this domain, represent mechanical axes. These axes are either rotation axes along which torque is transferred and around which inertias rotate, or translation axes along which force is transferred and along which masses translate.

Certain blocks defined in Simscape domains also require input or output signals that carry physical units, or physical signals. Simscape physical signal lines and ports  represent and connect physical signals with units. Conversion blocks allow you to convert dimensionless Simulink signals to Simscape physical signals, and back.

Model Driveline Systems

Simscape Driveline software extends Simulink and Simscape software with blocks to model driveline components and properties, represent drivelines as physical networks, and to solve the equations of motion.

To build and run a Simscape Driveline model representation of a driveline:

- 1** Specify rotational inertia or translational mass for each body. Connect the bodies with physical connection lines representing driveline axes at mechanical conserving ports.

If needed, ground the driveline to one or more mechanical references fixed in space.
- 2** Constrain the driveline axes to rotate or translate together by connecting them with gears. Gears impose static constraints on driveline motions and transfer torques and forces at fixed ratios.
- 3** As necessary, add dynamic elements that transfer torque, force, and motion among driveline axes in a nonstatic way. These elements include internal torque-generating

components such as damped springs, clutches, clutch-like elements, transmissions, and torque converters. You can also construct and connect your own dynamic elements.

Similarly, add dynamic sources and environmental interactions, such as engines, vehicle bodies, and tires.

- 4 Set up mechanical sources and sensors to initiate and record body motions, and to apply external torques and forces to the driveline.
- 5 Connect the Simscape Solver Configuration to the driveline, then configure it. Start the simulation, calling the Simulink and Simscape solvers to find the motions of the system. Display and analyze the motion.

Model Inertias and Gears

Simscape Driveline software defines a driveline as a collection of rotating and translating bodies, defined by their rotational inertias and translational masses. Rotational and translational degrees of freedom (DoFs) originate on inertias and masses, but are carried by physical connection lines. Directly connecting one body to another constrains both bodies to rotate at the same angular or linear velocity. A torque or force applied to one body is applied to both. You can also ground driveline axes to mechanical references that do not move and that represent infinite effective inertia or mass.

Note All Simscape Driveline DoFs are absolute in an implicit global coordinate system at rest, but are measured and used in a relative way, between one component and another. To measure regarding the global rest frame, ground sensors or other components with mechanical reference blocks.

In a real driveline, the bodies can also be connected indirectly by gears that couple driveline axes. The gears constrain the axes to rotate together. These gears can be simple or complex and can couple two or more axes. The gears have two roles:

- Constraining the connected axes to rotate or translate together at velocities in fixed ratio or ratios.
- Transferring the torques or forces flowing along one or more axes to other axes, also in fixed ratio or ratios.

Model Dynamic Driveline Elements

To create more realistic driveline models, you elaborate on simple drivelines consisting of inertias, masses, and gears. You add complex mechanical elements that generate torques and forces internally within the driveline, between one axis and another. Certain Simscape Driveline blocks encapsulate as subsystems entire models of complex driveline elements:

- Load-dependent loss models of nonideal gears.
- Clutches and clutch-like elements that model the locking and unlocking of pairs of driveline axes by applying kinetic and static friction
- Vehicle component models that represent engines, tires, and vehicle bodies
- Specialized torque and force models, such as torque converters, hard stops, and damped spring-like torsion

Model Custom Driveline Elements

The blocks provided in the Simscape Foundation library can serve as starting points for developing variant or entirely new models to simulate the same components. You can also study masked subsystems by looking under their masks. If necessary, break the link between the block and the library before modifying it, and then create your own version. Or, create your own new blocks using Simscape Driveline and Simscape components, or with the Simscape language.

For more information on specialized driveline components, see “Specialized and Customized Driveline Components” on page 9-2.

Actuate and Sense Motion

Simscape motion sources and sensors are the blocks that you use to insert and extract basic kinematic and dynamic information:

- Source blocks impart motion to driveline axes and impose externally defined torques and forces on the bodies of a driveline.
- Sensor blocks measure the motions of, and the torques and forces transferred along, the axes of a driveline system.

Source inputs and sensor outputs (Simscape) are physical signals that carry units.

Simulate and Analyze Motion

Once you specify all the rotational inertias and translational masses of the bodies and interconnect the bodies with gears and other driveline elements, the dynamic problem of finding the system motion is solvable. To finish a driveline model and prepare it for simulation, you connect the driveline to the Simscape solver. This solver defines certain aspects of the simulation and integrates the Newtonian dynamics for the system, applying all internal and external torques and constraints to find the motions of the bodies.

Once your model is ready for simulation, run it and analyze its motions, torques, and forces.

Inverse Dynamics – Trimming and Linearization

You typically do not know the torques and forces necessary to produce a given set of motions. By motion-actuating your driveline with motion sources and measuring the resulting torques and forces, you can find the torques and forces required to produce specified motions. This technique inverts the canonical approach to dynamics, which consists of finding motions from torques and forces.

A special case of inverse dynamics is *trimming*. This technique involves searching for steady-state motions of the bodies, when their accelerations and the torques and forces they experience vanish. Using the specialized tools in Simscape and Simulink, you can perturb such a steady motion state slightly to find how the system responds to small disturbances. The response indicates the system stability and suitability for controllers.

Generating Code – Real-Time and Hardware-in-the-Loop Simulation

Simscape Driveline software is compatible with Simulink Acceleration modes, Simulink Coder, and Simulink Real-Time™ software. With these products, you can generate code versions of the models that you originally create in Simulink with block diagrams, enhancing simulation speed and model portability. A common application of generated code is defining real-time and hardware-in-the-loop simulations.

The presence of clutches in a driveline model induces locking-unlocking iterations and dynamic discontinuities. These discontinuities place certain restrictions on code generation. For more information about these restrictions, see “Driveline Simulation Performance” on page 16-2 and “Simscape Driveline Limitations” on page 16-64.

Modeling Driveline Systems

- “Start a New Simscape Driveline Model” on page 2-2
- “Build a Drivetrain Model” on page 2-3

Start a New Simscape Driveline Model

You can use the Simscape model template as a starting point for your Simscape Driveline models. The template provides the required Solver Configuration block and the commonly used Simulink-PS Converter and PS-Simulink Converter blocks.



To open the Simscape model template, at the MATLAB command prompt, enter `ssc_new`. Simscape software opens the model template along with the Foundation block library. To open the Simscape Driveline block library, enter `sd_l lib`.

Build a Drivetrain Model



The example model in “Drivetrain Model” on page 1-4 illustrates a typical drivetrain system that you can model with Simscape Driveline software. It also illustrates the key rules for connecting driveline blocks to each other and the dual roles of Simscape physical connection lines in driveline modeling. Within the Simscape mechanical domain:

- The *across* variable is angular or linear velocity, depending on the type of mechanical ports you are connecting to, rotational or translational. Along any connection line, the velocity is the same.
- Depending on the type of mechanical ports, you are connecting to, the *through* variable is torque or force. Torques and forces are conserved along a connection line and sum to zero at line branch points.

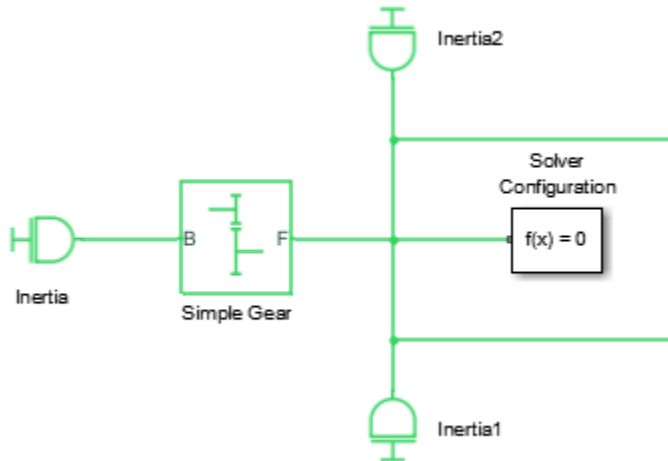
Before building and running the tutorial models, review these rules.

- Driveline blocks feature mechanical conserving ports  and, in some cases, physical signal ports . You can connect mechanical ports only to other mechanical ports, and physical signal ports only to one another.

You cannot mix rotational and translational ports, or connect a mechanical conserving port to a physical signal port.

- The physical connection lines interconnecting mechanical ports  represent driveline axes and enforce physical relationships. Unlike physical signal and Simulink lines, they do not represent signals or mathematical operations, and they have no inherent directionality.
- A driveline connection line represents an idealized massless and perfectly rigid spinning shaft or translating axis. A driveline connection line between two ports constrains the two driveline components that are connected to the line to rotate or translate at the same velocity.
- You can branch mechanical connection lines. Connect the end of any branch of a mechanical connection line to a mechanical port .
- Branching a driveline connection line modifies the physical constraints that it represents. All driveline components connected to the ends of a set of branched lines rotate or translate at the same velocity. For lines branched from a branch point, the sum of all torques or forces flowing in equals the sum of all torques or forces flowing out. How the torque or force is divided depends on the defining equations of the attached blocks in the rest of the system.

- Mechanical connection lines satisfying the velocity constraint must have the same initial velocities.



Branching Driveline Connection Lines

The Solver Configuration block in this example does not use any torque. It does share the angular velocity constraint from the branch point. Symbolically, the branching conditions on driveline connection lines are:

$$\omega_1 = \omega_2 = \omega_3 = \dots$$

and

$$\tau_1 + \tau_2 + \tau_3 + \dots = 0$$

The sign convention is that torques flowing in are positive. Like all driveline axes, these elements have no inherent directionality. Torque flow directions are defined by overall system equations during simulation.

Torque and motion are transferred through the driveline from some driveshafts to others. Certain Simscape Driveline blocks require explicit directionality and represent it by designating one driveline connector port as the input *base* (B) and the other as the output *follower* (F), or some equivalent pair. When required, positive relative motion of driveline axes or shafts is measured as follower relative to base.

Motion Is Absolute Except when relative motion is explicitly required, all motion in Simscape Driveline and Simscape models is measured in implicit absolute coordinates. The Mechanical Rotational Reference and Mechanical Translational Reference blocks define the absolute zero velocity. If they are connected to a driveline axis, these blocks enforce this zero-motion state on that axis.

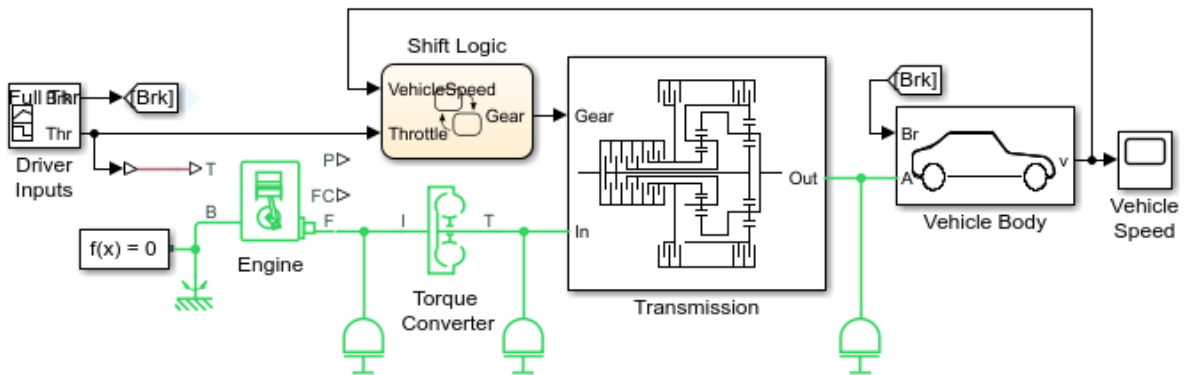
Complete Vehicle Model

Complete Vehicle Model

The full car drivetrain simulation of the `sd1_car` example encompasses all the basic methods of driveline modeling and many key Simscape Driveline features. It includes engine and transmission models and a model of the drivetrain-wheel-road coupling. The engine and transmission are coupled with a torque converter. Programmed clutch control steps the transmission through four gears during the simulation. The clutch pressure signals are smooth and more realistic than the sharp clutch pressure signals in the simpler drivetrain examples. This section describes these features, subsystems, and their relationship and purposes, leading you to actual simulation.

Understanding the Global Structure of the Model

Open the example. The model contains model workspace variables for parameterizing some of the blocks. For information on creating, accessing, and changing model workspace variables, see “Specify Source for Data in Model Workspace” (Simulink) and “Change Model Workspace Data” (Simulink).



Vehicle with Four-Speed Transmission

1. [Plot speeds](#) of shafts and vehicle ([see code](#))
2. [Explore simulation results](#) using [sscexplore](#)
3. [Learn more](#) about this example

Vehicle with Four-Speed Transmission Model

The main driveline subsystems and components are:

- Driver Inputs — Throttle/brake profile
- Engine — System-level model of spark-ignition and diesel engine
- Torque Converter — Three-part torque converter consisting of an impeller, a turbine, and a stator.
- Transmission subsystem — CR-CR 4-speed transmission
- Shift Logic — Stateflow[®] implemented transmission controller
- Vehicle Body — Vehicle, tire, and brake dynamics

While the engine is idling initially at a nonzero speed, the transmission output and the vehicle as a whole are initially not moving.

Model the Throttle/Brake Profile

The Driver Inputs block is a Simulink Signal Builder block that provides throttle and brake signals to the engine and transmission control system. Open the Driver Inputs block to view the throttle/brake profile for the simulation.

The throttle signal is programmed to produce a realistic acceleration profile and to agree with the gear shifting sequence described in “Control the Clutches” on page 3-7. The throttle signal feeds to the engine and to the transmission controller.

The brake signal supplies the input force that actuates braking in a Double-Shoe Brake block in the Vehicle Body subsystem.

Model the Engine

For the purposes of system modeling, an engine or motor specifies an output torque as a function of driveline speed. The engine has a connection port coupling it rotationally to the rest of the system.

Using an Engine Block from Vehicle Components

The Engines library contains blocks that you control using an input physical signal for the throttle. You can parameterize the Generic Engine block using vectors to specify speed and torque. The block calculates the maximum possible torque as a function of the engine speed at any instant. The throttle signal controls how much of the maximum torque the engine can deliver. The Piston Engine block accounts for the instantaneous torque transmitted to the engine drive shaft. The instantaneous torque enables you to model

vibrations in the drivetrain due to piston revolution. To model just the piston mechanism of a combustion engine, use the Piston block.

The `sd1_car` example uses a Generic Engine block, configured as spark-ignition type. The block properties specified in the dialog box include the maximum power, speed at maximum power, and maximum possible speed of the engine. To view engine settings, click the Engine block. The engine torque and motion are modeled relative to the rotational ground, which is taken as the base reference of the engine and the starting point of the driveline, or mechanical rotational, connections in this model.

Alternative and Advanced Methods for Modeling Engines

Simscape Driveline allows you to create complex, custom engine models. Several important engine features to consider in a complex model are:

- Distinguishing steady-state behavior from engine start-up, when the engine speed-engine torque function has not yet reached its maximum possible envelope
- Details of mechanical power production, such as air-fuel compression and combustion
- Additional controls beyond what can be represented by a single throttle signal

Model the Transmission

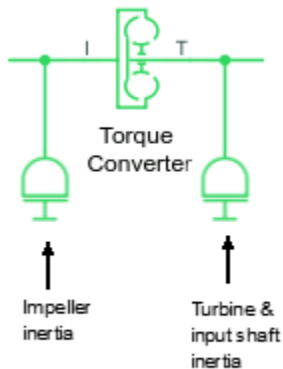
The CR-CR 4-speed transmission subsystem in the `sd1_car` model is similar to other examples with the same transmission. The clutch and planetary gear properties are set in the blocks with model workspace variables.

Workspace Variable	Description
<code>eff_tor_rad</code>	Clutch: effective torque radius (m)
<code>num_fric_surf</code>	Clutch: number of friction surfaces in contact
<code>engagement_area</code>	Clutch: friction surface area in contact (m ²)
<code>fric_coeff</code>	Clutch: kinetic friction coefficient of surfaces in contact
<code>peak_normal</code>	Clutch: static (locking) friction coefficient of surfaces in contact
<code>velTol</code>	Clutch: clutch velocity locking tolerance (rad/s)
<code>pressThresh</code>	Clutch: Normalized pressure threshold
<code>p0</code>	Clutch: Physical pressure normalization (Pa)

For more about gears, clutches, and transmissions, see the Disk Friction Clutch block reference page.

Couple the Engine to the Transmission

The `sd1_car` model couples the engine and the transmission through a torque converter block.



Torque Converter Stage

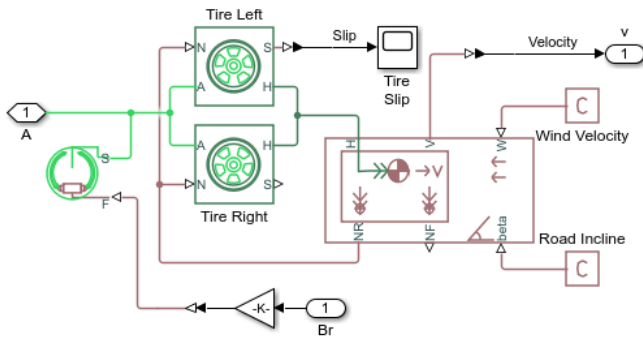
Like a clutch, a torque converter couples two independent driveline axes to transfer angular motion and torque from an input to an output shaft. However, unlike a clutch, a torque converter never locks. The torque converter transfers motion by hydrodynamic viscosity, not by surface friction. Thus a torque converter does not step through discrete stages and avoids the motion discontinuities inherent in friction clutches.

To mimic engine idling at the start of the simulation, the initial condition of the impeller inertia is a nonzero angular velocity. The initial condition of the turbine & input shaft inertia is zero speed.

For more details about these blocks, see the Torque Converter and Inertia block reference pages.

Model the Tires, Brakes, Wheels, and Road

The transmission feeds its output torque to the final drive subsystem, Vehicle Body. This subsystem represents the vehicle inertia (the load on the transmission), the wheels, the brakes, the driving conditions, and the wheel contact with the road. The subsystem models only the rear wheels as driven by the transmission.



Final Drive Subsystem: Vehicle Body

The subsystem has two major areas.

Modeling the Tires and Brakes

The right and left tire blocks accept the driveline torque and rotation from the transmission at their wheel axle rotational ports (A). Given a normal or vertical load (N), this torque and rotation are converted to a thrust force and translation at the wheel hub translational ports (H).

The tires rotate nonideally, slipping before they fully generate traction and react against the road surface. The tire slip of the left tire is reported as a physical signal and converted to Simulink for use with the Tire slip scope.

The Double-Shoe Brake block represents a brake arranged as two pivoted rigid shoes that are symmetrically installed inside or outside of a drum and operated by one actuator. The brake block converts the braking signal from the Driver Inputs block to an actuator force that exerts a friction torque on the shaft that connects the brake drum to the tire blocks.

Modeling the Vehicle Body and Load

The driveline connection line sequence of the model ends with the Vehicle Body block, which specifies the vehicle geometry, mass, aerodynamic drag, and initial velocity (zero). This block generates the normal forces that the Tire blocks accept as vertical loads. Vehicle Body accepts the developed thrust force and motion at its horizontal motion translational port (H). The vehicle body model also accepts a wind velocity (W) and a road incline (beta), both provided by physical constants.

The rear wheel vertical load force (NR) is reported back to the Tire blocks. The forward wheel vertical load (NF) is not used.

The forward velocity (V) of the vehicle is converted and reported, through the subsystem outport, to the Vehicle velocity scope.

Alternative Differential, Wheel, Road, and Braking Models

The `sdl_car` example models only the rear wheels, the rear tires, and the vehicle body, without the more realistic drivetrain components of differential gears and brakes. The `sdl_vehicle_4wd` example illustrates how to model a vehicle that has four wheels and front and rear differential gears.

For information on modelling brake systems using clutches, see “Brake Motion Using Clutches” on page 7-9 and “Model a Two-Speed Transmission with Braking” on page 8-3.

Control the Clutches

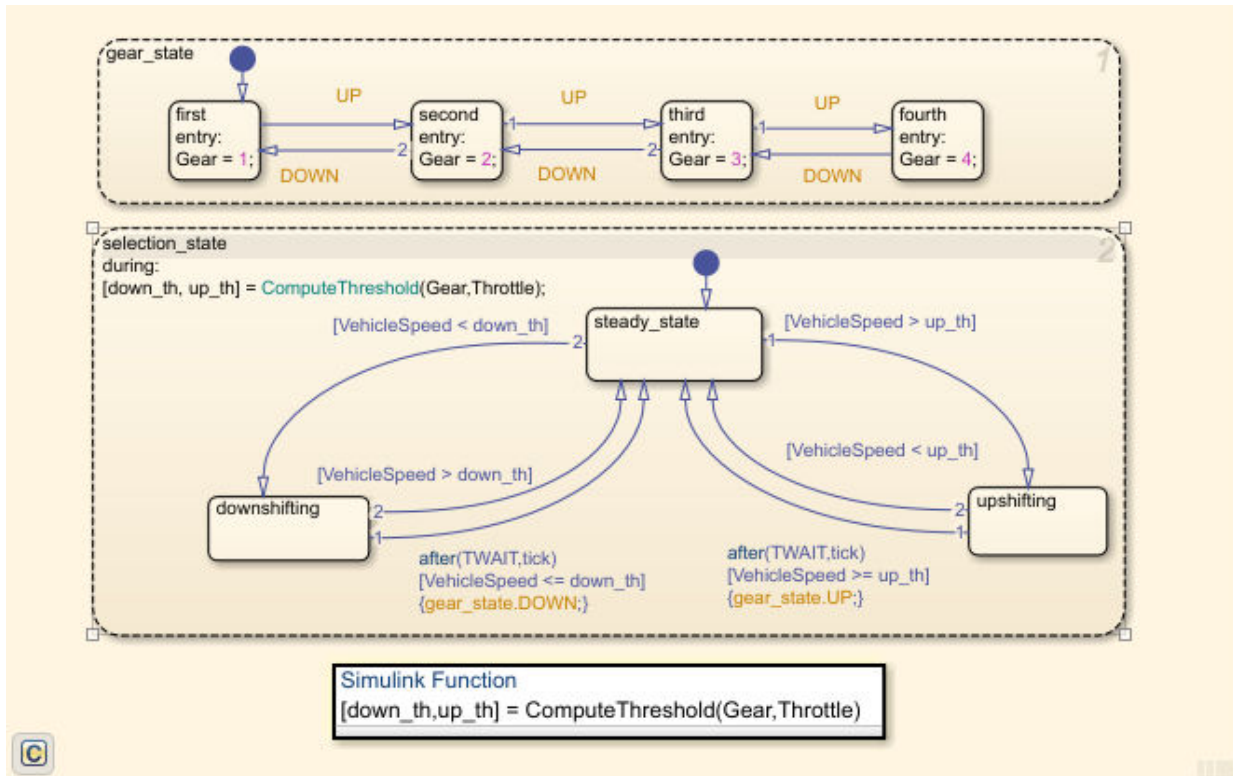
To select and engage the appropriate gear set, the model uses a Stateflow block and clutch schedule. To see how these components work, return to the main model of `sdl_car`.

State-Controlled Gear Selection

The Stateflow block, which is labeled Shift Logic, implements gear selection for the transmission. The block determines whether to shift up or down based on input from two other components in the model. Driver Inputs block supplies throttle and braking information. The Vehicle Body subsystem supplies the velocity of the vehicle body via a feedback loop.

To open the Stateflow diagram, double-click the Shift Logic block. The Model Explorer is utilized to define the inputs as throttle and vehicle speed and the output as the desired

gear number. Two dashed AND states keep track of the gear state and the state of the gear selection process. The overall chart is executed as a discrete-time system. The Stateflow diagram shown in the figure illustrates the functionality of the block.



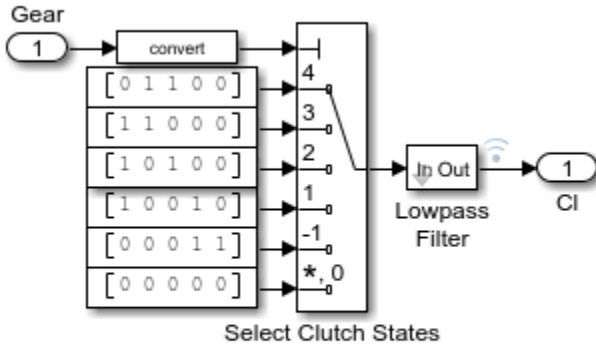
The model computes the upshifting and downshifting speed thresholds as a function of the instantaneous values of gear and throttle. While in `steady_state`, the model compares these values to the present vehicle speed to determine if a shift is required. If so, it enters one of the confirm states (upshifting or downshifting), which records the time of entry.

If the vehicle speed no longer satisfies the shift condition, while in the confirm state, the model ignores the shift and it transitions back to `steady_state`. The steady-state condition prevents extraneous shifts due to noise conditions. If the shift condition remains valid for a duration of `TWAIT` ticks, the model transitions through the lower junction and, depending on the current gear, it broadcasts one of the shift events. The model again

activates `steady_state` after a transition through one of the central junctions. The shift event, which is broadcast to the `gear_selection` state, activates a transition to the appropriate new gear. The Stateflow block outputs the gear information to a clutch schedule subsystem that is in the transmission subsystem.

Clutch Schedule Subsystem

The signal from the Stateflow block to the clutch schedule controls the five clutches of the CR-CR 4-Speed transmission. To see the clutch schedule, open the Transmission subsystem, and then the Clutch Schedule subsystem.



Clutch Schedule

Gear	A	B	C	D	R	Ratio
R	0	0	0	1	1	-g1
1	1	0	0	1	0	g2
2	1	0	1	0	0	$(g1+g2)/(1+g1)$
3	1	1	0	0	0	1
4	0	1	1	0	0	$g1/(1+g1)$

0 - Disengaged, 1 - Engaged

g1: Input planetary ring/sun gear ratio

g2: Output planetary ring/sun gear ratio

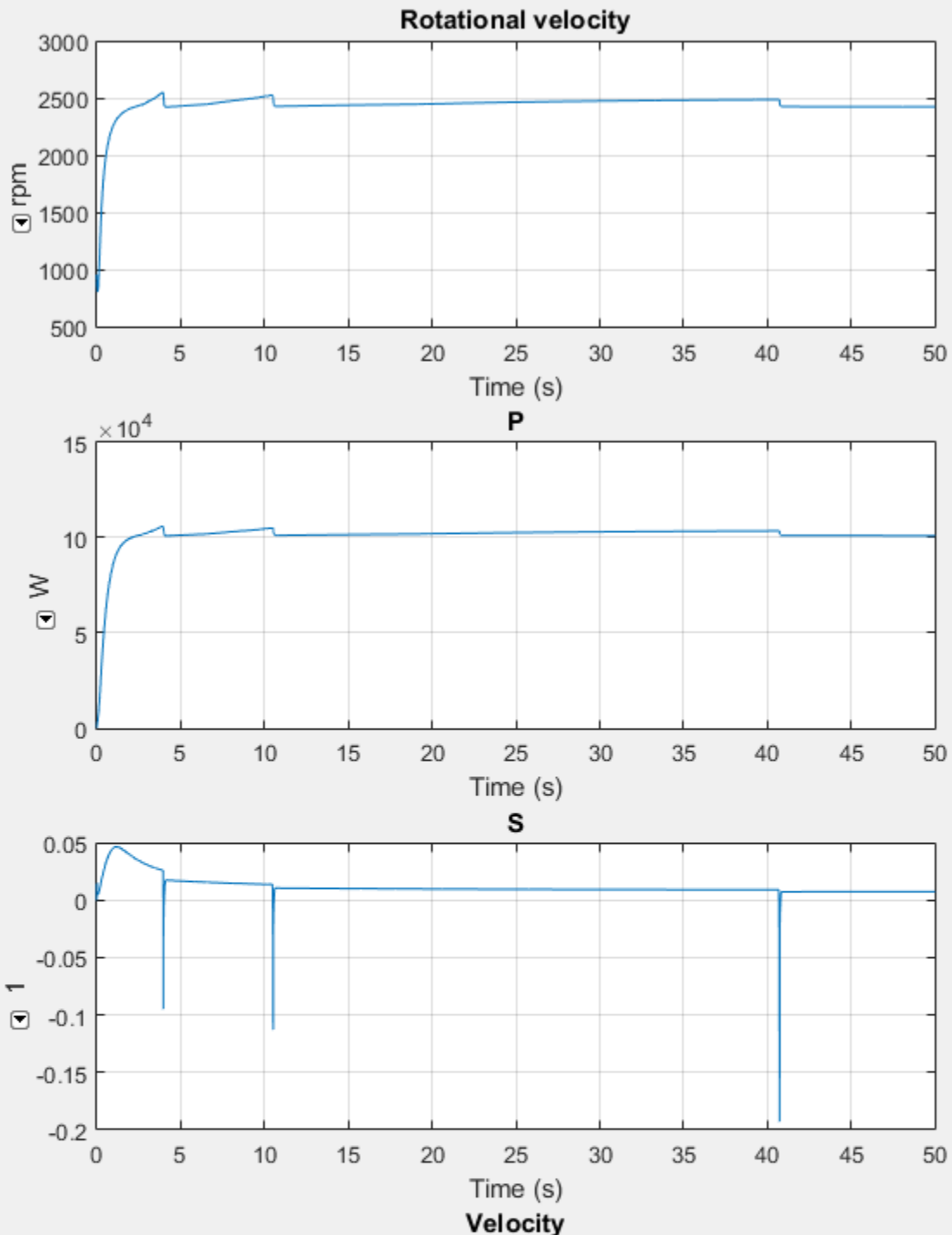
Run the Model

The model is configured to simulate for 50 seconds. The table shows the gear profile for the simulation.

Time Ranges (s)	CR-CR Gear Settings
0-3.96	1
3.96-10.48	2
10.48-40.68	3

Time Ranges (s)	CR-CR Gear Settings
40.68-50	4

- 1 Simulate the car.
- 2 To see the results using the Simscape Results Explorer, in the description in the model window, click **Explore simulation results**.
- 3 To plot the rotational velocity in RPMs and power in Watts for the engine:
 - a In the left pane of the Results Explorer window, expand the node for the **Engine**
 - b Click the **F** node, and then the **w** node.
 - c To change the units for the y-axis to revolutions per minute, click the arrow button below the y-axis label (rad/s) and select rpm.
 - d To add a plot of the power that the engine delivers to the torque converter, Ctrl+click the **P** node.
- 4 Add a plot of the tire slip.
 - a Ctrl+click to expand the **Vehicle_body** node.
 - b Ctrl+click to expand the **Tire_Left** node.
 - c Ctrl+click the **S** node.
- 5 Add a plot of the vehicle velocity.
 - a Ctrl+click to expand the second **Vehicle_body** node.
 - b Ctrl+click the **v** node.
 - c To change the units to kilometers per hour, click the arrow button below the y-axis label (m/s), select **Specify**, and for **Specify your unit**, enter km/hr.



The plots show that for:

- Engine speed and power — When the transmission shifts to second gear at 3.96 seconds, the engine reaches its maximum speed and power.
- Tire slip — As the transmission steps into higher gears, the speed ratio rises. The drive ratio falls, and the tire slip decreases. The tire motion more closely approaches ideal (nonslipping) motion at higher speeds.
- Vehicle velocity — The speed increases less with each upshift for gears one, two, and three. The velocity decreases slightly before it starts to stabilize when the car is in fourth gear.

Basic Motion, Torque, and Force Modeling

The purpose of a gear set is to transfer rotational motion and torque at a known ratio from one driveline axis to another. Simscape Driveline allows you to model simple and custom gears for coupling bodies that are rotating on a driveline axis.

The rules that apply to angular gears in relation to rotational motion and torque are analogous to the rules apply to linear gears in relation to translational motion and force.

Couple Rotational Motion with Gears

A gear set consists of two or more meshed gears rotating together at some specified gear ratios. By convention, Simscape Driveline gear ratios are constant. The gear ratios determine how angular velocity and torque are transferred from one driveline component to another.

Gear Coupling Rules

Ideal gears mesh and rotate together at a point of contact without frictional loss or slippage.

The simplest gear coupling consists of two circular gear wheels of radii r_1 and r_2 , spinning with angular velocities ω_1 and ω_2 , respectively, and lying in the same plane. Their connected shafts are parallel and carry torques τ_1 and τ_2 . The *gear ratio* of gear 2 to gear 1 is the ratio of their respective radii: $g_{12} = r_2/r_1$. The power transferred along either shaft is $\omega \cdot \tau$.

The gear coupling is often specified in terms of the number of gear teeth on each gear, N_1 and N_2 . The gear ratio of gear 2 to gear 1 is then $g_{12} = N_2/N_1 = r_2/r_1$.

The fundamental conditions on the simple gear coupling of rotational motion are $\omega_2/\omega_1 = \pm 1/g_{12}$ and $\tau_2/\tau_1 = \pm g_{12}$. That is, the ratio of angular velocities is the reciprocal of the ratio of radii, while the ratio of torques is the ratio of radii. The transferred power, being the product of angular velocity and torque, is the same on either shaft.

The choice of signs indicates that the gears can spin in the same or in opposite directions. If the gears are external to one another (rotating together on their respective outside surfaces), they rotate in opposite directions. If the gears are internal to one another (rotating together with the outside of the smaller gear meshing with inside of the larger gear), they rotate in the same direction.

Caution Gear ratios in driveline model blocks must be strictly positive. Vanishing or negative gear ratios cause Simscape Driveline simulation to stop with an error at model initialization. If you need to reverse the relative rotation direction of a shaft connected to a gear, you can change the direction in the gear block dialog box.

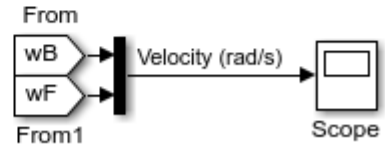
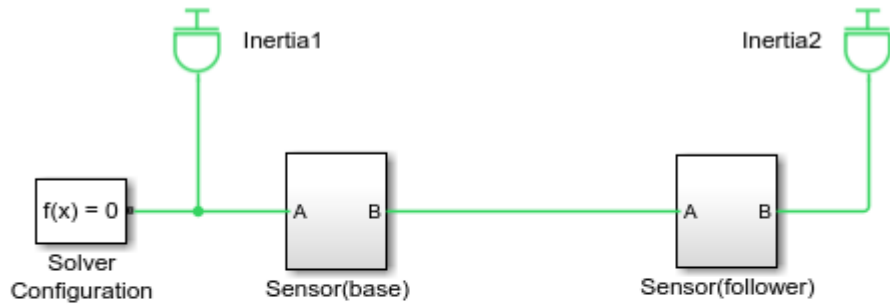
Couple Two Spinning Inertias with a Simple Gear

In this example, you couple two spinning inertias. In the first coupling, the inertias spin with the same angular velocity along a single shaft (driveline axis). Then the inertias spin at different velocities as they spin along two shafts and are coupled by a gear. Finally, the inertias are coupled by a gear and actuated by an external torque, so that they spin at different rates and experience different torques. For each model, the example uses basic Simscape mechanical and Simscape Driveline blocks, such as Inertia, Simple Gear, and Solver Configuration.

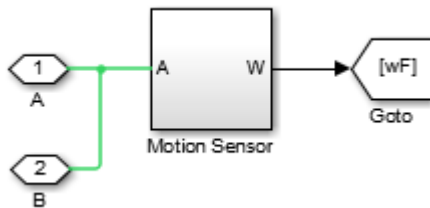
Modeling Two Spinning Inertias

Create the first version of the simplest, nontrivial driveline model, two inertias spinning together along the same axis. Open the Simscape Driveline, Simscape, and Simulink block libraries and a new Simulink model window.

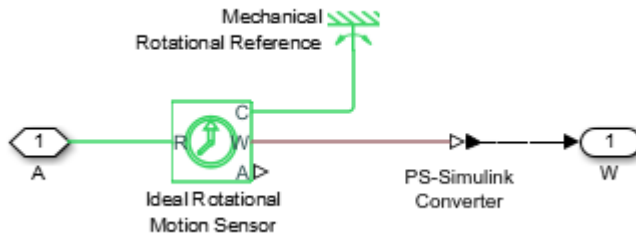
- 1** Drag and drop two Inertia, two Ideal Rotational Motion Sensor, two Mechanical Rotational Reference, and two PS-Simulink Converter blocks into the model window.
- 2** From the Simscape Utilities library, drag a Solver Configuration block. Every topologically distinct driveline block diagram requires exactly one instance of this block.
- 3** From the Simulink library, drag and drop a Scope, a Mux, and two pairs of Goto and From blocks. Connect the blocks as shown in the following figures. The sensor subsystems are arranged hierarchically.



Model with Two Spinning Inertias

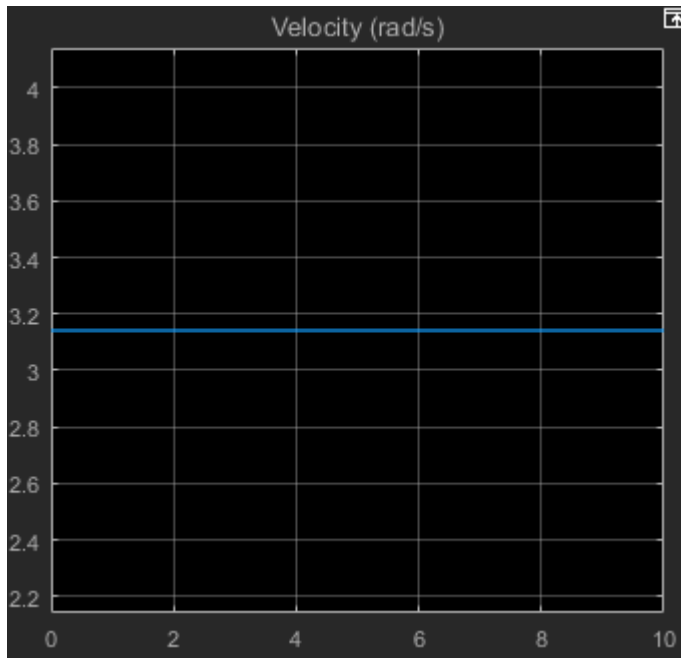


Sensor Subsystem



Motion Sensor Subsystem

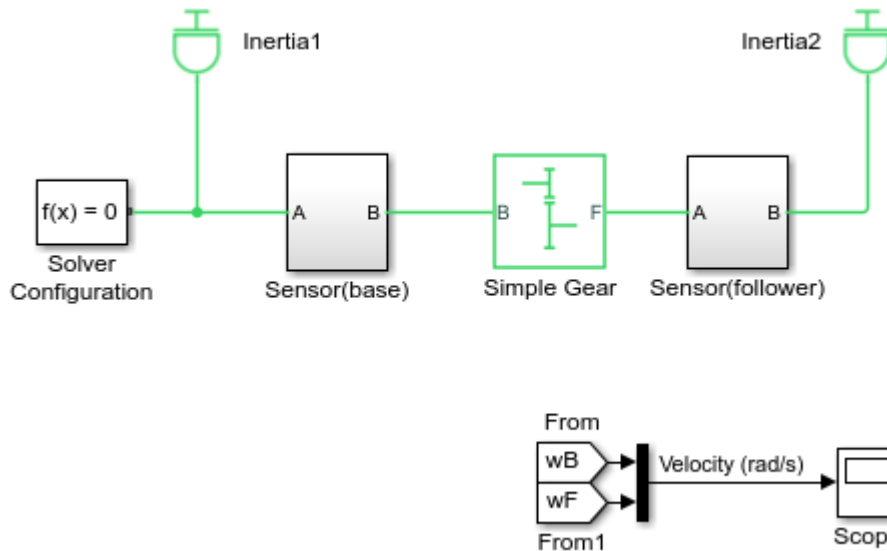
- 4 At the start of the simulation, because there is no damping, the inertias rotate at the initial velocity that you specify. The connection line between the two Inertia blocks requires them to have the same rotational velocity. To specify the initial rotational velocity, open each Inertia block. In the **Variables** tab, select the **Rotational velocity** check box and set the **Value** parameter to π radians/second (rad/s).
- 5 Open the Scope block and start the simulation. The two angular velocities are constant at 3.14 radians/second.



Coupling Two Spinning Inertias with a Simple Gear

Modify the model you created by coupling the two spinning inertias with a simple, ideal gear with a fixed gear ratio.

- From the Simscape Driveline block library, drag and drop a Simple Gear block into your model. Open the block. Change the default follower-base gear ratio value to 1. Change the **Output shaft rotates** menu to **In same direction as input shaft** and click **OK**. The simple gear then represents two gear wheels rotating together at the same rate in the same direction, with one wheel inside the other. Connect the blocks as shown in the following figure.



Model with Two Spinning Inertias Coupled by a Gear

Leave the initial angular velocities at π in the Inertia blocks.

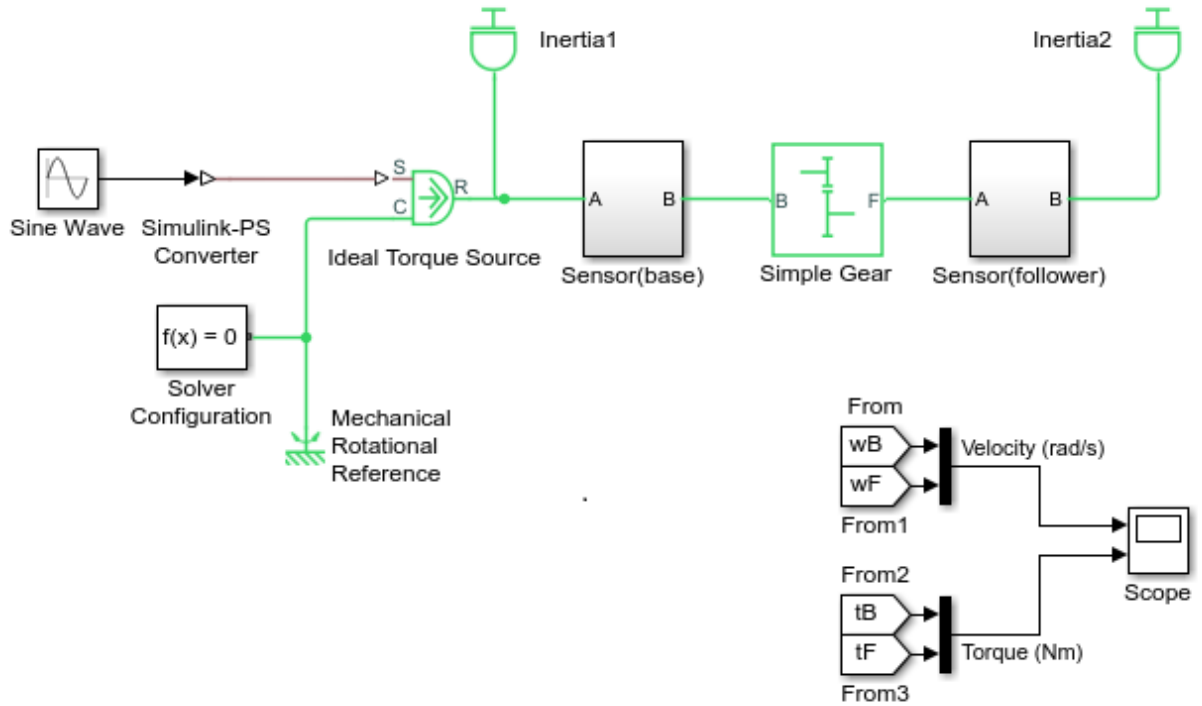
- Open the Scope and start the simulation. The two angular velocities are constant at 3.14 radians/second for both Inertias.
- Change the **Output shaft rotates** menu back to **In opposite direction to input shaft**. The simple gear then becomes two wheels rotating together in opposite directions, with the two wheels meshed on their respective outer surfaces. Change initial velocity in Inertia2 to $-\pi$.

- 4 Restart the simulation. The two angular velocities are 3.14 and -3.14 radians/second for Inertia1 and Inertia2, respectively. The second angular velocity is the same, but with opposite sign, because the two bodies are spinning in opposite directions.
- 5 Change the **Output shaft rotates** menu again to **In same direction as input shaft**.

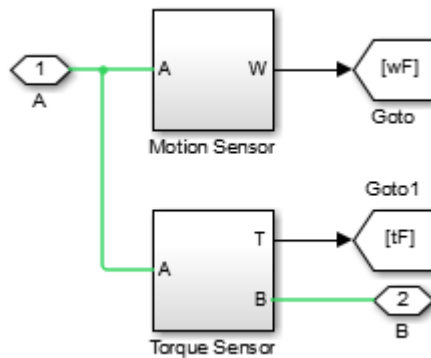
Torque-Actuating Two Coupled, Spinning Inertias

In the final version of the simple gear model, you actuate the inertias with an external torque instead of starting them with fixed initial angular velocities. The external torque varies sinusoidally. You can find a completed version of this model in the `sd1_gear` example model.

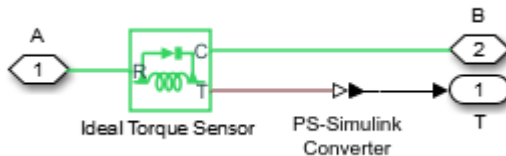
- 1 From the Simscape Foundation library, copy an Ideal Torque Source and two Ideal Torque Sensor blocks, plus a Simulink-PS Converter block and another Mechanical Rotational Reference block. From the Simulink library, drag and drop a Sine Wave block and two more pairs of Goto and From blocks.
- 2 Connect the blocks as shown in the following figures. The Torque Sensor subsystems are arranged in parallel with the Motion Sensor subsystems inside the Sensor subsystem blocks. Set the initial velocities of both Inertias to zero. Change the default follower-base gear ratio value to 2. Modify the Scope block to add another axis for measuring the torques. Connect the other blocks as shown.



Model with Two Spinning Inertias Coupled by a Gear and Actuated with Torque



Updated Sensor Subsystem



Torque Sensor Subsystem

- 3 Open the Scope block and start the simulation.



The measured torques and angular velocities vary sinusoidally. As in the preceding models, the angular velocity of Inertia2 is half that of Inertia1. The torque in the second (follower) shaft is twice that in the first, as required by the laws of gear coupling.

For the Simple Gear block, change the **Output shaft rotates** menu to **In opposite direction to input shaft** and restart the simulation. The same angular velocities and torques result, except that the values associated with Inertia2 and the second shaft are negative because the second body and second shaft are spinning in opposite directions.

Sensing and Actuating Motion and Torque

The mechanical sensor and source blocks that you use in the preceding models illustrate their dual nature. They act as driveline components themselves, but also let you inject and extract physical signals associated with motion and torque, including the appropriate physical units. You can use these physical signals with other blocks in the Simscape

physical modeling environment, or convert them to dimensionless Simulink signals for use in the nonphysical part of your model. Both sensor and source blocks have pairs of mechanical ports and are connected either in series with or across physical connection lines.

- Mechanical sensor and source blocks have both mechanical conserving ports  and physical signal ports .

Many Simscape Driveline blocks also feature a mix of mechanical conserving and physical signal ports.

