Simulink[®] Design Optimization[™] User's Guide

MATLAB&SIMULINK®



R

R2021**b**

How to Contact MathWorks



Latest news:

Phone:

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



 \searrow

508-647-7000

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

Simulink[®] *Design Optimization*[™] *User's Guide*

© COPYRIGHT 1993-2021 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

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

Revision History

March 2009 September 2009 March 2010 September 2010 April 2011 September 2011 March 2012 September 2012 March 2013 September 2013 March 2014 October 2014 March 2015 September 2015 March 2016 September 2016 March 2017 September 2017 March 2018 September 2018 March 2019 September 2019 March 2020 September 2020 March 2021 September 2021

Online only Online only

New for Version 1 (Release 2009a) Revised for Version 1.1 (Release 2009b) Revised for Version 1.1.1 (Release 2010a) Revised for Version 1.2 (Release 2010b) Revised for Version 1.2.1 (Release 2011a) Revised for Version 2.0 (Release 2011b) Revised for Version 2.1 (Release 2012a) Revised for Version 2.2 (Release 2012b) Revised for Version 2.3 (Release 2013a) Revised for Version 2.4 (Release 2013b) Revised for Version 2.5 (Release 2014a) Revised for Version 2.6 (Release 2014b) Revised for Version 2.7 (Release 2015a) Revised for Version 2.8 (Release 2015b) Revised for Version 3.0 (Release 2016a) Revised for Version 3.1 (Release 2016b) Revised for Version 3.2 (Release 2017a) Revised for Version 3.3 (Release 2017b) Revised for Version 3.4 (Release 2018a) Revised for Version 3.5 (Release 2018b) Revised for Version 3.6 (Release 2019a) Revised for Version 3.7 (Release 2019b) Revised for Version 3.8 (Release 2020a) Revised for Version 3.9 (Release 2020b) Revised for Version 3.9.1 (Release 2021a) Revised for Version 3.10 (Release 2021b)



Data Analysis and Processing

Model Requirements for Importing Data	1-2
Select Input Signals	1-2
Select Output Signals	1-3
Import Data for Parameter Estimation	1-5
Create Experiment	1-5
Time-Domain Data	1-6
Time-Series Data	1-9
Complex Data	1-9
Plot and Analyze Data	1-11
Why Plot the Data Before Parameter Estimation	1-11
Plot Data	1-11
Preprocess Data	1-13
Ways to Preprocess Data	1-13
Remove Offset	1-13
Scale Data	1-14
Extract Data	1-14
	1-14
Filter Data	-
Resample Data	1-15
Replace Data	1-16

Parameter Estimation

2

What Is an Experiment?	2-3
Specify Estimation Data	2-4 2-4
Create Experiment	2-4 2-5
Specify Parameters for Estimation	2-7
Choosing Which Parameters to Estimate First	2-7
Add Model Parameters as Variables for Estimation	2-7
Specify Parameters for Estimation	2-9
Specify Known Initial States	2-14
	2-14
Specify Model Initial States	2-14