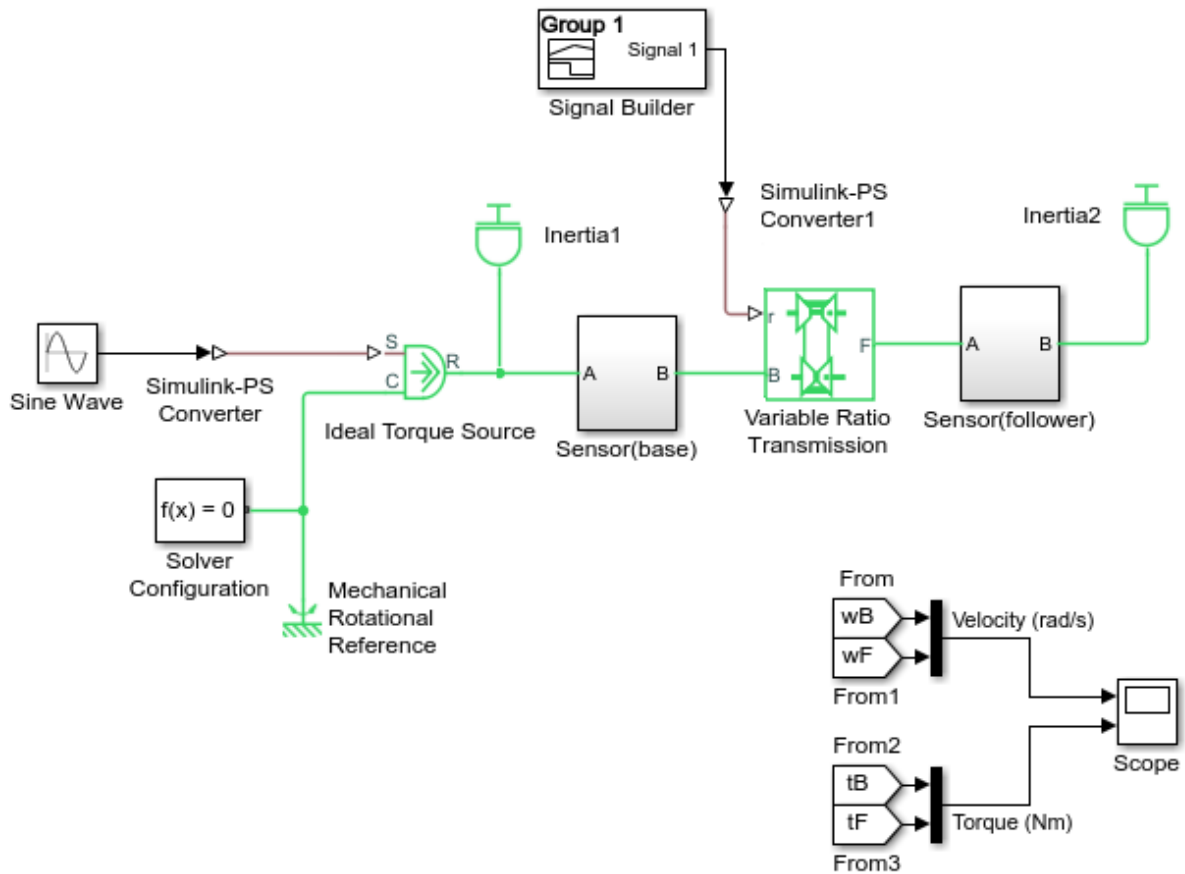
- An Ideal Torque Source injects torque along, or in series with, the driveline connection line. An Ideal Torque Sensor measures the torque flowing along, or in series with, the driveline connection line.
- An Ideal Rotational Motion Sensor reports the *difference* between the motions at its two connection ports.

To extract the absolute motion at its R port, connect the C port to a mechanical reference block that grounds that port to zero motion.

Couple Two Spinning Inertias with a Variable Ratio Transmission

You can modify the gear model from the “Couple Two Spinning Inertias with a Simple Gear” on page 4-3 example by replacing the fixed-ratio gear with a transmission whose gear ratio varies in time. You specify the gear ratio variation with a Simulink signal converted to a unitless physical signal. Start with the gear model that you built in the “Couple Two Spinning Inertias with a Simple Gear” on page 4-3 example or by opening and editing the `sd_l_gear` model.

- 1 From the Simscape Driveline block library, drag and drop a Variable Ratio Transmission block and replace the Simple Gear block with it. The two shafts spin in the same direction because the **Output shaft rotates** parameter for the Variable Ratio Transmission block is set to **In same direction as input** (the default setting).
- 2 The Variable Ratio Transmission block accepts the continuously varying gear ratio as a physical signal Simulink signal through the extra physical signal input labeled `r`. For this example, create a variable signal for the gear ratio with a Signal Builder block from the Simulink block library and Simulink-PS Converter block. Build a signal that rises with constant slope from 1 to 2 over 10 seconds. Then connect the converted physical signal to the `r` port.



Simple Variable Ratio Transmission Model

- Do not change the other, original settings of the simple gear model. Open the Scope and start the simulation.

The angular velocities and torques of the two shafts have the same signs. The ratios of angular velocities and torques start at 1, because the initial gear ratio is 1. As the gear ratio increases toward 2, the angular velocity of Inertia2 becomes smaller than the velocity of Inertia1, while the associated torque in the second shaft becomes larger than

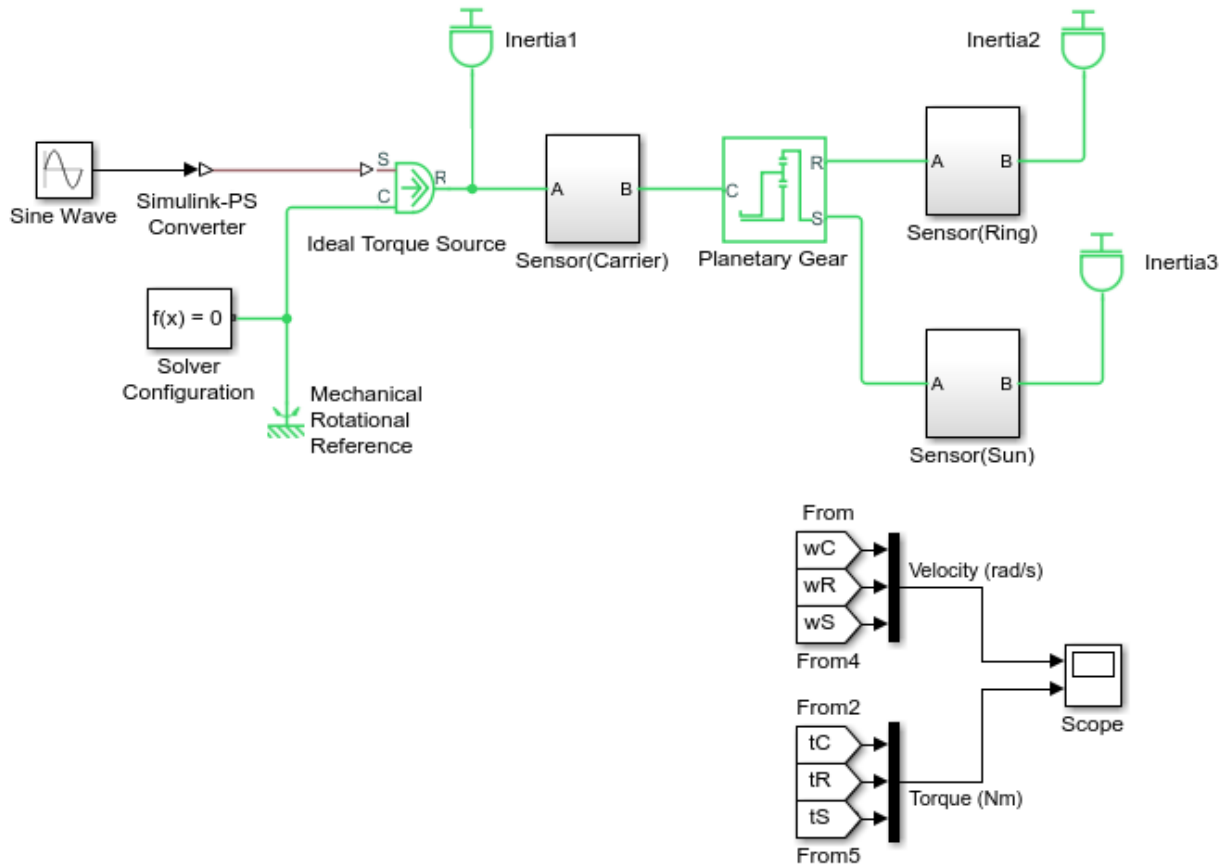
the torque in the first shaft. Because of the changing gear ratio, the motion and the torques are no longer strictly sinusoidal, even though the actuating external torque is.

The `sdl_gear_variable` example is a full model of this type. To learn more about how to use variable gear ratios, consult the Variable Ratio Transmission block reference page.

Couple Three Spinning Inertias with a Planetary Gear

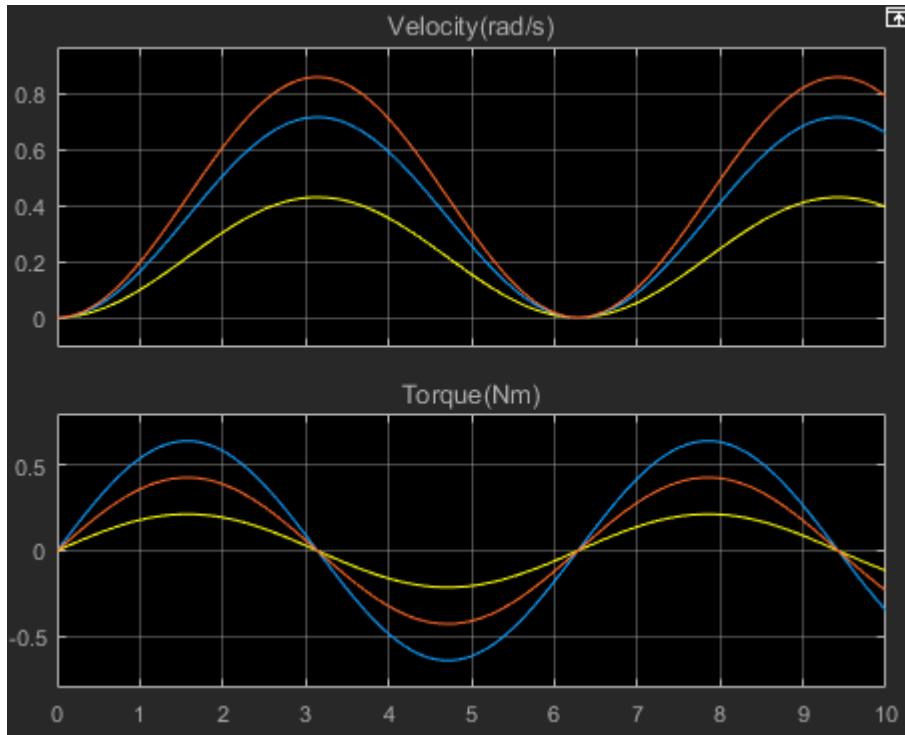
You can modify the gear model from the “Couple Two Spinning Inertias with a Simple Gear” on page 4-3 example and use it as a starting point for studying complex gear sets. One of the most important complex gear sets is the planetary gear, which has three wheels, the ring, the sun, and the planet, all held in place by a common carrier body. The planetary gear is important because it is a common component in complex, realistic transmissions.

- 1** Start with the simple gear model you built or by opening the `sd1_gear` example.
- 2** Replace the Simple Gear in your model with a Planetary Gear from the Simscape Driveline block library. A planetary gear splits input angular motion from the carrier between the ring and sun wheels, each connected to their respective bodies.
- 3** Copy the Sensor (Follower) subsystem, the connected Inertia block, and a From block.
- 4** Rename the sensor subsystems to match the gears that they attach to: carrier, ring, and sun. Rename the signals on the Goto and From block tags to match the gears that the signals represent:
 - For the carrier gear, rename the tags signals as wC and tC .
 - For the ring gear, rename the tags signals as wR and tR .
 - For the sun gear, rename the tags signals as wS and tS .
- 5** Add an input to each of the two Mux blocks.
- 6** Connect the blocks to form the new diagram as shown in the figure.

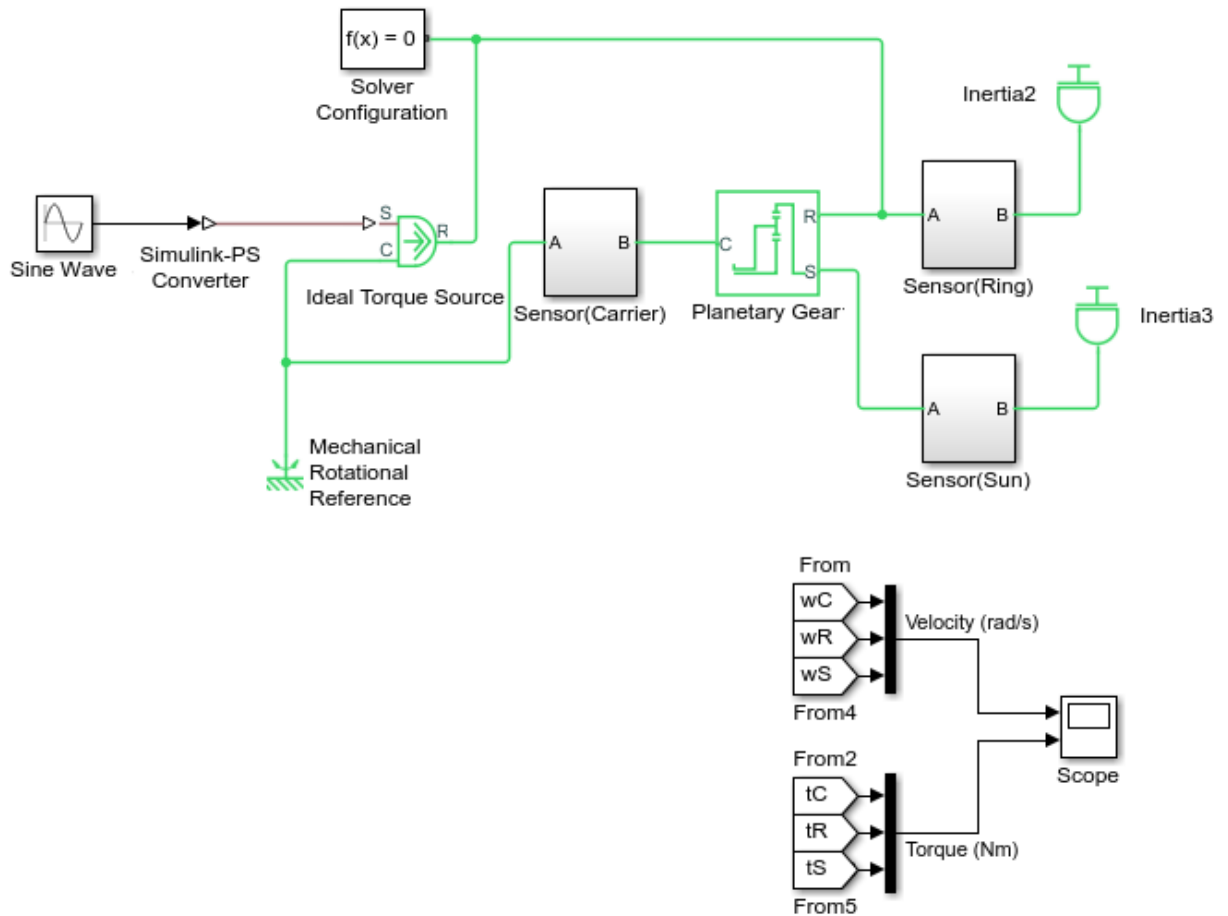


Simple Planetary Gear Model

- Open the Scope and start the simulation to observe the angular velocities of the ring, carrier, and sun, from largest to smallest. The ratio of the ring to sun gear velocities is always 2.

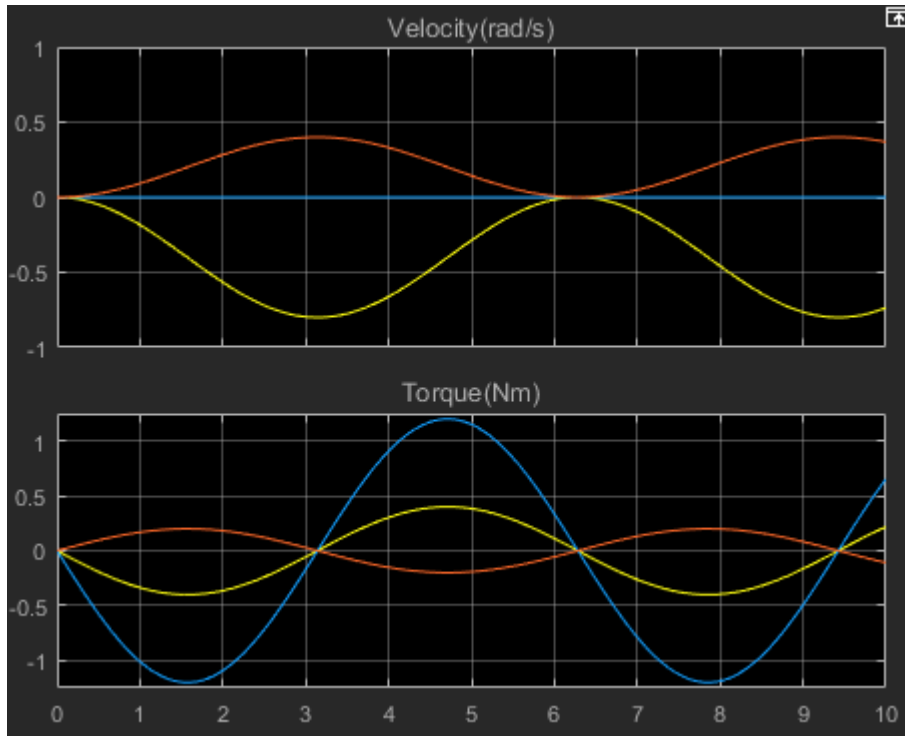


- 8** To see the ring and sun wheels spinning alone, you must lock the carrier. To do so:
- 1** Delete the Inertia1 block and the associated connector.
 - 2** Delete the connector between the Ideal Torque Source block and the Sensor (Carrier) subsystem.
 - 3** Connect the Sensor (Carrier) subsystem to the connector between the Mechanical Rotational Reference block and the Ideal Torque Source block.
 - 4** Connect the Ideal Torque Source block to the connector between the Planetary Gear block and the Sensor (Ring) subsystem.
 - 5** Delete the connector that connects the Solver Configuration block to the connector between the Mechanical Rotational Reference block and the Ideal Torque Source block.
 - 6** Reposition the Solver Configuration block.
 - 7** Connect the Solver Configuration block to the connector between the Ideal Torque Source block and the Sensor (Ring) subsystem.



Simple Planetary Gear Model with Locked Carrier

- 9 Open the Scope and start your model. Observe the angular velocities of the ring, carrier, and sun.



The carrier, connected to Mechanical Rotational Reference, does not move. The ring is driven with a sinusoidal torque, and the sun responds by spinning in the opposite direction (ring and sun gear wheels are external to one another) at twice the rate. The ring wheel has twice the radius (or twice the number of teeth) as the sun, so it spins half as fast.

Driveline Actuation

From the torques and forces applied to driveline inertias and masses, a Simscape Driveline simulation determines the resulting motion from the driveline component connections and defining equations. However, a simulation can also accept motions imposed on a driveline and solve for the torques and forces to produce those motions. In general, a driveline simulation is a mixture of these two requirements, solving dynamics both forward (torque and force to motion) and inverse (motion to torque and force). Imposing motions and applying torques and forces to a driveline are together forms of mechanical *actuation*.

Best Practices for Modeling Torque-Force Actuation and Motion Actuation

All actuation, except for initial conditions, requires physical signal inputs to define time-varying functions that carry physical units.

Torque-force actuation and motion actuation are complementary and mutually exclusive. In all cases, exercise care as you apply a mixture of actuation types to a driveline and its degrees of freedom (DoFs). The complete effect of the actuation types must be such that:

- Driveline DoFs actuated by torques and forces are not also subject to motion actuation. (They can be subject to motion initial condition settings.)
- Driveline DoFs actuated by motions are not also subject to torque or force actuation.

For a Simscape Driveline model to simulate nontrivial motion, torque and motion actuation types complement one another exactly to account consistently for the motion of all the DoFs. If this criterion is not satisfied, one of these outcomes results:

- The motion of the driveline is trivial, staying in its initial motion state for the entire simulation.
- The actuation types are inconsistent with each other, and the simulation stops with an error.
- The actuation types leave the driveline motion underdetermined or overdetermined, and the simulation stops with an error.

For more information, see “Troubleshoot Driveline Modeling and Simulation Issues” on page 18-2.

Actuate a Driveline Using Torques and Forces

You can apply a torque to a rotational driveshaft, or a force to a translational driveshaft, in the following ways:

- Directly, with an Ideal Torque Source or Ideal Force Source block.
- Indirectly, with a dynamic element that generates torque or force. Such blocks include torque converters, clutches and clutch-like elements, and engines.

A torque or force source accepts a physical signal input and originates, from its mechanical conserving port, a mechanical connection line carrying that torque or force.

The Simscape Driveline simulation solves for the motion of the spinning or sliding driveshaft, given the torque or force that it is subject to. Therefore you cannot also subject that same driveshaft to motion actuation.

Note Simscape Driveline might generate an error if the combined torques and forces at any given node have a nonzero sum.

Actuate a Driveline Using Motions

You can apply a motion to a driveshaft directly, with an Ideal Angular Velocity Source or Ideal Translational Velocity Source block.

A motion source accepts a physical signal input and originates, from its mechanical conserving port, a mechanical connection line spinning or sliding with the specified motion.

The Simscape Driveline simulation solves for the torque or force carried by the spinning or sliding driveshaft, given its motion. Therefore you cannot also subject that same driveshaft to torque or force actuation.

Set Initial Conditions of Driveline Motion

When driveline simulation starts, the complete driveline determines the initial motion of all driveshafts by a combination of constraints, motion sources, and initial condition settings. You set the initial conditions for the rotational and translational motion of inertias and masses in their respective Inertia and Mass blocks. The block default for initial velocities is zero (no initial motion).

For more information about constraints and degrees of freedom, see “Driveline Degrees of Freedom” on page 16-38.

Note Ensure that the initial conditions that you impose on the Inertia and Mass blocks in your driveline are consistent with all of the constraints and motion sources for the driveline. If an inconsistency occurs, Simscape Driveline simulation stops with an error at model initialization.

Resolving Undetermined Motions in Complex Gears

A simple gear has two ports and imposes one constraint between them, leaving one independent DoF. Once one port is connected to a driveshaft, the motion of the driveshaft of the other port is determined.

A complex gear has three or more ports and imposes one or more constraints among them. A complex gear can have any number of independent DoFs, including none.

If a simulation apportions the initial motions of a complex gear in an unsatisfactory way, enforce your preferred division by setting initial conditions on the connected Inertia and Mass blocks.

However you divide the initial motion among the gear shafts, ensure that this division is consistent with all constraints in your driveline, and with any motion sources.

For more information about complex gears, see “Basic Motion, Torque, and Force Modeling” .

Power Transmission Using Pulley Networks

The Simscape Driveline Couplings and Drives library includes pulley blocks that allow you to transmit power. The Belt Pulley block represents a simple belt-driven friction pulley. The Belt Drive block combines two Belt Pulley blocks that you can configure as an open or crossed belt system. You can model a simple hoist using the Belt Pulley. To model more complex mechanisms, combine several pulley blocks with tensioners, which you can build using springs and dampers. Build high-fidelity models by including other Simscape and Simscape Driveline blocks.

Best Practices for Modeling Pulley Networks

In this section...

“Belt Direction” on page 6-2

“Inertia” on page 6-3

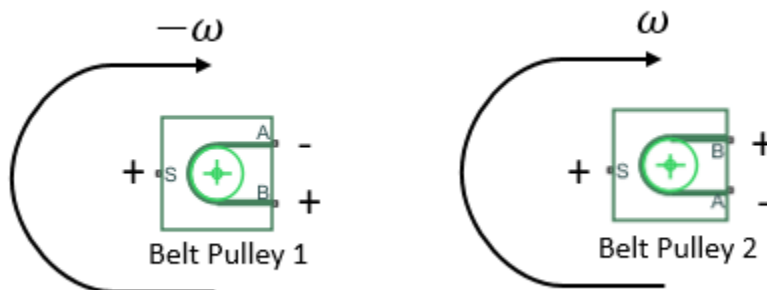
“Belt Tension” on page 6-3

“Power Window System Pulley Mechanism” on page 6-3

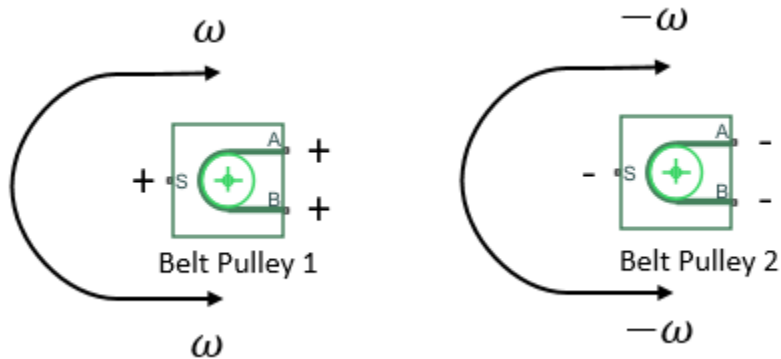
Like real-world pulleys, Simscape Driveline pulley blocks rely on belt tension and inertia for motion. To prevent initialization errors and to obtain the desired power transmission from your pulley system, apply these modeling methods.

Belt Direction

Pulley belt direction is not a geometric or physical constraint; it is purely a sign convention. For example, for a Belt Pulley block with **Belt direction** set to **Ends move in opposite direction**, the sign convention is such that a positive rotation in port **S** tends to give a negative translation for port **A** and a positive translation for port **B**. According to this convention, the angular velocity is the same for the two belt pulleys in the figure.



For a Belt Pulley block with **Belt direction** set to **Ends move in same direction**, the sign convention is such that a positive rotation in **S** tends to give a positive translation for port **A** and a positive translation for port **B**.



The Ends move in same direction option is applicable to most pulley systems. The Ends move in same direction option allows you to model a simplified representation of a complex block-and-tackle system with belt ends that move in the same direction.

Inertia

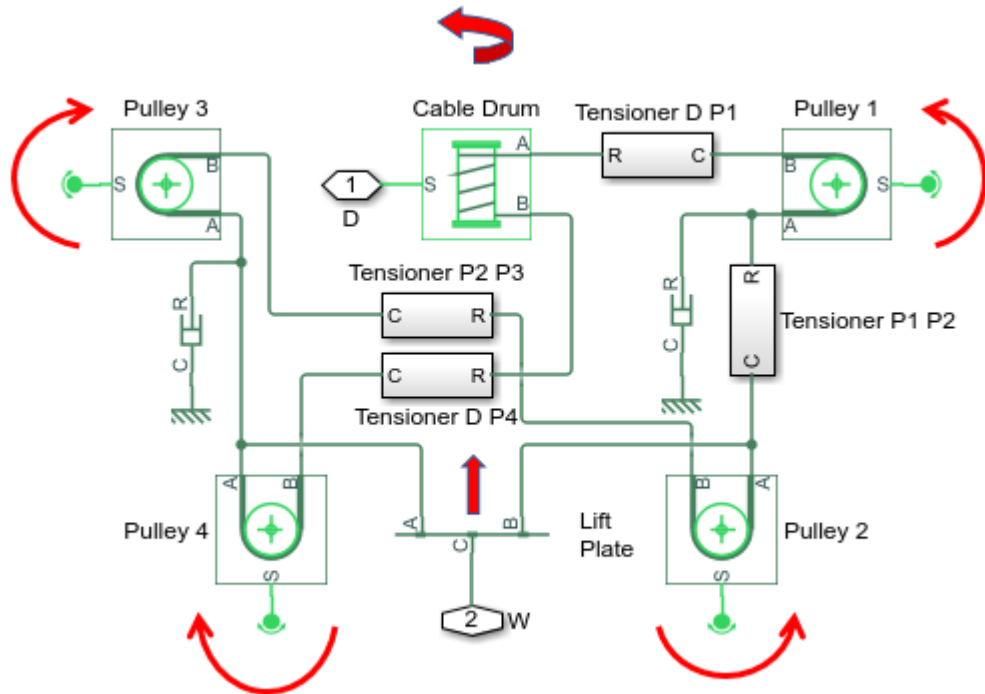
To facilitate motion, include inertia in the pulley system. You can include inertia in a pulley block by specifying a nonzero value for the **Inertia** parameter in the block configuration settings. Another way to include inertia is to add a downstream inertia block from the Simscape Driveline Inertias and Loads library or from the Simscape Rotational Elements library. Attribute some initial velocity to the inertia, as needed, to initiate motion in your pulley system.

Belt Tension

Maintain belt contact by including tensioners in your pulley system. Include no fewer than the number of pulley pairs less one. For example, if there are five pulley pairs, include at least four tensioners. You can build the tensioners using spring and damper blocks.

Power Window System Pulley Mechanism

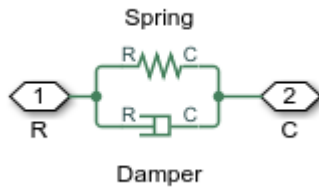
The Simscape Driveline “Power Window System” example contains a pulley network that includes tensioners and inertia and follows recommended belt-direction practices.



The arrows show how the four Belt Pulley blocks rotate in response to the rotation of the **Cable Drum** block. If the drum rotates in the opposite direction, the pulley directions reverse, and the **Lift Plate** is lowered. There are six pulley pairs:

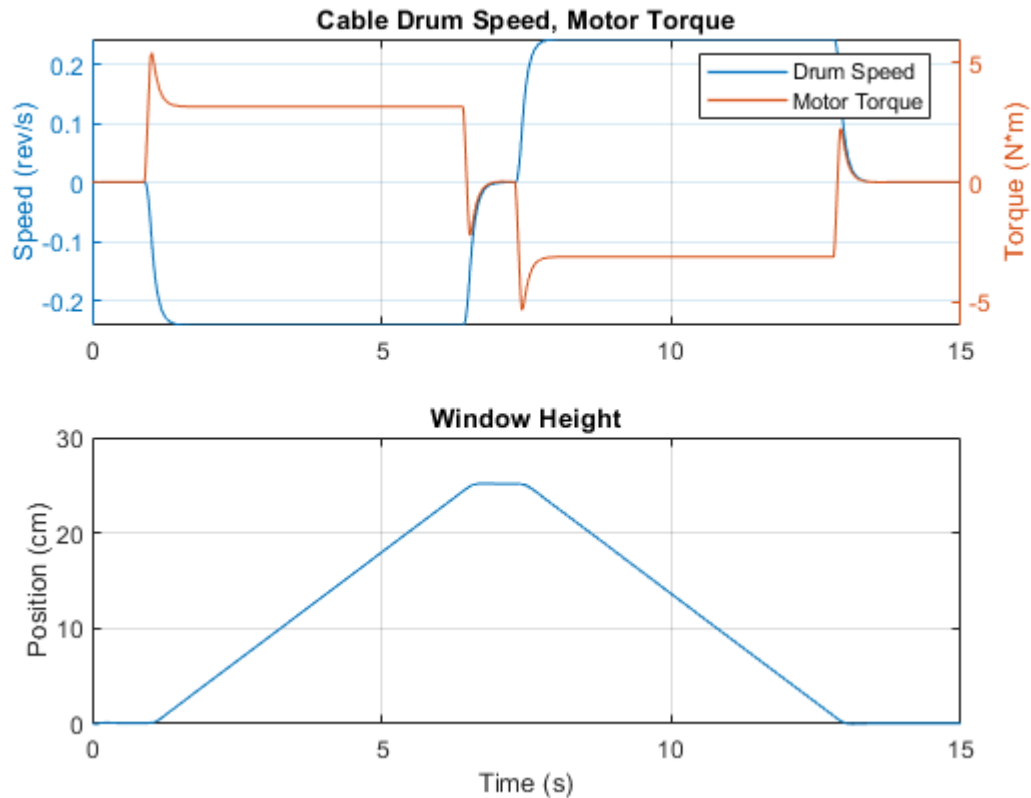
- **Cable Drum** and **Pulley 1**
- **Pulley 1** and **Pulley 2**
- **Pulley 2** and **Pulley 3**
- **Pulley 3** and **Pulley 4**
- **Pulley 4** and **Cable Drum**
- **Pulley 2** and **Pulley 4**


Therefore, it is recommended that the system includes at least five tensioners. The **Lift Plate** acts as a tensioner for the **Pulley 2** and **Pulley 4** pulley pair. The system contains four additional tensioners. Open one of the tensioner subsystems.

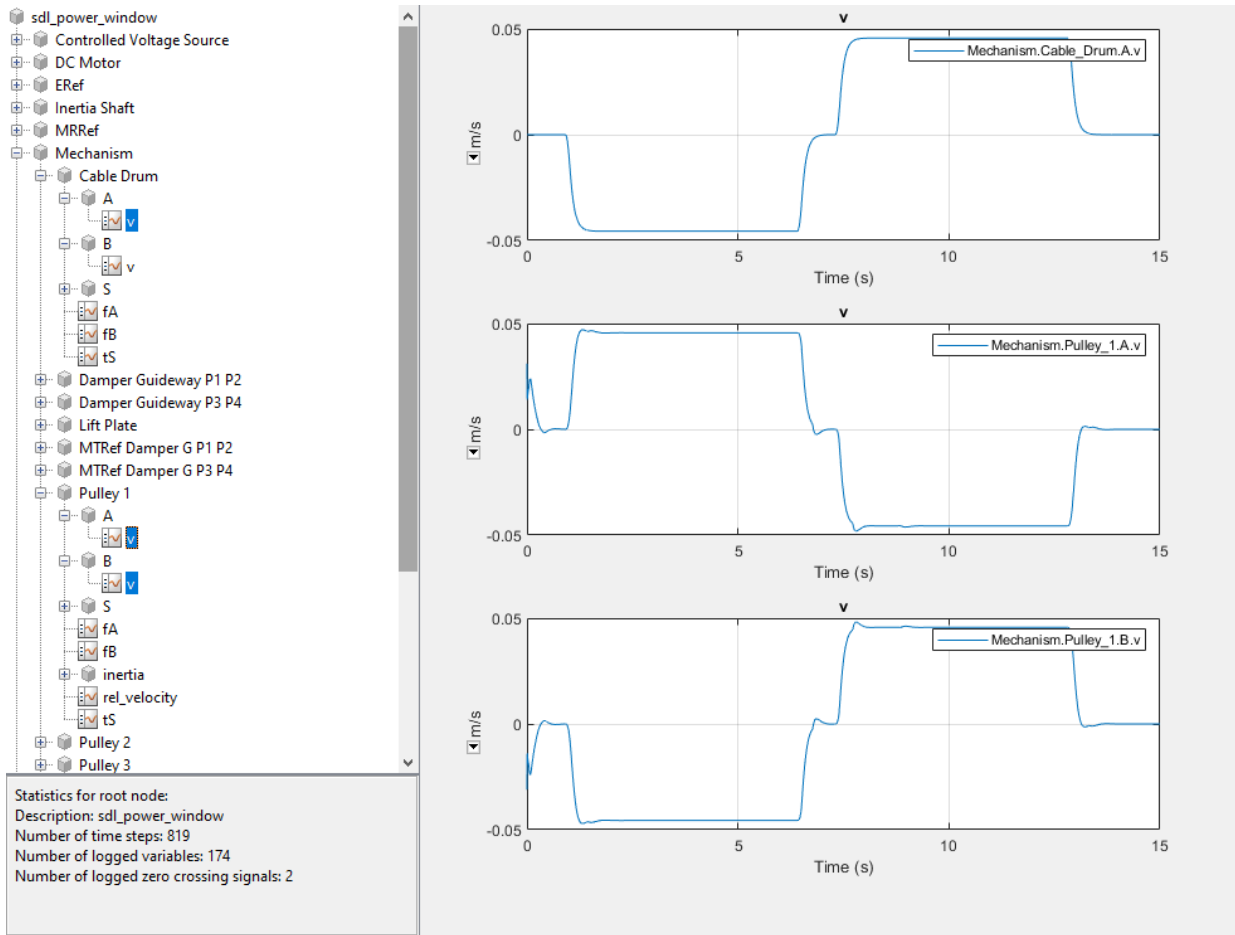


Each tensioner contains a spring and damper network that is parameterized with the spring and damping coefficients of the cable.

- 3 Run the simulation and plot the results by clicking the **Plot motor torque** link in the model canvas. When the **Cable Drum** has a negative velocity, the **Lift Plate** tends to go up, as does the window. When the **Cable Drum** has a positive velocity, both the **Lift Plate** and window tend to go down.



- 4 Open the Results Explorer by clicking the **Explore simulation results** link in the model canvas. On the Results Explorer toolbar, click the settings button  and, for the **Plot signals** parameter, select Separate. Use **Ctrl+click** to open plots for:
- **Mechanism > Cable Drum > A > v**
 - **Pulley 1 > A > v**
 - **Pulley 1 > B > v**



As expected, the velocity of the drum belt end at port **B** matches the velocity of the **Pulley 1** belt end at port **B** and is the opposite of the velocity of the **Pulley 1** belt end at port **A**.

See Also

Belt Drive | Belt Pulley | Inertia | Mechanical Translational Reference | Rope Drum | Rotational Free End | Shock Absorber | Translational Damper | Translational Spring | Variable Inertia | Worm Gear

Related Examples

- “Power Window System”

More About

- “Troubleshoot Pulley Network Issues” on page 18-6

Gear Coupling Control Using Clutches

- “How a Clutch Works” on page 7-2
- “Model Friction Clutches at a Fundamental Level” on page 7-3
- “Engage and Disengage Gears Using a Clutch” on page 7-4
- “Brake Motion Using Clutches” on page 7-9

An important requirement of a practical drivetrain is the ability to transfer rotational motion and torque among spinning components at different speeds and gear ratios. In general, a single set of gears is not sufficient to accomplish this transfer. Clutches allow the drivetrain to transfer motion, torque, and force at different gear ratios under manual or automatic control.

How a Clutch Works

A clutch makes two shafts spinning at different rates spin at a single rate by applying torques that tend to accelerate one shaft and decelerate the other. The most common way for a clutch to accomplish this action is with surface friction. Such a clutch can operate in one of these modes of motion:

- *Disengaged*: the clutch applies no friction at all.
- *Engaged but unlocked*: the clutch applies kinetic friction, and the two shafts spin at different rates.
- *Engaged and locked*: the clutch applies static friction, and the two shafts spin together.

A clutch consists of mated frictional surfaces overlapping one another and connected on either side to a shaft. If the clutch is disengaged, the frictional surfaces have no contact and the shafts spin independently. To engage the clutch, contact between two surfaces is induced by applying pressure normal to the clutch surfaces. The two surfaces in contact and moving relative to one another experience kinetic friction, which causes them to narrow their relative velocity. The friction acts to reduce the relative motion between the two clutch plates and their connected shafts. At some critical combination of reduced relative speed and pressure (normal force), the clutch locks, so that the two shafts are spinning at the same rate. The shafts remain locked together as long as the transmitted torque remains less than the static friction, which is proportional to the applied normal force. If the clutch unlocks but is still engaged, it again applies kinetic rather than static friction.

The transition between unlocked and locked states is discontinuous. Modeling a clutch locking or unlocking event requires searching for the correct combination of pressure and torque acting on the clutch. The locking and unlocking events are determined during simulation by repeated and accurate zero-crossing detection. On simulating events and solving constraints together with dynamics in Simscape models, see “Desktop Simulation” (Simscape).

Model Friction Clutches at a Fundamental Level

The Disk Friction Clutch block requires only a single pressure signal to modulate the kinetic friction. You fix all its other characteristics before starting simulation.

Modeling a friction clutch at a fundamental level requires direct control over the kinetic and static friction torques. The Fundamental Friction Clutch block gives you that control. With this block, you must specify, by either external signals or internal dynamics, the kinetic friction and static friction limits (positive and negative) of the clutch as functions of time.

Engage and Disengage Gears Using a Clutch

This example shows a gear being engaged, then disengaged, by a custom clutch. Torque and motion are transferred from one shaft to another over a finite time interval.

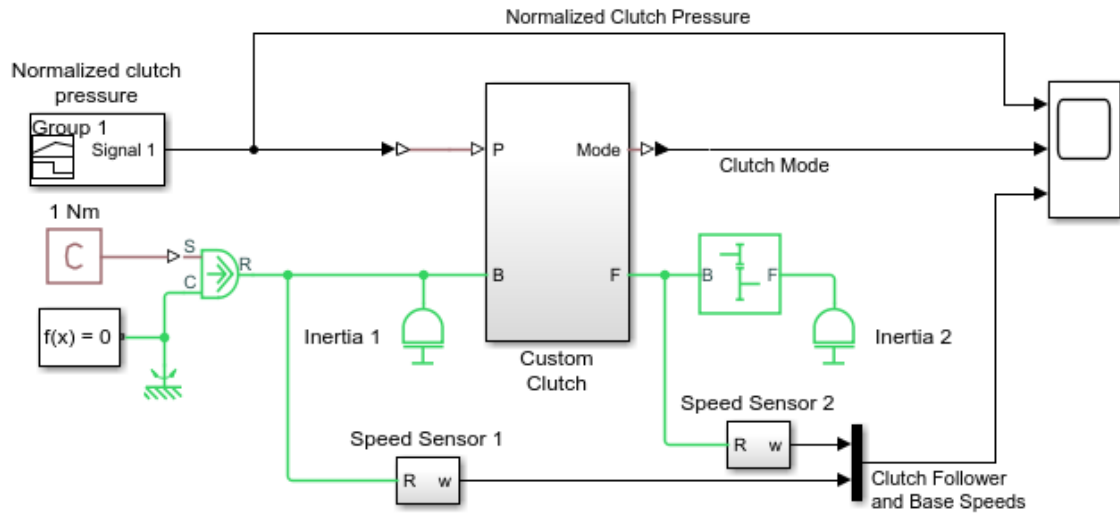
A common task in drivetrain design is transferring motion and torque at different fixed gear ratios. Drivetrains are typically designed to switch among a set of distinct gear ratios. Implementing the switch from one gear ratio to another requires gradually disengaging one set of driveline couplings and engaging another set. Clutches allow you to engage and disengage driveline shafts from one another gradually. The Disk Friction Clutch block represents a standard surface friction-based clutch that models this behavior.

The model in this example uses a custom clutch subsystem that contains a Fundamental Friction Clutch block. The Fundamental Friction Clutch block requires you to specify the static and kinetic clutch friction more completely than the Disk Friction Clutch block requires because it models clutches in greater detail. See also “Model Friction Clutches at a Fundamental Level” on page 7-3.

Note You can model continuous motion-torque transfer with the Torque Converter block, which simulates fluid viscosity instead of surface friction and does not lock.

Simulate Gear Engagement and Disengagement

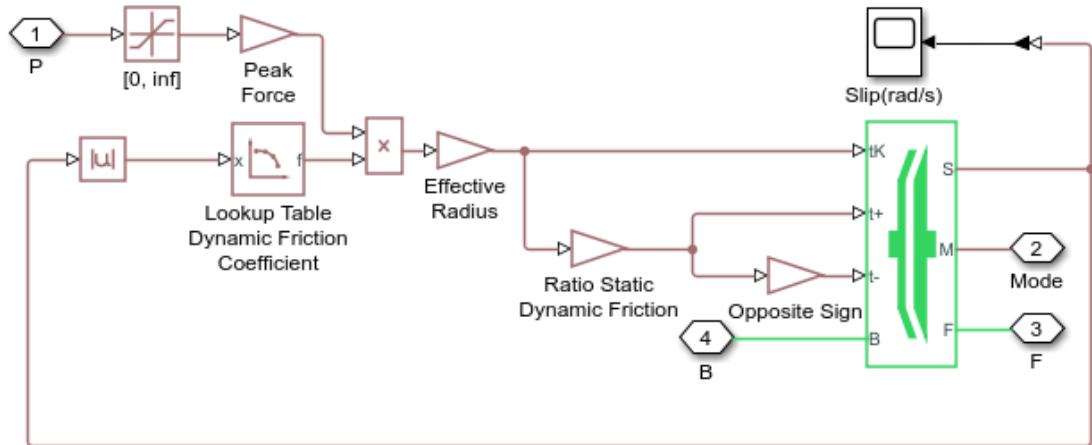
- 1 Open the model. At the MATLAB command prompt, enter
`sdl_clutch_custom`



Custom Clutch

1. [Plot speeds](#) of clutch shafts ([see code](#))
2. [Explore simulation results](#) using [sscexplore](#)
3. [Learn more](#) about this example

Custom Clutch Model with Programmed Clutch Pressure



Custom Clutch Subsystem

Model Components

- The clutch subsystem is positioned between the Inertia 1 and Simple Gear blocks and reports the clutch mode (forward, reverse, locked).
- The PS Constant block replaces the sinusoidal signal as the torque input. The torque sensor blocks are omitted.
- Simulink-PS Converter and PS-Simulink Converter blocks communicate between physical signals in the Simscape environment and Simulink blocks such as Signal Builder and Scope.
- The Signal Builder block provides the programmed clutch pressure signal, normalized between 0 and 1, as shown in the following table. This signal is converted to a physical pressure inside the clutch subsystem.

Time Range (Seconds)	Signal Value
0-2	0
2-4	0-0.8 with constant slope
4-6	0.8

Time Range (Seconds)	Signal Value
6-7	0.8-0 with constant slope
7-10	0

- 2** Open the Scopes and start the simulation. The normalized clutch pressure signal follows the profile that you created in Signal Builder and determines the behavior of the model.
- a** From 0 to 2 seconds, the velocity of Inertia 1 increases linearly because it is subject to a constant torque.
 - b** At 2 seconds, the clutch begins to engage, and Inertia 2 begins to spin. The velocity of Inertia 1 continues to rise, although at a slower rate, because the two inertias now share the external torque.
 - c** At 4 seconds, the pressure reaches its maximum. At about 5.32 seconds, the clutch locks. The driveshafts connected by the clutch now spin together. Inertia 1 and Inertia 2 continue to speed up at constant accelerations, Inertia 2 at half the velocity of Inertia 1.
 - d** At 6 seconds, the clutch begins to disengage as the pressure drops. Inertia 1 and Inertia 2 continue to accelerate with the applied torque.

The clutch unlocks at about 6.73 seconds and fully disengages at 7 seconds. (The clutch unlocks a little before completely disengaging because the pressure, even before vanishing, becomes too small to maintain the lock.) Inertia 1 is still accelerating. But Inertia 2, now free of the driveshaft and its torque, no longer accelerates and instead spins at a constant rate without frictional loss.

While the two shafts are locked, from 5.32-6.73 seconds, Inertia 1 and Inertia 2 spin in a fixed 2:1 ratio, because of the Simple Gear.



How the Clutch Mode Indicates Locking and Unlocking

The Clutch mode signal indicates the relative motion of its two connected shafts. From 0 to 5.32 seconds, the two shafts are moving relative to one another. The follower (driven) shaft is slower than the base (drive) shaft, so the mode signal is -1. Once the two shafts lock, their relative velocity is 0, and the mode signal switches to 0. At 6.73 seconds, they unlock, and the drive (base) shaft starts accelerating faster than the driven (follower) shaft. The mode signal switches back to -1.

Brake Motion Using Clutches

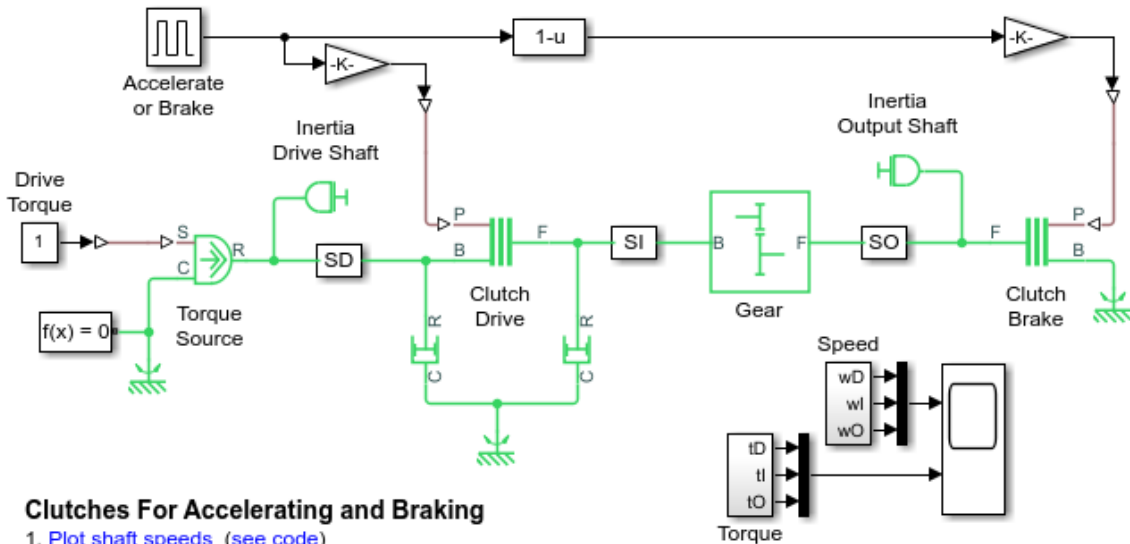
A special case of transferring motion occurs when you want to brake the spinning of a driveline component, slowing it down until it stops. The common way to brake the motion is to couple the spinning component to a rotational ground. You can represent a rotational ground with a Mechanical Rotational Reference block from the Simscape Foundation library. Because a rotational ground cannot move, a driveline axis locked to a rotational ground also cannot move. You can implement the gradual engagement or disengagement of a driveline component with a rotational ground using a clutch, just as you use a clutch to couple or uncouple two spinning shafts gradually.

Braking with a Two-Clutch System

- 1 Open the model. At the MATLAB command prompt, enter

```
sd1_clutch_acc_brake
```

The model features two clutches, one of which acts as a brake. The model also includes frictional damping for greater realism. The simulation time is set to `inf` (infinity). For simplicity, the model uses the Disk Friction Clutch block.



1. [Plot shaft speeds](#) (see code)
2. [Explore simulation results](#) using [sscexplore](#)
3. [Learn more](#) about this example

Clutch Model with Brake Clutch

This model uses the basic structure of inertia—clutch—gear—inertia. The first body, Inertia Drive Shaft block, is driven by an external torque, and the initial velocities are 0. There is, however, another clutch for the second body, Inertia Output Shaft block, that can couple Inertia Output Shaft to the Mechanical Rotational Reference block and bring it to a stop.

The switching assembly is based on the clutch switch. You can change this switch to apply a constant clutch pressure signal to either the Clutch Drive block or the Clutch Brake block. The Fcn 1-u block ensures that the full clutch pressure is applied to either one or the other, but not both at once. The Damper blocks apply viscous (velocity-dependent) friction to the spinning of the Inertia Drive Shaft and the Inertia Output Shaft.

The Accelerate or Brake block is programmed to provide a signal of 1 for the first 100 seconds of the simulation. It provides a signal of 0 for the second 100 seconds of the simulation.

2 Start the model.

During the first 100 seconds, when the Accelerate or Brake block is set to 1, the clutch pressure is applied to the Gear clutch. The Gear clutch engages and locks the driver and driven shafts and causes them to rotate at the same velocity.

The Inertia Output Shaft is on the other side of the Simple Gear. The angular velocity of the Inertia Drive Shaft is twice that of the Inertia Output Shaft because the gear ratio of the Simple Gear block is 2, follower to base. In this switch mode, no clutch pressure is applied to Brake Clutch, which remains unengaged.

After an initial transient, the system settles into a steady state of motion where the external torque balances the friction losses.

At $t = 100$ seconds, the Accelerate or Brake block switches to 0 to disengage the gear clutch and engage the brake clutch. The system undergoes another transient while the Gear clutch disengages and Brake clutch engages.

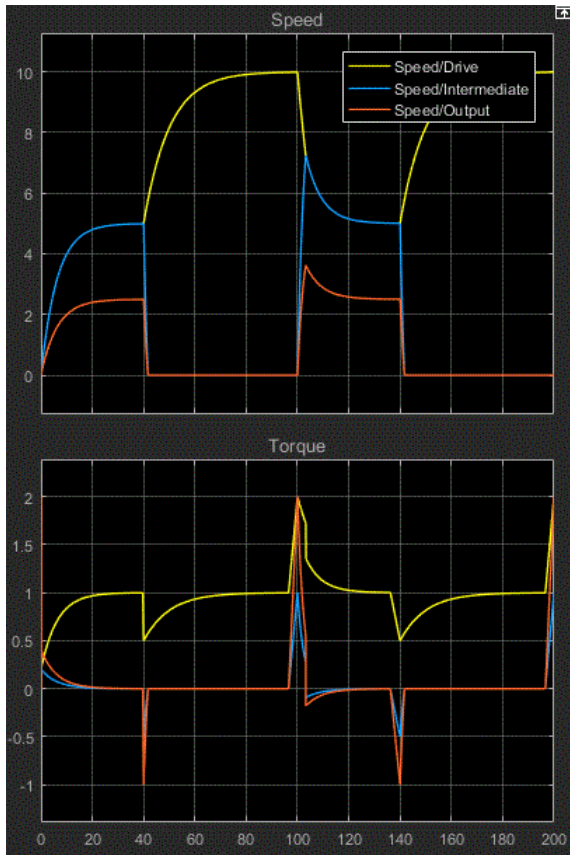
The angular velocity of Inertia Drive Shaft and the driver shaft settles down to a new steady state of 10 radians/second, twice its old speed.

Because the Gear clutch is now disengaged, the driven shaft and the Inertia Output Shaft are no longer subject to a driving torque through Gear clutch. But the Brake clutch is engaged and couples the Inertia Output Shaft to the immobile Mechanical Rotational Reference. Once engaged, the kinetic friction of the Clutch Brake brings the driven shaft and the Inertia Output Shaft to a stop.

To see the transient behavior at simulation start and when you switch the clutches:

- 1 Start the simulation and let it run for a short time. Then switch Clutch Switch to the other mode.
- 2 After a short time, stop the simulation. Use the **Autoscale** feature of the Scopes to capture the entire simulation sequence. The transients from the starting behavior and the switching transition are visible.

For example, in these plots, the model was started with Clutch Switch set to 1 (Gear clutch locked, Brake clutch disengaged, no braking). The velocities quickly climbed to their steady-state values. Then Clutch Switch was changed at about 1830 seconds of simulation time. Gear clutch disengaged and Brake clutch engaged, braking Inertia2. The angular velocity of the driver shaft rose from 5 to 10 radians/second. The angular velocity of the driven shaft dropped from 5 to 0. The angular velocity of Inertia2 dropped from 2.5 to 0.



Model Transmissions Using Gear Ratios and Clutch Schedules

- “Transmission Design Principles and Best Practices” on page 8-2
- “Model a Two-Speed Transmission with Braking” on page 8-3
- “Model a CR-CR 4-Speed Transmission Driveline with Braking” on page 8-7

In a real drivetrain, you couple an input or drive shaft to one of many output or driven shafts, or to one driven shaft with a choice of several gear ratios. The drivetrain then requires several clutches to switch between gears. You couple one of the driven shafts or one of the gear sets by engaging one of the clutches. You then switch to another output shaft or another gear ratio by disengaging one clutch and engaging another.

You can also engage more than one clutch at a time to use multiple gear sets simultaneously. Transmissions engage multiple gear sets at the same time to produce a single effective gear ratio, or *drive ratio*. Changing gears requires disengaging one set of clutches and engaging another set. You can specify the set of clutches to engage and disengage for each gear ratio in a *clutch schedule*. Designing a clutch schedule and shaping and sequencing the clutch pressure signals frequently constitute the most difficult part of transmission design. A realistic transmission model must also include losses due to friction and imperfect gear meshing.

To learn how to model transmissions using blocks from the Simscape Driveline gear and clutch libraries, see “Model a Two-Speed Transmission with Braking” on page 8-3 and “Model a CR-CR 4-Speed Transmission Driveline with Braking” on page 8-7.

Transmission Design Principles and Best Practices

When creating or modifying a transmission model:

- For a realistic simulation and to prevent acceleration singularities when torques are applied, connect inertia blocks with nonzero inertia values to gear shafts.
- To specify the engaged and free clutches for proper transmission gearing, set up a clutch schedule. Set all clutch pressures to 0 only if you want to disengage the transmission completely, that is, place it in neutral.

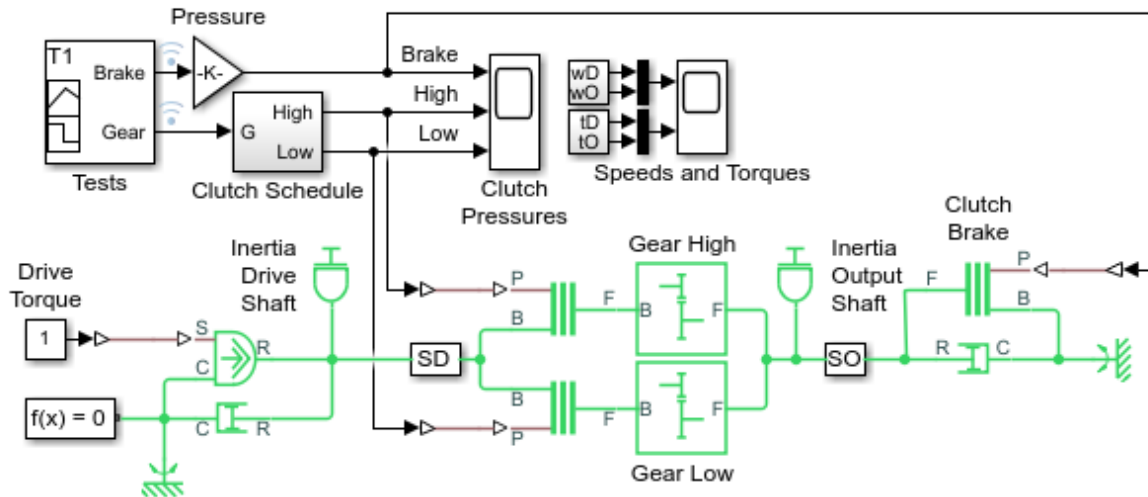
Do not engage any more or fewer clutches than necessary, at any time during simulation.

- If you redesign a transmission by adding or removing gears, you might also have to:
 - Add or remove clutches.
 - Redesign the clutch schedule.
 - Add or remove gear shaft inertias.

On the relationship clutch pressure signals to solver choices and settings, see “Driveline Simulation Performance” on page 16-2.

Model a Two-Speed Transmission with Braking

The example model `sdL_transmission_2spd` contains a driveline system that makes up a simple yet complete transmission.



Two-Speed Transmission

1. [Plot shaft speeds](#) (see code)
2. [Explore simulation results](#) using [sscexplore](#)
3. [Learn more](#) about this example

Simple Transmission with Two Gear-Clutch Pairs and Braking

The model is built on the `sdL_clutch_acc_brake` example model. This model contains two driveline shafts or axes, with a constant actuating torque of 1 Newton-meter applied to the driver shaft. Both the driver and the driven shafts are subject to small viscous damping torques. The viscous torque constant μ is 0.001 newton-meters/(radians/second). In the steady state, the driving and damping torques balance one another; the two shafts spin at constant rates, the driver shaft at $(1 \text{ N-m})/(0.001 \text{ N-m/(rad/s)}) = 1000 \text{ rad/s}$. If braking occurs, the driven shaft stops. There are now two selectable gears to couple the two axes, instead of one. For more information on modeling viscous losses with nonideal gear bearings instead of dampers, see “Model Gears with Losses” on page 11-4 and “Constant and Load-Dependent Gear Efficiencies” on page 11-6.

This transmission model couples the gears in a simple way, with each gear and the brake associated with its own clutch. Coupling one gear requires engaging and locking the corresponding clutch, while ensuring that the other two clutches are disengaged. The brake clutch is directly activated by its own switch.

Setting Up the Gears, Clutches, and Brake

The two gears are Simple Gear blocks with different gear ratios, each connected in series with its corresponding clutch. The two gear-clutch pairs are coupled in parallel. This parallel assembly then couples the driver shaft to the driven shaft, with their two spinning inertias. One gear is a “low” gear, the other a “high” gear. Following common usage for automobile gears, the “low” and “high” labels refer to the angular velocity ratios.

Note The ratio of speeds in a gear is the *reciprocal* of the gear ratio.

- The low gear is the Gear High block. You can couple the low gear by engaging its corresponding clutch, modeled by the Low gear clutch block. The gear ratio is 5:1, so that the ratio of output to input (follower to base) angular speeds is 1/5. Such a gear has a high torque transfer ratio of 5, from base to follower. In an automobile, such low gears are used to accelerate the vehicle from a stop by transferring a large torque down the drivetrain from the engine.
- The high gear is the Gear Low block, coupled by engaging its own clutch, represented by the High gear clutch block. The gear ratio is 2:1, and the angular velocity ratio of follower to base is 1/2, or 5/2 times the ratio in the low gear. The torque transfer ratio is only 2 from base to follower. An automotive high gear is used for milder acceleration or coasting once a vehicle is moving at a significant speed. The vehicle acceleration generated by this gear is less than the acceleration that is generated by the low gear.

Switching on either the Neutral switch or the Brake switch disengages both gear clutches. In either case, the driver shaft continues to spin, approaching a steady velocity, subject to the competing driving and damping torques.

- Switching the transmission to neutral leaves the brake clutch disengaged and the driven shaft free to spin. But without a driving torque, damping gradually brings the driven shaft to a stop.
- Switching on the brake immediately locks the brake clutch and stops the driven shaft.

This simple transmission is based on mapping each transmission state one-to-one with an engaged clutch. You cannot engage more than one clutch at a time without creating conflicts between gear ratios or between the driver shaft and the rotational ground.

Controlling the Transmission State with a Clutch Schedule

The requirement to engage a certain clutch or set of clutches and disengage others, both to implement transmission functions and to avoid motion conflicts between gears, is the basis for all clutch schedules. Simulink provides a number of ways to implement clutch schedules, depending on the complexity of the transmission and how much realism you require for the clutch pressure signals.

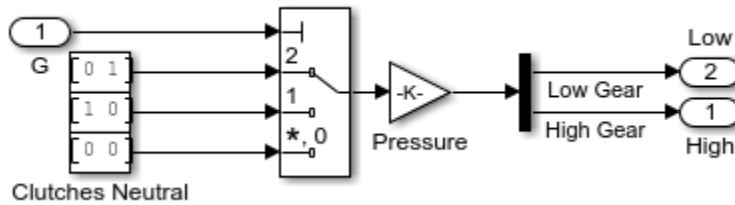
Caution To ensure that the transmission states are implemented correctly and to avoid motion conflicts among gear sets, check the clutch schedule for the transmission. To make sure that the clutches are engaged, locked, unlocked, and disengaged in a realistic and conflict-free manner, check the clutch pressure signal profiles. Unphysical or conflicting clutch schedules and clutch pressure signals lead to simulation errors in Simscape Driveline models.

For the `sdl_transmission_2spd` model, avoiding such conflicts leads to a unique clutch schedule.

Clutch Schedule for the Simple Two-Speed Transmission

Transmission State	Brake Clutch State	Low Gear Clutch State	High Gear Clutch State
Neutral/Braked	Disengaged/ Locked	Disengaged	Disengaged
Low Gear	Disengaged	Locked	Disengaged
High Gear	Disengaged	Disengaged	Locked

The model contains a simple Clutch Control subsystem to implement the clutch schedule and to output the clutch pressure signals to lock each clutch as needed.



Clutch Control Subsystem for Simple Transmission Model

Adding Realistic Clutch Signals

The clutch control subsystem of this example is adequate for a simple model, but not realistic. It contains unrealistic clutch pressure signals that rise and fall sharply. A full clutch control model requires realistic clutch pressure signals that rise from and fall back to zero in a smooth way. Greater realism requires a potentially more complex model. During simulation, the Simscape and Simulink solvers can determine transmission motion only if exactly two clutches are locked, or if all four clutches are unlocked. This model is similar to a real transmission where improperly constrained clutches can lead to lockup or damage to the transmission components. Changing the gear settings for the transmission while maintaining this requirement is an example of the central problem of transmission design.

For transmission and car examples with smoothed clutch pressure signals, see “Model a CR-CR 4-Speed Transmission Driveline with Braking” on page 8-7 and “Complete Vehicle Model” on page 3-2.

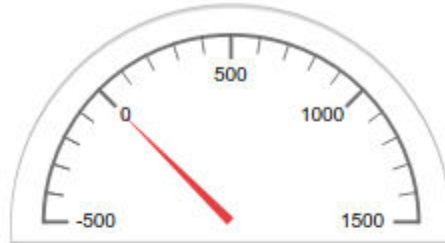
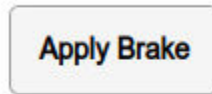
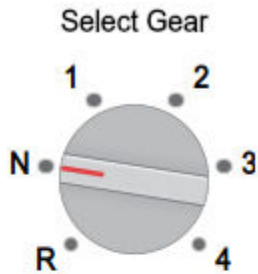
The CR-CR 4-Speed transmission subsystem couples the driver to the driven shaft (Inertia block on the right). If the transmission is disengaged, a brake clutch and fixed housing allow you to brake the driven shaft.

For clarity, the major signal buses of the model have been bundled as vectors and directed using Goto and From blocks. The Clutch Pressures are collected in the Scopes subsystem for convenience.

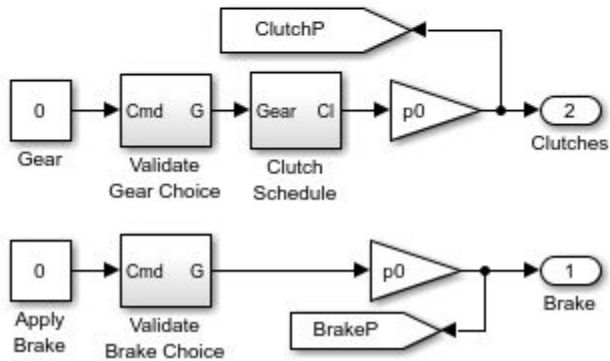
Replacing Programmed with Manually Controlled Clutch Pressures

The model represents the clutch control system using a Variant Subsystem block. To switch between **Programmed** and **Manual**, the two clutch control modes that the variant provides, click the links in the model window. During simulation, the manual subsystem provides direct control over gear changes. To switch gears in manual control mode:

- 1 In the Simulink toolbar, change the simulation time to `inf`.
- 2 Run simulation.
- 3 Change gears during using the **Select Gear** widget.



Output Shaft Speed (RPM)



Manually Switchable Clutch Control for the CR-CR Transmission

Use this subsystem as a manually switchable clutch control for the CR-CR transmission.

For complete control, change the simulation time to infinity.

Manual Clutch Control for CR-CR Transmission

Modeling Driveline Components

These sections introduce you to modeling specialized driveline components in the Simscape Driveline environment. Using predefined blocks from the Simscape Foundation and Simscape Driveline libraries and constructing custom components of your own are both emphasized.

- “Specialized and Customized Driveline Components” on page 9-2
- “Rotational-Translational Couplings” on page 9-5
- “Modeling Transmissions” on page 9-7

Specialized and Customized Driveline Components

In this section...
“Optimal Physical Modeling in the Simscape Environment” on page 9-2
“Reasons for Specialized Driveline Components” on page 9-2
“Greater Model Fidelity and Performance” on page 9-3

Optimal Physical Modeling in the Simscape Environment

Within the Simulink environment, you follow best practice if you model driveline systems by representing as much of the physical system as possible with Simscape Driveline and Simscape components. Major advantages include these features that make it easier to create accurate simulations of physical systems:

- Physical ports and connection lines supporting physical units
- Data logging
- Specialized solvers
- Consistent treatment of differential and algebraic constraint equations
- Customization with Simscape Foundation blocks and Simscape language

For custom block modeling with Simscape language, see “Custom Components” (Simscape).

Reserve Simulink blocks and signals for nonphysical aspects of modeling, such as nonphysical signals, algorithmic control, and model-level input/output tasks.

Reasons for Specialized Driveline Components

Simscape Driveline and Simscape physical connections help you create model architectures with clear physical component boundaries. You can then increase the fidelity of the model overall, or make only certain components more accurate representations of the system.

In driveline modeling, there are several reasons for making a model more complex and accurate.

Complex Component Geometries

Driveline models abstract the motion of three-dimensional mechanical systems constrained to move in one dimension. Some driveline components require extra specification to capture underlying two- and three-dimensional geometry. An example in the Simscape Driveline library is the Differential gear.

Internal Compliance Dynamics

Many standard Simscape Driveline components allow you to enable or disable modeling of internal dynamics. For example, Generic Engine allows you to represent an engine with instantaneous response for a simple, idealized model. To access a more complex and accurate representation, enable the internal time lag for the Generic Engine block. You can also create your own custom components (with internal compliance dynamics, for example), to whatever degree of fidelity and complexity that you want.

Frictional Losses

Much of complex internal dynamic Simscape Driveline modeling comes from representing the effect of both viscous and Coulomb friction.

- Viscous friction is proportional to the relative velocity of two surfaces in contact.
- Coulomb, or “sliding-sticky,” friction is proportional to the force normal to surfaces in contact. For low relative velocities, Coulomb friction causes surfaces to lock and cease relative motion.

Thus, friction models involve specification of relative geometry and motion, friction coefficients, and normal forces, of surfaces in contact.

Components with internal friction models include gears, clutches, tires, and other couplings, at different levels of optional complexity.

Greater Model Fidelity and Performance

Typically, greater fidelity of model components results in reduced simulation performance and changes the tradeoff between simulation accuracy and speed. You can adapt your simulation methods to handle greater fidelity, or reduce model fidelity to enhance performance.

For a discussion of how model fidelity and performance are related to simulation settings, see “Adjust Model Fidelity” on page 16-2 in “Driveline Simulation Performance” on page 16-2.

Rotational-Translational Couplings

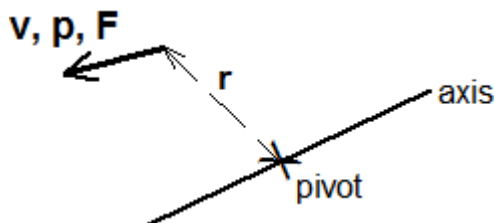
In this section...

“Convert Between Rotational and Translation Motion” on page 9-5

“Use Simscape and Simscape Driveline Elements to Couple Rotation and Translation” on page 9-5

Convert Between Rotational and Translation Motion

In general, mechanical systems mix rotational and translational motion. Rotational dynamics and motion about an axis couples to translational dynamics and motion (velocity \mathbf{v} , momentum \mathbf{p} , force \mathbf{F}) at some distance (moment arm \mathbf{r}) from the rotational pivot through a constraint that captures only motion normal to the moment arm.



While most driveline components involve rotational motion around fixed axes, certain key components transform translational and rotational motions from one to the other. Such components map motion along lines and motion around circles to one another.

The rack-and-pinion is an example of a mixed-motion gear constraint. Tires and propellers use contact friction to change driven rotational motion into forward or backward linear motion.

Use Simscape and Simscape Driveline Elements to Couple Rotation and Translation

Simscape and Simscape Driveline components that couple rotational and translational motion have mixed mechanical conserving ports of both rotational and translational type. Such blocks include wheels, tires, and certain gears:

- Leadscrew

- Rack & Pinion
- Tire (Magic Formula)
- Wheel and Axle

For an example of rotational-translational coupling with Tire (Magic Formula), see the model in “Complete Vehicle Model” on page 3-2.

Modeling Transmissions

In this section...

“Transmission Templates” on page 9-7

“Transmission Ports” on page 9-7

“Gear Input Signal” on page 9-7

“Initial States” on page 9-8

“Clutch Control” on page 9-9

“Inertias and Friction Losses” on page 9-9

“Real-Time Simulation” on page 9-10

Transmission Templates

The Transmissions library provides subsystem templates for modeling geared transmission systems with four to nine speed settings. The templates use Simscape Driveline and Simscape blocks to represent the transmission components—their gears, clutches, and brakes. An embedded Simulink subsystem defines the clutch schedule.

Use the templates as starting points and examples for your own transmission models. The template blocks are not library-linked, so you can modify them to suit your needs. Add, remove, or reconnect blocks to change the transmission structure. To capture transmission losses due to gear meshing or viscous damping, modify the block parameters.

Transmission Ports

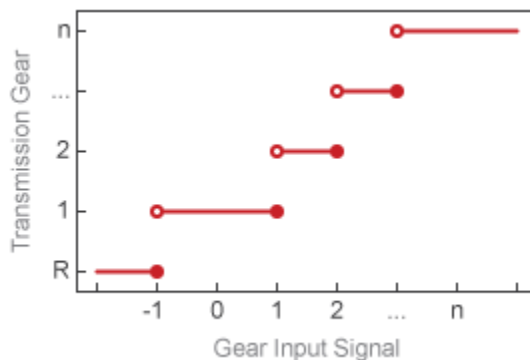
The transmission template blocks each have three ports. Ports **B** and **F** are rotational conserving ports. They represent, interchangeably, the input and output shafts. The **Gear** port is a Simulink input port that you can use to shift gears during simulation.

Gear Input Signal

The input signal to the **Gear** port sets the transmission gear according to the expression:

$$\text{Gear Setting} = \begin{cases} \text{Rev, if Gear} \leq -1 \\ 1^{\text{st}}, \text{ if } -1 < \text{Gear} \leq +1 \\ 2^{\text{nd}}, \text{ if } +1 < \text{Gear} \leq +2 \\ \dots, \dots \\ n^{\text{th}} \text{ if } (n-1) < \text{Gear} \leq n \end{cases}$$

The figure shows the correspondence between the transmission gear and the Gear input signal.



Initial States

The transmission templates are configured to begin simulation in first gear. To change the default initial gear, you must set the clutch and brake initial states to the values shown in the transmission clutch schedules. You change the initial states in the **Initial Conditions** tab of the block dialog boxes.

Consider the Lepelletier 6-Speed Transmission template. The transmission clutch schedule shows the clutch and brake initial states for reverse gear to be [1 0 0 1 0] in the order A-E. To begin simulation in reverse gear, you must then set the initial states in the block dialog boxes as follows:

- Clutch A locked
- Clutch B unlocked
- Clutch C unlocked

- Clutch D locked
- Clutch E unlocked

The Gear input signal must agree with the clutch and brake initial states. If the initial states correspond to reverse gear, then the first value in the Gear input signal must also correspond to reverse gear ($Gear \leq -1$). Matching the two in this way helps to avoid inconsistent states known to cause simulation errors.

Clutch Control

The clutch schedule converts the Gear input signal into clutch and brake input signals. These signals drive the clutches and brakes, causing some to lock and others to unlock in an orchestrated fashion. The resulting configuration determines which gears power flows through—and therefore which gear the transmission is in.

A Gain block scales the input signals to the proper magnitudes for actuating the clutches and brakes. These magnitudes depend on the actuation inputs expected by the Clutch and Brake blocks. The actuation inputs can be:

- Shift linkage displacements in Dog Clutch blocks
- Normal forces in Cone Clutch and Loaded-Contact Rotational Friction blocks
- Plate pressures in Disk Friction Clutch blocks

Transfer Function blocks smooth the otherwise discrete input signals using first-order filters. The smoothing allows the clutch state transitions to occur gradually over short periods of time rather than instantaneously. The transfer function time constants determine the characteristic time periods over which the clutch transitions occur.

Inertias and Friction Losses

Simscape Inertia blocks represent the inertias of transmission gears. These blocks enhance the accuracy of the model. They also prevent simulation errors due to zero-inertia gear sets disconnected from input and output shafts due to clutch unlocking.

By default, all frictional losses are set to zero. Frictional losses include losses due to meshing in gears and viscous damping in gears, clutches, and brakes. To account for frictional losses in your model, you must specify gear efficiencies and friction coefficients in the block dialog boxes.

Real-Time Simulation

Transmission components such as gears and clutches can slow down simulation by introducing zero-crossing events and complex state-change calculations to your model. To minimize their impact on simulation speed, several blocks provide optional parameterizations suited for real-time simulation. These parameterizations include:

- **Friction clutch approximation** in dog clutches — Reduces model stiffness due to backlash-induced vibrations.
- **No meshing losses** in gears — Eliminates gear friction calculations and associated zero crossings due to motion reversals.

These block parameterizations are labeled **Suitable for HIL**. Use them if you intend to run any type of real-time simulation, including hardware-in-loop (HIL) and software-in-loop (SIL) simulation. By default, all gear and clutch blocks in the transmission templates are set to use these parameterizations.

See Also

4-Speed CR-CR | 4-Speed Ravigneaux | 6-Speed Lepelletier | 7-Speed Lepelletier | 8-Speed | 9-Speed

Related Examples

- “Transmission Testbed”

Effective Inertias and Driveshafts

- “Model a Variable Inertia” on page 10-2
- “Model Driveshafts with Loss” on page 10-4

Model a Variable Inertia

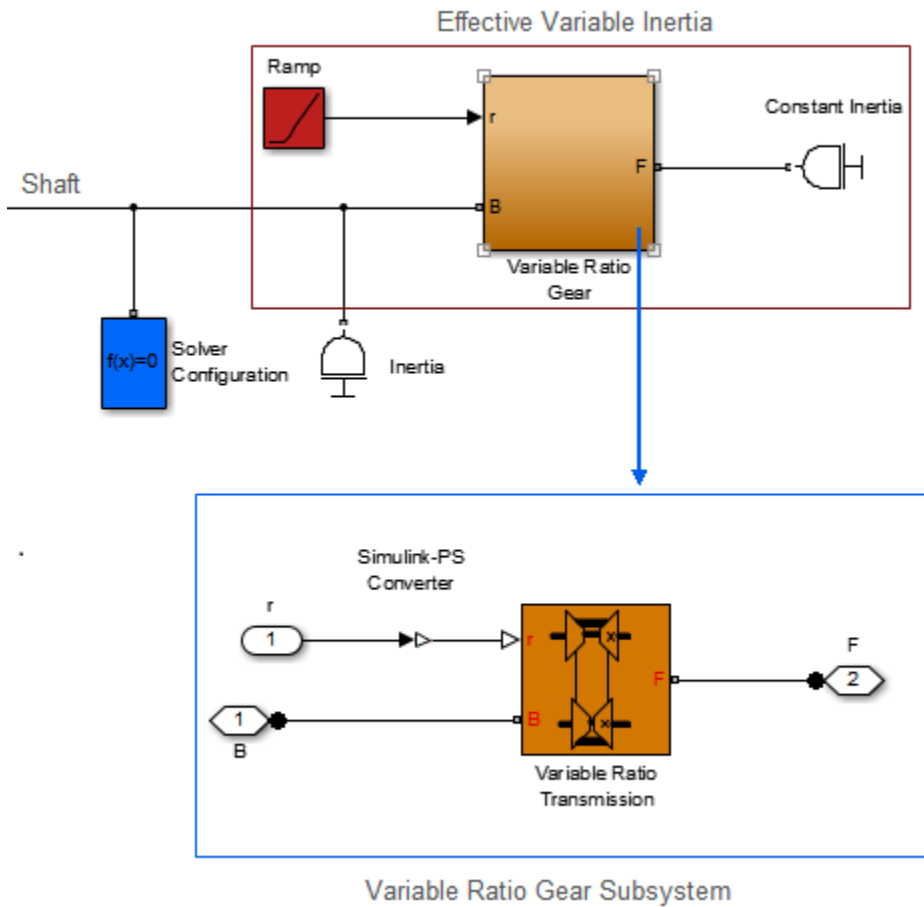
You cannot vary the inertia value of an Inertia block during a simulation. However, you can model a time-varying inertia indirectly with a Variable Ratio Transmission block. The Inertia block interacts with the rest of the system, including any applied torque, only through the gear with the time-varying gear ratio.

- 1 Place a Variable Ratio Transmission between a shaft and an Inertia.
- 2 Connect this constant Inertia to the Transmission base (B) or follower (F) port.
- 3 Vary the gear ratio $g(t)$ of the Variable Ratio Transmission with an incoming physical signal.

By changing the gear ratio, you change the effective inertia I_{eff} on the shaft from the constant Inertia (value I). I_{eff} is the effective inertia presented to the rest of the system as torque is applied, through the variable ratio gearbox, on the Inertia.

- If the B port is connected to the constant Inertia, $I_{\text{eff}} = I \cdot [g(t)]^2$
- If the F port is connected to the constant Inertia, $I_{\text{eff}} = I / [g(t)]^2$

In this diagram, the Variable Ratio Transmission is contained within the Variable Ratio Gear subsystem.



Effective Variable Inertia with a Variable Ratio Gearbox

Model Driveshafts with Loss

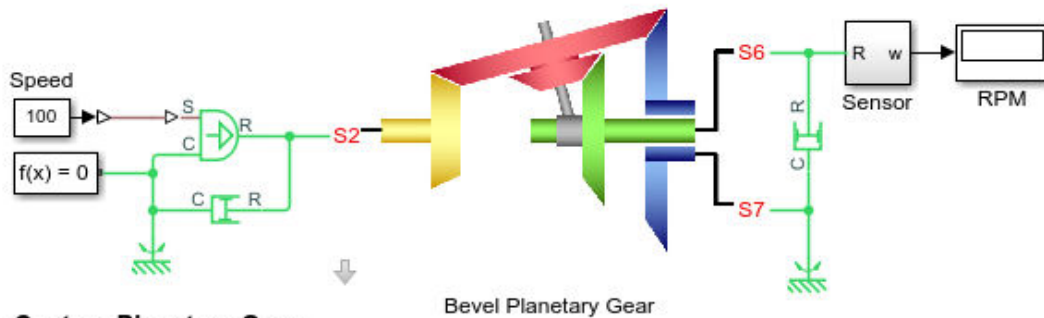
Realistic driveshafts experience damping from viscous friction, which is proportional to the driveshaft angular velocity. You can model such damping with the Rotational Damper and, if necessary, build complex damping subsystems from this block.

Specialized Gears

Make custom gears blocks the represent gear subcomponents. Create realistic models with gears that experience frictional, thermal, or load-dependent losses.

Custom Planetary Gear Model

While the Simscape Driveline library contains a Planetary Gear, you can create your own custom planetary gear using the **Planetary Subcomponents** sublibrary. The `sdl_gear_planetary_custom` example model combines three Sun-Planet Bevel subgears into a masked subsystem to model a coupled planetary gear train. The model uses an Ideal Angular Velocity Source to place a fixed velocity demand on the gearbox input, while damping the system on both the input and output driveshafts with Rotational Damper blocks.

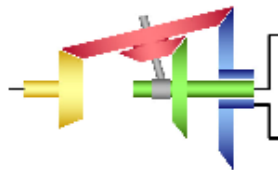
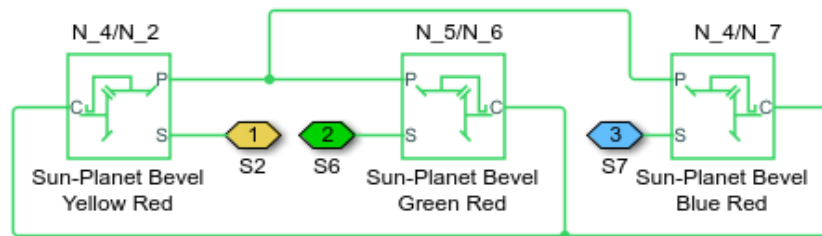


Custom Planetary Gear

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



Custom Planetary Gear System and Subsystem



See Also

Compound Planetary Gear | Ideal Angular Velocity Source | Rotational Damper | Sun-Planet Bevel

Model Gears with Losses

The blocks of the Simscape Driveline Gears Library contain optional built-in models of frictional losses, allowing you to represent nonideal gear couplings. In a nonideal gear pair (1,2), the angular velocity, gear radii, gear teeth constraints, and gear ratio $g_{12} = r_2/r_1 = \omega_1/\omega_2$ are unchanged. The transferred torque and power are reduced by:

- Coulomb friction between imperfectly meshing teeth surfaces on gears 1 and 2, parameterized by an efficiency η , $0 < \eta \leq 1$. This efficiency depends on the torque load on the teeth. But it is often approximated as constant.
- Viscous coupling of driveshafts with bearings, parameterized by viscous friction coefficients μ .

Constant Efficiency

In the simplest nonideal gear loss model, the efficiency η_{12} of meshing in gear pair (1,2) is constant, independent of load (torque or power transferred).

- The friction loss represented by η_{12} is effectively applied in full only if the transmitted power is greater than the power threshold p_{th} . Below this value, a hyperbolic tangent function smooths the efficiency factor, lowering the efficiency losses to zero when no power is transmitted.
- For gear sets with a carrier, η_{12} represents the ordinary efficiency, defined when the carrier is not moving.

For gears with different efficiencies for the forward and reverse power flow:

- $ForwardLoss = (1 - \eta_{FB})$, η_{FB} is the torque transfer efficiency from the follower shaft to the base shaft.
- $BackwardLoss = (1/\eta_{BF} - 1)$, where η_{BF} is the torque transfer efficiency from the base shaft to the follower shaft.

The frictional torque is calculated as:

$$T_f = T / 2((ForwardLoss + BackwardLoss)\tanh(4p / p_{th}) + ForwardLoss - BackwardLoss)$$

where:

- T is the transferred torque.

- p is the transferred power.
- p_{th} is the power threshold at the base shaft above which full efficiency losses are in effect.

For certain gear models, such as the Simple Gear, efficiency is assumed equal for both the forward and reverse power flow, $\eta_{BF} = \eta_{FB}$.

Load-Dependent Efficiency

Making η dependent on the load is a way to make the loss model more accurate. For an example of load-dependent efficiency, see the Simple Gear block reference page.

Geometry-Dependent Efficiency

Making η dependent on the geometry of gear meshing is another way to make the loss model more accurate. For an example of geometry-dependent efficiency, see the Leadscrew block reference page.

Viscous Friction

On a driveshaft mounted to a gear wheel by lubricated, nonideal bearings, the viscous friction experienced by the axis is controlled by the viscous friction coefficient μ . The viscous friction torque on a driveshaft "a" is $-\mu_a \cdot \omega_a$, where ω_a is the angular velocity of the driveshaft with respect to its mounting or carrier (if a carrier is present).

See Also

More About

- "Constant and Load-Dependent Gear Efficiencies" on page 11-6

Constant and Load-Dependent Gear Efficiencies

Here, you revisit the `sdL_transmission_2spd` model in “Model a Two-Speed Transmission with Braking” on page 8-3. You reconfigure the Simple Gears to model power loss due to nonideal meshing. The effect of viscous bearing losses is ignored.

- 1 Open the `sdL_transmission_2spd` model and simulate to check the ideal gear behavior.
- 2 Open the Gear High and the Gear Low blocks. Under **Meshing Losses**, in the **Friction model** drop-down list, choose `Constant efficiency` for both. Enter efficiencies less than 1, but greater than 0. For example, for the Gear Low block, enter 0.7; and for the Gear High block, enter 0.95.
- 3 Leave the other settings as they are, including zero viscosity. Close the blocks.
- 4 Restart the model. The driveline runs at a lower efficiency and slightly smaller angular velocities, because of the power losses. If you enter different efficiency factors for the two gears, the effect of the loss is different if you switch between gears.

Experiment with load-dependent efficiency. In the **Friction model** drop-down menu, choose `Load-dependent efficiency` instead. In that case, you need more efficiency model details to specify.

See Also

More About

- “Model Gears with Losses” on page 11-4

Specialized Clutches

Create realistic models with clutches that experience frictional losses and smooth pressure signals.

Clutches, Clutch-Like Elements, and Coulomb Friction

Coulomb friction acts along the plane of contact between two solid surfaces, in opposition to their actual or potential relative motion, and in proportion to the normal force pushing the surfaces together. It encompasses both kinetic friction, applied when the surfaces are in relative motion, and static friction, applied when they are locked together. Coulomb friction is the basis for clutches and clutch-like elements that rely on normal forces to keep surfaces in contact. When the relative speed of the surfaces becomes small enough and a normal force is applied, these elements lock and move together.

Realistic friction models often include viscous friction. This type of frictional force or torque is proportional to the relative translational or rotational velocity of the two surfaces in contact.

The Clutches library contains various clutch types, including single- and multi-plate, friction, cone, and dog (positive) clutches. You can customize the fundamental clutch blocks to meet your requirements. The Brakes & Detents library provides brake, detent, and friction blocks. These clutch-like elements apply Coulomb friction forces or torques between pairs of translating or rotating axes in loaded contact. Many also allow inclusion of viscous friction. Once engaged, clutches and brakes act to decelerate the relative motion of surfaces in contact and can lock the surfaces together under certain conditions.

Clutches and clutch-like elements have a dual role in a driveline model. When engaged but not locked, they act as *dynamic elements*, generating torques and forces between driveline axes in relative motion. When locked, they act as *conditional* or *dynamic constraints*, locking driveline axes to move together. Such constraints are conditional, because they can unlock, unlike gears.

For more information on clutches and dynamic constraints, see “Driveline Degrees of Freedom” on page 16-38 and “Driveline States — Effect of Clutches” on page 16-52.

See Also

More About

- “Model Clutches with Viscous Friction Loss” on page 12-3

Model Clutches with Viscous Friction Loss

A source of loss in a clutch system coupling two driveshafts comes from viscous friction at the two shaft bearings. Consider the `sdL_clutch_custom` model presented in “Engage and Disengage Gears Using a Clutch” on page 7-4. Here you add a kinetic friction torque proportional to the angular velocity on both sides of the clutch (viscous friction). The Simscape Foundation library provides a Rotational Damper block that represents such a damper. The angular motion of the driveshafts is relative to another component. Here the angular velocities of the shafts are measured relative to rotational ground, represented by Mechanical Rotational Reference. You can make a friction subsystem that applies such a torque to any driveline axis connected to it. You can copy the subsystem and modify the existing clutch model by connecting the two copies on either side of the clutch.

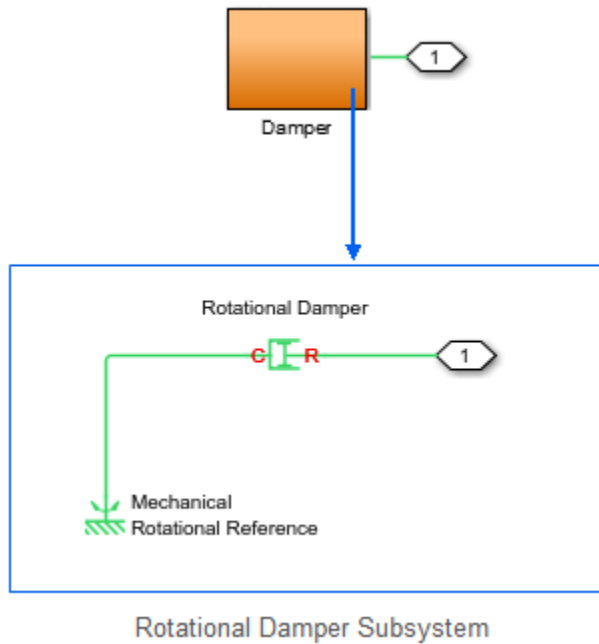
Note The velocity used in this damping is the absolute velocity of a single shaft relative to rest. If you had *two* rotating driveline shafts and wanted to exert a relative damping between them as a function of their *relative* velocities, use the same Rotational Damper block connected between the two axes.

Creating a Torque Damping Subsystem

The viscous friction torque is $\tau_{\text{fric}} = -\mu\omega$, where μ is the viscous friction coefficient. To implement this torque:

- 1 You can start with your modified model from the “Engage and Disengage Gears Using a Clutch” on page 7-4 tutorial or with the `sdL_clutch_custom` model. In either case, open the model.
- 2 From the Simscape library, copy Mechanical Rotational Reference, Rotational Damper, and Connection Port into your model window.
- 3 From the Simscape Driveline **Couplings & Drives > Springs & Dampers** library, copy the Rotational Damper block into your model window.
- 4 Connect the Mechanical Rotational Reference to the case (C) port of the Rotational Damper and the rod (R) port of the Rotational Damper to the Connection Port.
- 5 For the Rotational Damper block, for the **Damping coefficient**, enter 0.3. Leave the default units.
- 6 Select the whole connected three-block set, and create a subsystem. Name the subsystem `Damper 1`.

- 7 Create a second copy of Damper. Name the new subsystem Damper 2.

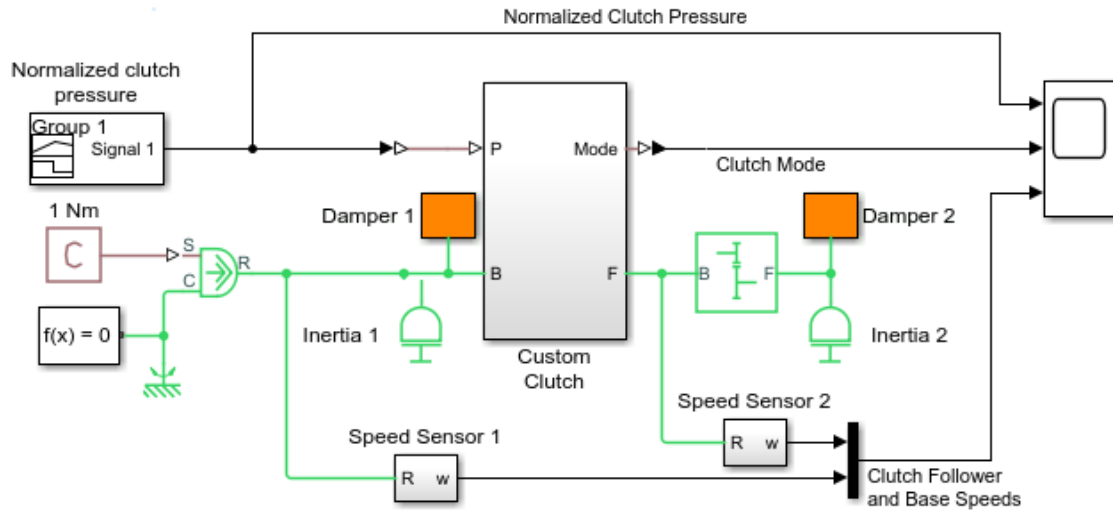


Rotational Damping Subsystem

Connecting and Simulating the Damped Clutch System

Complete and run the model.