Specify Estimation Options	2-17
Estimate Parameters and States	2-19
Validate Estimation Results	2-25
Configure and Perform Validation	2-25
Compare Measured and Simulated Responses	2-27
Speed Up Parameter Estimation Using Parallel Computing	2-30
When to Use Parallel Computing for Parameter Estimation	2-30
How Parallel Computing Speeds Up Estimation	2-30
Use Parallel Computing for Parameter Estimation	2-33
Configure Your System for Parallel Computing	2-33
Model Dependencies Estimate Parameters Using Parallel Computing in the Parameter Estimator	2-33
App	2-34
Estimate Parameters Using Parallel Computing (Code)	2-35
Troubleshooting	2-36
Estimating Initial Conditions for Blocks with External Initial Conditions	2.20
	2-38
Save and Load Estimation Sessions	2-39
Structure of an Estimation Session	2-39
Save Parameter Estimator App Sessions	2-39
Load Parameter Estimator App Sessions	2-39
Load Legacy Projects	2-39
How the Software Formulates Parameter Estimation as an Optimization	
Problem	2-41
Overview of Parameter Estimation as an Optimization Problem	2-41
Cost Function	2-41
Bounds and Constraints	2-42
Optimization Methods and Problem Formulations	2-43
Specify Steady-State Operating Point for Parameter Estimation	2-47
What is a Steady-State Operating Point?	2-47
Setting up a Steady-State Operating Point	2-47
	- 1/
Write a Cost Function	2-49
Anatomy of a Cost Function	2-49
Specify Inputs of the Cost Function	2-50
Compute Requirements	2-51
Specify Outputs of the Cost Function	2-52
Convenience Objects as Additional Inputs	2-52
	2-34
Gradient Computations	2-57
Estimate Model Parameter Values (Code)	2-58
Estimate Model Parameters and Initial States (Code)	2-67
Estimate Model Parameters Using Multiple Experiments (Code)	2-76

Estimate Model Parameters Per Experiment (Code)	2-86
Set Model to Steady-State When Estimating Parameters (Code) $\ldots \ldots$	2-97
Parameter Estimation for Power Plant Excitation System Starting at Steady-State (GUI)	2-106
Set Model to Steady-State When Estimating Parameters (GUI) \ldots .	2-117
Estimate Model Parameters with Parameter Constraints (Code) \ldots .	2-124
Importing and Preprocessing Experiment Data (GUI)	2-132
Estimate Model Parameter Values (GUI)	2-143
Estimate Model Parameters Per Experiment (GUI)	2-154
Estimate Model Parameters and Initial States (GUI)	2-168
Generate MATLAB Code for Parameter Estimation Problems (GUI) \dots	2-178
Improving Optimization Performance Using Fast Restart (GUI)	2-181
Improving Optimization Performance Using Fast Restart (Code)	2-187
Deployed Application of Parameter Estimation	2-191
Muscle Reflex Parameter Estimation	2-198
DC Servo Motor Parameter Estimation	2-205
Engine Speed Model Parameter Estimation	2-212
Clutch Friction Coefficient Estimation	2-214
Inverted Pendulum Parameter Estimation	2-221
Simplified Alternator Parameter Estimation	2-230

Response Optimization

How the Optimization Algorithm Formulates Minimization Problems	3-3
Feasibility Problem and Constraint Formulation	3-3
Tracking Problem	3-5
Gradient Descent Method Problem Formulations	3-6
Simplex Search Method Problem Formulations	3-7
Pattern Search Method Problem Formulations	3-7
Gradient Computations	3-8
Specify Signals to Log	3-10

Specify Custom Requirements in the App	3-11
Move Constraints	3-14 3-14 3-15
Specify Time-Domain Design Requirements in the App Specify Piecewise-Linear Lower and Upper Bounds	3-16 3-16
Specify Signal Property Requirements	3-17 3-19
Track Reference Signals Impose Elliptic Bound on Phase Plane Trajectory of Two Signals	3-19 3-21 3-22
Specify Custom Requirements	3-24 3-26
Edit Design Requirements Edit Design Requirement Dialog Box Parameters	3-29 3-29
Specify Variable Requirements in the App	3-31
Impose Monotonic Constraint Requirement on Variable Impose Upper Bound on Gradient Magnitude of Variable	3-31 3-33
Specify Linear or Quadratic Function Matching Constraint Specify Requirement on a Vector Property	3-36 3-39
Impose Relational Constraint Between Two Variables	3-41
Specify Frequency-Domain Design Requirements in the App Specify Lower Bounds on Gain and Phase Margin Specify Piecewise-Linear Lower and Upper Bounds on Frequency Response	3-43 3-43
Specify Bound on Closed-Loop Peak Gain	3-44 3-45
Specify Lower Bound on Damping Ratio Specify Lower Bounds on Natural Frequency Specify Upper and Lower Bounds on Natural Frequency Specify Lower Bounds on Natural Frequency	3-47 3-48
Specify Upper Bound on Approximate Settling Time Specify Piecewise-Linear Upper and Lower Bounds on Singular Values	3-49 3-51
Specify Step Response Characteristics	3-51 3-52 3-53
	3-56
Specify Design Variables Add Model Parameters as Variables for Optimization Specify Design Variables Specify Design Variables	3-50 3-56 3-58
Update Model with Design Variables Set	3-60
Specify Optimization Options	3-62
Create Linearization I/O Sets	3-64
Interact with Plots	3-67 3-67
Spider Plots	3-68 3-69
Compare Requirements and Design Variables Using Spider Plot	3-71
Save Design Variable Values for Specific Iteration	3-74