- 1 Connect the two Damper subsystems to the driveline of the clutch model, as shown in the figure.

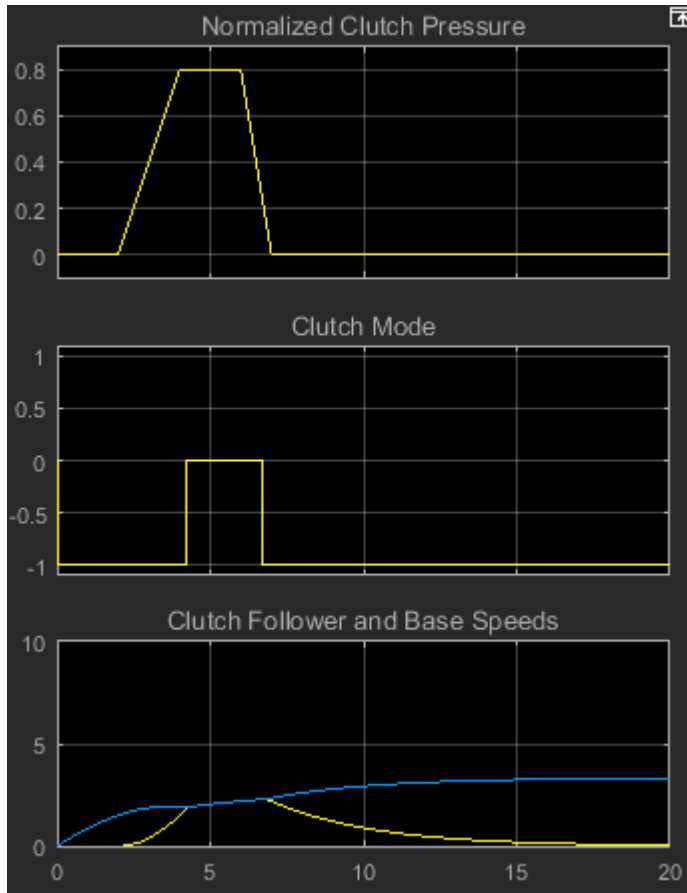


Custom Clutch

1. [Plot speeds](#) of clutch shafts ([see code](#))
2. [Explore simulation results](#) using [sscexplore](#)
3. [Learn more](#) about this example

Damped Custom Clutch Model

- 2 Change the simulation time to 20 seconds.
- 3 Open the Scope blocks and click **Start**. To see the full plots, readjust the horizontal axes of the Scope using **Autoscale**.



The clutch pressure and external torques are applied as before. The shafts now rotate slower because of the damping.

As before, Inertia2 begins to spin when the clutch starts to engage at 2 seconds. After the clutch locks at 4 seconds, the body continues to accelerate, at a slower rate than it did without damping. At about 6.7 seconds, the clutch begins to disengage and completely disengages at 7 seconds. Subject to friction, Inertia2 now starts to slow down, unlike in the friction-free case. Once the external torque is removed, its angular velocity drops exponentially with time.

The behavior of Inertia1 is more complex. It begins to spin up, at a lower rate than before, because of the damping. From 2-7 seconds in the simulation, Inertia1 shares the external torque with Inertia2 via the Clutch and the Simple Gear. After 7 seconds, the external torque applies to Inertia1 alone. It continues to accelerate, at an ever-slowing rate, because of the damping. If you let the simulation run without stopping, Inertia approaches its terminal angular velocity, a state where the frictional torque exactly balances the externally applied torque. This terminal velocity is $\omega_{\text{term}} = \tau_{\text{ext}}/\mu$ or $1/0.3 = 3.3333$ radians/second. The third Scope plot approaches this terminal value.

See Also

More About

- “Clutches, Clutch-Like Elements, and Coulomb Friction” on page 12-2

Model Realistic Clutch Pressure Signals

The most critical addition that you can make to clutch models for greater realism is to change the clutch pressure signals from step functions (0 to 1, or 1 to 0) to signals with a smooth rise and fall. This greater realism results in a more complex model. At any simulation time, it is critical for your model to determine transmission motion by locking exactly the correct number of clutches. If all clutches are unlocked, the transmission is in neutral. Changing the gear settings of a transmission while maintaining this requirement is one of the central problems of transmission design.

Such transmission and vehicle models as `mdl_transmission_4spd_crcr` and `mdl_car` switch gear settings without placing their transmissions in neutral. Controlling an actual manual transmission requires moving the transmission out of gear and into neutral, picking a new gear setting, and then putting the transmission into the new gear.

In the `mdl_transmission_4spd_crcr` example, with manual transmission control, you can mimic these steps by turning on the Neutral Switch, changing the gear setting, then turning off slipping the Neutral Switch.

In a programmed transmission control model, you can filter clutch pressures with Transfer Fcn blocks, shaping the pressure signals from sharp steps to smooth rises or falls.

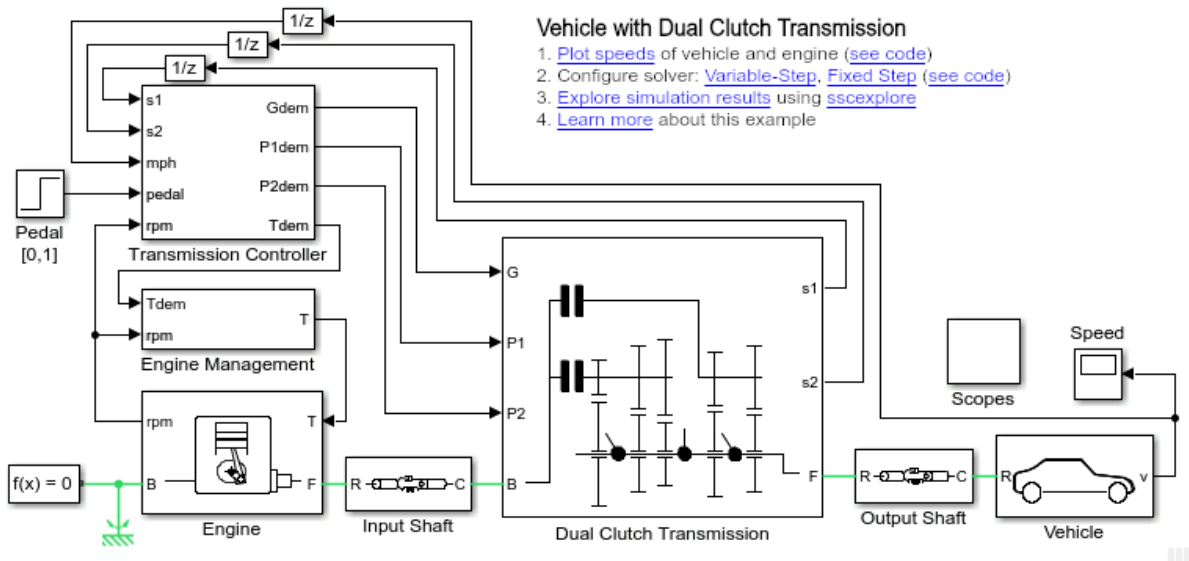
“Automatic Transmission with a Dual Clutch” on page 12-9

Automatic Transmission with a Dual Clutch

The `sdl_vehicle_dual_clutch` example model illustrates important points about both physical and control design modeling using the Simscape and Simscape Driveline environment and libraries.

The model represents an automatic transmission with two clutches. In an automatic transmission, an engine management subsystem decides when to change gear ratio and what the next gear ratio is. The pedal deflection imposed by a vehicle driver is converted into a demanded engine torque. The torque demanded and the current forward vehicle speed together determine which gear ratio the transmission switches to before it actually switches (gear preselection). By gradually lowering the clutch pressure, the transmission control system smoothly unlocks and disengages the clutch configuration for the current gear ratio. At the same time, the control system gradually raises the clutch pressures to achieve the new gear ratio by engaging and locking the new clutch configuration.

- The physical components of the transmission, from the engine, gears, and clutches, to the vehicle body and tire, are modeled using Simscape Driveline blocks, with physical ports and connections.
- The algorithmic control of the transmission, including the gear-switching and transmission control, is modeled using normal Simulink blocks, with signal ports, signal lines, and enabled subsystems.



Vehicle with Dual Clutch Model

Predefined Simulation Options

The model is also set up to allow you to switch between two common simulation configurations. To configure the model to simulate with a variable-step global solver or a fixed-step local solver, click the links in the description.

You can directly adjust all the solver options by opening the model Configuration Parameters and the network Solver Configuration dialog boxes.

See Also

More About

- “Model Realistic Clutch Pressure Signals” on page 12-8

Control Vehicle Velocity

Control Vehicle Throttle Input Using a Powertrain Blockset Driver

In this section...

“Open-Loop Simulation Using a Signal Builder Block” on page 13-2

“Closed-Loop Simulation Using a Longitudinal Driver Block” on page 13-4

“Simulation Comparison” on page 13-10

This example shows how to control throttle input to a Simscape Driveline vehicle model using a Powertrain Blockset Longitudinal Driver block. You add the driver to an open-loop model that uses a Signal Builder block for feedforward control. Adding the driver allows you to model closed-loop control by supplying a reference velocity and a feedback loop.

Open-Loop Simulation Using a Signal Builder Block

In the open-loop simulation, a Simulink Signal Builder block is used to ramp up the throttle. Simulate the model to see the open-loop response.

- 1 Open the model. At the MATLAB command prompt, enter this code.

See Code

```
%% Model information
modelName = 'sdl_vehicle_4wd';
```


```
%% Run the original model
open_system(modelName)
```

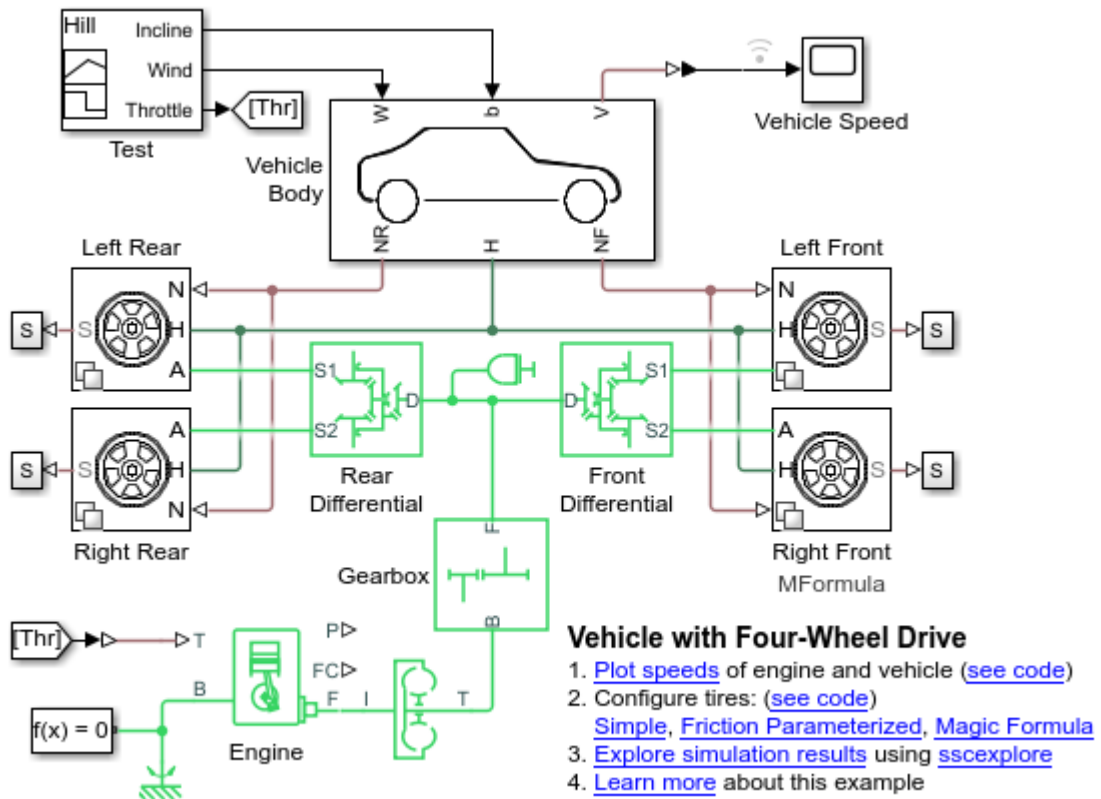
- 2 Enable the signal that goes to the **Motor RPM** scope block for Simulink data logging and viewing with the Simulation Data Inspector.

See Code

```
%% PS-S Converter1 information
pStoSConverter1Name = 'PS-Simulink Converter1';
pStoSConverter1Path = [modelName, '/', pStoSConverter1Name];
pStoSConverter1PortHandles = get_param(pStoSConverter1Path, 'PortHandles');
pStoSConverter1Inport = pStoSConverter1PortHandles.LConn(1,1);
pStoSConverter1Outport = pStoSConverter1PortHandles.Outport(1,1);
```

```
%% Enable the signal that goes to the Vehicle Speed scope block for
% Simulink(TM) data logging and viewing with the Simulation Data Inspector
set_param(pStoSConverter1Outport, 'DataLogging', 'on')
```

The logging badge  marks the signal in the model.



- 3 Increase simulation time to get steady-state results. Simulate the model.

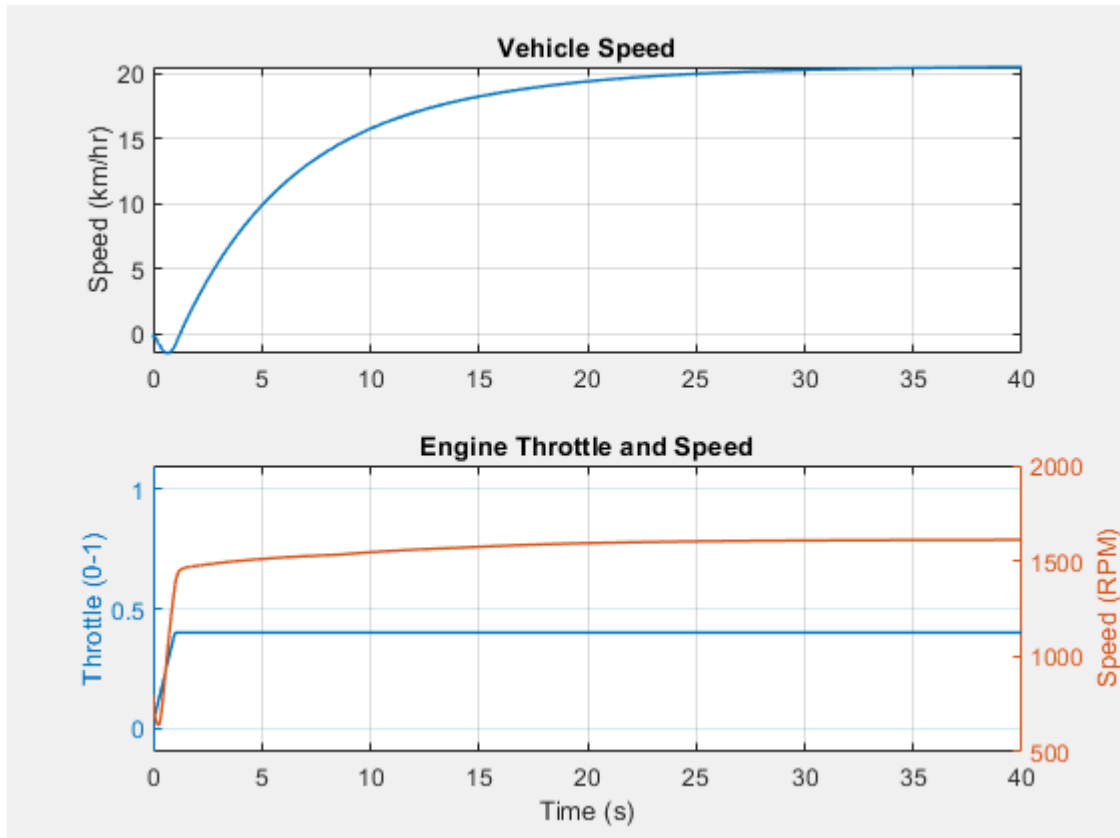
See Code

```

%% Increase simulation time
set_param(modelName, 'StopTime', '30')

%% Simulate the model
sim(modelName)
% Plot the vehicle speed, the engine speed, and the throttle input.
sdl_vehicle_4wd_plotspeed

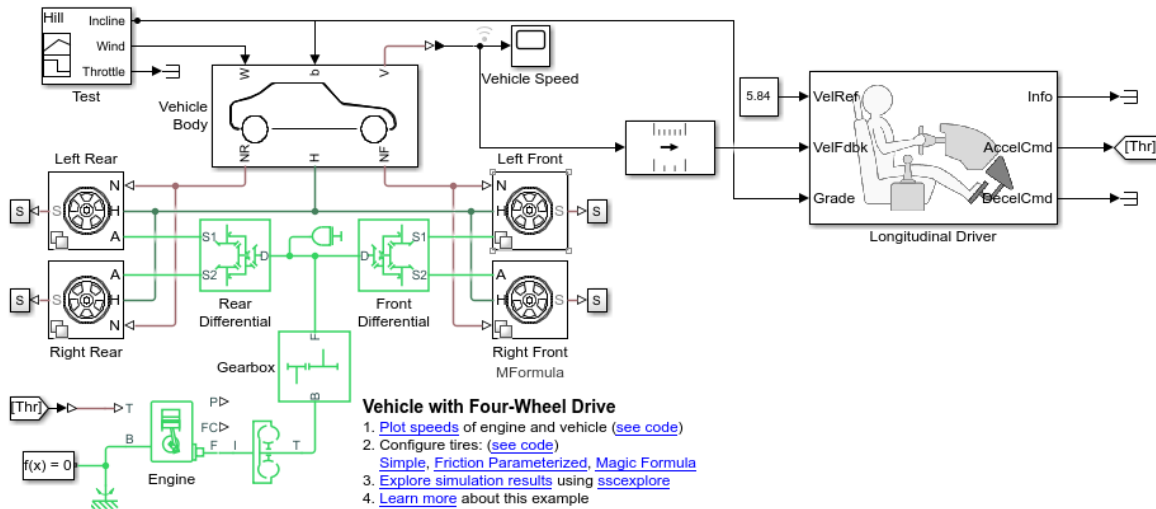
```



Even though the throttle input is nonzero at simulation time 0, the vehicle rolls down the hill at first because the input is too small to overcome the gravitational force of the incline. When the input is large enough, the vehicle accelerates up the hill and settles at a velocity of ~20 km/h.

Closed-Loop Simulation Using a Longitudinal Driver Block

To control throttle input, add a Powertrain Blockset Longitudinal Driver block.



- 1 Add a Longitudinal Driver block to the model.
 - a Expand the model window to fit the Longitudinal Driver block.

See Code

```
%% Update the model canvas
modelWindowNewLocation = [47 100 1072 723];

% Increase the size of the model window
set_param(modelName, 'location', modelWindowNewLocation);
```

- b Add the Longitudinal Driver block.

See Code

```
%% Longitudinal Driver block name, path, %
% and position information
longDriverName = 'Driver';
longDriverLibPath = 'autolibscenario/Longitudinal Driver';
longDriverPath = [modelName, '/Longitudinal Driver'];
longDriverPos = [710 15 905 135];

% Add a driver block
```

```

add_block(longDriverLibPath,longDriverPath,...
    'position',longDriverPos)

% Longitudinal Driver block port information
longDriverPortHandles = get_param(longDriverPath,'PortHandles');
longDriverPortVelRef = longDriverPortHandles.Inport(1,1);
longDriverPortVelFdbk = longDriverPortHandles.Inport(1,2);
longDriverPortGrade = longDriverPortHandles.Inport(1,3);
longDriverPortInfo = longDriverPortHandles.Outport(1,1);
longDriverPortAccelCmd = longDriverPortHandles.Outport(1,2);
longDriverPortDecelCmd = longDriverPortHandles.Outport(1,3);

```

- c Change the source of throttle input from the Signal Builder block **Throttle** port to the Longitudinal Driver block **AccelCmd** port and terminate the unconnected **Throttle** port with a Terminator block.

See Code

```

%% Signal Builder block information
signalBuildName = 'Test';
signalBuildPath = [modelName,'/',signalBuildName];
signalBuildPortHandles = get_param(signalBuildPath,'PortHandles');
signalBuildPortIncline = signalBuildPortHandles.Outport(1,1);
signalBuildPortWind = signalBuildPortHandles.Outport(1,2);
signalBuildPortThrottle = signalBuildPortHandles.Outport(1,3);

%% Throttle Goto block information
goToThrottleName = 'Goto';
goToThrottlePath = [modelName,'/',goToThrottleName];
goToThrottlePortHandle = get_param(goToThrottlePath,'PortHandles');
goToThrottleInport = goToThrottlePortHandle.Inport(1,1);
goToThrottleNewPos = [950 66 985 84];

% Delete the connection line from the Signal Builder to the Throttle Goto
delete_line(modelName,signalBuildPortThrottle,goToThrottleInport)

% Move the throttle Goto to the Longitudinal Driver
set_param(goToThrottlePath, 'position', goToThrottleNewPos)

% Add a connection line from the Longitudinal Driver AccelCmd
% port to the Throttle goto
add_line(modelName,longDriverPortAccelCmd,goToThrottleInport)

%% Terminator0 Block Information
% Terminator0 block name, path, and position information
terminator0Name = 'Terminator0';
terminator0LibPath = 'simulink/Sinks/Terminator';
terminator0Path = [modelName,'/',terminator0Name];
terminator0Pos = [200 5 220 25];

% Add Terminator0 block
add_block(terminator0LibPath,terminator0Path,...

```

```

        'position', terminator0Pos)

% Terminator0 port information
terminator0PortHandle = get_param(terminator0Path, 'PortHandles');
terminator0Inport = terminator0PortHandle.Inport(1,1);

% Connect Longitudinal Driver AccelCmd Output to the Thr Goto inport
add_line(modelName, signalBuildPortThrottle, terminator0Inport)

```

- d Terminate the **Info** and **DeclCmd** outputs on the Longitudinal Driver block

See Code

```

%% T1
% Terminator1 block name, path, and position
% information
terminator1Name = 'Terminator1';
terminatorLibPath = 'simulink/Sinks/Terminator';
terminator1Path = [modelName, '/', terminator1Name];
terminator1Pos = [955 105 975 125];

% Add Terminator1 block
add_block(terminatorLibPath, terminator1Path, ...
    'position', terminator1Pos)

% Terminator1 port information
terminator1PortHandle = get_param(terminator1Path, 'PortHandles');
terminator1Inport = terminator1PortHandle.Inport(1,1);

% Connect Longitudinal Driver DecelCmd Output
% to Terminator1 inport
add_line(modelName, longDriverPortDecelCmd, terminator1Inport)

%% T2
% Terminator2 block name, path, and position information
terminator2Name = 'Terminator2';
terminatorLibPath = 'simulink/Sinks/Terminator';
terminator2Path = [modelName, '/', terminator2Name];
terminator2Pos = [955 25 975 45];

% Add Terminator2 block
add_block(terminatorLibPath, terminator2Path, ...
    'position', terminator2Pos)

% Terminator2 port information
terminator2PortHandle = get_param(terminator2Path, 'PortHandles');
terminator2Inport = terminator2PortHandle.Inport(1,1);

% Connect Longitudinal Driver Info Output to Terminator2 inport
add_line(modelName, longDriverPortInfo, terminator2Inport)

```

- e Input a 5.84 reference velocity to the Longitudinal Driver block **VelRef** port using a Constant block.

See Code

```
%% Reference Velocity
ref_vel = '5.84';
constantLibPath = 'simulink/Commonly Used Blocks/Constant';
referenceVelocityName = 'Constant';
referenceVelocityPath = [modelName, '/', referenceVelocityName];
referenceVelocityPos = [655 20 685 50];
add_block(constantLibPath, referenceVelocityPath, ...
    'position', referenceVelocityPos)
set_param(referenceVelocityPath, 'Value', ref_vel)

% Reference Velocity port information
referenceVelocityPortHandle = get_param(referenceVelocityPath, 'PortHandles');
referenceVelocityOutputport = referenceVelocityPortHandle.Output(1,1);

add_line(modelName, referenceVelocityOutputport, longDriverPortVelRef)
```

- f** Input the incline angle signal from the Signal Builder block to the Longitudinal Driver block by connecting the **Incline** output to the **Grade** input.

See Code

```
%% Connect incline output port to Grade input port
add_line(modelName, signalBuildPortIncline, ...
    longDriverPortGrade, 'autorouting', 'on')
```

- g** Input the velocity feedback signal to the Longitudinal Driver block using a Unit Conversion block to convert from km/hr to m/s.

See Code

```
%% Unit Conversion
% UnitConvert0 block name, path, and position information
unitConvert0Name = 'Unit Conversion';
unitConvert0LibPath = 'simulink/Signal Attributes/Unit Conversion';
unitConvert0Path = [modelName, '/', unitConvert0Name];
unitConvert0Pos = [565 54 635 96];

% Add block
add_block(unitConvert0LibPath, unitConvert0Path, ...
    'position', unitConvert0Pos)

%% UnitConvert0 port information
unitConvert0PortHandle = get_param(unitConvert0Path, 'PortHandles');
unitConvert0Inport = unitConvert0PortHandle.Inport(1,1);
unitConvert0Outputport = unitConvert0PortHandle.Output(1,1);

% Connect Longitudinal Driver AccelCmd Outputport to Goto Throttle inport
add_line(modelName, unitConvert0Outputport, ...
    longDriverPortVelFdbk, 'autorouting', 'on')

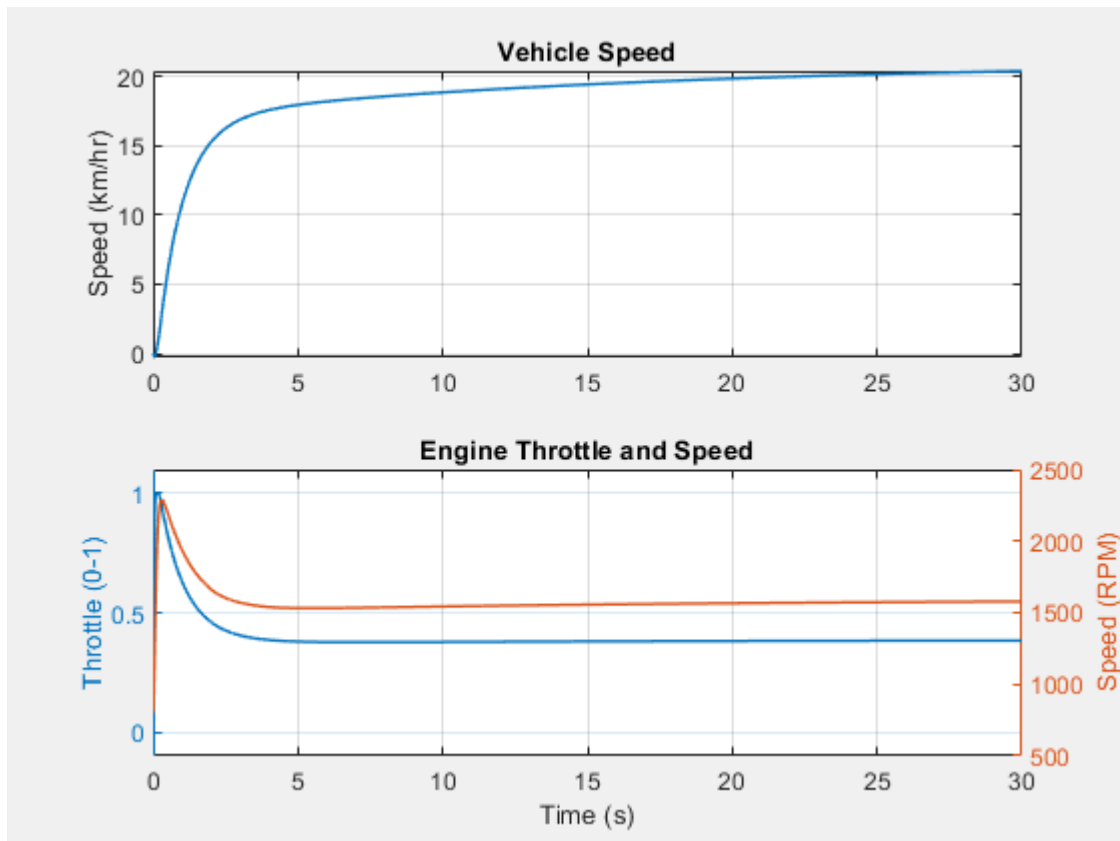
% Connect PS-S Converter Outputport to Longitudinal Driver AccelCmd Inport
```

```
add_line(modelName,pStoSConverter1Output,...
unitConvert0Inport,'autorouting','on')
```

- 2 Simulate the closed-loop model using the Simple Tire blocks and plot the results.

See Code

```
%% Simple
sdl_vehicle_4wd_settires(bdroot,'Simple');
sim(modelName)
% Plot the vehicle speed, the engine speed, and the Throttle input.
sdl_vehicle_4wd_plot1speed
```



The drive block increases the throttle input rapidly at the beginning of simulation due to the difference between the velocity feedback and reference signals.

Simulation Comparison

Compare the open and closed-loop results using the Simulation Data Inspector.

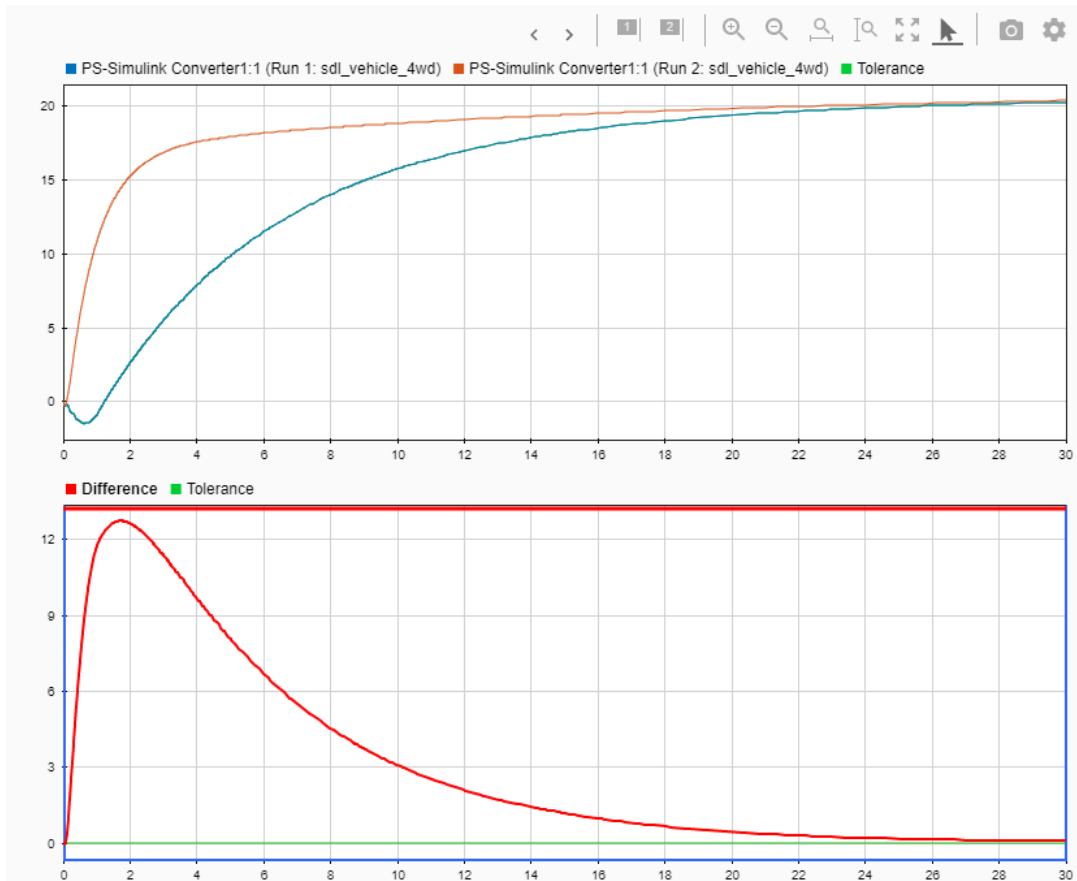
See Code

```
%% Get Simulation (TM) Data Inspector run IDs for
%   the last two runs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 1);
runSimple = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaselinePartition = Simulink.sdi.compareRuns(runBaseline,...
    runSimple);
```

To see the results in the Simulation Data Inspector, click the **Compare** icon and then, under **Filter Comparisons**, click **PS-Simulink Converter1:1**.



The first plot overlays the results from the open and closed-loop simulations. It shows how much faster the controlled vehicle goes to steady state.

The second plot shows the numerical difference in the results from the two simulations. It shows how much the two signals differ at the beginning of the simulation and how they eventually reach the same steady state.

You can also examine the results for other tire blocks.

See Code for Magic Formula Tire

```
%% Magic
sdl_vehicle_4wd_settires(bdroot, 'Mformula');
```

```
sim(modelName)
% Plot the vehicle speed, the engine speed, and the throttle input.
sdl_vehicle_4wd_plot1speed
```

See Code for Friction Parameterized Tire

```
%% Friction
sdl_vehicle_4wd_settires(bdroot, 'Friction');
sim(modelName)
% Plot the vehicle speed, the engine speed, and the throttle input.
sdl_vehicle_4wd_plot1speed
```

See Also

[Constant](#) | [Terminator](#) | [Tire \(Friction Parameterized\)](#) | [Tire \(Magic Formula\)](#) | [Tire \(Simple\)](#) | [Unit Conversion](#)

Related Examples

- “Vehicle with Four-Wheel Drive”

More About

- “Compare Simulation Data” (Simulink)

Drivetrain Disturbances

- “Model Drivetrain Noise” on page 14-2
- “Model and Detect Drivetrain Faults” on page 14-13

Model Drivetrain Noise

This example shows how to inject a fault into a drivetrain using a Torque Noise Source block. Injecting noise into your model allows you to predict how your actual physical system responds when it experiences environmental or internal disturbances. It also allows you to test the robustness and responsiveness of your control system.

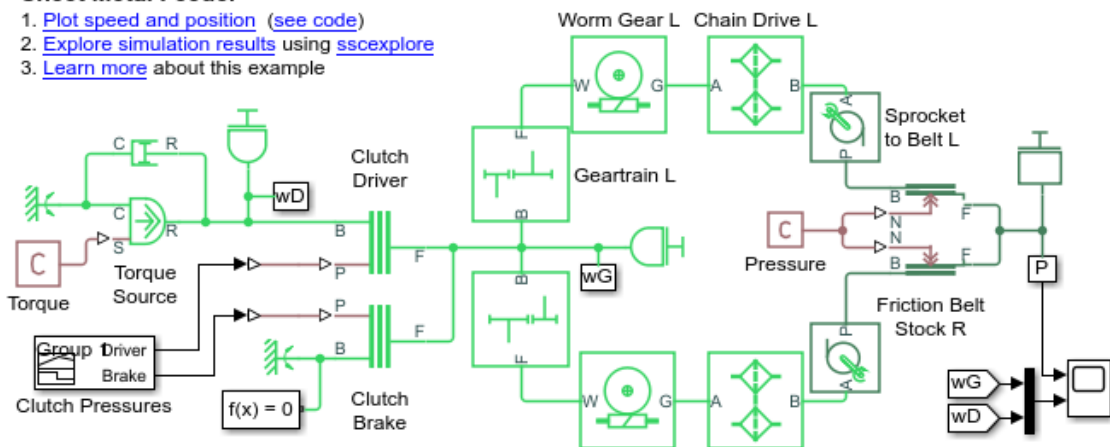
In the example, the noise is injected into one of the sprockets on a metal sheet feeder that is moving a piece of stock. Although you can perform most of the steps in this example using tools that the Simulink and Simscape Driveline user interfaces provide, programmatic commands are supplied. You can combine the programmatic commands to create a script for parameter sweeps.

- 1 Open the model. At the MATLAB command prompt, enter:

```
model = 'sdl_sheet_metal_feeder';
open_system(model)
```

Sheet Metal Feeder

1. [Plot speed and position](#) ([see code](#))
2. [Explore simulation results](#) using [sscexplore](#)
3. [Learn more](#) about this example



- 2 Simulate the model and plot the results.

Script for Generating and Plotting Simulation Results

```

%% Simulate
sim(model)

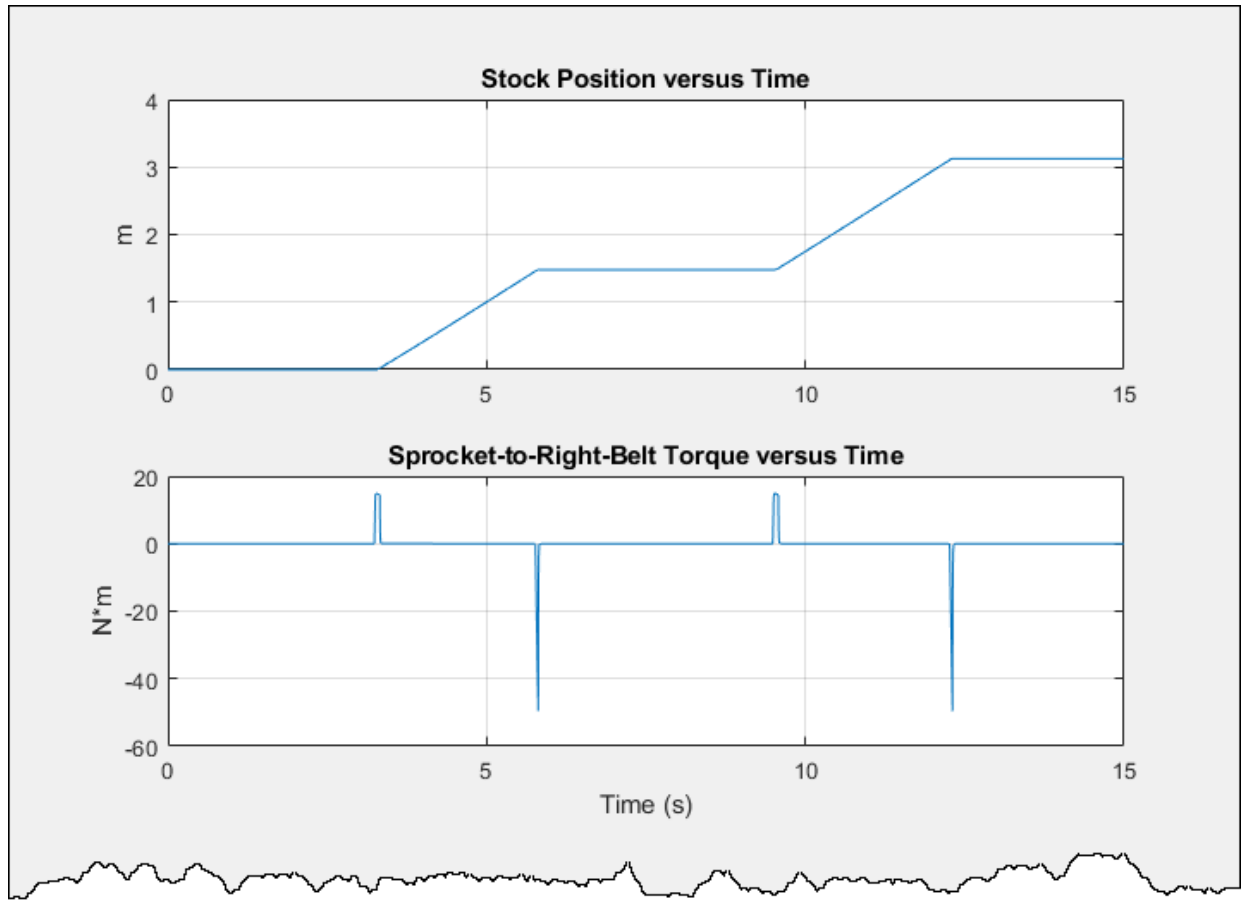
%% Define the data
simlog1 = simlog_sdl_sheet_metal_feeder;
time = simlog1.Sensor_Stock.Motion_Sensor.x.series.time;
sheetPosition = simlog1.Sensor_Stock.Motion_Sensor.x.series.values;
sprocketTorque = simlog1.Sprocket_to_Belt_R.t.series.values;

%% Create figure
figure1 = figure('NumberTitle','off',...
    'Name','Simscape Results: sdl_sheet_metal_feeder_noise',...
    'OuterPosition',[565 52 733 822]);

%% Create subplot 1: Sprocket to Belt L Tourque versus Time
% Create axes for
axes1 = axes('Parent',figure1,...
    'Position',[0.13 0.709 0.775 0.216]);
hold(axes1,'on')
box(axes1,'on')
grid(axes1,'on')
% Create title & axis labels
title('Stock Position versus Time')
% xlabel('Time (s)')
ylabel('m')
% Create plot
plot(time,sheetPosition,'Parent',axes1)

%% Create subplot 2: Sprocket to Belt R Tourque versus Time
% Create axes
axes2 = axes('Parent',figure1,...
    'Position',[0.13 0.409 0.775 0.2156]);
hold(axes2,'on')
box(axes2,'on')
grid(axes2,'on')
% Create title & axis labels
title('Sprocket-to-Right-Belt Torque versus Time')
xlabel('Time (s)')
ylabel('N*m')
% Create plot with linked axes
plot(time,sprocketTorque,'Parent',axes2)
linkaxes([axes1,axes2],'x')

```



- 3 Add, configure, connect, and arrange these blocks as shown:
- Step block — Specify 7.5 for the **Step time** parameter and 300 for the **Final value** parameter.
 - Simulink-PS Converter block
 - Mechanical Rotational Reference block
 - Torque Noise Source block — For the **Sample time**, specify $1e-1$, for **Repeatability**, select Specify seed, and for **Seed** specify 0.

Script for Adding, Configuring, Connecting, and Arranging Blocks

```

%% Expand the model window to make room for new blocks
set_param(model,'Location',[108 73 881 682])

%% Add and configure Step block
% Define block
stepPath = [model,'/Step'];
stepLib = 'simulink/Sources/Step';
stepPosition = [-195 205 -165 235];
stepTime = '7.5';
stepValue = '300';
% Add block
add_block(stepLib,stepPath,'Position',stepPosition)
% Configure Step block
set_param(stepPath,'Time',stepTime,...
    'After',stepValue)
% Check block configuration
open_system(stepPath)
pause(3);
close_system(stepPath)
% Get output port
stepPortHandle = get_param(stepPath,'PortHandles');
stepOutputport = stepPortHandle.Outputport;

%% Add and configure S-PS Converter block
% Define block
sPSConvPath = [model,'/Simulink-PS Converter'];
sPSConvLib = ...
    'nes1_utility/Simulink-PS Converter';
sPSConvPathPosition = [-115 205 -85 235];
% Add block
add_block(sPSConvLib,sPSConvPath,'Position',sPSConvPathPosition)
% Get output port
sPSConvPortHandle = get_param(sPSConvPath,'PortHandles');
sPSConvInport = sPSConvPortHandle.Inport;
sPSConvConPort = sPSConvPortHandle.RConn(1,1);

%% Add Mechanical Rotational Reference block
% Define block
mechRotRefPath = [model,'/Mechanical Rotational Reference'];
mechRotRefLib = ...
    'fl_lib/Mechanical/Rotational Elements/Mechanical Rotational Reference';
mechRotRefPosition = [-40 255 -20 275];
% Add block
add_block(mechRotRefLib,mechRotRefPath,'Position',mechRotRefPosition)
%% Get output port
mechRotRefPortHandle = get_param(mechRotRefPath,'PortHandles');
mechRotRefConPort = mechRotRefPortHandle.LConn(1,1);

%% Add and configure Torque Noise Source block
% Define block
torqueNoisePath = [model,'/Torque Noise Source'];
torqueNoiseLib = 'sd_lib/Sources/Torque Noise Source';
torqueNoisePosition = [25 210 65 250];
% Add block
add_block(torqueNoiseLib,torqueNoisePath,'Position',torqueNoisePosition)
%% Get output port
torqueNoisePortHandle = get_param(torqueNoisePath,'PortHandles');
torqueNoiseConPort1 = torqueNoisePortHandle.LConn(1,1);
torqueNoiseConPort2 = torqueNoisePortHandle.LConn(1,2);

```

```
torqueNoiseConPort3 = torqueNoisePortHandle.RConn(1,1);
%% Configure block
set_param(torqueNoisePath, 'sample_time','1e-1',...
    'repeatability','3')
%% Check block configuration
open_system(torqueNoisePath)
pause(3);
close_system(torqueNoisePath)

%% Get block points for connecting as a branched line
torqueNoisePortPoints = get_param(torqueNoisePath,'PortConnectivity');
[torqueNoiseLConnPoints1,torqueNoiseLConnPoints2,torqueNoiseRConnPoints]...
    = torqueNoisePortPoints.Position;
sprocketPath = [model,'/Sprocket to Belt R'];
sprocketPortHandle = get_param(sprocketPath,'PortHandles');
sprocketConPort1 = sprocketPortHandle.RConn(1,1);
sprocketPortPoints = get_param(sprocketPath,'PortConnectivity');
[sprocketLConnPoints,sprocketRConnPoints] = sprocketPortPoints.Position;

%% Connect blocks
add_line(model,stepOutport,sPSConvInport)
add_line(model,sPSConvConPort,torqueNoiseConPort1)
add_line(model,mechRotRefConPort,torqueNoiseConPort2)
add_line(model,[sprocketLConnPoints;torqueNoiseRConnPoints])

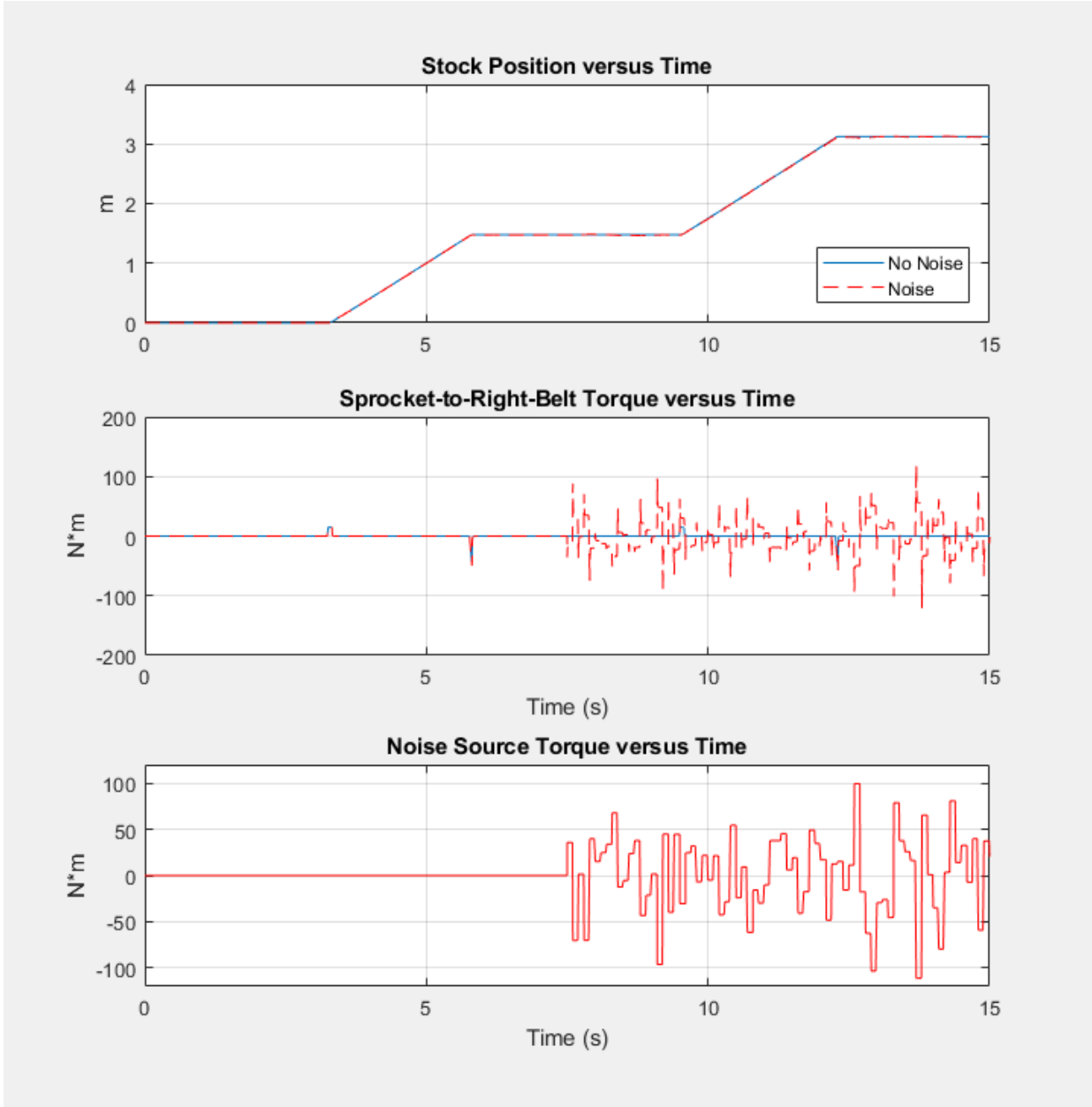
%% Hold to check connections
pause(5);
```



```
% Create subplot 3: Sprocket to Belt R Torque versus Time
% Create axes
axes3 = axes('Parent',figure1,...
    'Position',[0.13 0.11 0.775 0.200]);
hold(axes3,'on')
box(axes3,'on')
grid(axes3,'on')

% Create title and axis labels
title('Noise Source Torque versus Time')
xlabel('Time (s)')
ylabel('N*m')
linkaxes([axes1,axes2,axes3], 'x')

% Create plot
plot(time,noiseTorque,'Parent',axes3,'Color','r')
legend(axes1,'No Noise','Noise','Location','southeast');
ylim(axes3,[-120 120]);
```

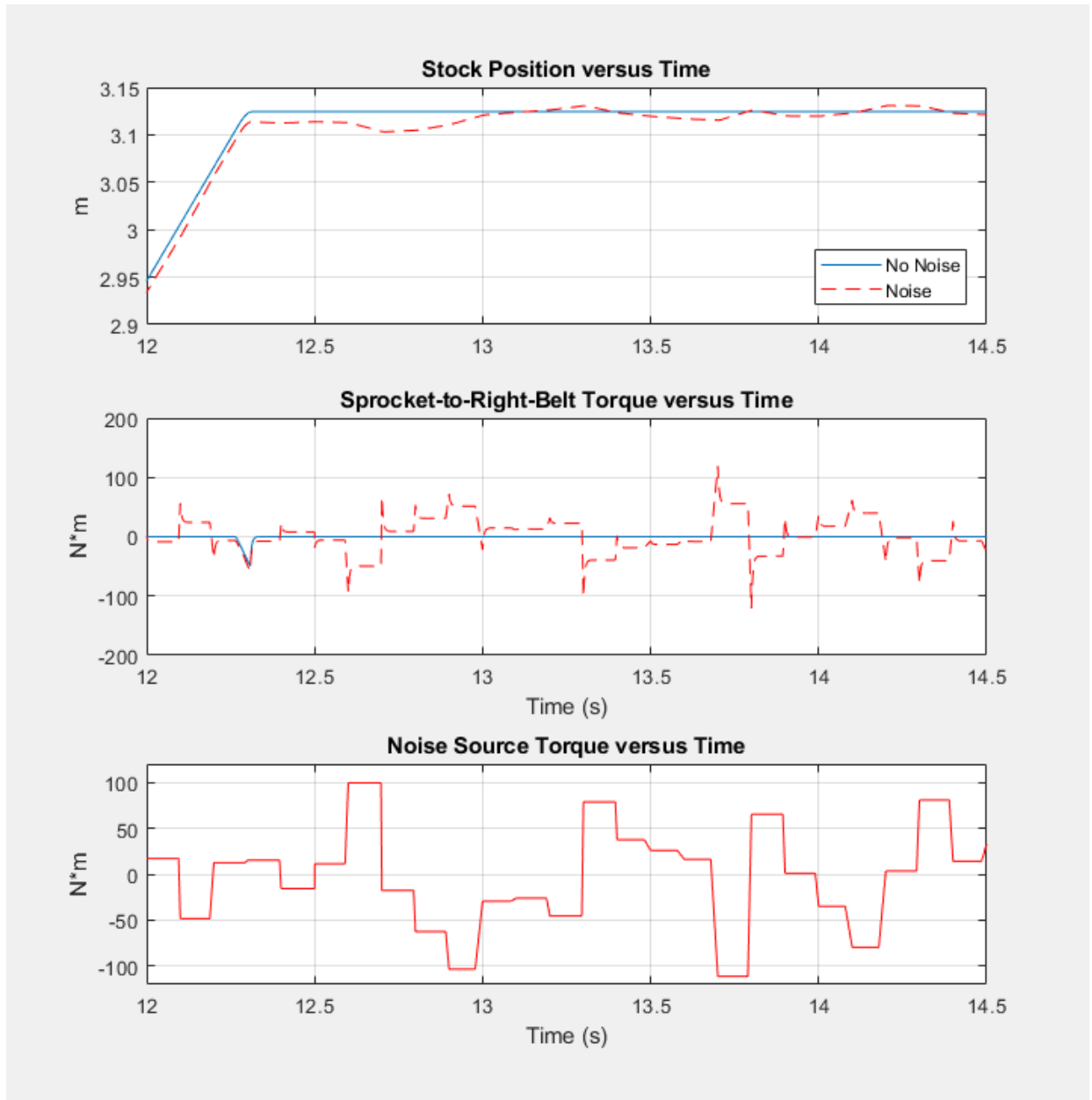



The noise injected at simulation time, $t = 7.5$ seconds introduces torque disturbances.

- 5 Zoom in to see the effects of the disturbance.

Script for Zooming In

```
%% Zoom to examine data at peak noise  
xlim(axes3,[12.0 14.50]);  
ylim(axes3,[-120 120]);
```



When the noise source torque exceeds $\pm 50 \text{ N}\cdot\text{m}$, it most significantly effects the torque applied by the sprocket and, therefore, the position of the stock.

See Also

[Mechanical Rotational Reference](#) | [Rotational Damper](#) | [Simulink-PS Converter](#) | [Sinusoidal Torque Source](#) | [Step](#) | [Torque Noise Source](#)

Simscape > Foundation Library > Mechanical > Rotational Elements library. The **Foundation Library** Rotational Damper block is not able to detect or respond to faults.

- 2 Simulate the model and plot the results.

Script for Generating and Plotting Simulation Results

```

%% Simulate Model
sim(model)

%% Get simulation results
simlog01 = simlog_sdl_flexible_shaft;
simlog_t = simlog01.Clutch.P.series.time;
simlog_pClutch = simlog01.Clutch.P.series.values('Pa');
simlog_wMotor = simlog01.Shaft_Motor.F.w.series.values('rad/s');
simlog_wLoad = simlog01.Shaft_Load.F.w.series.values('rad/s');

%% Plot results
X1 = simlog_t;
YMatrix1 = [simlog_wMotor simlog_wLoad];
Y1 = simlog_pClutch;

% Create figure
figure1 = figure('Name','sdl_flexible_shaft',...
    'OuterPosition',[107 336 733 822]);

% Create axes
axes1 = axes('Parent',figure1,...
    'Position',[0.13 0.709 0.775 0.216]);
hold(axes1,'on');

% Activate the left side of the axes
yyaxis(axes1,'left');

% Create multiple lines using matrix input to plot
plot1 = plot(X1,YMatrix1,'LineWidth',2);
set(plot1(1),'DisplayName','Motor Shaft','Color','k');
set(plot1(2),'DisplayName','Load Shaft','Color','b');

% Create ylabel
ylabel('Speed (rad/s)');

% Comment/uncomment the following line to remove/
% preserve the Y-limits of the axes
ylim(axes1,[-100 250]);

% Set the remaining axes properties
set(axes1,'YColor',[0 0.447 0.741]);

% Activate the right side of the axes
yyaxis(axes1,'right');

% Create plot
plot(X1,Y1,'DisplayName','Clutch Pressure','LineWidth',2, 'Color', 'r');

% Create ylabel
ylabel('Pressure (Pa)');

% Comment/uncomment the following line to remove/

```

```

% preserve the Y-limits of the axes
ylim(axes1,[0 2000000]);

% Set the remaining axes properties
set(axes1,'YColor',[0.85 0.325 0.098]);
% Create xlabel
xlabel('Time (s)');

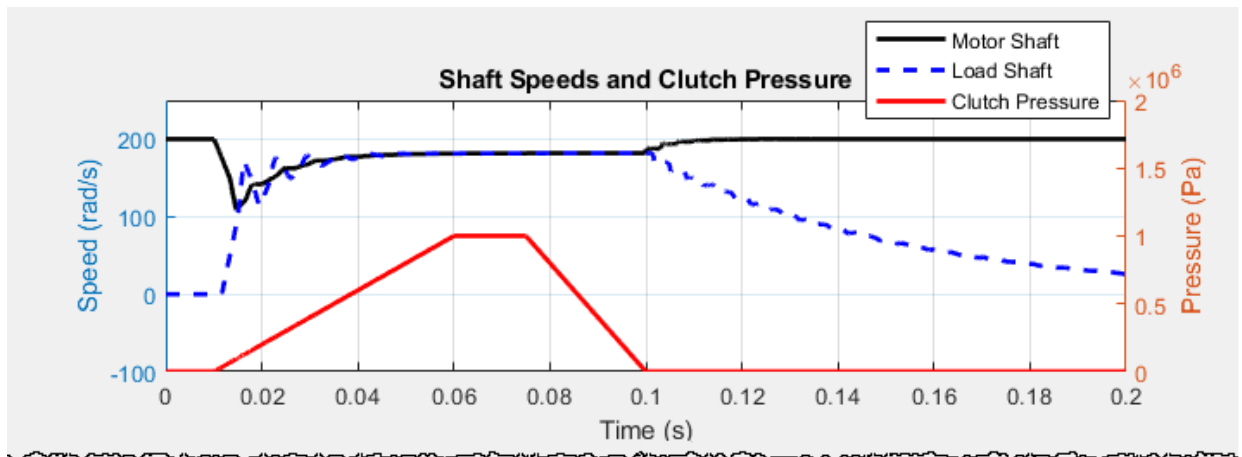
% Create title
title('Shaft Speeds and Clutch Pressure');

% Uncomment the following line to preserve the X-limits of the axes
% xlim(axes1,[0 0.2]);

% Set the remaining axes properties
set(axes1,'LineStyleOrderIndex',2);
box(axes1,'on');
grid(axes1,'on');

% Create legend
legend1 = legend(axes1,'show');
% set(legend1,'Location','best');
set(legend1,...
    'Position',[0.6944 0.9125 0.1982 0.07270]);

```



At the start of the simulation, the clutch is unlocked and the driven shaft is free. The initial velocity of the motor shaft is the specified 200 rad/s and the system starts at steady state. The oscillations triggered by the engaging and disengaging of the clutch are due to the flexibility in the shafts.

- 3 Replace the Simscape damper with the Simscape Driveline Rotational Damper, which is in the **Simscape > Driveline > Couplings & Drives > Springs & Dampers** library. Label the new block **Faultable Damper**.

Script for Replacing the Rotational Damper Block

```

%% Replace Rotational Damper from Foundation Library with
% Faultable Rotational Damper from the Simscape Driveline Library

% Define Unfaultable Damper Block
foundationDamper = [model, '/Damping Bearing'];

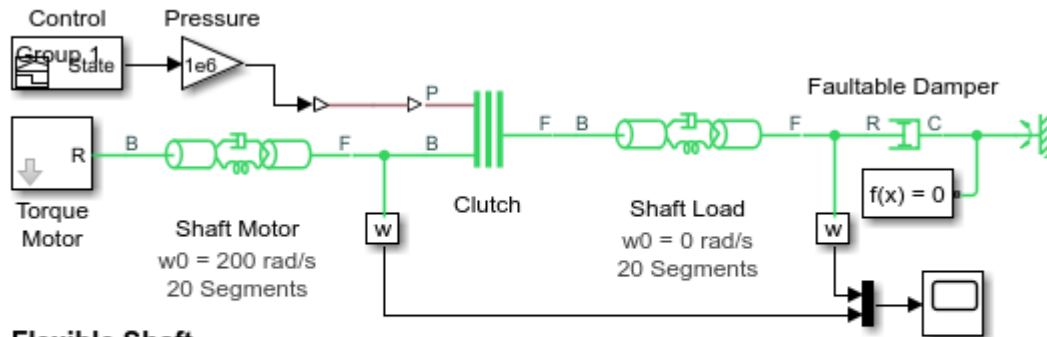
% Get Unfaultable Damper Block Damping Coefficient
dampingCoefficient = get_param(foundationDamper, 'D');

% Get Unfaultable Damper Block Position
damperPosition = get_param(foundationDamper, 'Position');

% Delete Unfaultable Damper Block
delete_block(foundationDamper)

% Add Faultable Damper Block
faultableDamperLib = ...
    'sd_lib/Couplings & Drives/Springs & Dampers/Rotational Damper';
faultableDamperPath = [model, '/Faultable Damper'];
add_block(faultableDamperLib, faultableDamperPath, ...
    'Position', damperPosition, ...
    'NamePlacement', 'alternate', ...
    'D', dampingCoefficient)

```



Flexible Shaft

1. [Plot shaft speed](#) (see code)
2. [Plot twist](#) in shaft compliance elements (see code)
3. [Explore simulation results](#) using [sscexplore](#)
4. [Learn more](#) about this example

4. Enable a time-based fault and specify a response that includes a change in the damping coefficient and the generation of a MATLAB warning. Use these values for the damper **Fault** parameters:

- **Enable faults** — Enabled

- **Faulted damping coefficient** — 10
- **Enable temporal fault trigger** — Enabled
- **Simulation time for fault event** — 0.07
- **Reporting when fault occurs** — Warning

Script for Configuring the Rotational Damper Block Using a Timed Fault

```

%% Define Faultable Damper Block Parameters
underDamp = '10';
overDamp = '150';

faultTime = '.06';

aMaxDef = '100';
aMaxHigh = '400';
aMaxLow = '50';

shockNMaxDef = '1';
shockNMaxHigh = '5';
shockNMaxLow = '2';

disabled = '0';
enabled = '1';

none = '1';
warning = '2';
error = '3';

%% Parameterize a Timed Fault
set_param(faultableDamperPath,...
    'enable_faults', enabled,...           % Enable faults
    'b_fault', underDamp,...              % Faulted damping coefficient
    'temporal_fault', enabled,...         % Enable temporal fault trigger
    'fault_time', faultTime,...           % Simulation time for fault event
    'shock_fault', disabled,...           % Disable behavioral fault trigger
    'acceleration_limit', aMaxDef,...     % Maximum permissible acceleration
    'shock_limit', shockNMaxDef, ...     % Maximum number of shocks
    'report_fault', warning)             % Reporting when fault occurs

```

- 5 Simulate the model and plot the results.

Script for Generating and Plotting Simulation Results

```

%% Simulate
sim(model)

%% Get Simulation Results
simlog02 = simlog_sdl_flexible_shaft;
simlog_t02 = simlog02.Clutch.P.series.time;
simlog_pClutch02 = simlog02.Clutch.P.series.values('Pa');
simlog_wMotor02 = simlog02.Shaft_Motor.F.w.series.values('rad/s');
simlog_wLoad02 = simlog02.Shaft_Load.F.w.series.values('rad/s');

%% Plot Results
X2 = simlog_t02;
YMatrix2 = [simlog_wMotor02 simlog_wLoad02];
Y2 = simlog_pClutch02;

```

```

% Create axes
axes2 = axes('Parent',figure1,...
    'Position',[0.125 0.4 0.775 0.216]);
hold(axes2,'on');

% Activate the left side of the axes
yyaxis(axes2,'left');

% Create multiple lines using matrix input to plot
plot1 = plot(X2,YMatrix2,'LineWidth',2);
set(plot1(1),'DisplayName','Motor Shaft','Color','k');
set(plot1(2),'DisplayName','Load Shaft','Color','b');

% Create ylabel
ylabel('Speed (rad/s)');

% Comment/uncomment the following line to remove/
% preserve the Y-limits of the axes
ylim(axes2,[-100 250]);

% Set the remaining axes properties
set(axes2,'YColor',[0 0.447 0.741]);

% Activate the right side of the axes
yyaxis(axes2,'right');
% Create plot
plot(X2,Y2,'DisplayName','Clutch Pressure','LineWidth',2,'Color','r');

% Create ylabel
ylabel('Pressure (Pa)');

% Comment/uncomment the following line to remove/
% preserve the Y-limits of the axes
ylim(axes2,[0 2000000]);

% Set the remaining axes properties
set(axes2,'YColor',[0.85 0.325 0.098]);

% Create title
title('Time-Faulted Speeds and Clutch Pressure');

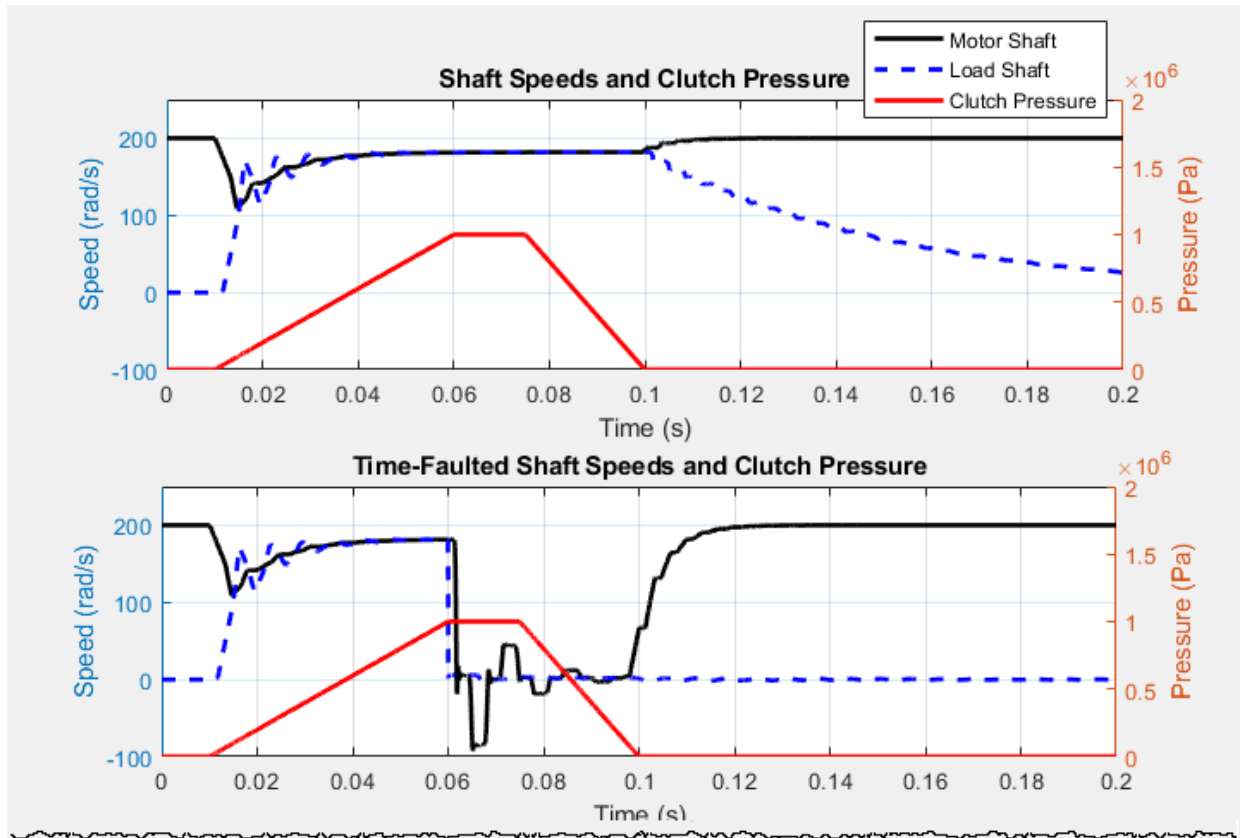
% Create xlabel
xlabel('Time (s)');

% Uncomment the following line to
% preserve the X-limits of the axes
% xlim(axes1,[0 0.2]);

% Set the remaining axes properties
box(axes2,'on');
grid(axes2,'on');
set(axes2,'LineStyleOrderIndex',2);

```

Warning: At time 0.060000, one or more assertions are triggered.
A fault event has occurred The assertion comes from:
Block path: sdl_flexible_shaft/Faultable Damper
Assert location: (location information is protected)



At simulation time $t = 0.06$ s, the time specified for the fault, a warning is reported. The damping coefficient drops and slows the speed of both shafts.

- 6 Enable a shock-based fault and specify a response that includes a change in the damping coefficient and the generation of a MATLAB warning. Then, simulate the model and plot the new results. Use these values for the damper **Fault** parameters:

- **Enable faults** — Yes
- **Faulted damping coefficient** — 150
- **Enable temporal fault trigger** — Disabled
- **Enable behavioral fault trigger** — Enabled
- **Maximum permissible acceleration** — 50

- **Maximum number of shocks** — 2
- **Reporting when fault occurs** — Warning

Script for Configuring the Rotational Damper Block Using a Timed Fault

```
%% Paramaterize a Shock Fault
set_param(faultableDamperPath,...
    'enable_faults', enabled,... % Enable faults
    'b_fault', overDamp,... % Faulted damping coefficient
    'temporal_fault', disabled,... % Disable temporal fault trigger
    'fault_time', faultTime,... % Simulation time for fault event
    'shock_fault', enabled,... % Enable behavioral fault trigger
    'acceleration_limit', aMaxHigh,... % Maximum permissible acceleration
    'shock_limit', shockNMaxLow ,... % Maximum number of shocks
    'report_fault', warning) % Reporting when fault occurs
```

- 7 Simulate the model and plot the results.

Script for Generating and Plotting Simulation Results

```
%% Simulate
sim(model)

%% Get Simulation Results
simlog03 = simlog_sdl_flexible_shaft;
simlog_t03 = simlog03.Clutch.P.series.time;
simlog_pClutch03 = simlog03.Clutch.P.series.values('Pa');
simlog_wMotor03 = simlog03.Shaft_Motor.F.w.series.values('rad/s');
simlog_wLoad03 = simlog03.Shaft_Load.F.w.series.values('rad/s');

%% Plot Results
X3 = simlog_t03;
YMatrix3 = [simlog_wMotor03 simlog_wLoad03];
Y3 = simlog_pClutch03;

% Create axes
axes3 = axes('Parent',figure1,...
    'Position',[0.125 0.08 0.775 0.216]);
hold(axes3,'on');

% Activate the left side of the axes
yyaxis(axes3,'left');

% Create multiple lines using matrix input to plot
plot1 = plot(X3,YMatrix3,'LineWidth',2);
set(plot1(1),'DisplayName','Motor Shaft', 'Color', 'k');
set(plot1(2),'DisplayName','Load Shaft', 'Color', 'b');

% Create ylabel
ylabel('Speed (rad/s)');

% Comment/uncomment the following line to
% remove/preserve the Y-limits of the axes
ylim(axes3,[-100 250]);

% Set the remaining axes properties
set(axes3,'YColor',[0 0.447 0.741]);

% Activate the right side of the axes
yyaxis(axes3,'right');
```

```
% Create plot
plot(X3,Y3,'DisplayName','Clutch Pressure','LineWidth',2, 'Color', 'r');

% Create ylabel
ylabel('Pressure (Pa)');

% Comment/uncomment the following line to
% remove/preserve the Y-limits of the axes
ylim(axes3,[0 2000000]);

% Set the remaining axes properties
set(axes3,'YColor',[0.85 0.325 0.098]);

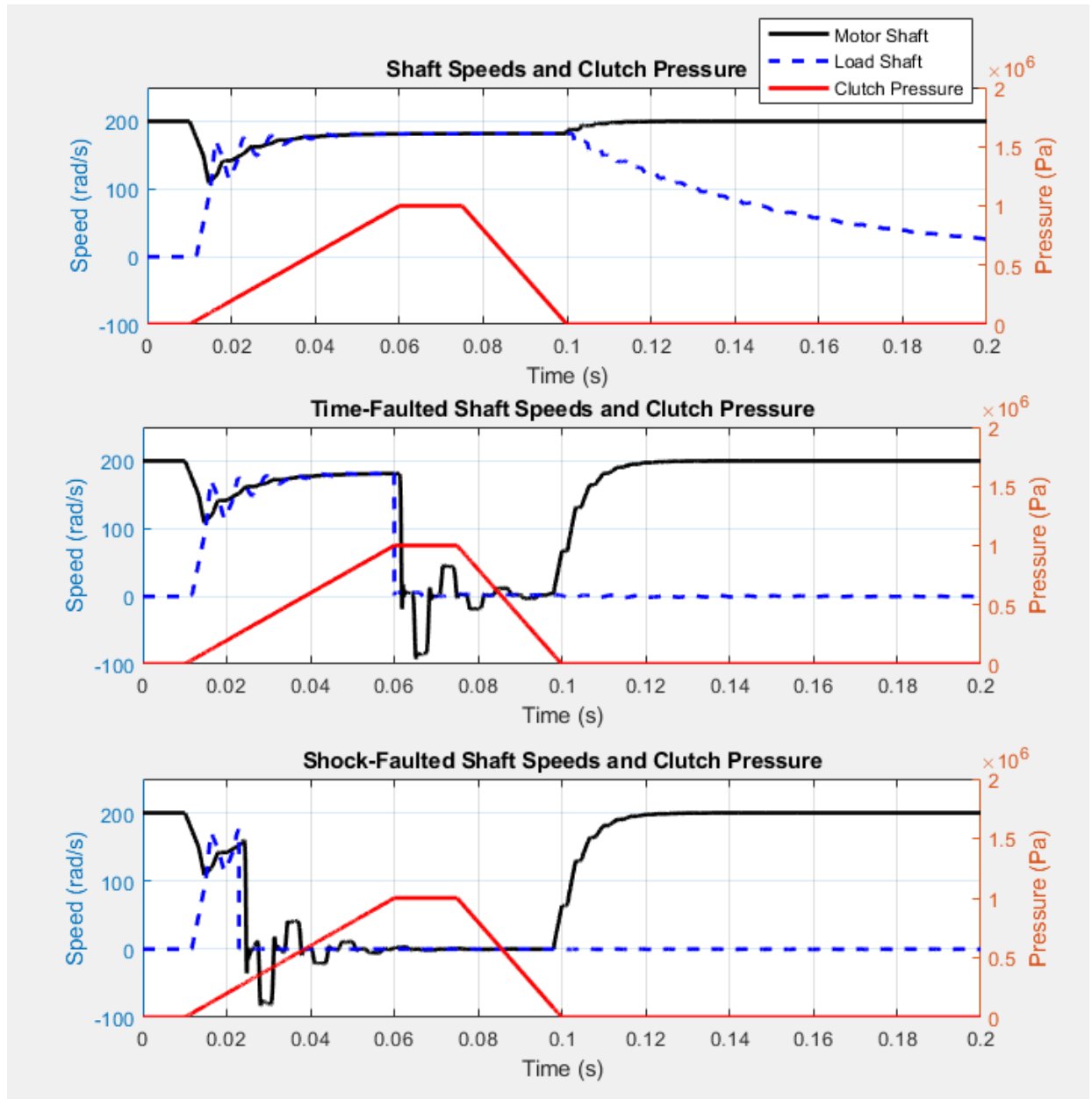
% Create title
title('Shock-Faulted Shaft Speeds and Clutch Pressure');

% Create xlabel
xlabel('Time (s)');

% Uncomment the following line to
% preserve the X-limits of the axes
% xlim(axes1,[0 0.2]);

% Set the remaining axes properties
box(axes3,'on');
grid(axes3,'on');
set(axes3,'LineStyleOrderIndex',2);
```

```
Warning: At time 0.026048, one or more assertions are triggered.
A fault event has occurred The assertion comes from:
Block path: sdl_flexible_shaft/Faultable Damper
Assert location: (location information is protected)
```



At simulation time $t = 0.026$ s, the maximum number of shocks for the specified acceleration is reached. A warning is reported and the damping coefficient increases and slows the speed of both shafts.

Rotational Damper | Translational Damper

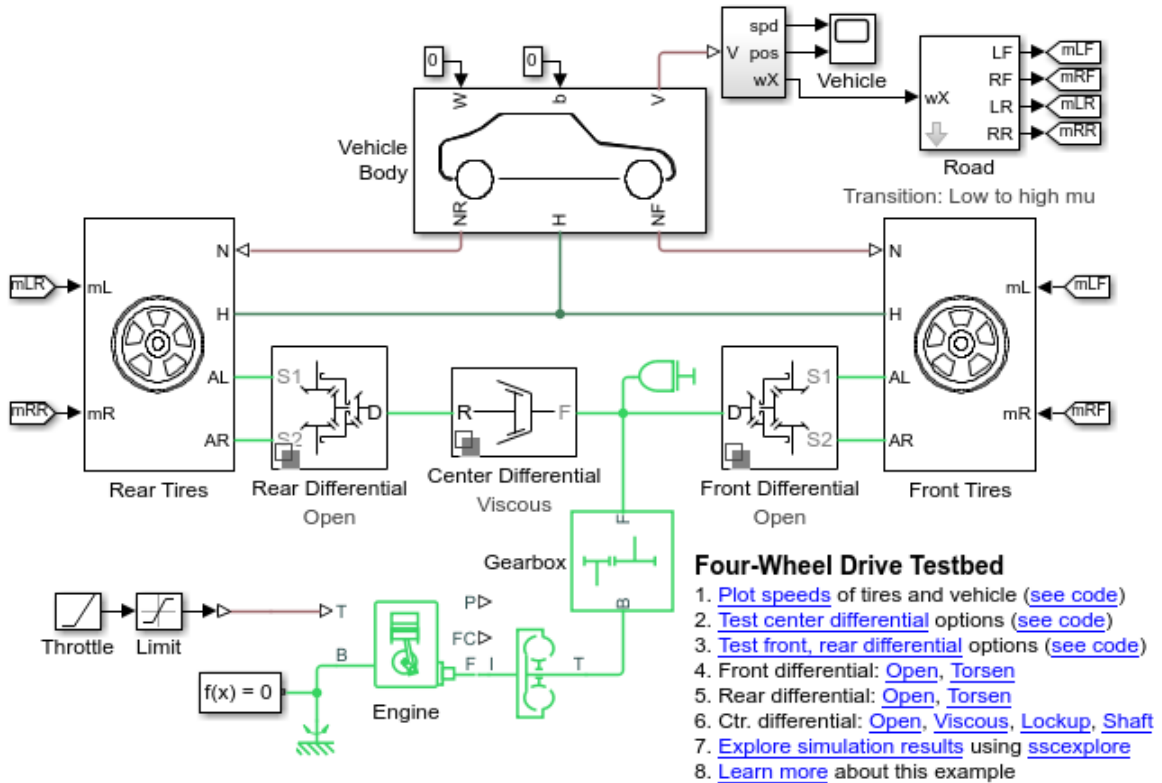
Modeling Driveline Environments

Model a Road Profile with Varying Elevation and Friction

This example shows how to vary road conditions throughout a simulation of a 4-wheel drive vehicle test-bed. The model is a version of the `sdl_vehicle_4wd_testbed` that is updated to include Road Profile blocks for both the front and rear tires. As the vehicle travels, the axle parameters and the position of the center of gravity (CG) determine the position of the front and rear axles. The Road Profile blocks use the axle positions to determine vehicle angle and tire friction coefficients based on parameters that you specify.

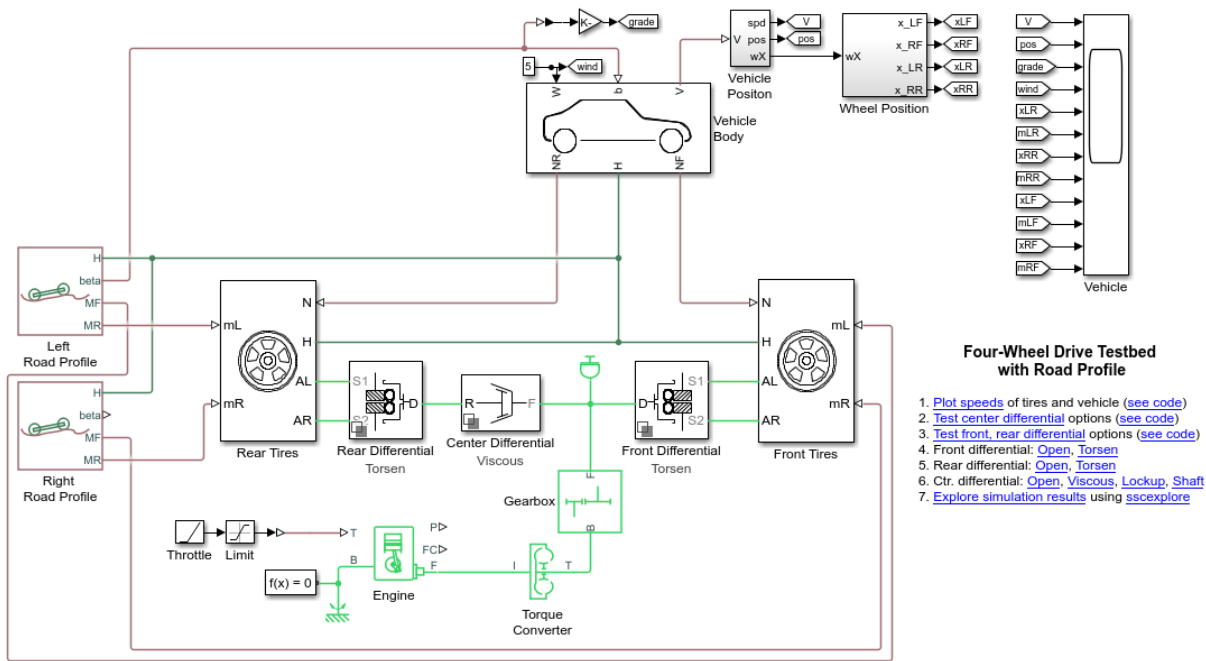
Updates to the Original Model

The original model determines the magic formula coefficients based on the position of the vehicle relative to its position at the simulation start. The figures show the original and the updated models.

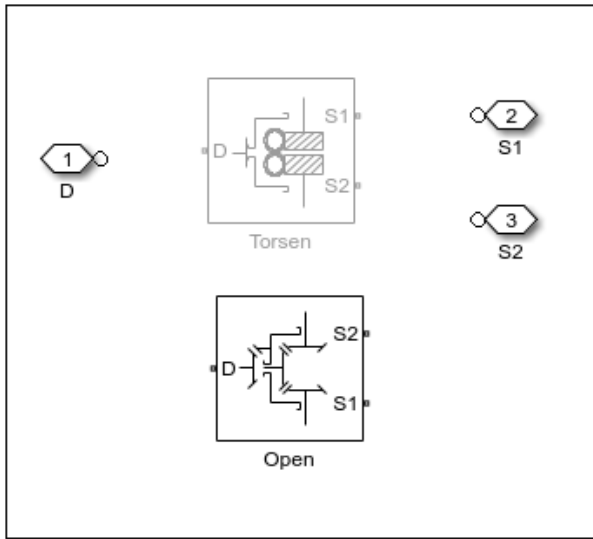


The new model includes Road Profile blocks for both the right- and left-side tires. To open the model, at the MATLAB command prompt, enter

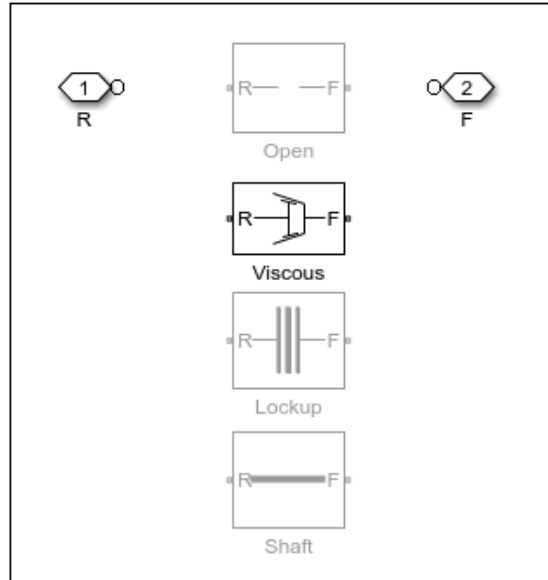
```
open_system('sdl_vehicle_road_4wd_testbed')
```



In both models, the front, rear, and center differentials are represented by variant subsystems.

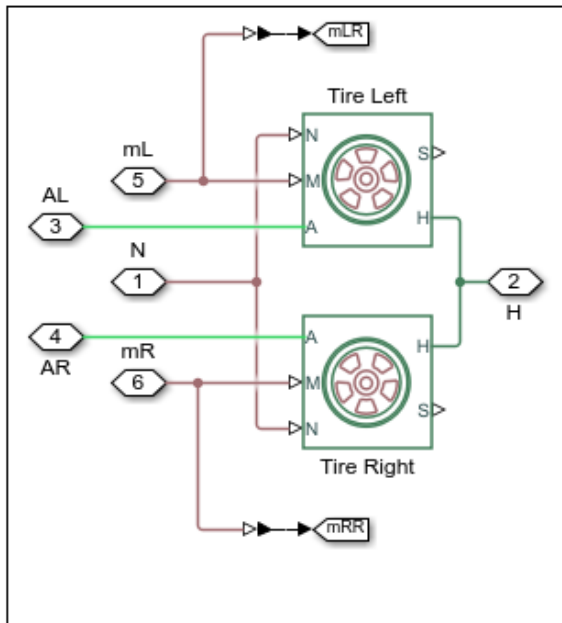


Front and Rear Differentials

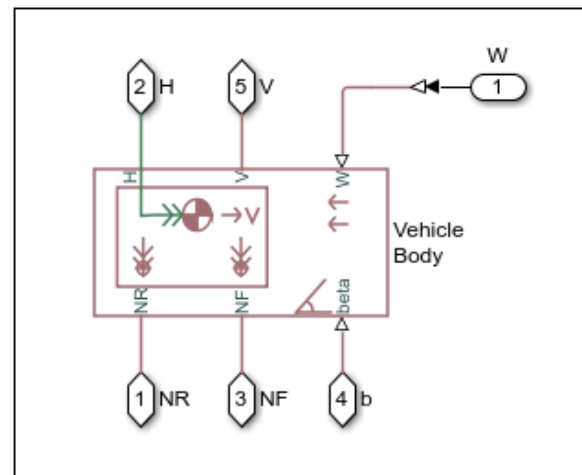


Center Differential

The front and rear tire subsystems contain Tire (Magic Formula) blocks, while the **Vehicle Body** subsystem is a mask for a Vehicle Body block.



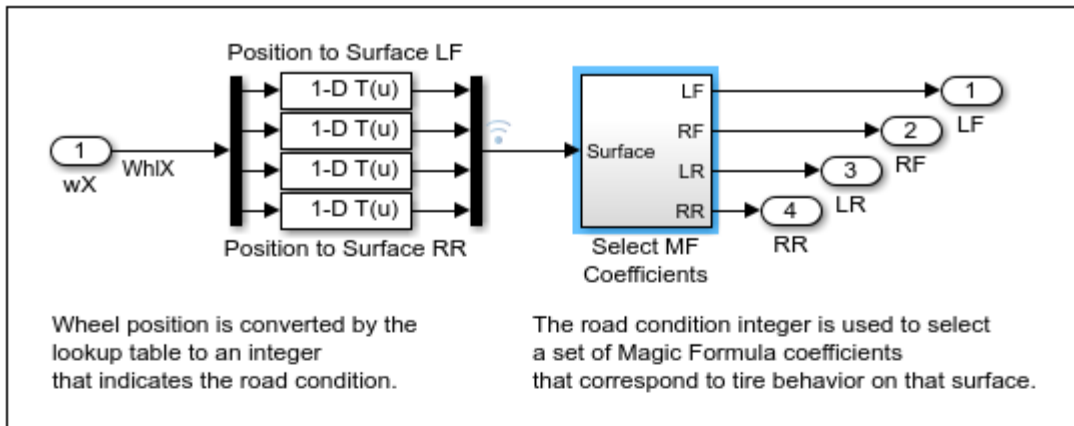
Front and Rear Tires



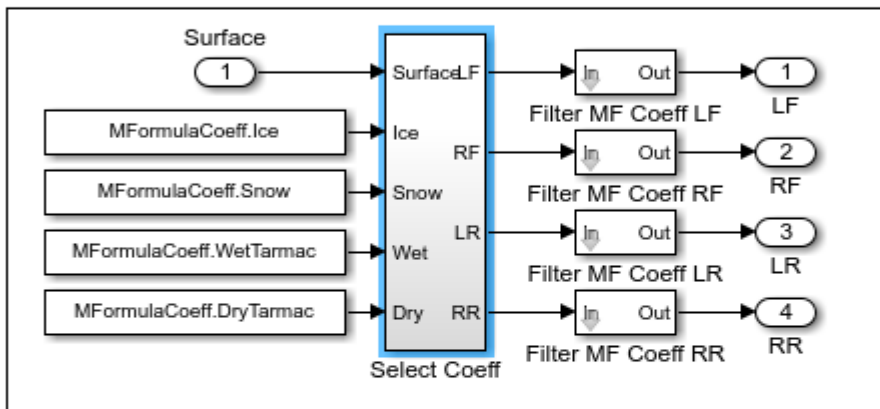
Vehicle Body

Updates that allow the model to determine road conditions using the Road Profile blocks are:

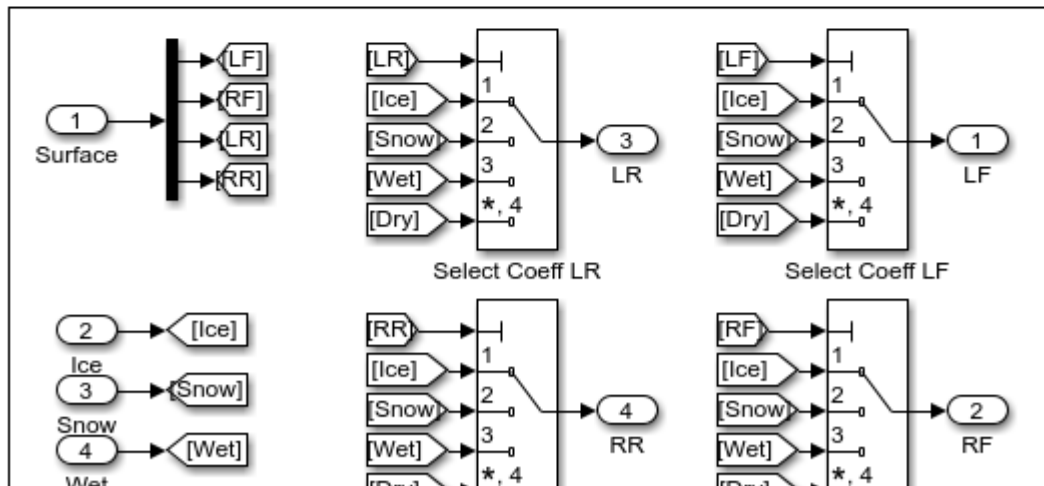
- 1 Replacement of the **Road** subsystem with the **Wheel Position** subsystem. The road subsystem contains three levels of subsystems that the model uses to determine the Magic Formula coefficients for the tires during simulation.



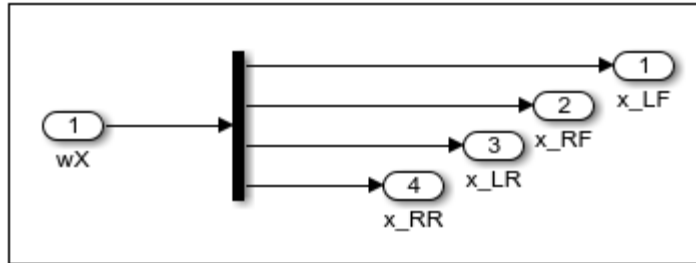
Road Subsystem



Select MF Coefficients Subsystem



The addition of the Road Profile blocks allows for the replacement of the **Wheel Position** system with the much simpler **Wheel Position** subsystem. The new subsystem demuxes the wheel position signals.



Wheel Position

- 2 Parameterization for the added Road Profile blocks for the right and left tires:

Main

- **Horizontal distance from CG to front axle** — x_f
- **Horizontal distance from CG to rear axle** — x_r
- **Horizontal distance for vertical profile** — x_{height_vector}
- **Vertical profile** — $height_vector$

Friction

- **Friction output** — Physical signal Magic Formula coefficients
- **Horizontal distance for friction profile** — $x_{friction_vector}$
- **Magic Formula coefficients for front axle** — MF_M_matrix
- **Magic Formula coefficients for rear axle** — MR_M_matrix

Position Variable

- **Override** — select
- **Beginning Value** — x_0

- 3 Vehicle Body block **Main** parameter updates:

- **Horizontal distance from CG to front axle** — x_f

- **Horizontal distance from CG to rear axle** — x_r

4 Variable definitions for the model:

```
x_f=1.4;
x_r=1.6;
x_height_vector=[-10, 0, 10];
height_vector=[0, 0, 0.25];
x_friction_vector = [ -10, 5, 10, 15 ];
MF_M_matrix = [10 1.9 1 0.97;...
               4 2 0.1 1;...
               12 2.3 0.82 1;...
               10 1.9 1 0.97];
MR_M_matrix = [10 1.9 1 0.97;...
               12 2.3 0.82 1;...
               12 2.3 0.82 1;...
               10 1.9 1 0.97];

x_0 = 0;
```

5 Additional environmental updates:

- The left-tire Road Profile block introduces a variable road grade. The Gain block converts the grade variable, *beta* from radians to degrees.
- A headwind is included by using a nonzero value for the Constant block.