Design Optimization to Meet Time-Domain and Frequency-Domain Requirements (GUI)	3-76
Design Optimization Tuning Parameters in Referenced Models (GUI) .	3-88
Design Optimization Tuning Parameters in Referenced Models (Code)	3-95
Specify Steady-State Operating Point for Response Optimization What is a Steady-State Operating Point? Setting up a Steady-State Operating Point	3-101 3-101 3-101
Design Optimization to Meet a Custom Objective (GUI)	3-103
Design Optimization to Meet a Custom Objective (Code)	3-118
Design Optimization to Meet Custom Signal Requirements (GUI) \ldots	3-125
Design Optimization to Meet Frequency-Domain Requirements (GUI)	3-129
Specify Custom Signal Objective with Uncertain Variable (GUI) \ldots	3-143
Design Optimization with Uncertain Variables (Code)	3-152
Generate MATLAB Code for Design Optimization Problems (GUI) \ldots	3-160
Skip Model Simulation Based on Parameter Constraint Violation (GUI)	3-163
Optimizing Parameters for Robustness	3-170 3-170 3-170 3-172
Use Accelerator Mode During Simulations	3-179 3-179 3-179 3-179
Speed Up Response Optimization Using Parallel Computing When to Use Parallel Computing for Response Optimization How Parallel Computing Speeds Up Optimization	3-180 3-180 3-180
Use Parallel Computing for Response Optimization Configure Your System for Parallel Computing Model Dependencies Optimize Design Using Parallel Computing (GUI) Optimize Design Using Parallel Computing (Code) Troubleshooting	3-183 3-183 3-183 3-184 3-186 3-187
Use Fast Restart Mode During Response Optimization Response Optimizer App Workflow for Fast Restart Command-Line Workflow for Fast Restart Troubleshooting	3-189 3-189 3-189 3-190

Optimization Does Not Make Progress	3-191 3-191
Why don't the responses and parameter values change at all? Why does the optimization stall?	3-191 3-191 3-191
Optimization Convergence	
Why does the optimization terminate before exceeding the maximum number of iterations, with a solution that does not satisfy all the	3-192
constraints or design requirements?	3-192 3-193
is close to a solution?	3-193 3-193
Optimization Speed and Parallel Computing How can I speed up the optimization? Why are the optimization results with and without using parallel computing	
different?	3-194 3-194
Why does the optimization using parallel computing not make any progress	? 3-195
Why does the optimization using parallel computing not stop when I click the Stop optimization button?	3-195
Undesirable Parameter Values	3-196 3-196 3-196
Reverting to Initial Parameter Values	3-198 3-198
Save and Load Optimization Sessions	3-199 3-199 3-199 3-199 3-199
Improving Optimization Performance Using Parallel Computing	3-200
Optimizing Time-Domain Response of Simulink Models Using Parallel Computing	3-209
Design Optimization to Meet Frequency-Domain Requirements (Code)	3-216
PID Tuning with Actuator Constraints	3-222
PID Tuning with Reference Tracking and Plant Uncertainty	3-227
Engine Design and Cost Tradeoffs	3-232

Magnetic Levitation Controller Tuning	3-240
LQG Controller Tuning	3-248
Inverted Pendulum Controller Tuning	3-252
Pitch Rate Controller Tuning	3-256
Tuning of Airframe Autopilot Gains	3-260
Distillation Controller Tuning	3-264
Heat Exchanger Controller Tuning	3-268
Power Converter Tuning	3-272
Servomechanism Tuning	3-276
Stewart Platform Controller Tuning	3-280
Phase Lock Loop Tuning	3-284
Surrogate Optimization in Simulink Design Optimization	3-288

Sensitivity Analysis

What is Sensitivity Analysis?	4-2
Specify Parameters for Design Exploration	4-4
Add Model Parameters as Variables	4-4
Select Parameters for Design Exploration	4-6
Generate Parameter Samples for Sensitivity Analysis	4-8
Generate Random Parameter Values	4-8
	4-16
Specify Time-Domain Requirements	4-21
	4-21
Specify Piecewise-Linear Lower and Upper Bounds	4-24
	4-26
	4-27
	4-29
	4-31
Specify Custom Requirements	4-33
Specify Parameters Requirements	4-36
	4-36
Impose Upper Bound on Gradient Magnitude of Variable	4-38
	4-41
Specify Requirement on a Vector Property	4-43

Impose Relational Constraint Between Two Variables	4-45
Specify Frequency-Domain Requirements Specify Lower Bounds on Gain and Phase Margin Specify Piecewise-Linear Lower and Upper Bounds on Frequency Response	
Specify Bound on Closed-Loop Peak GainSpecify Lower Bound on Damping Ratio	4-50 4-51 4-52
Specify Upper and Lower Bounds on Natural Frequency Specify Upper Bound on Approximate Settling Time Specify Piecewise-Linear Upper and Lower Bounds on Singular Values	4-53 4-54 4-54
Specify Step Response Characteristics	4-56 4-57
Evaluate Design Requirements	4-60
Analyze Relation Between Parameters and Design Requirements	4-67 4-67
Visual Analysis	4-67 4-68
Use Sensitivity Analysis to Configure Estimation and Optimization Export Sensitivity Analysis Results	4-74 4-76
Interact with Plots in the Sensitivity Analyzer	4-79
Parameter Set Plots	4-79
Requirement Plots Evaluated Result Scatter Plots	4-84 4-86
Evaluated Result Contour Plots	4-00 4-92
Statistical Analysis Tornado Plots	4-92 4-93
Validate Sensitivity Analysis	4-96
Inspect the Generated Parameter Set	4-96
Check Evaluation Results	4-97
Perform Sensitivity Analysis with Different Parameter Set	4-99
Store Intermediate Data in the App	4-100
	4-102
	4-102
Setting up a Steady-State Operating Point	4-102
Use Parallel Computing for Sensitivity Analysis	4-104
	4-104
1	4-104
\mathbf{J}	4-104
	4-107
Troubleshooting	4-108
5 5	4-109
5 5	4-109
	4-109
Troubleshooting	4-110
Design Exploration Using Parameter Sampling (GUI)	4-112

Identify Key Parameters for Estimation (GUI)	4-131
Explore Design Reliability Using Parameter Sampling (GUI)	4-145
Design Exploration Using Parameter Sampling (Code)	4-157
Identify Key Parameters for Estimation (Code)	4-169
Generate MATLAB Code for Sensitivity Analysis Statistics to Identify Key Parameters (GUI)	4-179
Generate MATLAB Code for Sensitivity Analysis for Design Space Exploration and Evaluation (GUI)	4-183