6 Signal block updates:

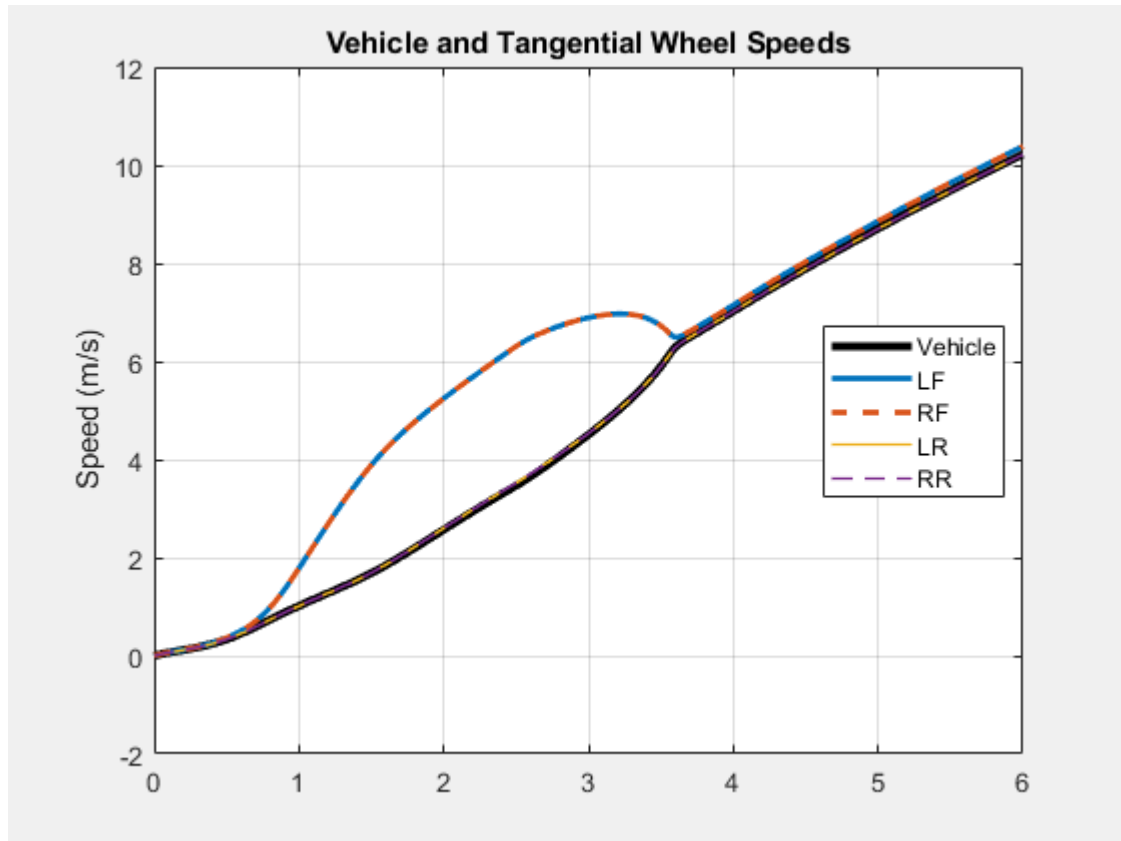
- Outports and Inports blocks are replaced with Connection Port blocks.
- Goto and From blocks are used to relay signals to the Scope.

7 Data visualization and logging updates:

- The Scope block is updated to show tire positions, Magic Formula Coefficients, headwind, and road elevation.
- The simlog name is updated to match the name of the updated model.
- The differential test and plot generation code is updated to use the new simlog name.

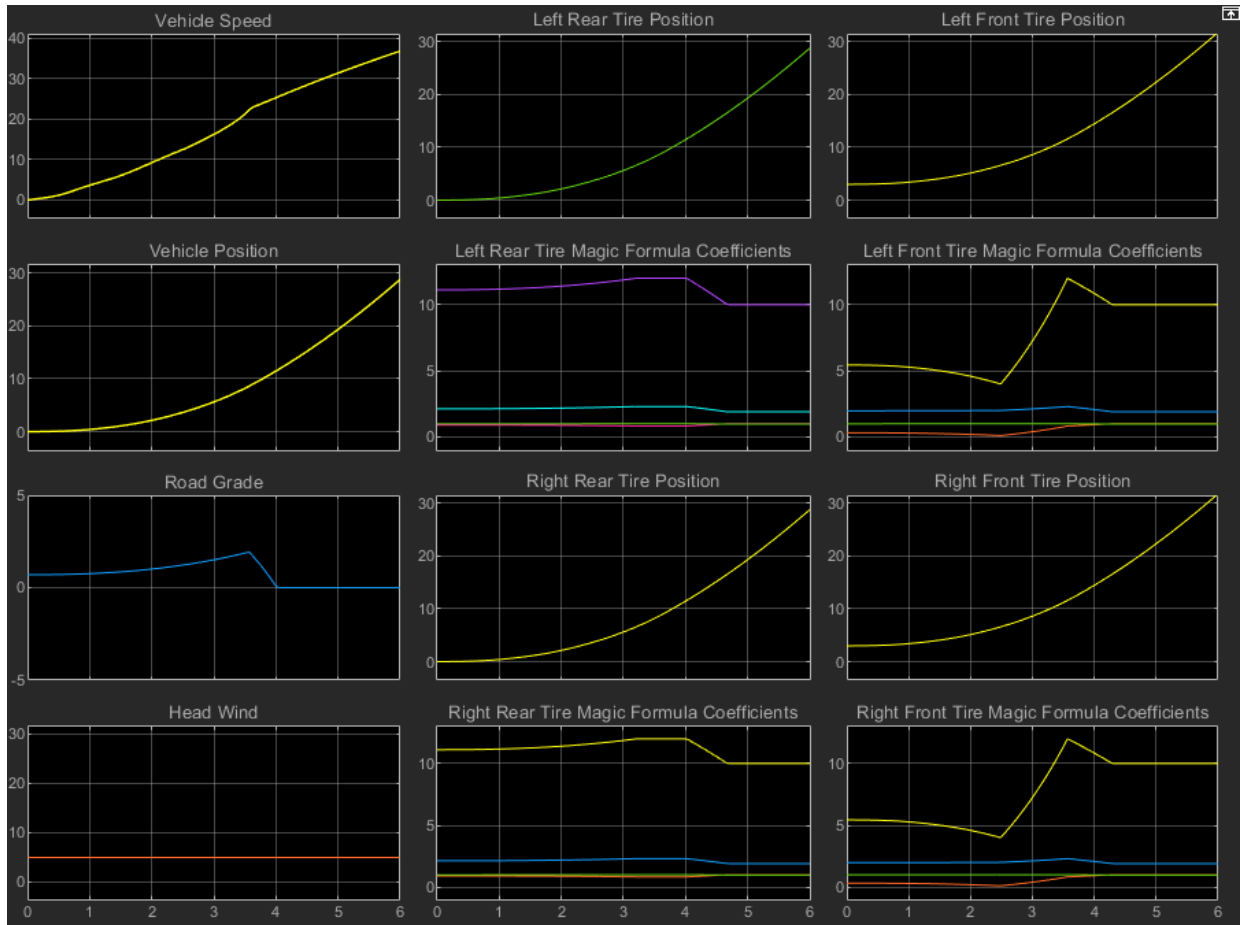
Run the Simulation

- 1 To run the simulation and generate a plot of the results, click **Plot speeds**.



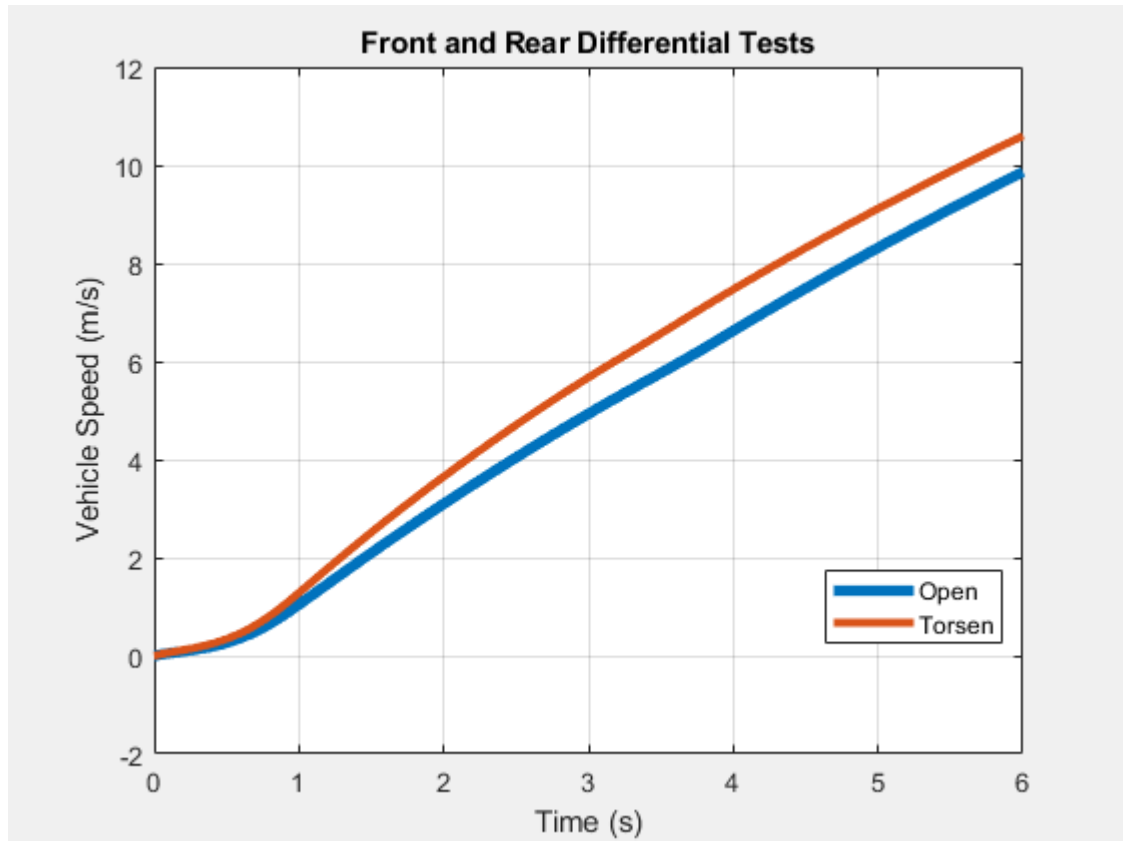
The front tires experience slip in the middle of the simulation due to the slippery conditions related to the $[4 \ 2 \ 0.1 \ 1]$ Magic Coefficients that the simulation uses when the positions of the front tires are 5 to 10 meters from the original positions.

- 2 To see how the road incline, headwind, tire position, and vehicle speed and position relate to each other and to the Magic Coefficients, open the Scope block.



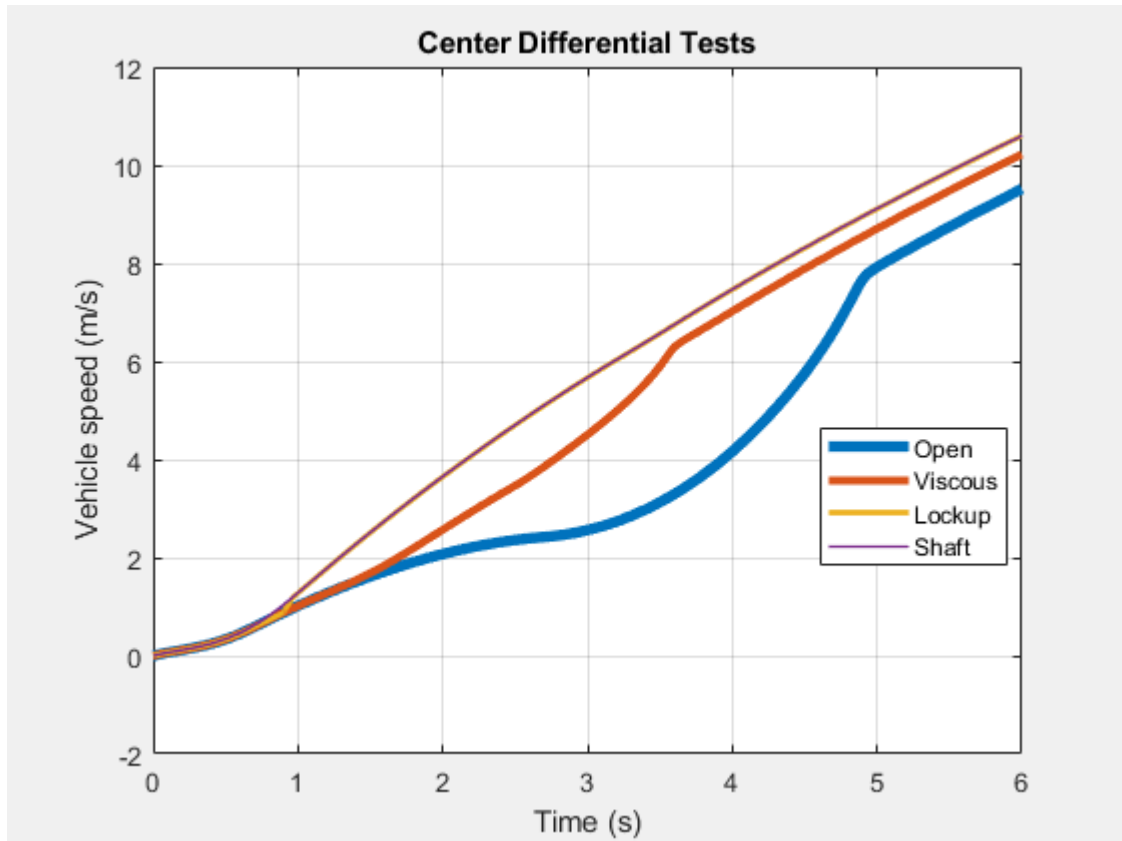
The vehicle velocity increases slightly as the road grade decreases.

- 3 To test the front and rear differentials in both the Open and Torsen variants, click **Test front, rear differential.**



The Torsen differential configuration results in higher velocity throughout the simulation.

- 4 To test all variants of the center differential, click **Test center differential**.



The open and viscous differential configurations result in lower, more variable velocity when the grade changes during the simulation.

See Also

[Road Profile](#) | [Tire \(Friction Parameterized\)](#) | [Tire \(Magic Formula\)](#) | [Tire-Road Interaction \(Magic Formula\)](#) | [Vehicle Body](#)

Analyzing Driveline Models and Simulations

These sections explore some of the more complex issues in driveline modeling, and some powerful techniques that can extend your driveline simulations.

- “Driveline Simulation Performance” on page 16-2
- “Resolve Partitioning Solver Simulation Issues for Simscape Driveline Models” on page 16-7
- “Driveline Degrees of Freedom” on page 16-38
- “Driveline States — Effect of Clutches” on page 16-52
- “How Simscape Driveline Simulates a Drivetrain System” on page 16-55
- “Model Thermal Losses in Driveline Components” on page 16-56
- “Simscape Driveline Limitations” on page 16-64

For information about the nature of Simscape models, states, and simulation, see the documentation for “Simulink” and “Simscape”.

Driveline Simulation Performance

In this section...
“About Simulation Performance” on page 16-2
“Adjust Model Fidelity” on page 16-2
“Improve Simulation Performance by Using the Partitioning Solver” on page 16-3
“Optimize Simulation of Stiff Drivelines” on page 16-3
“Optimize Simulation of Clutches” on page 16-4

About Simulation Performance

Driveline simulation involves the tradeoff between accuracy and speed inherent in all numerical simulation. Accuracy bundles two distinct issues, the accuracy or *fidelity* of the model, versus the accuracy of the simulation methods. This section describes the inherent complexity of driveline models, as distinct from general simulation issues.

About solvers and simulation methods, see “Setting Up Solvers for Physical Models” (Simscape) and “Making Optimal Solver Choices for Physical Simulation” (Simscape).

Adjust Model Fidelity

Improving the fidelity of driveline models involves making blocks that are more accurate representations of the actual physical components. For example, you can make the internal dynamics of the components represented by blocks more or less accurate and realistic by:

- Turning physical effects on and off, such as nonideal gear meshing losses (gear efficiency)
- Including or omitting compliance (including damped spring reactions), hard stops, and time lags
- Including or omitting Coulomb friction from clutches and clutch-like elements
- Steepening or softening sharp gradients in physical thresholds, such as velocity thresholds in clutches and nonideal gears

Modeling these physical effects requires additional dynamics and algebraic constraints, generates computationally more intensive simulations, and can reduce simulation speed, often considerably.

Model Fidelity in Ordinary Desktop Simulation

- Very small velocity threshold values and short time lags can degrade numerical convergence or simulation performance. Consider whether you can make these values larger in your simulation.
- If your model includes gears with efficiency loss, select adaptive zero-crossing in the **Model Configuration Parameters** menu.

Model Fidelity in Fixed-Step, Real-Time, and Hardware-in-the-Loop Simulation

Apart from clutches, MathWorks® does not recommend including fidelity enhancements in fixed-step/fixed cost, real-time, or hardware-in-the-loop (HIL) simulation.

To model compliance or efficiencies, consider reducing the number of such elements by:

- Deleting unnecessary lossy elements
- Combining lossy elements into as few elements as possible

If you simulate with a fixed-step solver, avoid:

- Very small velocity thresholds.
- Time lags that are short compared to the fixed time step.

Improve Simulation Performance by Using the Partitioning Solver

The Partitioning solver is a Simscape fixed-step local solver that improves performance for certain models. For more information about the Partitioning solver, including limitations for the types of models that it can solve, see “Increase Simulation Speed Using the Partitioning Solver” (Simscape). For an example that shows how to simulate a Simscape Driveline model using the Partitioning solver, see “Resolve Partitioning Solver Simulation Issues for Simscape Driveline Models” on page 16-7.

Optimize Simulation of Stiff Drivelines

When modeling a driveline, consider not modeling all the compliances, depending on the purpose of your model. If there are specific compliances that are more dominant than others, then try modeling only the dominant compliances.

The coupling of drivelines to external loads — for an automobile, the wheel-tire-road load — is often stiff. Driving and road conditions typically change over seconds or tens of

seconds. However, the internal changes of the drive system of an automobile can change over fractions of a second, especially if clutch changes and braking are at work. In addition, clutch locking and unlocking events create dynamic discontinuities.

For example, a tire is “stiff” in responding slowly to imposed forces and experiencing slip. A tire also has a broad range of frequency responses. Consider modeling tire compliance only when you model the automobile accelerating from rest.

Optimize Simulation of Clutches

Clutch locking and unlocking events generate discontinuous changes in driveline dynamics and can cause major inaccuracies, particularly if you are simulating with a large variable-step solver tolerance or a large fixed time step.

- Clutch discontinuities change the number and nature of the degrees of freedom of the driveline during the simulation.
- Because clutch discontinuities are idealized events, they cause the driveline torques to change abruptly, as the clutch switches abruptly between static and kinetic friction.

Smoothing and Offsetting Clutch Control Signals

You exert dynamic control on the locking and unlocking of clutches through their input pressure or other locking signals.

The simplest way to force a locking is to change a clutch pressure abruptly from zero to some predetermined value. You can then force an unlocking by abruptly changing the clutch pressure back to zero. Such abrupt clutch pressure changes are not realistic. The best solution is to model full clutch actuation. However, you can use simplified models to reduce model complexity.

You can improve your clutch modeling and make it more realistic by ensuring that the clutch pressure signals rise and fall smoothly, not suddenly. The Simulink Sources library provides many ways to create such signals. You can also reshape existing signals using blocks such as State-Space and Transfer Fcn.

These example models illustrate smoothed clutch pressure signals:

- `sdL_clutch_custom` ramps up and down the input clutch pressure.
- `sdL_car` uses Transfer Fcn blocks to reshape and smooth sharp clutch pressure signals.

For more information about smoothing clutch signals, see “Model Realistic Clutch Pressure Signals” on page 12-8.

Adjusting Clutch Parameters

You can adjust internal parameters within each clutch block to control when and how the clutch locks and unlocks.

Changing Pressure or Force Threshold

The locking signal coming into a clutch is physical, with units of force or pressure. With some clutches, you can specify a force or pressure threshold F_{th} or P_{th} . This threshold imposes a cutoff on the clutch pressure such that the effective controlling pressure is $P - P_{th}$ rather than P . If $P < P_{th}$, no pressure at all is applied. (Normal force between clutch surfaces can substitute for pressure.) Raising the pressure or force threshold of a clutch that has an adjustable threshold makes it harder for the clutch to engage.

Tip If a clutch in your simulation engages too easily, consider raising its pressure or force threshold. If the clutch has difficulty engaging, consider lowering this threshold.

Changing Velocity Tolerance

Most clutch blocks have a velocity tolerance parameter ω_{Tol} that controls when the clutch locks or unlocks.

- A clutch can lock only if the relative shaft velocity ω lies in the range $-\omega_{Tol} < \omega < +\omega_{Tol}$.
- A clutch unlocks if the torque across the clutch exceeds the static friction limit, which depends in turn on the normal force across the clutch.

You specify ω_{Tol} values through each clutch block.

Tip If a clutch switches between locked and unlocked too easily during simulation, consider increasing its velocity tolerance.

Adjusting Solvers for Clutch Discontinuities

If you use a solver tolerance or step size that is too large, clutch discontinuities can cause major inaccuracies.

- If the variable-step tolerances are too large, the solver finds it difficult or impossible to track the dynamic change associated with the change of friction torques acting on the driveline accurately.
- If the fixed step size is too large, the solver cannot accurately resolve abrupt changes such as clutch locking and unlocking events. A fixed-step solver cannot adaptively reduce its step size to compensate.

Tip If you encounter convergence failures or abrupt driveline state (velocity) changes at or around the instant of clutch state changes, consider reducing the solver tolerances (for a variable-step solver) or the step size (for a fixed-step solver). Set the variable-step solver tolerance or the fixed-step solver step size to the smallest value possible that produces an acceptable simulation speed (not too slow).

Adjustment	Solver Type and Setting	Effect on Accuracy	Effect on Speed	Effect on Clutch Simulation
Reduce	Variable-step: tolerances	Increases	Reduces	Improves resolution and simulation of abrupt locking and unlocking
	Fixed-step: step size			
Increase	Variable-step: tolerances	Reduces	Increases	Degrades resolution and simulation of abrupt locking and unlocking
	Fixed-step: step size			

Resolve Partitioning Solver Simulation Issues for Simscape Driveline Models

In this section...

“Resolving Issues for Blocks with Stiffness or Friction” on page 16-7

“Using the Partitioning Solver” on page 16-7

“Resolve Initial Condition Errors and Warnings” on page 16-8

“Reduce Chatter Due to Friction” on page 16-17

“Resolve Chatter Due to Stiffness” on page 16-8

The Partitioning solver is a Simscape fixed-step, local solver that improves performance for certain models. However, when using the Partitioning solver, some Simscape Driveline models generate warnings, stop and generate errors, fail to initialize, or yield signal chatter due to numerical difficulties. These examples show how to eliminate errors, mitigate warnings, and reduce chatter by resolving numerical difficulties.

Resolving Issues for Blocks with Stiffness or Friction

Numerical difficulties that prevent models from simulating to completion, yield warnings, or introduce chatter are typically related to blocks that have high stiffness or friction. Simscape Driveline blocks with high stiffness or friction include clutches, belt pulleys, tires, and flexible shafts.

To resolve numerical difficulties in models that contain these blocks, use one or more of the methods:

- Adjust the solver settings.
- Remove high-priority variable redundancies.
- Unlock clutch initial conditions.
- Loosen friction-related tolerances.
- Eliminate high-stiffness blocks.
- Eliminate degrees of freedom.

Using the Partitioning Solver

When simulating models using the Partitioning solver:

- 1 Simulate the model by using a global variable-step solver to capture baseline results that agree with your mathematical model or empirical data.
- 2 Configure the local solver for a Partitioning solver simulation.
- 3 Run the Partitioning solver simulation. If the simulation:
 - Runs to completion — Compare the Partitioning solver simulation results to the baseline results. If the results do not agree, adjust the solver settings or model components and simulate again. For example, decrease the step size or simplify model dynamics. For more information, see “Reduce Numerical Stiffness” (Simscape), “Choose Step Size and Number of Iterations” (Simscape), and “Reduce Fast Dynamics” (Simscape).

Repeat until the simulation returns results that agree with the baseline results.

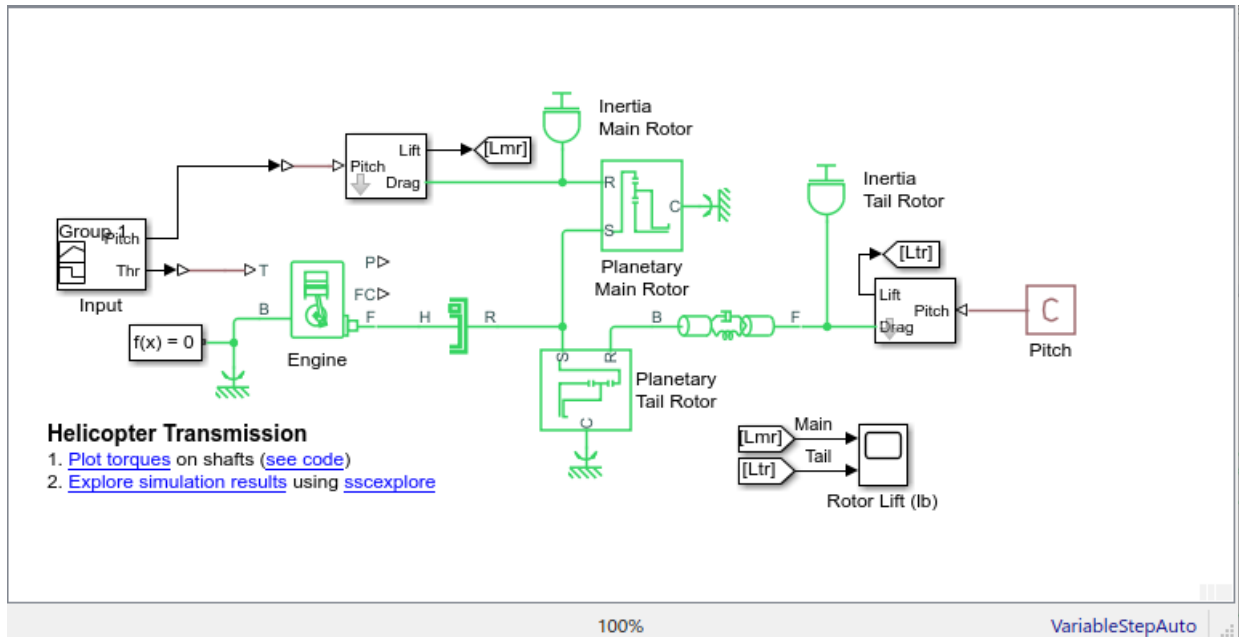
- Fails to simulate to completion due to numerical issues — Resolve the issues by applying one or more of the methods listed in “Resolving Issues for Blocks with Stiffness or Friction” on page 16-7. Rerun the simulation. Repeat until the simulation runs to completion, then compare the results to the baseline results. If the results do not agree, adjust the solver settings or model components and rerun the simulation. Repeat until the Partitioning solver simulation returns results that agree with the baseline results.

Resolve Initial Condition Errors and Warnings

This example shows how to resolve numerical difficulties that generate initial condition errors and warnings. When an initial condition issue prevents a simulation from initiating or running to completion, MATLAB stops the simulation and generates an error. When a simulation is unable to satisfy high-priority targets, the simulation continues to run, but MATLAB generates a warning.

- 1 Open the model. At the MATLAB command prompt, enter:

```
%% Open the Model
model = 'sdl_transmission_helicopter_base';
open_system(model)
```



- The model window indicates that the Simulink global solver is a variable-step solver.
- 2 To examine the Simscape local solver configuration, open the Solver Configuration block settings.

See Code

```

%% Define the Solver Configuration Block and Path
solvConfig = 'Solver Configuration';
solvConfigPath = [model, '/', solvConfig];

%% Open the Solver Configuration Block Settings
open_system(solvConfigPath)

```

- The model is configured to simulate using the Simulink global solver because the **Use local solver** check box is cleared.
- 3 Simulate the model and then, to examine the baseline results from the variable-step simulation, open the Scope block.

See Code

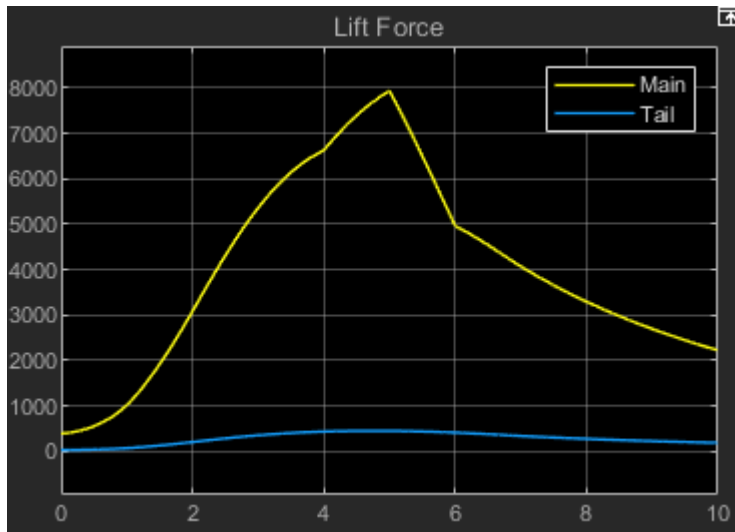
```

%% Simulate the Model
sim(model)

%% Examine the Baseline Results
% Define the Scope Block and Path
scope = 'Rotor Lift (lb)';
scopePath = [model, '/', scope];

% Open the Scope Block to View the Results
open_system(scopePath)

```



- 4 Configure the model for a Partitioning solver simulation. In the Solver Configuration block settings, select the **Use local solver** check box.

See Code

```

%% Configure the Partitioning Solver
% Open the Solver Configuration Block Settings
open_system(solvConfigPath)

% Configure the Solver Configuration Block
set_param(solvConfigPath, ...
    'UseLocalSolver', 'on', ...
    'DoFixedCost', 'on')

```



```

%% To Confirm the Configuration Change by Refreshing
% the Solver Configuration Block Settings,
% Uncomment the Next Two Lines of Code (Optional)
% close_system(solvConfigPath)
% open_system(solvConfigPath)

```

When you select the **Use local solver** check box, related parameters are enabled. By default, the parameters in the Solver Configuration block are set to::

- 1 **Solver type** — Partitioning
 - 2 **Sample time** — 0.05
 - 3 **Partition method** — Robust simulation
 - 4 **Partition storage method** — Exhaustive
 - 5 **Use fixed-cost runtime consistency iterations** — Selected
 - 6 **Nonlinear iterations** — 3
- 5 Simulate the model.

See Code

```

%% Simulate the Model
sim(model)

```

The simulation generates two warnings and one error. The error prevents compilation because it stops the simulation.

```

Warning: Initial conditions for eliminated differential variables not
supported by partitioning solver. The following states may deviate from
requested initial conditions:
   ['sdl_transmission_helicopter_base/Inertia Main Rotor'] Inertia_Main_Rotor.w
   ['sdl_transmission_helicopter_base/Inertia Tail Rotor'] Inertia_Tail_Rotor.w

Warning: Simscape succeeded in finding consistent states with which to start the simulation,
but the states found may deviate from requested initial conditions.

Error: ['sdl_transmission_helicopter_base/Solver Configuration']:
At time 0.050000, one or more assertions are triggered.
See causes for specific information.

Caused by:
Argument of sqrt must be nonnegative. The assertion comes from:
Block path: sdl_transmission_helicopter_base/Lift and Drag Main Rotor/L^.5

Assert location:
In between line: 48, column: 19 and line: 48, column: 20 in file:
Dir:\Program\Files\MATLAB\R20XXx\toolbox\physmod\simscape\library\m\
physical_signal_legacy\+foundation\physical_signal\+functions\math_function.ssc

```

- 6 The error message indicates that the solver returned a negative value for a square-root computation. If a variable-step simulation runs to completion, but a fixed-step

simulation generates this type of error for the same model, typically, the local solver sample time is too large or the local solver number of nonlinear iterations is too small. Increase the number of nonlinear iterations for the local solver. In the Solver Configuration block settings, set **Nonlinear iterations** to 9. Then simulate the model and view the results in the Scope block.

See Code

```
%% Increase the Number of Nonlinear Iterations for the Local Solver
% Increase the Number of Nonlinear Iterations
set_param(solvConfigPath, 'MaxNonlinIter', '9')
```

```
%% To Confirm the Configuration Change by Refreshing
% the Solver Configuration Block Settings,
% Uncomment the Next Two Lines of Code (Optional)
% close_system(solvConfigPath)
% open_system(solvConfigPath)
```

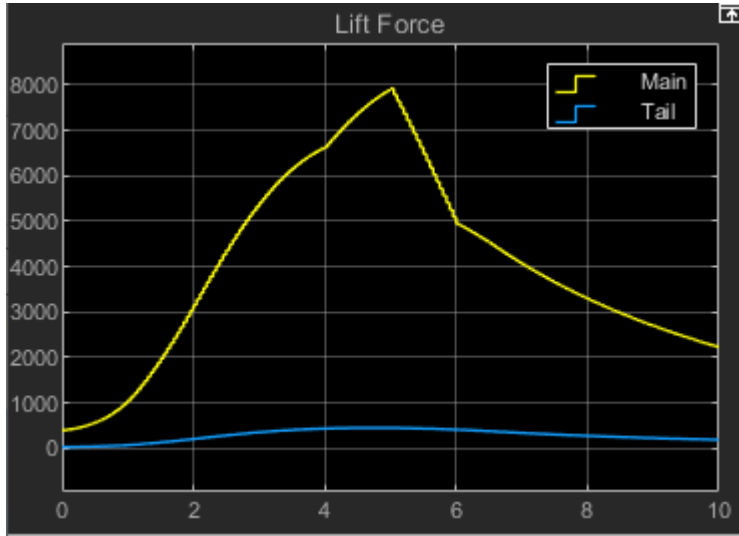
```
%% Simulate the Model
sim(model)
```

```
% Open the Scope Block to View the Results
open_system(scopePath)
```

```
Warning: Initial conditions for eliminated differential variables not
supported by partitioning solver. The following states may deviate
from requested initial conditions:
```

```
['sdL_transmission_helicopter_base/Inertia Main Rotor']
Inertia_Main_Rotor.w
['sdL_transmission_helicopter_base/Inertia Tail Rotor']
Inertia_Tail_Rotor.w
```

```
Warning: Simscape succeeded in finding consistent states with which to
start the simulation, but the states found may deviate from requested
initial conditions.
```



The simulation runs to completion, but it still generates two warnings. The results agree with the baseline results in terms of the significant characteristics for the model (such as the slopes, magnitudes, and inflection points).

- Both warnings are caused by initial condition issues. Investigate the warnings by examining the variables in the model. To open the Variable Viewer, on the **Apps** tab, in the **Physical Modeling** category, click **Simscape Variable Viewer**. To filter for variables that have targets that the simulation is unable to satisfy, click the arrow to the right of the **Status** column header, and in the drop-down list, clear the **OK** check box.

Name	Stat...	Priority	Target	Start	Unit
Engine	<input type="checkbox"/>				
engine_inertia	<input type="checkbox"/>				
w	<input type="checkbox"/>	High	800.0	809.3796089952091	rpm
Flexible Shaft	<input type="checkbox"/>				
fsTorsion	<input type="checkbox"/>				
phi	<input type="checkbox"/>	High	[-1.8E-4;-1.35E-4;-9.0E-5;-4.5E-5;...	[-2.119130861320607;-2.119098570415258;-2.119066340917349;-2.119034172858798;-2.11900206...	rad
w	<input type="checkbox"/>	High	[-400.0;-400.0;-400.0;-400.0;-400.0]	[-404.6898044976045;-404.6922317558877;-404.6946707421279;-404.6971214624212;-404.699583...	rpm
Inertia Main Rotor	<input type="checkbox"/>				
w	<input type="checkbox"/>	High	-80.0	-80.9379608995209	rpm
Inertia Tail Rotor	<input type="checkbox"/>				
w	<input type="checkbox"/>	High	-400.0	-404.699583922893	rpm
Lift and Drag Tail Rotor	<input type="checkbox"/>				
Ideal Rotational Motion Sens...	<input type="checkbox"/>				
phi	<input type="checkbox"/>	High	0.0	2.094395102393195	rad

There are several variables that have High priority targets that the simulation is unable to satisfy.

The two inertia blocks that are mentioned in the first warning are among the blocks that have problematic high-priority targets. The Partitioning solver has converted these variables to nondifferential variables, which are algebraically related to remaining differential variables. Initial conditions for nondifferential variables are not supported by the Partitioning solver.

The Inertia Tail Rotor initial rotational velocity, -400 (rpm), is algebraically constrained to match the rotational velocity specified in the Flexible Shaft block. Likewise, the Inertia Main Rotor initial rotational velocity of -80 (rpm) is algebraically constrained to match the rotational velocity of the **R** node of Planetary Main Rotor block.

- 8 Eliminate the first warning by removing the priority for the problematic initial condition targets. Open both the Inertia Main Rotor block and the Inertia Tail Rotor block, and, in the **Variables** settings, set the priority for the **Rotational velocity** variable to **None**. Then simulate the model, examine the variables, and view the results.

See Code

```
%% Remove the Target Priorities

% Define the Inertia Main Rotor Block and Path
inertiaMainRotor = 'Inertia Main Rotor';
inertiaMainRotorPath = [model, '/', inertiaMainRotor];

% Define the Inertia Tail Rotor Block and Path
inertiaTailRotor = 'Inertia Tail Rotor';
inertiaTailRotorPath = [model, '/', inertiaTailRotor];

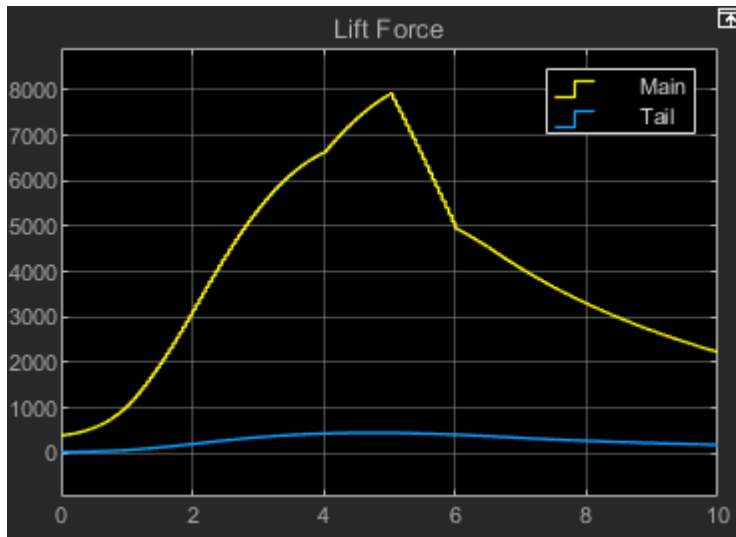
% Eliminate the High Priorities
set_param(inertiaMainRotorPath, 'w_priority', 'None')
set_param(inertiaTailRotorPath, 'w_priority', 'None')

% Simulate the Model
sim(model)

% Open the Scope Block to View the Results
open_system(scopePath)
```

Warning: Simscape succeeded in finding consistent states with which to start the simulation, but the states found may deviate from requested initial conditions.

Name	Stat...	Priority	Target	Start	Unit
Engine	■				
engine_inertia	■				
w	■	High	800.0	809.3796089952091	rpm
Flexible Shaft	■				
fsTorsion	■				
phi	■	High	[-1.8E-4;-1.35E-4;-9.0E-5;-4.5E-5;...	[-2.119130861320607;-2.119098570415258;-2.119066340917349;-2.119034172858798;-2.11900206...	rad
w	■	High	[-400.0;-400.0;-400.0;-400.0;-400.0]	[-404.6898044976045;-404.6922317558877;-404.6946707421279;-404.6971214624212;-404.699583...	rpm
Lift and Drag Tail Rotor	■				
Ideal Rotational Motion Sens...	■				
phi	■	High	0.0	2.094395102393195	rad



The inertia initial condition warning is not generated and the Variable Viewer no longer shows the velocity for the inertia among the failed targets. The results agree with the baseline results in terms of the significant characteristics for the model.

- Examine the Variable Viewer data. The remaining warning is issued because there is a large difference between the **Target** and **Start** values for several high priority targets. To reduce the difference, decrease the sample time for the local solver. In the Solver Configuration block settings, set **Sample time** to 0.01 . Then simulate the model and view the results.

See Code

```
% Increase the Local Solver Sample Time
set_param(solvConfigPath, 'LocalSolverSampleTime', '0.01')
```

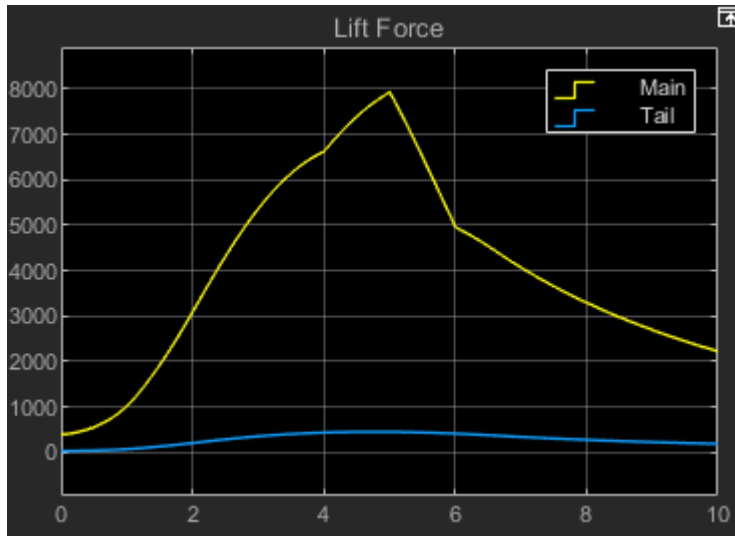
```
% Simulate the Model
sim(model)
```

```
% Open the Scope Block to View the Results
open_system(scopePath)
```

Warning: Simscape succeeded in finding consistent states with which to start the simulation, but the states found may deviate from requested initial conditions.

The screenshot shows a table of variables from a Simscape model. The table has columns for Name, Status (Stat...), Priority (Prior...), Target, Start, and Unit. The variables are grouped into three main sections: Engine, Flexible Shaft, and Lift and Drag Tail Rotor.

Name	Stat...	Prior...	Target	Start	Unit
Engine	<input checked="" type="checkbox"/>				
engine_inertia	<input checked="" type="checkbox"/>				
w	<input checked="" type="checkbox"/>	High	800.0	801.2725692424472	rpm
Flexible Shaft	<input checked="" type="checkbox"/>				
fsTorsion	<input checked="" type="checkbox"/>				
phi	<input checked="" type="checkbox"/>	High	[-1.8E-4;-1.35E-4;-9.0E-5;-4.5E-5...	[-0.4197253361758484;-0.419679659252492;-0.419634023885098;-0.4195884300340003;-0.41954287...	rad
w	<input checked="" type="checkbox"/>	High	[-400.0;-400.0;-400.0;-400.0;-400...	[-400.6362846212236;-400.635638207034;-400.6350314758654;-400.6344643898394;-400.633936913...	rpm
Lift and Drag Tail Rotor	<input checked="" type="checkbox"/>				
Ideal Rotational Motion Sen...	<input checked="" type="checkbox"/>				
phi	<input checked="" type="checkbox"/>	High	0.0	0.4188790204786391	rad



There is still a warning for several high-priority targets, but the difference between the **Target** and **Start** values is smaller. The results match the baseline results in terms of the significant characteristics for the model and, due to the decrease in sample time, the **Main** signal is smoother.

Reduce Chatter Due to Friction

This example shows how to reduce chatter, a type of signal noise. Chatter can occur when you use the Partitioning solver to simulate a model that includes a block that models friction. Certain Simscape Driveline brake, clutch, drive, gear, and tire blocks can model friction.

- 1 Open the model. At the MATLAB command prompt, enter:

```
%% Open the Model
model = 'sdl_capstan';
open_system(model)

% Expand the Model Window to Accomodate the Property Inspector
modelLocation = get_param(model, 'location');
modelLocationAdjustment = [0 0 400 100];
modelLocationAdjusted = modelLocation + modelLocationAdjustment;
set_param(model, 'location', modelLocationAdjusted)
```



```

set_param(model,...
    'SimscapeLogType','local',...
    'SimscapeLogToSDI', 'on',...
    'SimscapeLogOpenViewer', 'on')

%% To Confirm the Configuration Change by Refreshing
% the the Model Configuration Parameters,
% Uncomment the Next Four Lines of Code (Optional)
% closeDialog(modelConfigObj)
% openDialog(modelConfigObj, 'Simscape')
% pause(5)
% closeDialog(modelConfigObj)

%% Define the Capstan Block and Path
capstan = 'Capstan';
capstanPath = [model, '/', capstan];

%% Enable Data Logging for the Capstan block
set_param(capstanPath, 'LogSimulationData', 'on')

%% To Confirm the Configuration Change in the Property Inspector (Optional):
% 1. Uncomment the next line of code
% set_param(capstanPath, 'Selected', 'on');
%
% 2. Manually, open the Property Inspector:
% 2.1. On the Simulink Modeling tab,
%     click the arrow on the right side of the Design section.
% 2.2. In the Data Management category, select Property Inspector.
% 3. In the Property Inspector, expand the Logging settings.

```

- 3** Simulate the model and then, to examine the results for the Capstan block **B**-node force, in the Simulation Data Inspector:

- a** Expand the **Capstan** node.
- b** Select the **fB** check box.

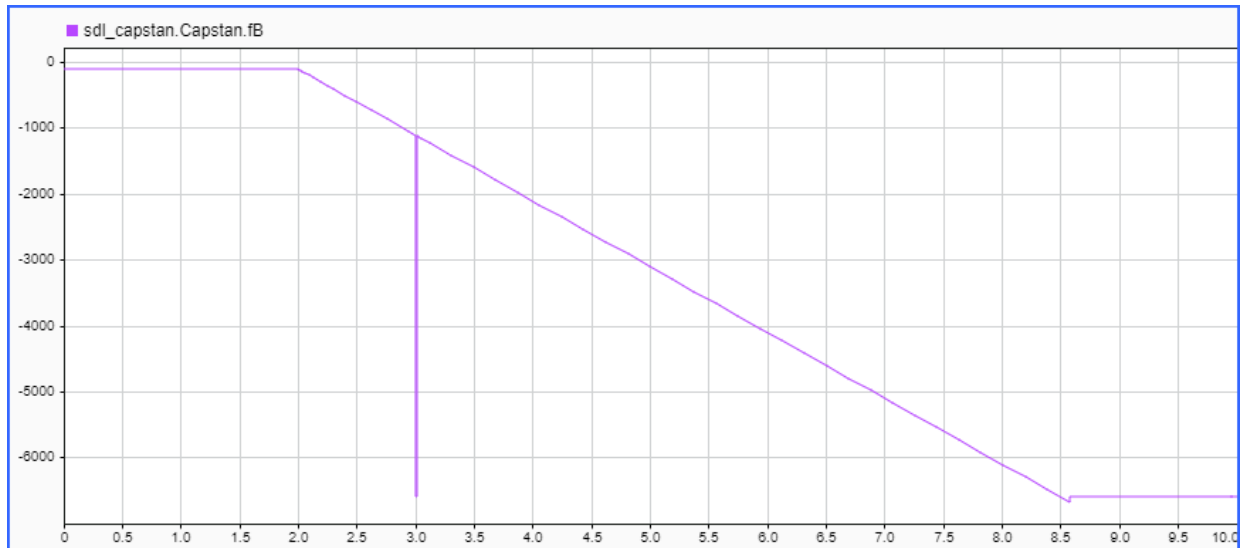
See Code

```

%% Simulate the Model
sim(model)

%% Select and Format the Data for the Simulation Data Inspector
runIDs = Simulink.sdi.getAllRunIDs;
baselineRunID = runIDs(end);
baselineRun = Simulink.sdi.getRun(baselineRunID);
baselinefB = baselineRun.getSignalByIndex(5);
baselinefB.Checked = true;
baselineRunTireLFSlip.LineColor = [0.7176    0.2745    1];
% baselinefB.LineDashed = '-.';

```



The results show that the load force causes slip when it exceeds the capstan force limit.