5

Optimization-Based Control Design

Time-Domain Design Requirements in Simulink	5-2
Specify Piecewise-Linear Lower and Upper Bounds	5-2
Specify Step Response Characteristics	5-3
Track Reference Signals	5-4
Specify Custom Requirements	5-6
Edit Design Requirements	5-7
Frequency-Domain Design Requirements in Simulink	5-9
Specify Lower Bounds on Gain and Phase Margin	5-9
Specify Piecewise-Linear Lower and Upper Bounds on Frequency Response	
	5-10
Specify Bound on Closed-Loop Peak Gain	5-11
Specify Lower Bound on Damping Ratio	5-13
Specify Upper and Lower Bounds on Natural Frequency	5-14 5-15
Specify Upper Bound on Approximate Settling Time	5-10 5-17
Specify Step Response Characteristics	5-18
Specify Custom Requirements	5-19
	J -15
Time- and Frequency-Domain Requirements in Control System Designer	
App	5-22
Root Locus Diagrams	5-22
Open-Loop and Prefilter Bode Diagrams	5-23
Open-Loop Nichols Plots	5-23
Step/Impulse Response Plots	5-24
Time-Domain Simulations in Control System Designer App	5-25
Design Optimization-Based Controllers for LTI Systems	5-26
Optimize LTI System to Meet Frequency-Domain Requirements	5-27
Design Requirements	5-27 5-27
Create an LTI Plant Model Open the Control System Designer App	5-27
Open Optimization Based Tuning Method	5-20
	2-21

Select Tunable Compensator Elements	5-33
Add Design Requirements	5-34
Optimize the System Response	5-42
Create and Display the Closed-Loop System	5-43
Design Linear Controllers for Simulink Models	5-46
Enforcing Time and Frequency Requirements on a Single-Loop Controller	
Design	5-48
Airframe Controller Tuning	5-63
DC Motor Controller Tuning	5-65
Hydraulic Piston Regulator Tuning	5-67

Lookup Tables

What are Adaptive Lookup Tables?	
How to Estimate Lookup Table Values	•
Estimate Constrained Values of a Lookup Table	
Lookup Table Output Estimate the Monotonically Increasing Table Values Using Default Setting	
Validate the Estimation Results	
Estimate Lookup Table Values from Data	
Objectives	
Open a Parameter Estimation Session Estimate the Table Values Using Default Settings	
Validate the Estimation Results	
Building Models Using Adaptive Lookup Table Blocks	
Selecting an Adaptation Method	
Sample Mean	
Model Engine Using n-D Adaptive Lookup Table	
Objectives	
Building a Model Using Adaptive Lookup Table Blocks Adapting the Lookup Table Values Using Time-Varying I/O Data	

Using Adaptive Lookup Tables in Real-Time Environment	6-43
Design Optimization Using Lookup Table Requirements for Gain Scheduling (Code)	6-44
Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)	6-59
2-D Adaptive Lookup Table Generation	6-70
Engine Volumetric Efficiency Surface Matching	6-71

Data Analysis and Processing

- "Model Requirements for Importing Data" on page 1-2
- "Import Data for Parameter Estimation" on page 1-5
- "Plot and Analyze Data" on page 1-11
- "Preprocess Data" on page 1-13

Model Requirements for Importing Data

Before you can analyze and preprocess the estimation data, you must assign the data to the model ports or signals. In order to assign the data, the Simulink model must contain one of the following elements:

Root-level model Inport block

Note You do not need an Inport block if your model already contains a fixed input block, such as a Step block.

- Root-level model Outport block
- Logged signal, which can be a root-level signal in the model or a signal in a model subsystem

To enable signal logging for a signal, in the Simulink Editor, select the signal, click the

Simulation Data Inspector button arrow and click Log Selected Signals. For more information, see "Export Signal Data Using Signal Logging".

When you create an experiment, as described in "Create Experiment" on page 2-4, the top level input and output ports as well as logged signals are selected by default. You can add or remove the input and output signals using the experiment editor on page 2-5. In the experiment editor, the rows in the **Inputs** panel correspond to the model's root-level Inport blocks.

✓ Experiments		
NewData	Edit Experiment: NewData	X
▼ Results	Outputs Define measured output signals for this experiment. engine idle speed/Sum:1 <1x1 Signal, 7501 points> Select Meaured Output Signals	×
	Inputs Optionally define inputs signals for this experiment. engine idle speed/BPAV:1 <7501x2 Signal> Select Inputs	× =

Similarly, the rows in the **Outputs** panel correspond to either the root-level Outport blocks or logged signals in the model.