- 4 Close the Simulation Data Inspector.

See Code

```
%% Close the Simulation Data Inspector
Simulink.sdi.close
```

- 5 Configure the model for a Partitioning solver simulation. Click the Solver Configuration and in the Solver Configuration block settings:

- a Select the **Use local solver** check box.
- b Set **Solver type** to Partitioning.
- c Set **Partition method** to Robust simulation.
- d Set **Solver type** to Partitioning.
- e Set **Partition storage method** to Exhaustive.

See Code

```
% Configure the Local Solver

%% Define the Solver Configuration Block and Path
solvConfig = 'Solver Configuration';
solvConfigPath = [model, '/', solvConfig];
```

```

%% Configure the Solver Configuration Block
set_param(solvConfigPath,...
    'UseLocalSolver','on',...
    'LocalSolverChoice','NE_PARTITIONING_ADVANCER',...
    'PartitionMethod','ROBUST',...
    'PartitionStorageMethod','EXHAUSTIVE',...
    'DoFixedCost','on')

%% To Confirm the Configuration Change in the Property Inspector (Optional):
% 1. Uncomment the next line of code
% set_param(solvConfigPath,'Selected','on');
%
% 2. If the Property Inspector is closed, manually, open it:
% 2.1. On the Simulink Modeling tab,
%      click the arrow on the right side of the Design section.
% 2.2. In the Data Management category, select Property Inspector.
% 3. In the Property Inspector, expand the Logging settings.

```

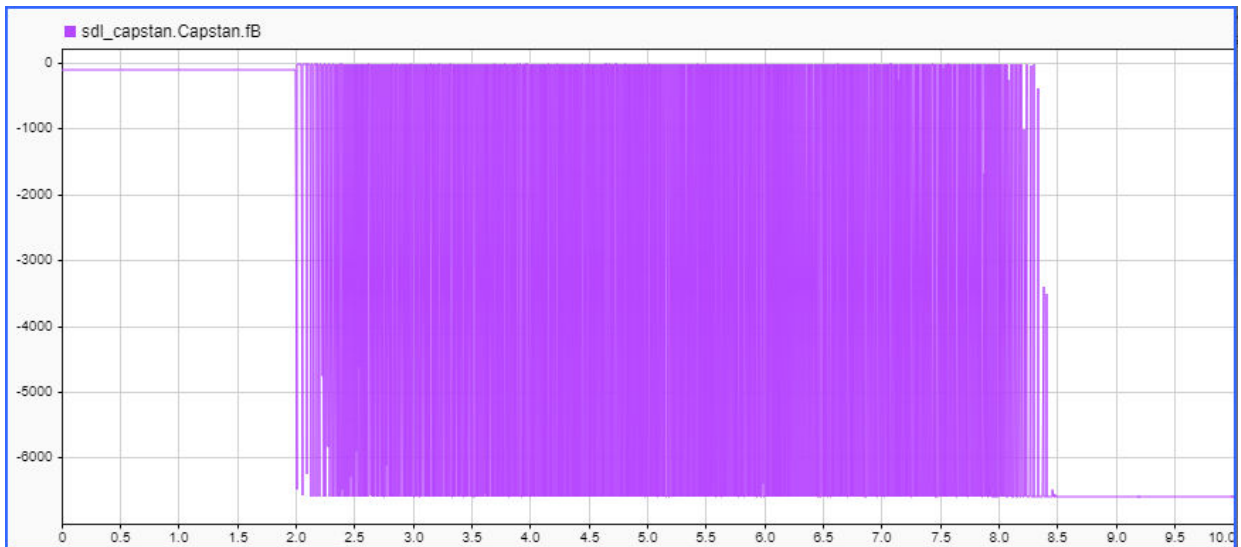
6 Simulate the model and examine the results in the Simulation Data Inspector.

See Code

```

%% Simulate the model
sim(model)

```



The Capstan block **B**-node force shows significant chatter during the Partitioning solver simulation. Refer to the documentation for the Belt Pulley block for sources of stiffness or discontinuities.

See Code

```
%% Open the documentation for the Belt Pulley block
web(fullfile(docroot, 'physmod/sdl/ref/beltpulley.html'))
```

According to the documentation, V_{rel} is the relative velocity between the belt and pulley periphery. The idealization of the discontinuity at $V_{rel} = 0$ is both difficult for the solver to resolve and not physically accurate. To alleviate this issue, the friction coefficient is assumed to change its value as a function of the relative velocity such that

$$\mu = -f * \tanh\left(4 * \frac{V_{rel}}{V_{thr}}\right),$$

where:

- μ is the instantaneous value of the friction coefficient.
- f is the steady-state value of the friction coefficient.
- V_{thr} is the friction velocity threshold.

Therefore, for small values of V_{thr} , the friction force stiffness near V_{rel} increases significantly.

- 7 To resolve this issue, increase the V_{thr} value in the Capstan block settings. In the **Contact** settings, increase **Velocity threshold** from 0.001 to 0.1.

See Code

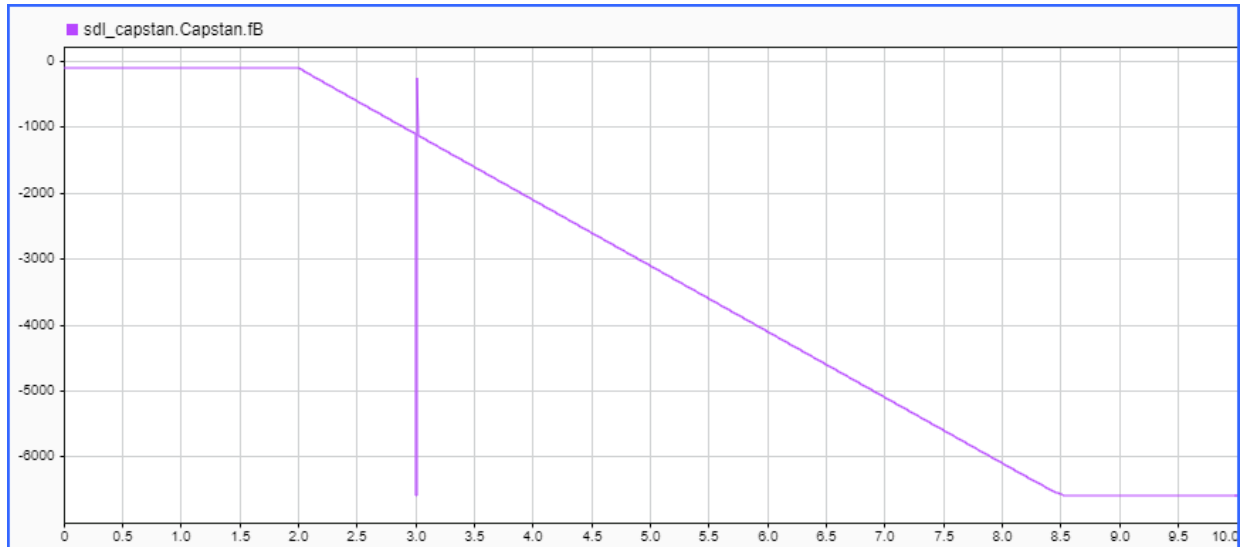
```
%% Increase the Velocity Threshold
set_param(capstanPath, 'v_thr', '0.1')

%% To Confirm the Configuration Change in the Property Inspector (Optional):
% 1. Uncomment the next line of code
% set_param(capstanPath, 'Selected', 'on');
%
% 2. If the Property Inspector is closed, manually, open it:
%   2.1. On the Simulink Modeling tab,
%       click the arrow on the right side of the Design section.
%   2.2. In the Data Management category, select Property Inspector.
%
% 3. In the Property Inspector, expand the Contact settings.
%
% 4. Uncomment the next line of code
% set_param(capstanPath, 'Selected', 'off');
```

- 8 Simulate the model and examine the results in the Simulation Data Inspector.

See Code

```
%% Simulate the Model
sim(model)
```



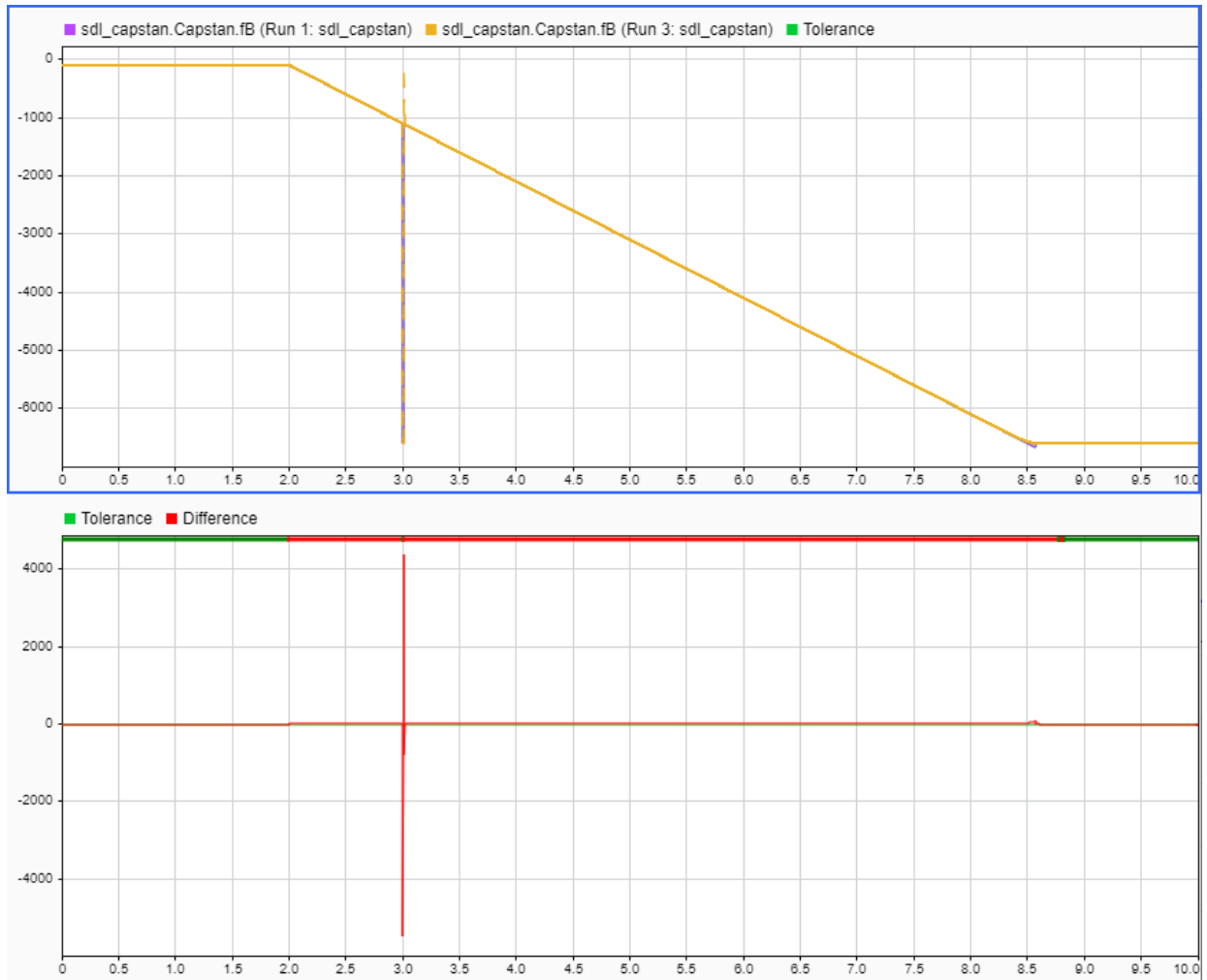
The Capstan block **B**-node force no longer shows significant chatter.

- 9** Compare the data from the variable-step simulation to the new data from the fixed-step solver simulation.
 - a** In the Simulation Data Inspector, select **Compare**.
 - b** To configure the comparison, in the top, right pane:
 - i** Click the arrow on the right side of the **Baseline** setting. At the bottom of the list, click **Signals** and then select `sdl_capstan.Capstan.fB` (Run 1: `sdl_capstan`).
 - ii** Click the arrow on the right side of the **Compare to** setting. At the bottom of the list, click **Signals** and then select `sdl_capstan.Capstan.fB` (Run 3: `sdl_capstan`).
 - iii** Click **Compare**.
 - c** To change the Partitioning solver results line color, in the **Properties** pane, in the **Compare to** column, click the colored **Line**, select a different color and style, such as yellow and dash-dot, and then click **Set**.

See Code

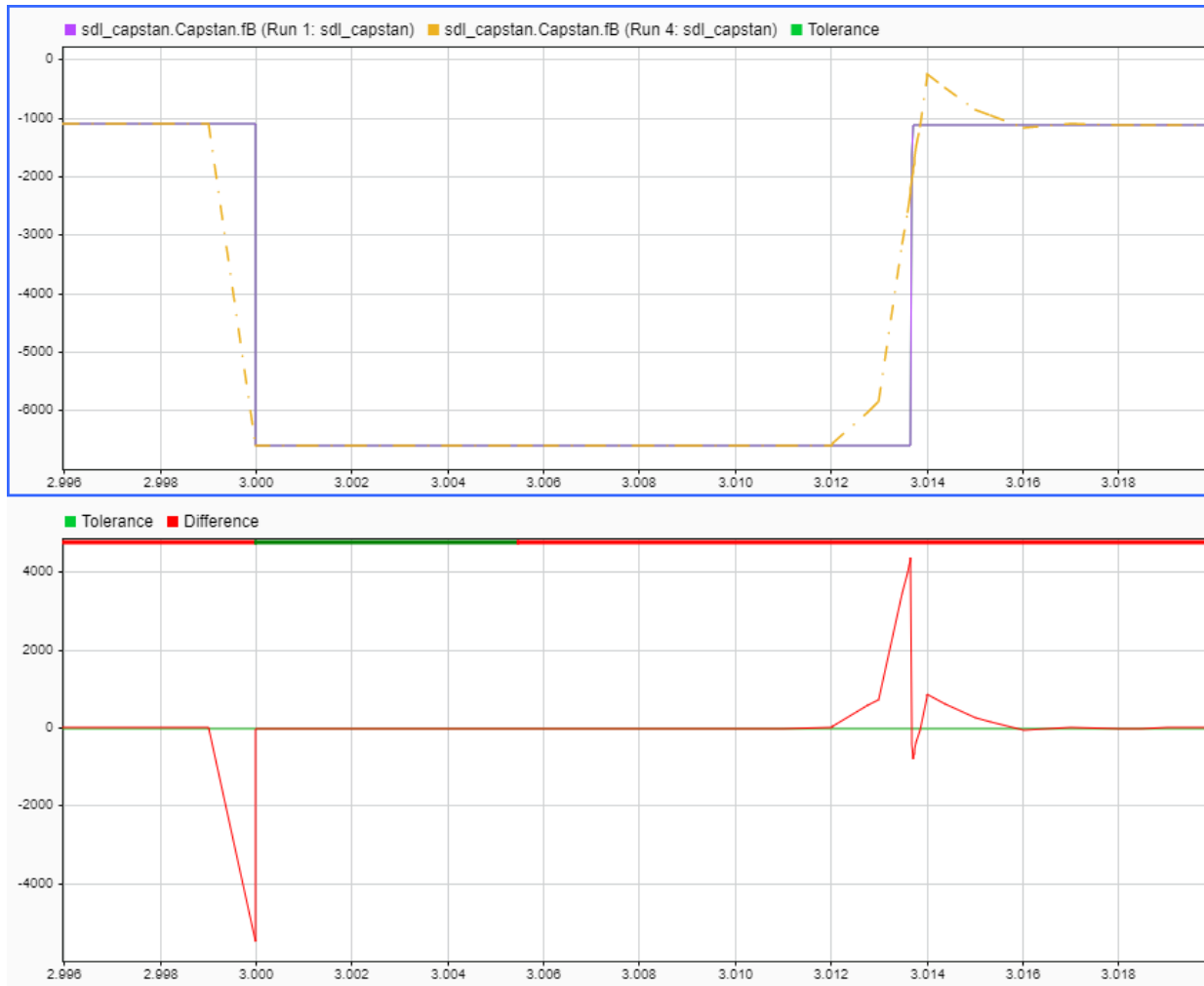
```
%% Select and Format Data for the Simulation Data Inspector Comparison
runIDs = Simulink.sdi.getAllRunIDs;
partSolverRun2ID = runIDs(end);
partSolverRun2 = Simulink.sdi.getRun(partSolverRun2ID);
partSolverRun2fB = partSolverRun2.getSignalByIndex(5);
% partSolverRun2fB.Checked = true;
partSolverRun2fB.LineColor = [0.9294    0.6941    .1255];
partSolverRun2fB.LineDashed = '-.-';

% Compare Runs
compBaselinePartition2 = Simulink.sdi.compareRuns(baselineRunID,...
    partSolverRun2ID);
```



The Capstan block **B**-node force no longer shows significant chatter. The Partitioning solver and Variable-step solver simulation results agree for much of the simulation. The difference in the results is the expected difference for a comparison of variable-step and fixed-step solver results.

The figure shows a time-scaled view of the difference between the results of the variable-step simulation and the results of the Partitioning solver fixed-step simulation.



By decreasing the step size during a simulation, the variable-step solver is able to capture fast dynamics that occur at simulation time, $t = 3$ s. The Partitioning solver steps over the fast dynamics because the fixed-step size is too large. You could reduce or resolve the difference by decreasing the step size for the Partitioning solver, but doing so increases simulation duration time and can result in a real-time simulation overrun.

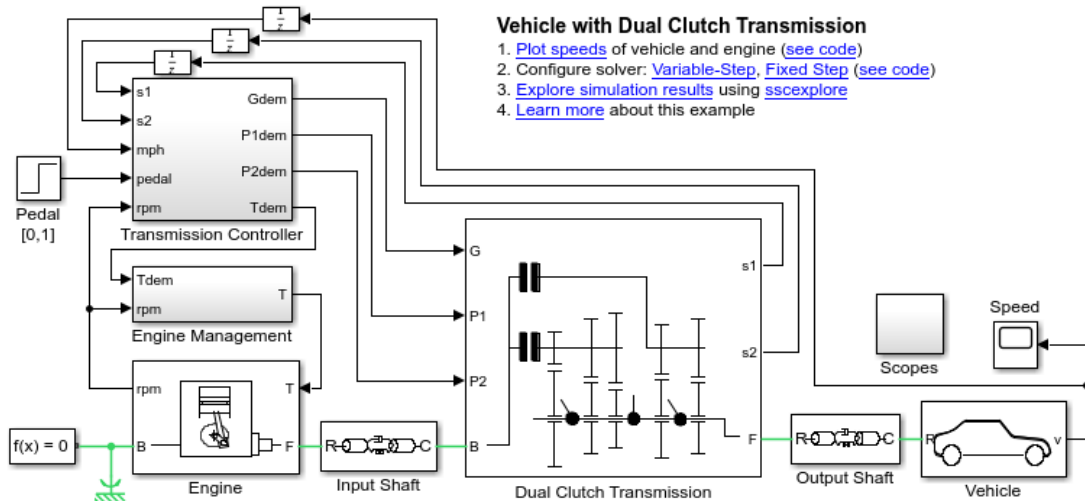
The overshoot of the fixed step solver at simulation time $t = 3.014$ s is also a characteristic of fixed-step solvers. Reducing the step size for the Partitioning solver can minimize the overshoot but can also result in a real-time simulation overrun.

Resolve Chatter Due to Stiffness

This example also shows how to resolve numerical difficulties that yield chatter in Simscape Driveline simulations that use the Partitioning solver. In this case, the chatter is caused by stiffness. A stiff model is one that contains both fast and slow dynamics.

- 1 Open the model with block. At the MATLAB command prompt, enter:

```
%% Open the Model
model = 'sdl_vehicle_dual_clutch';
open_system(model)
```



The model is configured for a variable-step simulation that uses the global solver. Data logging is enabled only for the signal that contains the gear state data.

- 2 Configure the model for data logging. For this example, tire slip is the data of interest and logging additional data increases the computational cost of simulation. Enable data logging for tire slip and disable data logging for the gear state.
 - a In the Vehicle subsystem, the Tire LF Tire (Magic Formula) block represents the left-front tire of the vehicle. The Tire LF **S** port, which transmits the tire slip data, is a physical signal port. You can log physical signal data using Simulink data logging. The destination for the signal is the Tire slip Scope block, which is in the Scopes subsystem.
 - i Open the Scopes subsystem.
 - ii The tire slip data is in the signal that the FrontSlip From block transmits to the Tire slip Scope block. Select the signal, and in the Simulink toolstrip, on the **Signal** tab, click the arrow on the right side of the **Monitor** section. In the **Signal Monitoring** category, click **Log Signals**.

See Code

```
%% Enable Tire Slip Data Logging
% Enable Logging for Front Left Tire Slip Signal

% Define Scope Subsystem and path
scopesSS = 'Scopes';
scopesSSPath = [model, '/', scopesSS];

% Define Front Slip From Tag Signal
fromTireLFLogSlip1 = 'From8';
fromTireLFLogSlip1Path = [scopesSSPath, '/', fromTireLFLogSlip1];

%% Enable the Front Slip From Tag Signal for
% Simulink(TM) data logging and viewing with the Simulation Data Inspector

fromTireLFLogSlip1PortHandles = get_param(fromTireLFLogSlip1Path, ...
    'PortHandles');
fromTireLFLogSlip1Output = fromTireLFLogSlip1PortHandles.Output;
set_param(fromTireLFLogSlip1Output, 'DataLogging', 'on')
```

- b In the Transmission Controller subsystem, in the Shift state subsystem, the z3 Unit Delay block transmits the gear state.
 - i Open the Transmission Controller subsystem.
 - ii Open the Shift state subsystem.
 - iii The Gear state G subsystem transmits the gear state to a Unit Delay block, which, in turn, transmits the data to the Gear state Output block. The signal that connects the Unit Delay block to the Gear state Output block is marked for data logging. Select the signal, and in the Simulink toolstrip, on the

Signal tab, click the arrow on the right side of the **Monitor** section. In the **Signal Monitoring** category, click **Log Signals**.

See Code

```

%% Disable Gear State Data Logging

% Define Transmission Controller Subsystem and Path
transCntrl = 'Transmission Controller';
transCntrlPath = [model, '/', transCntrl];

% Define Shift State Subsystem and path
shiftState = 'Shift state';
shiftStatePath = [transCntrlPath, '/', shiftState];

% Define Unit Delay block and path
unitDelay = 'z3';
unitDelayPath = [shiftStatePath, '/', unitDelay];

% Disable Unit Delay block signal logging
unitDelayPortHandles = get_param(unitDelayPath, 'PortHandles');
unitDelayOutputport = unitDelayPortHandles.Outputport;
set_param(unitDelayOutputport, 'DataLogging', 'off')

```

- 3 Obtain and examine the baseline results. Simulate using the global variable-step solver and review the results in the Simulation Data Inspector.
 - a Run the simulation. In the Simulink toolstrip, on the **Simulation** tab, in the **Simulate** section, click **Run**.

See Code

```

%% Simulate the Model
sim(model)

```

- b Open the Simulation Data Inspector. In the Simulink toolstrip, on the **Simulation** tab, click the arrow on the right side of the **Review Results** section, and, in the **Signal Logging Results** category, click Signal Logging Results. To inspect the tire slip data, select the **From8:1** check box.

See Code

```

%% Examine Baseline Results in the Simulation Data Inspector

% Define Baseline Run
runIDs = Simulink.sdi.getAllRunIDs;
baselineRunID = runIDs(end);
baselineRun = Simulink.sdi.getRun(baselineRunID);
baselineRunTireLFSlip = baselineRun.getSignalByIndex(1);

% Set Line Color
baselineRunTireLFSlip.LineColor = [0 0.4470 0.7410];

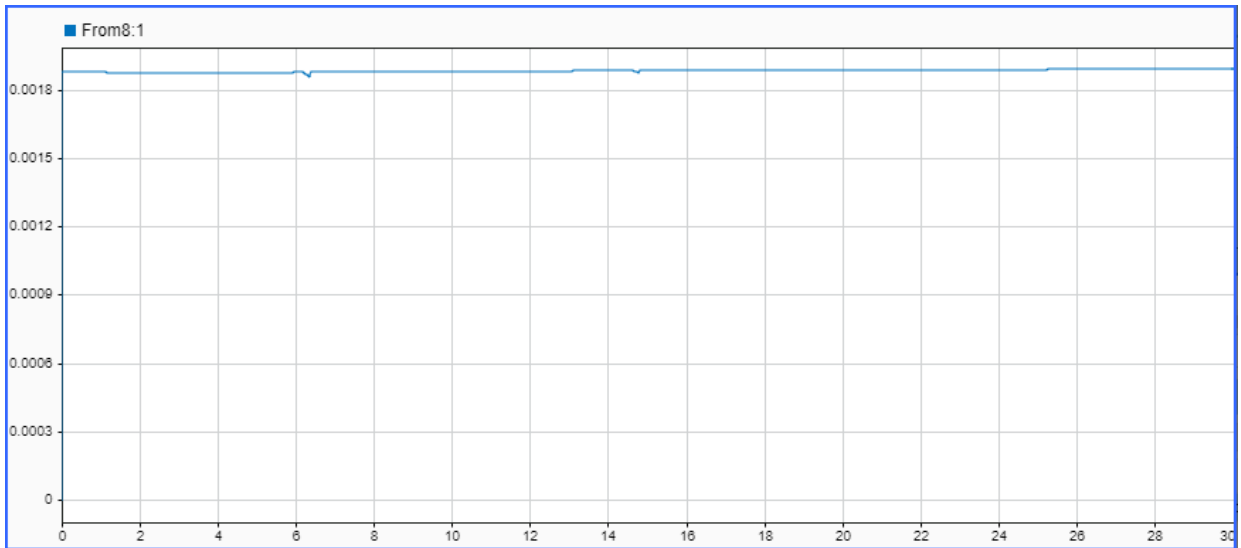
```

```

% Select Signal (From8:1) Check Box
baselineRunTireLFSlip.Checked = true;

%% Open Simulation Data Inspector
Simulink.sdi.view

```



- 4 Configure the local solver for fixed-step simulation using the Partitioning solver.
 - a In the model, open the Solver Configuration block settings.
 - b Select the **Use local solver** check box.

See Code

```

% Define the Solver Configuration Block and Path
solvConfig = 'Solver Configuration';
solvConfigPath = [model, '/', solvConfig];

% Configure the Solver Configuration Block
set_param(solvConfigPath, ...
    'UseLocalSolver', 'on', ...
    'DoFixedCost', 'on')

```

- 5 Simulate using the Partitioning solver.

See Code

```

%% Simulate the Model
sim(model)

```

The simulation runs to completion, but generates two initial condition warnings.

Warning: Initial conditions for eliminated differential variables not supported by partitioning solver. The following states may deviate from requested initial conditions:

```
['sdl_vehicle_dual_clutch/Vehicle/Tire LR'] Vehicle.Tire_LR.tire_inertia.w
  o In sdl.tires.tire_magic
['sdl_vehicle_dual_clutch/Vehicle/Tire RF'] Vehicle.Tire_RF.tire_inertia.w
  o In sdl.tires.tire_magic
['sdl_vehicle_dual_clutch/Vehicle/Tire RR'] Vehicle.Tire_RR.tire_inertia.w
  o In sdl.tires.tire_magic
```

Warning: Simscape succeeded in finding consistent states with which to start the simulation, but the states found may deviate from requested initial conditions.

- 6** Compare the baseline and Partitioning solver results in the Simulation Data Inspector.
 - a** Open the Simulation Data Inspector.
 - b** In the top, left pane, select **Compare**.
 - c** Configure the comparison. In the top, right pane:
 - i** On the right side of the **Baseline** setting, click the down arrow and select Run 1: `sdl_vehicle_clutch`.
 - ii** On the right side of the **Compare to** setting, click the down arrow and select Run 2: `sdl_vehicle_clutch`.
 - iii** Click **Compare**.
 - d** To change the Partitioning solver results line color, in the **Properties** pane, in the **Compare to** column, click the colored **Line**, select a different color, such as yellow, and then click **Set**.

See Code

```
% Compare Partitioning Solver Run 1 to Baseline Run
% Open Simulation Data Inspector
Simulink.sdi.view

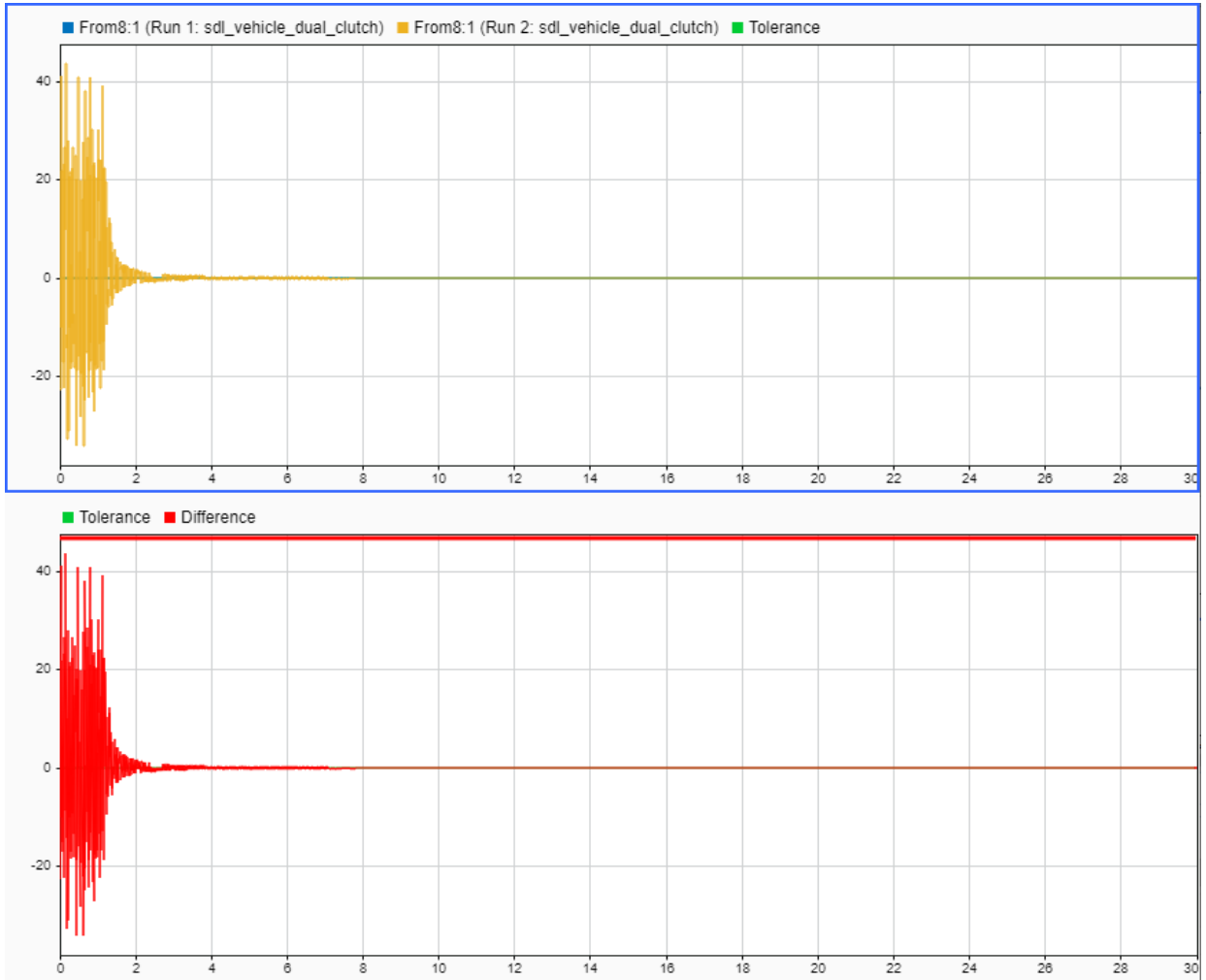
% Define Partitioning Solver Run 1
runIDs = Simulink.sdi.getAllRunIDs;
pSolverRun1ID = runIDs(end);
pSolverRun1 = Simulink.sdi.getRun(pSolverRun1ID);
psolverRun1TireLFSlip = pSolverRun1.getSignalByIndex(1);

% Select From8:1 Check Box
% psolverRun1TireLFSlip.Checked = true;

% Set Line Color
psolverRun1TireLFSlip.LineColor = [0.9294    0.6941    .1255];

% Compare Runs
```

```
compBaselinePartition1 = Simulink.sdi.compareRuns(baselineRunID,...
pSolverRun1ID);
```



The results from the Partitioning solver simulation contain significant chatter.

- 7 The signal chatter is due to stiffness that is related to inertia in the Tire (Magic Formula) blocks. The inertia is also the cause of the first initial condition warning. To resolve the signal chatter and the initial condition warning, simplify the tire dynamics:

- a** Open the Vehicle subsystem.
- b** Open the Tire LF block settings.
- c** In the **Dynamics** settings, set **Inertia** to **No Inertia**.
- d** Using the same process, omit the inertia for the Tire RF, Tire LR, and Tire RR blocks.

See Code

```

%% Resolve Chatter and First Initial Conditions Warning
% Simplify Tire Dynamics by Removing Inertia

% Define Vehicle Subsystem and Paths
vehicle = 'Vehicle';
vehiclePath = [model, '/', vehicle];

%% Define Tire Blocks and Paths and Omit Tire Inertia
%% Define Tire LF
tireLF = 'Tire LF';
tireLFPath = [vehiclePath, '/', tireLF];

%% Set Tire LF Settings Dynamics > Inertia to "No inertia"
set_param(tireLFPath, 'model_inertia', '0')

%% Omit Inertia from Tire RF, LR, RR Blocks

% Tire RF
tireRF = 'Tire RF';
tireRFPath = [vehiclePath, '/', tireRF];
set_param(tireRFPath, 'model_inertia', '0')

% Tire LR
tireLR = 'Tire LR';
tireLRPath = [vehiclePath, '/', tireLR];
set_param(tireLRPath, 'model_inertia', '0')

% Tire RR
tireRR = 'Tire RR';
tireRRPath = [vehiclePath, '/', tireRR];
set_param(tireRRPath, 'model_inertia', '0')

```

- 8** The second initial condition warning is due to the locked state of the Dog Clutch 1 block at the beginning of the simulation. The Partitioning solver cannot solve the dynamics for the initial torque that the locked clutch transmits between the ring and the hub. To resolve the warning:
 - a** Open the Dual Clutch Transmission subsystem.
 - b** Open the Gears subsystem.
 - c** Open the Dog Clutch 1 block settings. In the **Initial Conditions** settings, set **Clutch initial state** to **Unlocked**.

See Code

```
%% Resolve Second Initial Condition Warning
% Set Dog Clutch 1 Block Clutch Initial State to Unlocked

% Define Dual Clutch Transmission Subsystem and Path
dualClutchTrans = 'Dual Clutch Transmission';
dualClutchTranPath = [model, '/', dualClutchTrans];

%% Define Gears Subsystem and Path
gearsSS = 'Gears';
gearsSSPath = [dualClutchTranPath, '/', gearsSS];

%% Define Dog Clutch 1 Block and Path
dogClutch1 = 'Dog Clutch 1';
dogClutch1Path = [gearsSSPath, '/', dogClutch1];

% Select Dog Clutch 1 Block
set_param(dogClutch1Path, 'Selected', 'on')

%% Set Initial Conditions > Clutch initial state to "Unlocked"
set_param(dogClutch1Path, 'initial_state_engaged', '0')
```

9 Simulate and then examine and compare the results in the Simulation Data Inspector.

- a Run the simulation.

See Code

```
%% Simulate the Model
sim(model)
```

The simulation runs to completion, and does not generate any initial condition warnings.

- b Open the Simulation Data Inspector and compare the results from the Partitioning solver fixed-step simulation to the baseline results from the variable-step simulation. To configure the comparison, in the top, right pane:
 - i On the right side of the **Baseline** setting, click the down arrow and select Run 1: `sdl_vehicle_clutch`.
 - ii On the right side of the **Compare to** setting, click the down arrow and select Run 3: `sdl_vehicle_clutch`.
 - iii Click **Compare**.

See Code

```
%% Open the the Simulation Data Inspector
Simulink.sdi.view

%% Define the baseline run and select the From tag check box
```



```
runIDs = Simulink.sdi.getAllRunIDs;
pSolverRun2ID = runIDs(end);
pSolverRun2 = Simulink.sdi.getRun(pSolverRun1ID);
psolverRun2TireLFSlip = pSolverRun1.getSignalByIndex(1);
psolverRun2TireLFSlip.LineColor = [0.9294 0.6941 .1255];
psolverRun2TireLFSlip.Checked = true;

% Compare the results
compBaselinePartition2 = Simulink.sdi.compareRuns(baselineRunID,...
    pSolverRun2ID);
```



The results from the Partitioning solver simulation no longer contain significant chatter and are much more similar to the baseline results.

See Also

Solver Configuration

Related Examples

- “Increase Simulation Speed Using the Partitioning Solver” (Simscape)
- “Reduce Numerical Stiffness” (Simscape)
- “Reduce Fast Dynamics” (Simscape)
- “Reduce Zero Crossings” (Simscape)

More About

- “Variable Viewer” (Simscape)
- “Model Statistics Available when Using the Partitioning Solver” (Simscape)

Driveline Degrees of Freedom

In this section...

“About Driveline Degrees of Freedom and Constraints” on page 16-38

“Identify Degrees of Freedom” on page 16-38

“Define Fundamental Degrees of Freedom” on page 16-39

“Define Connected Degrees of Freedom” on page 16-41

“Define Constrained Degrees of Freedom” on page 16-42

“Actuate, Sense, and Terminate Degrees of Freedom” on page 16-46

“Count Independent Degrees of Freedom” on page 16-47

“Count Degrees of Freedom in a Simple Driveline with a Clutch” on page 16-48

About Driveline Degrees of Freedom and Constraints

Identifying rotational degrees of freedom (DoFs) is important for building and analyzing a driveline, particularly a complex system with many constraints and external actuations. Simulink represents driveline DoFs and other Simscape system variables as *states*, among all states of a model, including the pure Simulink states.

This section explains how to identify driveline DoFs, handle constraints, and extract the true or *independent* DoFs from a complete driveline diagram.

- The basic elements of a driveline diagram:
 - Connection lines
 - Constraints, including branchings
 - Dynamic elements
- Sensors and sources
 - Actuating drivelines with motion sources and recording motions with motion sensors
 - Terminating DoFs

Identify Degrees of Freedom

In a Simscape Driveline model, mechanical motions can be rotational or translational: motion around or along one axis. The simplest way to identify a driveline *degree of*

freedom (DoF) is from an angular or linear velocity. A DoF represents a single, distinct angular or linear velocity. Each DoF responds to the torques and forces acting on the inertias and masses making up the driveline. Integrating Newton's equations of motion determines the angular and linear motions. Mechanical DoFs are properties of rotating inertias and translating masses. It is nonetheless consistent and simpler to identify a single Simscape Driveline DoF as a driveline axis with its connected inertias and masses.

To identify and count DoFs in a driveline, look at a Simscape Driveline diagram starting with its mechanical connection lines first, before considering its blocks. Driveline blocks modify the DoFs represented by connection lines by:

- Generating torques and forces that act relatively between driveline axes
- Adding constraints between driveline axes
- Imposing externally actuated torques, forces, and motions


For the basic rules of connection lines and ports, see “Build a Drivetrain Model” on page 2-3.

Define Fundamental Degrees of Freedom

The basic unit of driveline motion is the DoF represented by an unbroken mechanical connection line. Such lines represent idealized massless and perfectly rigid driveline axes.

Represented by Inertia blocks, rotating bodies with inertias are rigidly attached to and rotate with their axes. Represented by Mass blocks, translating bodies with masses are rigidly attached to and translate along their axes. A single connection line or a set of branched connection lines represents either rotational or translational motion and must be connected to either rotational or translational ports.

Driveline Axes as Fundamental Degrees of Freedom — Mechanical Ports

A connection line anchored by physical network connector ports  represents an idealized driveline axis. The connection line enforces the constraint that the two connected driveline components rotate or translate at the same angular or linear velocity, respectively.



You measure the angular or linear velocity of an axis with an Ideal Rotational Motion Sensor or Ideal Translational Motion Sensor block.

Defining Relative and Absolute Angles and Positions

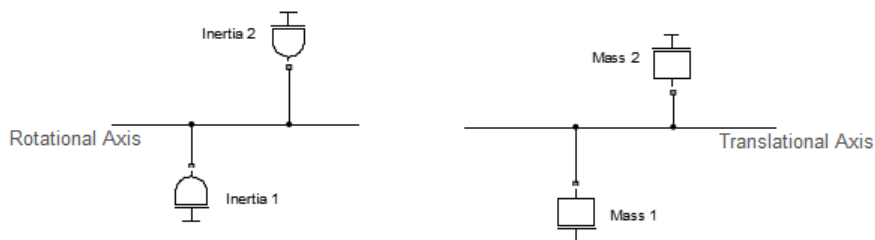
Relative angle or position is sometimes necessary to compute internally generated torques or forces between pairs of axes (see “Define Connected Degrees of Freedom” on page 16-41). To determine a relative angle or position, a motion sensor block integrates the relative angular or linear velocity of the pair of axes and adds the result to the initial relative angle or position specified in the block dialog box.

You can define an absolute rotation angle or translation position for a single axis when you measure its motion with a motion sensor, connecting the other physical connection port of the sensor to a Mechanical Rotational Reference or Mechanical Translational Reference. The sensor defines an absolute angle or position by integrating the velocity of the axis and adding the absolute reference angle or position that you provide in the motion sensor dialog box.

Rotating Inertias and Translating Masses Attached to Driveline Axes

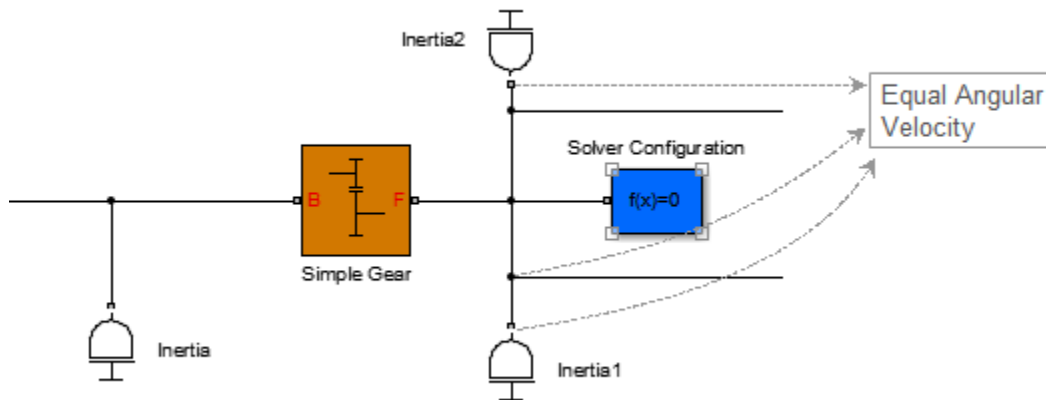
You cannot subject a driveline connection line, by itself, to any torques or forces, because it lacks inertia or mass. The other basic element to construct a functioning driveline model is one or more Inertia blocks, one or more Mass blocks, or both. In a real mechanical system, the spinning (or sliding) bodies carry both inertia (or mass) and DoFs.

You attach Inertias and Masses to mechanical connection lines by branching the lines. The attached inertias or masses are subject to whatever torque or force is transmitted by the connection line. The connection line imposes the constraint that everything attached to a single line must be spinning or sliding at the same velocity.



Driveline Axis Branching Rules and Constraints

You can branch connection lines. You can connect the end of any branch of a driveline connection line to a mechanical conserving connection port \square only. A set of unbroken, branched connection lines represents a single DoF.



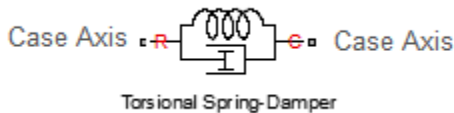
Branched Connection Lines and Angular Velocity Constraints

Define Connected Degrees of Freedom

You can connect two independent driveline axes, representing two independent degrees of freedom (DoFs), by an internal *dynamic element*. A dynamic element generates a torque or force from the relative angle, position, or motion of the two axes. This torque or force acts between the two axes, which remain independent DoFs, and which transmit the torque or force to their respective attached inertias or masses.

Dynamic Elements — Internal Torque and Force Generation

Apart from gears, most of the Simscape Driveline library blocks are dynamic elements, as are the mechanical rotational and translational blocks of the Simscape Foundation library. These blocks generate internal torques and forces. On a block with two mechanical conserving ports, a single torque or force is applied with positive sign to one axis and negative sign to the other axis. In this figure, torque is applied equally and oppositely to the rod and case axes of the Torsional Spring-Damper.



On blocks with more than two mechanical conserving ports, the total torques or forces flowing in and out of the block still sum to zero, but the torque or force is divided among the ports in a more complex way that depends on the driveline dynamics.

Clutch and Clutch-Like Elements – Conditional Connections

A clutch or clutch-like element is a *conditional* or *dynamic* constraint.

If unlocked, a clutch connects two driveline axes and can impose a relative torque between them, leaving the two axes independent. The unlocked clutch is either unengaged, imposing no torque at all; or engaged, imposing kinetic friction as a function of the relative velocity of the two connected axes.

If a clutch locks and applies only static friction between the two connected axes, the two axes are no longer independent. Instead, they act as a single axis, spinning at the same rate. See “Define Constrained Degrees of Freedom” on page 16-42.

Several other, clutch-like blocks also have locking and unlocking Coulomb friction:

- Torsional Spring-Damper
- Loaded-Contact Rotational Friction and Loaded-Contact Translational Friction

Define Constrained Degrees of Freedom

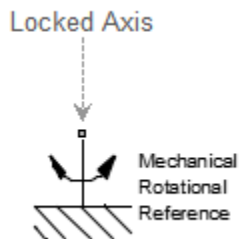
Certain driveline elements couple driveline axes in a way that eliminates their freedom to move independently. Such elements impose constraints on the motions of the connected axes. A constrained axis is no longer independent of other axes and does not count toward the total net or independent motions of the driveline. Such constraints remove independent degrees of freedom (DoFs) from the system.

Not all constraints are independent. Closing branched connection lines into loops makes some of the constraints within the loops redundant. The number of effective or independent constraints is the number of constraints arising from blocks minus the number of independent closed driveline connection line loops.

Except for clutches and clutch-like elements, driveline constraints are *unconditional* or *static* constraints; that is, unchanging over the simulation.

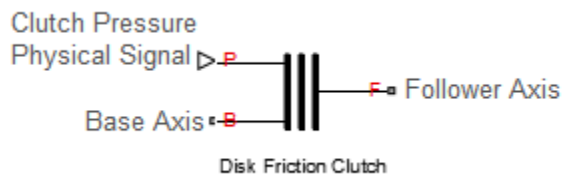
Locking a Driveline Axis

Connecting a driveline connection line to a Mechanical Rotational Reference or Mechanical Translational Reference block freezes the motion of the corresponding driveline axis. It cannot move, and its angular or linear velocity is constrained to be zero during a simulation. Such an axis has no associated independent DoF.



Locking Two Driveline Axes Together with a Clutch or Clutch-Like Element

As long as the conditions for locking are valid, a locked clutch or clutch-like element constrains the two connected driveline axes to spin or slide together. The two axes remain distinct, but only one represents an independent DoF. The other is dependent.

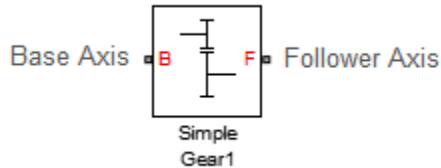


Even if it continues to apply kinetic friction between the axes, an unlocked clutch or clutch-like element no longer imposes a constraint. Instead, it acts as a dynamic element. See “Define Connected Degrees of Freedom” on page 16-41.

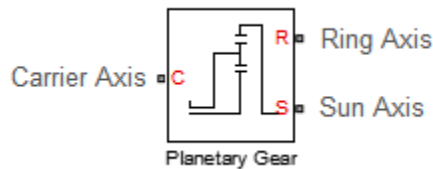
Coupling Driveline Axes with Gears

A gear coupling between two or more driveline axes reduces the independent DoFs of the driveline by imposing constraints. The nature of those constraints depends on the gear

that you use. Gear blocks with two connected axes impose one such constraint and reduce the two axes to a single independent DoF.



Multiaxis gears impose more than one constraint. For example, a planetary gear imposes two constraints on three axes, reducing the axes to one independent DoF. (This count does not include the fourth, internal DoF, the planetary wheel, which is not connected to an axis with a mechanical port.)



Closed Loops, Effective Constraints, and Constraint Consistency

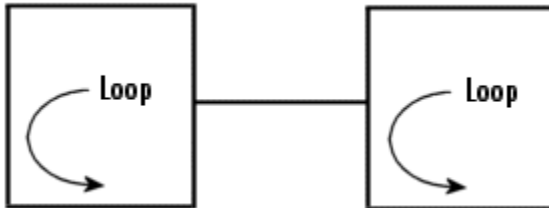
The actual constraint count to determine the number of DoFs is the number of effective or *independent* constraints. When connection lines form closed loops, take extra care in counting constraints in a driveline diagram. The presence of closed loops in a diagram reduces the effective constraint count by rendering some of the constraints redundant:

$$N_{\text{constr}} = N_{\text{bconstr}} - N_{\text{loop}}$$

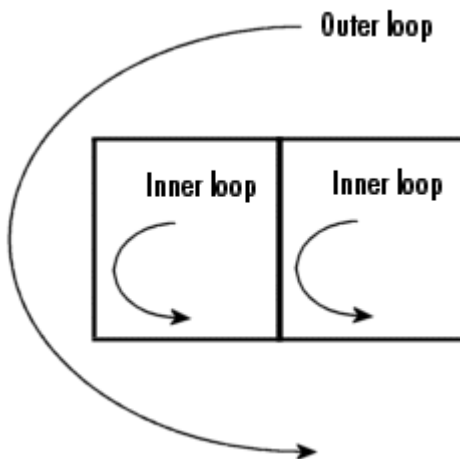
- N_{constr} Number of independent constraints
- N_{bconstr} Number of constraints from blocks
- N_{loop} Number of independent loops

You can reliably count the number of independent loops by counting the fundamental loops. Fundamental loops have no subloops. You can trace a fundamental loop with only one path. By counting only fundamental loops, you avoid overcounting loops that overlap.

For example, this diagram has two independent loops.



In this diagram, you can draw three loops: two inner loops, left and right, and the outer loop. The outer loop encompasses both inner loops.



There are two independent loops in this diagram, because only two are fundamental. The outer loop is not fundamental.

Consistency of Constraints

As long as all the velocities constrained by line branch points are equal over the whole loop, a closed loop renders redundant one of the constraints contained within it. (See "Driveline Axis Branching Rules and Constraints" on page 16-41.) The velocities not directly connected by lines must also be consistent if, for example, they are transferred through gears.

If the velocities along a closed loop cannot be made consistent, the driveline is overconstrained and cannot move.

Actuate, Sense, and Terminate Degrees of Freedom

You can use Simscape Driveline and related blocks with only one driveline connector port **□** to originate or terminate a physical connection line. Terminating a connection line limits the DoF.

Such blocks include:

- Inertia and Mass, which accept torque and force and respond with acceleration.
- Mechanical Rotational Reference and Mechanical Translational Reference, which ground DoFs to zero velocity.
- Vehicle Body, which implicitly connects the driveline to ground.

These blocks do not have to end a connection line, but can instead be branched off a connection line.

Directionality of Degrees of Freedom

Driveline connection lines have no inherent directionality. The direction of motion and torque flow is determined by the driveline dynamics when you simulate a model.

Effect of Torque and Force Actuation on Degrees of Freedom

Connecting an Ideal Torque Source or Ideal Force Source into a driveline connection line adds the torque or force specified by a physical signal input into that driveline axis. Such an actuation has no effect on the number of system DoFs. The driveline axes transmit torques and forces to their connected Inertias and Masses. The driveline is free to respond to these imposed torques or forces. The motion is simulated by integrating the driveline accelerations (a result of the imposed torques and forces) to obtain the driveline velocities.

Effect of Motion Actuation on Degrees of Freedom

Connecting an Ideal Angular Velocity Source or Ideal Translational Velocity Source to a driveline axis removes the freedom of that axis to respond to torques or forces. Instead, it specifies the axis motion during the simulation from the actuating physical signal input. Unlike torque actuation, motion actuation removes an independent DoF from the system.

For more information about driveline actuation with torques, forces, and motions, see “Driveline Actuation”.

Count Independent Degrees of Freedom

To determine the number of independent degrees of freedom (DoFs) in your driveline:

- 1 Count all the continuous, unbroken driveline connection lines (grouping connected sets of branched lines) in the Simscape Driveline portion of your model diagram. Call the total of such lines N_{CL} .

These lines connect two driveline connector ports or terminate on one mechanical connector port \blacksquare . For details, see “Define Fundamental Degrees of Freedom” on page 16-39 and “Actuate, Sense, and Terminate Degrees of Freedom” on page 16-46.

- 2 Count all the constraints arising from blocks that impose constraints on their connected driveline axes. Call the total of such constraints $N_{bconstr}$.

Usually, each such block imposes one constraint, but complex gears impose more than one. For details, see “Define Constrained Degrees of Freedom” on page 16-42.

- 3 Count the number of independent loops, N_{loop} . The effective number of constraints is $N_{constr} = N_{bconstr} - N_{loop}$. For details, see “Closed Loops, Effective Constraints, and Constraint Consistency” on page 16-44.

- 4 Count all the motion actuations in your driveline, by counting each motion source block. Call the total of such motion actuations N_{mact} . For details, see “Actuate, Sense, and Terminate Degrees of Freedom” on page 16-46.

The number N_{DoF} of independent DoFs in your driveline is:

$$N_{DoF} = N_{CL} - N_{constr} - N_{mact} = N_{CL} - [N_{bconstr} - N_{loop}] - N_{mact}$$

A necessary (although not sufficient) condition for driveline motion and successful driveline simulation is that N_{DoF} is positive. Count rotational and translational DoFs separately.

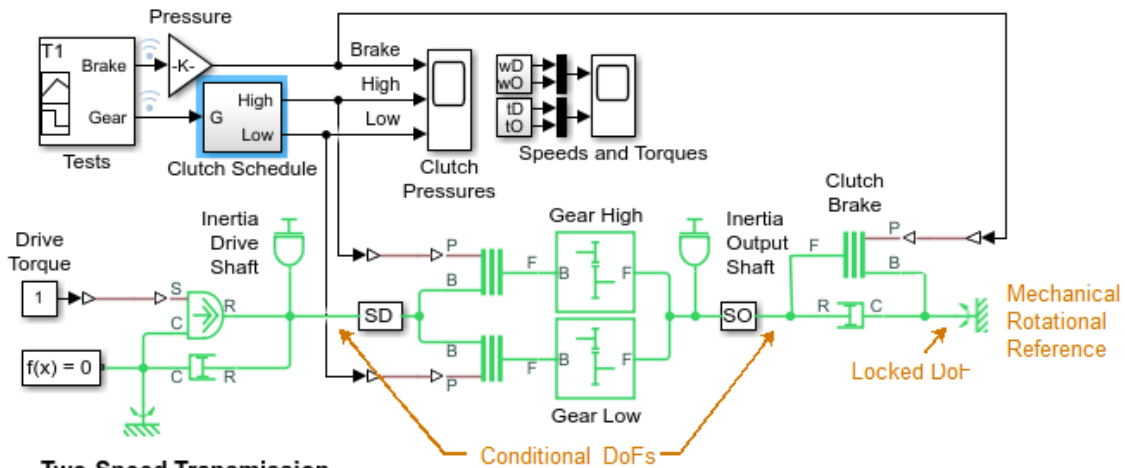
Conditional Degrees of Freedom with Clutches and Clutch-Like Elements

Unlike other driveline components, clutches, and clutch-like elements can undergo a discontinuous state change during a simulation. In general, the number of independent DoFs of a driveline is not constant during its motion. Each state change of one or more clutches changes the independent DoF count. Taken as a whole, different collective states

of the clutches of a driveline can have different total net DoFs. To understand a driveline completely, examine each possible collective state of its clutch states to identify its independent DoFs and possibly invalid configurations.

Count Degrees of Freedom in a Simple Driveline with a Clutch

Consider the two-speed transmission model `sd1_transmission_2spd`.



Two-Speed Transmission

1. [Plot shaft speeds](#) ([see code](#))
2. [Explore simulation results](#) using `sscexplore`
3. [Learn more](#) about this example

Simple Transmission

This system has five apparent DoFs, represented by these driveline axes:

- Branched axis with the Inertia Drive Shaft block
- Branched axis with the Inertia Output Shaft block
- Axis connecting the high gear clutch block (the clutch for the high clutch schedule) to the Gear High block
- Axis connecting the low gear clutch block (the clutch for the low clutch schedule) to Gear Low block

- Axis connecting the Clutch Brake block to the Mechanical Rotational Reference block (rotational ground)

There is an apparent closed loop formed by the gear blocks and gear clutch blocks. This loop is real only if both gear clutch blocks are locked.

The actual number of independent DoFs depends on the state of the clutches. The model has no motion sources, so we need consider only gears and clutches as constraints:

- The two gears blocks are always acting, therefore yielding two ever-present constraints.
- The fifth axis is always connected to the housing (rotational ground).

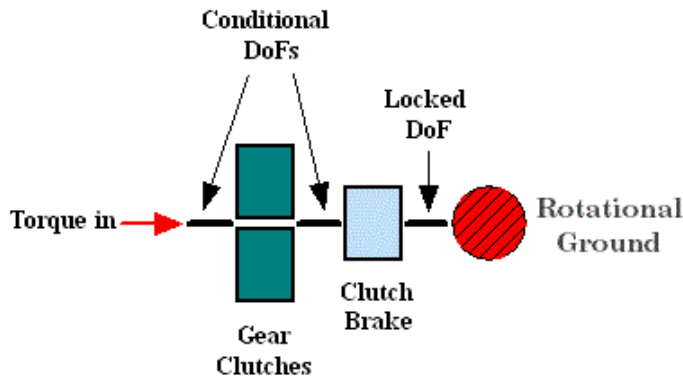
These three constraints reduce five DoFs to two DoFs.

Now consider the clutches.

- Consider first the case where the Clutch Brake block is disabled (free).
 - If both the high gear clutch and low gear clutch blocks are unlocked, the system has two independent DoFs, one on the left of the gear clutch blocks and the other between the gear clutch blocks and the Clutch Brake block.
 - If one of these gear clutch blocks is locked, the additional constraint reduces the system to one independent DoF, everything to the left of the Clutch Brake block. (The clutch control schedule is set up to prevent both of these clutch blocks from being locked at the same time.)
- If the Clutch Brake block is enabled, the clutch control schedule keeps the two gear clutch blocks disabled.
 - If the Clutch Brake block is unlocked, the driveline has two independent DoFs: to the left of the gear clutch blocks and between the gear clutch blocks and the Clutch Brake block.
 - If the Clutch Brake block is locked, the system is reduced to one DoF, to the left of the gear clutch blocks. Everything to the right of the gear clutch blocks is locked to the housing.

This table and abstract diagram summarize the possibilities available in this model.

Brake Enabling	Clutch Locking	Independent DoFs
Brake disabled	Both gear clutch blocks unlocked	Two: On the left and on the right of the gear clutch blocks
	One gear clutch block locked	One: On the left of the Clutch Brake block
Brake enabled	Clutch Brake block unlocked	Two: On the left and on the right of the gear clutch blocks
	Clutch Brake block locked	One: On the left of the gear clutch blocks



Degrees of Freedom in the Simple Transmission

Nonphysical Configurations

The clutch schedule design implemented in the Clutch Schedule subsystem excludes nonphysical configurations. It is worth considering them anyway, for the sake of a complete understanding of driveline design. For more information about clutch issues, see “Troubleshoot Driveline Modeling and Simulation Issues” on page 18-2 and “Modeling Transmissions” on page 9-7.

Both Gear Clutches Locked, Clutch Brake Unlocked

This configuration creates a conflict of DoFs and reduces the independent DoFs to one. The driveline axis to the right of the gear clutch blocks tries to spin at two different rates, as required by two different gear ratios. Two locked clutches enforce two additional constraints on the two remaining DoFs, but form a closed loop, nominally leaving one

freedom in the mechanism. Because of the DoF conflict, attempting to simulate such a configuration leads to a Simscape Driveline error.

If the two Gears had identical gear ratios, the DoFs would not conflict, and the simulation would run without error.

One Gear Clutch Locked, Clutch Brake Locked

This configuration also creates a conflict of DoFs and yields zero DoFs. The two locked clutches enforce two additional constraints on the two remaining DoFs and leave no freedom in the mechanism. Driven by the driveline axis to the left, the driveline axis between the gear clutch blocks tries to spin but finds itself locked to the Mechanical Rotational Reference. Attempting to simulate such a configuration leads to a Simscape Driveline error.

Both Gear Clutches Locked, Clutch Brake Locked

This configuration is also overconstrained. Three locked clutches enforce two effective constraints on the remaining two DoFs (after taking into account the closed loop) and yield $N_{\text{DoF}} = 0$. In addition, the driveline axis to the right of the gear clutch blocks tries to spin at two different nonzero rates, while remaining locked to the Mechanical Rotational Reference, creating two distinct DoF conflicts.

Driveline States — Effect of Clutches

In this section...
“Driveline States and Degrees of Freedom” on page 16-52
“Find and Use Driveline States” on page 16-53

Driveline States and Degrees of Freedom

It is best to have some familiarity with advanced Simulink modeling techniques before using this section. For more information on driveline degrees of freedom, see “Driveline Degrees of Freedom” on page 16-38.

Simulink and Simscape represent driveline degrees of freedom (DoFs) and other information about the dynamics of a model with *states*. The driveline states are a subset of the total states of the model. Although the number of independent driveline states in a model is equal to the number of independent DoFs (with all clutches unlocked), the driveline states in general are linear combinations of the velocities, not the velocities of particular driveline axes. Before you simulate a model, this DoF-to-state transformation is not known.

You can extract state and model output data from your simulation. In the **Model Configuration Parameters** dialog box, select the appropriate check boxes in the **Data Import/Export** pane. The default state and output vectors are `xout` and `yout`, respectively.

Discontinuous Clutch State Changes

In part, the overall state of the driveline is the set of all its clutch states. Because clutches are dynamic constraints, the nature of the driveline states in a model with clutches and clutch-like elements can change during simulation. When a clutch locks, two independent driveline states become dependent on one another.

For software to design and analyze transitions among discontinuous states such as those found in clutches and transmissions, see .

Inverse Dynamics

State information is also useful for analyzing the *inverse dynamics* of a driveline. Often, you apply torques and forces to a driveline in *forward dynamics* and then determine the

motions. Inverse dynamics means specifying motions to determine what torques and forces produce those motions.

If you motion-actuate some parts of your driveline instead, those axes and the equivalent states are no longer independent. If you want outputs from these axes, measure the torques and forces flowing along them. Knowing these torques and forces is the starting point of inverse dynamic analysis.

Find and Use Driveline States

This section explains how you locate and use Simscape Driveline states.

Locating Driveline States in Simulink

Your driveline model consists of a mixture of Simscape Driveline, Simscape, and ordinary Simulink blocks. In general, a model has Simulink states associated with the Simulink blocks. The Simscape Driveline and Simscape states of a single driveline system are associated with the Solver Configuration block of that driveline.

You can list all model states with the Simulink `Simulink.BlockDiagram.getInitialState` method:

- 1 Open a model. In this example, use `sdl_gear` as an example.
- 2 At the command line, enter:

```
sigt = Simulink.BlockDiagram.getInitialState('sdl_gear');  
sigt.time  
sigt.signals
```

The `Simulink.BlockDiagram.getInitialState` method initializes the model at zero time and captures the model states within the `.signals` structure. This list is the total set of states, not just the independent states. The Simscape and driveline states are a subset of the total states.

Trimming and Linearization — Clutch States

An important part of analyzing a driveline system is finding stable steady states of motion and understanding how the driveline responds to small changes in inputs, such as changes to initial conditions or to the applied forces and torques. Trimming and linearization are the formal steps of such an analysis.

If you implement clutch state changes in your simulation, trimming requires that you start by specifying which clutches are locked and unlocked. The trimming procedure then

determines the state of continuous motion. During linearization, simulation starts with the clutch states that you specify and iterates to find a consistent state of all clutches. It then implements the perturbation of continuous states, holding the clutch states fixed.

For more information about trimming and linearizing Simscape models, see “Finding an Operating Point” (Simscape) and “Linearizing at an Operating Point” (Simscape).

How Simscape Driveline Simulates a Drivetrain System

In this section...

“About Simscape Driveline and Simscape Simulation” on page 16-55

“Clutch State Determination” on page 16-55

About Simscape Driveline and Simscape Simulation

Apart from clutches and clutch-like elements, Simscape Driveline simulation is a special case of Simscape simulation.

- For information on clutches, degrees of freedom, and states, see “Driveline Degrees of Freedom” on page 16-38 and “Driveline States — Effect of Clutches” on page 16-52.
- For information on fixing simulation errors, see “Troubleshoot Driveline Modeling and Simulation Issues” on page 18-2.
- On how Simscape models work, see:
 - “How Simscape Models Represent Physical Systems” (Simscape)
 - “How Simscape Simulation Works” (Simscape)

Clutch State Determination

During simulation, Simscape Driveline software checks the clutch and clutch-like blocks in your model for locking and unlocking events. If a locked clutch meets the criteria for unlocking, or an unlocked clutch the criteria for locking, the respective clutch states change.

If one or more clutch and clutch-like constraints change, the driveline states are repartitioned into new sets of dependent and independent states. The repartitioning requires a partial reinitialization of the driveline that preserves the state of the driveline before the clutch changes, except for the subset of constraints and states affected by the clutch transitions.

Model Thermal Losses in Driveline Components

In this section...

“Thermal Ports” on page 16-56

“Thermal-Modeling Parameters” on page 16-57

“Model Thermal Losses for a Simple Gear” on page 16-57

Thermal modeling provides data that helps you to design efficiency and thermal protection into your system. Certain blocks in the Simscape Driveline Brakes & Detents, Clutches, and Gears libraries have thermal variants that allow you to determine how heat generation affects the efficiency and temperature of driveline components. For example, the Simple Gear block, which models a gear of base and follower wheels, has a thermal variant that can simulate the heat generated by meshing losses. Selecting a thermal variant for a block adds a thermal port to the block and enables the associated thermal-modeling parameters.

Thermal Ports

Thermal ports are physical conserving ports in the Simscape thermal domain. You can model thermal effects like heat exchange and insulation by connecting blocks, from other Simscape products, that use the thermal domain to the thermal ports on Simscape Driveline thermal variants.

Thermal ports are associated with temperature and heat flow which are the Across and Through variables of the Simscape thermal domain. To measure thermal variables, you can use one or both of these methods:

- 1 Log simulation data using a Simscape logging node. View the data using the `sscexplore` function.
- 2 Add a sensor from the **Simscape > Foundation Library > Thermal > Thermal Sensors** library to your model. To measure temperature, use a parallel-connected Ideal Temperature Sensor block. To measure heat flow, use a series-connected Ideal Heat Flow Sensor block.

There are several advantages to using data logging for desktop simulation. Data logging is less computationally costly than using a gauge block and it allows you to:

- View post-simulation results easily using the Simscape Results Explorer.

- Output data easily to the MATLAB Workspace for post-processing analysis.

However, if you use only data logging to measure a variable, you cannot output a feedback signal for that variable to a control system during simulation as you can when you use only a sensor to measure the variable. Also, because data logging is not supported for code generation, you cannot use Simscape data logging when you perform real-time simulation on target hardware.

Thermal-Modeling Parameters

Thermal-modeling parameters are device-specific characteristics that determine how thermal dynamics affect device temperature and performance during simulation.

For some blocks, the default variant includes requisite parameters for simulating thermal dynamics. For such blocks, parameter dimensions change when you select a thermal variant. For example, to parameterize meshing losses based on a constant efficiency friction model for the default variant of the Simple Gear block, you specify the **Efficiency** parameter using a scalar value. If you select a thermal variant for the Simple Gear block, you must use a vector quantity to specify the **Efficiency** parameter.

Selecting a thermal variant enables additional thermal-modeling parameters. For example, selecting the thermal variant of the Simple Gear block enables the **Temperature** parameter. To determine the extent of thermal losses, the block performs a table lookup based on the values that specified for the **Efficiency** and **Temperature** parameters.

Model Thermal Losses for a Simple Gear

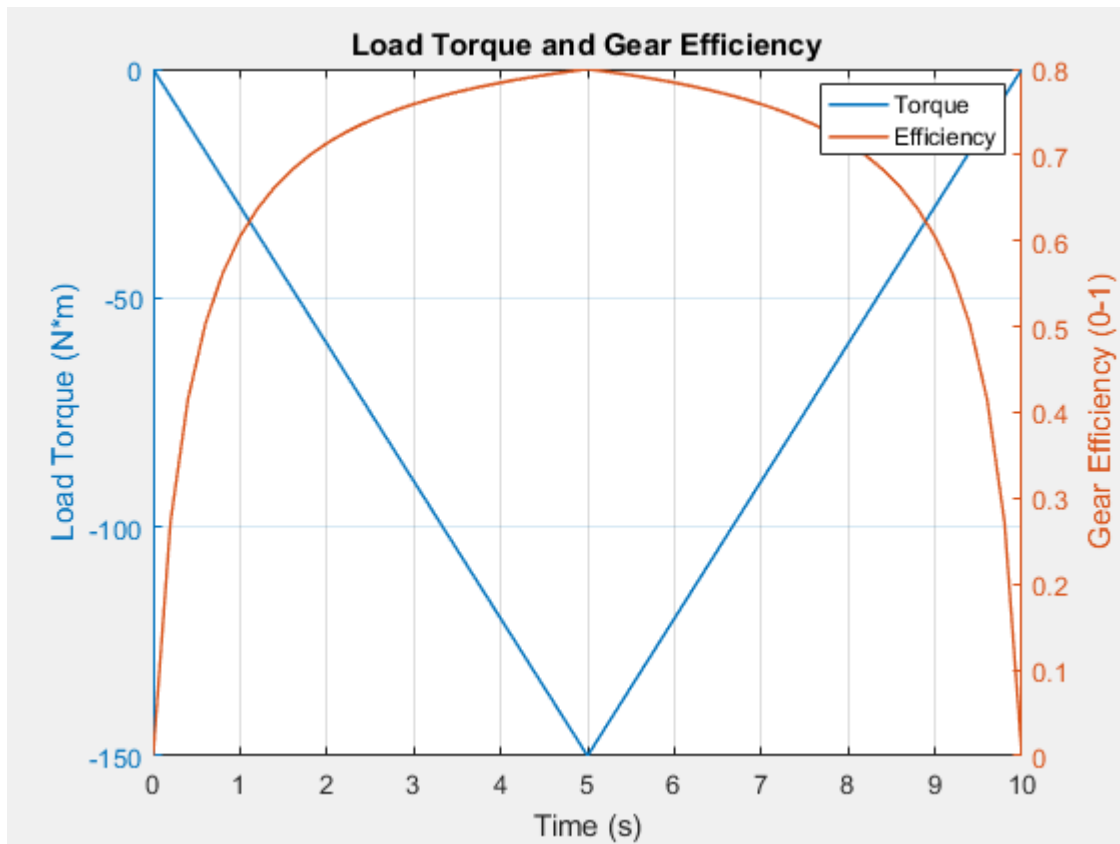
This example shows how to enable a thermal port, parameterize a thermal variant, and analyze the results of a simulation that models thermal losses.

Measure Load-Dependent Efficiency

- 1 Open the model. At the MATLAB command prompt, enter
`hdl_gear_efficiency`
- 2 Examine the parameters for the **Gear** block.

For **Meshing Losses**, the **Friction model** is set to Load-dependent efficiency. The **Nominal output torque** is 150 N*m and the **Efficiency at nominal output torque** is 0.8.

- 3 To simulate the model and plot gearbox efficiency, in the model window, click **Plot efficiency**.



The efficiency at the nominal point exactly matches the parameter values in the block. However, the efficiency is dependent only on the torque. Temperature does not factor into the efficiency calculation.

Use a Thermal Variant for the Gear Block

To include temperature as a factor in the efficiency calculation, select a thermal variant for the gear block.

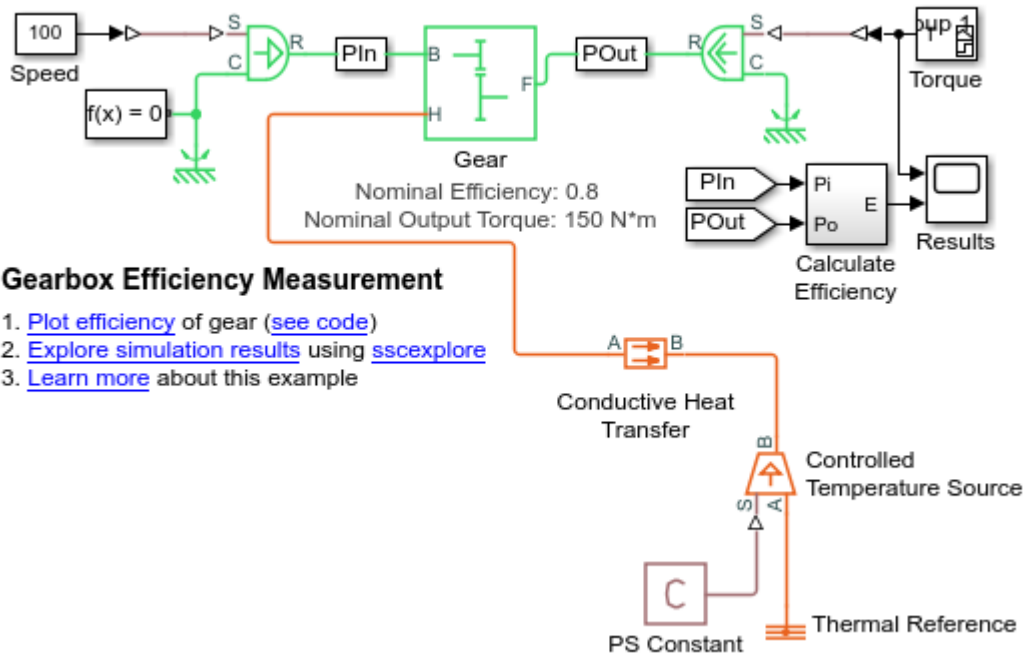
- 1 Right-click the **Gear** block and, from the context menu, select **Simscape > Block choices**. Select **Show thermal port**.

- 2 Parameterize the thermal variant. If the block dialog box is open, close and then open it to make the thermal parameters visible. For the **Meshing Losses** parameters:
 - a Set **Friction Model** to Temperature and load-dependent efficiency.
 - b For **Temperature**, specify [280 400 500].
 - c For **Efficiency matrix**, specify [0.65 0.65 0.7; 0.7 0.7 0.75; 0.75 0.75 0.8].
- 3 For the **Thermal Port > Initial Temperature** parameter, specify 320.

Add Thermal Library Blocks

To model heat transfer, add blocks from the Simscape Foundation Thermal library.

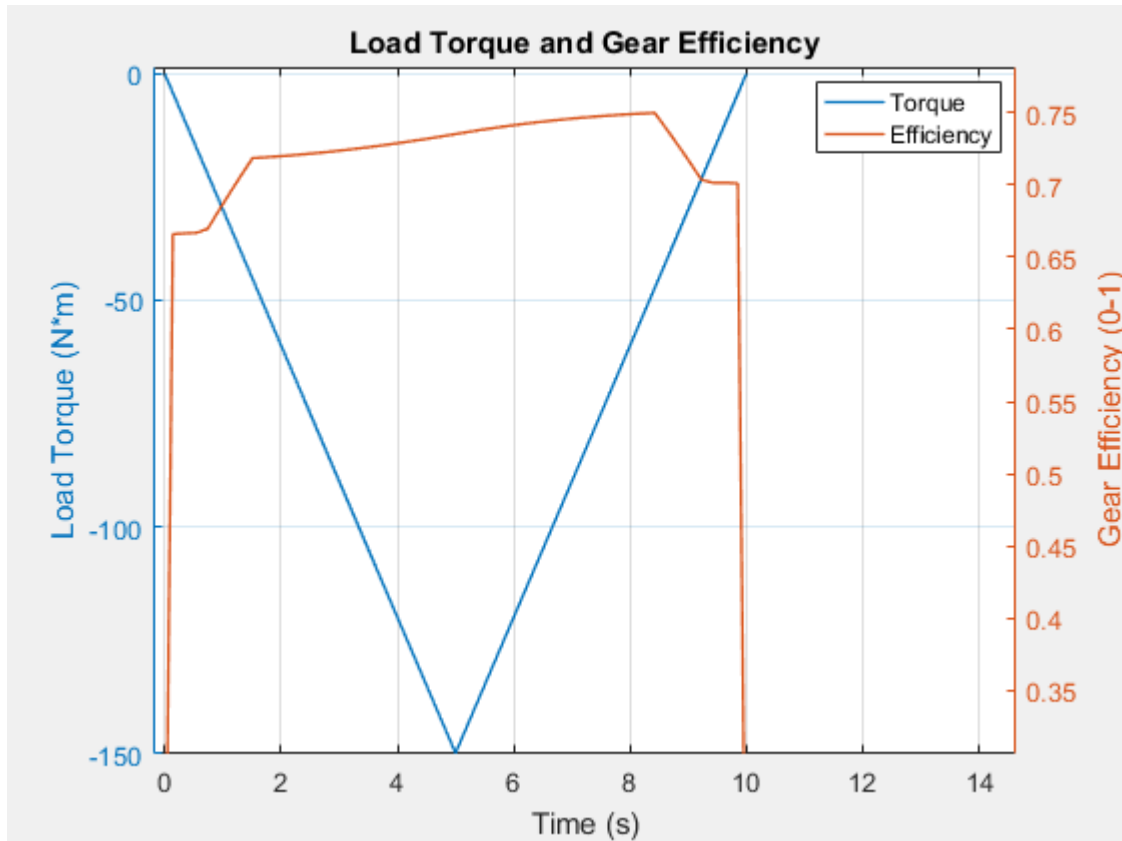
- 1 Add a block that represents heat flow between the gear and the environment. Open the Simulink Library Browser. From the **Simscape > Foundation Library > Thermal > Thermal Elements** library, add a Conductive Heat Transfer block to the model.
- 2 Add a block that represents a thermal reference point. Also from the **Thermal Elements** library, add a Thermal Reference block to the model.
- 3 Add blocks for modeling the ambient temperature as a constant, ideal source of thermal energy.
 - From the **Simscape > Foundation Library > Thermal > Thermal Sources** library, add an Controlled Temperature Source block.
 - From the **Simscape > Foundation Library > Physical Signals > Sources** library, add a PS Constant block. Specify a value of 320 for the PS Constant block.
- 4 Arrange and connect the blocks as shown in the figure.



Measure Temperature and Load-Dependent Efficiency

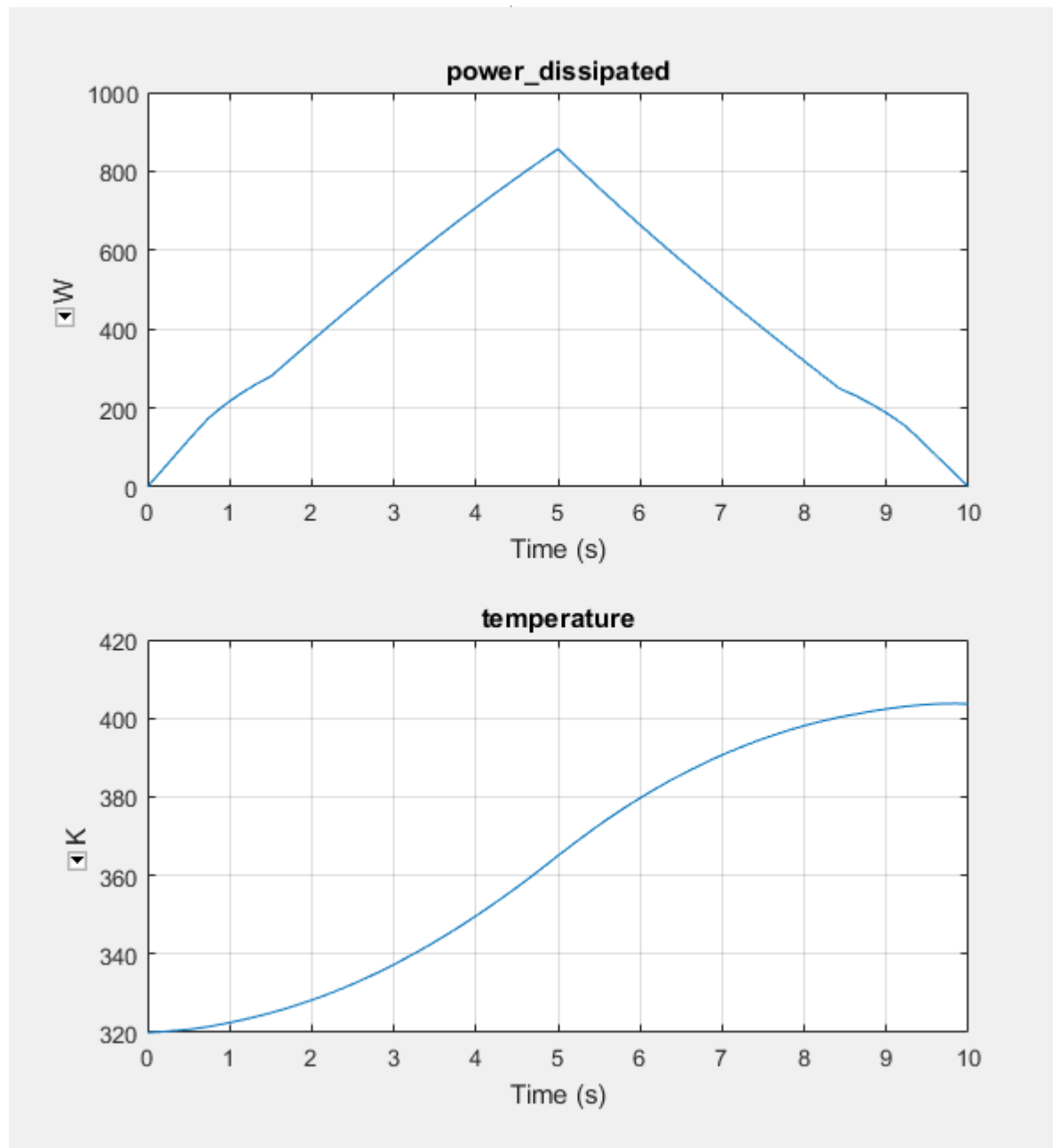
Evaluate efficiency as a function of both the load torque and the gear temperature.

- 1 Simulate the model.
- 2 To plot gearbox efficiency, in the model window, click **Plot efficiency**. Zoom out for a better view of the efficiency curve.



The efficiency peaks at 8.5 seconds when the magnitude of the load torque is approximately 33% of its maximal value. The efficiency is no longer dependent only on the load torque.

- 3 To view the data for the dissipated power and for gear temperature:
 - a In the model window, click **Explore simulation results**.
 - b In the node tree window, expand the **Gear > simple_gear_model** nodes.
 - c CTRL + click the **power_dissipated** and **temperature** nodes.



As the magnitudes of the torque load and temperature increase during the first half of the simulation, so does the amount of power that is dissipated. The amount of power that is dissipated decreases in the second half of the simulation due to the decrease in the torque load. However, the temperature of the gear continues to rise, as does the efficiency due to the vector specified for the **Temperature** parameter. The efficiency depends on both the load torque and the gear temperature.

See Also

Simple Gear

More About

- “About Simulation Data Logging” (Simscape)

Simscape Driveline Limitations

In this section...
“Simscape Driveline and Simulink Limitations” on page 16-64
“Additional Simscape Driveline Limitations” on page 16-64

Simscape Driveline and Simulink Limitations

Simscape Driveline software shares the limitations of Simscape software concerning Simulink features and tools. For Simscape limitations, see “Limitations” (Simscape).

Additional Simscape Driveline Limitations

Simscape Driveline software also has additional limitations of its own.

Index-2 Differential-Algebraic Equations from Variable Ratio Transmission

If you use the Variable Ratio Transmission block in your model, you might make your simulation slower or less accurate, depending on where the variable ratio comes from. A physical signal input defines the variable ratio as a function of time.

- If this time function is defined autonomously, independently of the dynamics of the physical system, the variable ratio is not a simulation problem.
- If this time function is defined in terms of feedback from the physical system itself, the variable ratio makes the physical-mathematical model into an *Index-2 differential-algebraic equation* (DAE) system. Its solution requires two differentiations of constraints and results in simulation warnings or errors.

To avoid this problem, do one of the following:

- Remove the Variable Ratio Transmission blocks that use physical system feedback to define their variable ratios.
- Delay the feedback signal, so that the feedback is no longer instantaneous.

Real-Time Simulation

Prepare Simscape Driveline Models for Real-Time Simulation Using Simscape Checks

If you have a Simulink Real-Time license, you can optimize your model for real-time execution using the `Execute real-time application` activity mode in the Simulink Performance Advisor. This mode includes several checks specific to physical models. For example, the Simulink Performance Advisor identifies Simscape Solver Configuration blocks with settings that are suboptimal for real-time simulation. For optimal results, Solver Configuration blocks should have the **Use local solver** and **Use fixed-cost runtime consistency iterations** options selected.

The checks are organized into folders. You can use the checks in the **Simscape checks** folder for all physical models. Subfolders contain checks that target blocks from Simscape Driveline and other add-on products such as Simscape Electrical.

Before you run the checks, use the processes described in “Real-Time Model Preparation Workflow” (Simscape), “Real-Time Simulation Workflow” (Simscape), and “Hardware-In-The-Loop Simulation Workflow” (Simscape).

To run the Simulink Real-Time Performance Advisor Checks:

- 1 In the Simulink Editor menu bar, select **Analysis > Performance Tools > Performance Advisor**.
- 2 In the Performance Advisor window, under **Activity**, select `Execute real-time application`.
- 3 In the left pane, expand the **Real-Time** folder, and then the **Simscape checks** folder.
- 4 Run the top-level Simscape checks and the Simscape Driveline checks. If your model contains blocks from other add-on products, also run the checks in the subfolder corresponding to that product.

See Also

More About

- “Model Preparation Objectives” (Simscape)
- “Real-Time Model Preparation Workflow” (Simscape)

- “Real-Time Simulation Workflow” (Simscape)
- “Use Performance Advisor to Improve Simulation Efficiency” (Simulink)

Troubleshoot Driveline Simulation Issues

- “Troubleshoot Driveline Modeling and Simulation Issues” on page 18-2
- “Troubleshoot Overconstrained and Conflicting Degrees of Freedom” on page 18-3
- “Troubleshoot Clutch and Transmission Errors” on page 18-4
- “Troubleshoot Inconsistent Initial Conditions” on page 18-5
- “Troubleshoot Pulley Network Issues” on page 18-6
- “Troubleshoot Engine Issues” on page 18-7

Troubleshoot Driveline Modeling and Simulation Issues

Various errors can cause your Simscape Driveline simulation to stop before completion. Some of these errors arise from nonphysical configurations of the driveline. To learn how to correct such issues, see:

- “Troubleshoot Overconstrained and Conflicting Degrees of Freedom” on page 18-3
- “Troubleshoot Clutch and Transmission Errors” on page 18-4
- “Troubleshoot Inconsistent Initial Conditions” on page 18-5
- “Troubleshoot Pulley Network Issues” on page 18-6
- “Troubleshoot Engine Issues” on page 18-7

Troubleshoot Overconstrained and Conflicting Degrees of Freedom

Analyzing and counting the driveline degrees of freedom (DoFs) are essential to fixing one type of simulation error. For more information, see “Driveline Degrees of Freedom” on page 16-38.

To run successfully, your driveline simulation must have a positive number of independent DoFs, from the start to the end of the simulation. Furthermore, the model DoFs must not conflict with each other.

If you encounter a simulation error where the driveline cannot move, check whether the number N_{DoF} of independent DoFs is positive, and whether the DoFs do not conflict with each other.

Checking the Number of DoFs

If N_{DoF} is not positive:

- Remove one or more constraining blocks, such as Gears, Clutches or clutch-like elements, and Mechanical Rotational References.
- Remove one or more Ideal Angular Velocity Source blocks.

Try one or both of these steps repeatedly until you locate the origin or origins of the simulation failure and make N_{DoF} positive.

Checking the Consistency of DoFs

Consider also whether two or more DoFs are in conflict. For example, check whether two velocity sources are trying to move a single DoF in two different ways. Such a configuration creates a motion conflict and leads to a simulation error.

Troubleshoot Clutch and Transmission Errors

Faulty clutch and transmission configurations generate many driveline motion failures and usually arise from DoF conflicts and errors. Clutches impose *conditional* or *dynamic* constraints.

To avoid or solve such problems, pay close attention to the collective state of your clutches, including clutches occurring inside transmission subsystems. The key to avoiding errors with transmissions is to work out and implement a complete and consistent clutch schedule.

Common mistakes include:

- Locking too many clutches simultaneously, leading to redundant dynamic constraints and overconstrained (not enough) DoFs.
- Locking conflicts among clutches, leading to nonredundant but still conflicting constraints.

Example: Locking one clutch locks one driveline axis to another. You could also lock the first driveline axis simultaneously to a third axis with another clutch. If the second and third axes cannot turn at the same velocity, these DoFs are in conflict.

- Locking too few clutches simultaneously. This error does not overconstrain DoFs or put them in conflict. However, it puts a transmission into a neutral state where it cannot transmit any torque or motion.

For information about adjusting simulation for clutches, see “Optimize Simulation of Clutches” on page 16-4 and “Transmissions with Gear Ratios and Clutch Schedules”.

Troubleshoot Inconsistent Initial Conditions

Like motion sources, initial conditions can cause motion conflicts. Unlike motion sources, they do not impose constraints or remove DoFs from the driveline, because they act only at the start of the simulation. However, under certain circumstances, initial conditions can cause errors when you start the simulation:

- Initial conditions conflict with one another.

Example: You couple two driveline axes through a Gear with a gear ratio of 2. The base axis must spin twice as fast as the follower, in the same direction. If you actuate the base with a velocity source, and the follower is connected to an inertia with initial velocity not set to half the initial base velocity, the simulation stops with an error.

- Initial conditions conflict with motion sources. When the simulation starts, the signal controlling a velocity source acting on an axis and the initial velocity value specified on an Inertia or a Mass attached to that axis must agree. Analogous requirements hold for velocities transformed by gear couplings.

Regardless of how you set the initial conditions of your driveline axes, the complete set of initial conditions must be consistent with itself. Driveline connection lines satisfying angular velocity constraints (for example, branched lines, or lines in closed loops) must have the same initial angular velocities.

Troubleshoot Pulley Network Issues

Like real-world pulleys, Simscape Driveline pulley blocks rely on belt tension and inertia for motion. To fix a pulley network in your model that is not acting as expected:

- Make sure that there are no conflicts in the settings for the belt direction. For more information, see “Best Practices for Modeling Pulley Networks” on page 6-2.
- Ensure that there is inertia in the system. If there is no inertia, add it by including an Inertia block from the Simscape Rotational Elements library, by adding inertia to a downstream component, or including inertia in one of the pulleys. As needed, specify an initial velocity for the inertia.
- Compare the number of pulley pairs to the number of tensioners. If the number of tensioners is not equal to either the number of pulley pairs or the number of pulley pairs less one, add tensioners. For example, if there are five pulley pairs, include at least four tensioners. Use spring and damper blocks to model the tensioners.

See Also

Belt Drive | Belt Pulley | Inertia | Inertia | Rope Drum | Rotational Free End | Translational Spring | Variable Inertia | Worm Gear

Related Examples

- “Power Window System”

More About

- “Best Practices for Modeling Pulley Networks” on page 6-2

Troubleshoot Engine Issues

Like real-world engines, blocks from the Simscape Driveline Engines library rely on the inertia from each cycle to initiate the next cycle. If a piston- or engine-driven network in your model is not responding to throttle input, either at the beginning of simulation or when the engine reaches stall speed, examine the engine output for the simulation. If there is no engine velocity in response to throttle input, try these engine-startup methods:

- Add initial velocity to the engine block — Specify a nonzero value for the initial velocity or crank velocity parameter in the engine block settings. Specify a value that is well above stall speed. Iterate to find the correct solution.
- Add an inertia with initial velocity to the engine network — Add an Inertia block from the Simscape Rotational Elements library or add inertia to a downstream component, for example a shaft. Specify an initial velocity using the Variables settings for the Inertia block or downstream component. Set the target velocity to **High Priority**.
- Add an electric starter motor — Use a starter motor, such as the DC motor in the “Permanent Magnet DC Motor” (Simscape) example, to initiate engine motion.

If a model that is not giving expected results contains a Piston block with the **Pressure parameterization** set to **By crank angle and throttle** and a **Pressure matrix (gauge)** that indicates zero velocity, include an external inertia with initial angular velocity.

See Also

Generic Engine | Inertia | Piston | Piston Engine

Related Examples

- “Permanent Magnet DC Motor” (Simscape)

More About

- “Troubleshoot Inconsistent Initial Conditions” on page 18-5