Select Input Signals

You can add the Inport block in the experiment editor by clicking the **Select Inputs** button in the **Inputs** panel to launch the **Select Inputs** dialog box. You can select the Inport block you want by selecting the check box corresponding to it and clicking **OK**. There is only one Inport block for the engine_idle_speed model.

Select Input	s		×
•	Inputs		
V	engine idle speed/BPAV:1 (BPAV)		
	🖋 ОК	🔀 Cancel	(?) Help
	~	vv	

Using the dialog box, you can import the input data by typing, for example, [time,iodata(:,1)] in the **Inputs** panel. To learn more about importing data, see Import Data on page 1-5.

Inputs	
Optionally define inputs signals for this experiment.	
engine idle speed/BPAV:1 (BPAV)	
0	🔻 🖶 🚽 💌

Select Output Signals

• • •

You can add the Output block in the experiment editor by clicking **Select Measured Output Signals** in the **Outputs** panel to launch the **Select Outputs** dialog box. You can select the Outport block you want by clicking the check box corresponding to it, and clicking **OK**. There is only one Outport block for the engine_idle_speed model.

Select Outp	uts		×
Click signal	s in the simulink model to add them to the outp	out table.	
•	Outputs		
V	engine idle speed/Sum:1 (Engine Speed)		
	🖋 ок 🖇	2 Cancel	(?) Help
	V 00 2		() neip

Using the dialog box, you can import the output data by typing, for example, [time,iodata(:,2)] in the **Inputs** panel. To learn more about importing data, see Import Data on page 1-5.

Outputs	
Define measured output signals for this experiment.	
engine idle speed/Sum:1 (Engine Speed)	
<1x1 Signal, 1 points>	🝷 🖶 🛧 🗶

See Also

Related Examples

• "Import Data for Parameter Estimation" on page 1-5

More About

• "What Is an Experiment?" on page 2-3

Import Data for Parameter Estimation

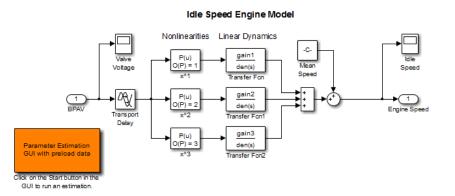
Create Experiment

Before you begin data import, create an experiment. Simulink Design Optimization software provides an app for setting up the estimation session.

To create an estimation session:

1 At the MATLAB[®] prompt, open the nonlinear idle speed model of an automotive engine by typing :

engine_idle_speed



The model contains the Inport block BPAV and Outport block Engine Speed for importing input and output data, respectively. To learn more, see "Model Requirements for Importing Data" on page 1-2.

2 In the Simulink model window, open the **Parameter Estimator** by selecting **Analysis** > **Parameter Estimation**.

A Paramet	A Parameter Estimation - engine_idle_speed								
PARAI	METER ESTIMA	ATION	VALIDATI	ON	VIEW	SH 1.	5000 X		2 🗗 🕐 🔼
					F			Cost Function: Sum Squared Error 💌	
Open Session 👻	Save Session 👻	New Experiment	Select Experiments	Select Parameters	Sensitivity Analysis 👻	Add Plot	Plot Model Response	More Options	Estimate T
FI	LE	EXPER	IMENTS	PARAM	ETERS	PL	OTS	OPTIONS	ESTIMATE
Data Brows	ser		\odot						
▼ Paramet	ers								
▼ Experim	ents								
✓ Results									
▼ Results									
▼ Preview									
+ rieview									

Parameter Estimator

You can organize the estimation and validation tasks inside **Experiments** under **Data Browser** panel on the left. You can assign each experiment to an estimation task or validation task.

To create an experiment, click the **New Experiment** button.



This creates an experiment called Exp under **Experiments**. To change the name of the experiment, right-click and select **Rename**. Call it NewData.

Note The Simulink model must remain open to perform parameter estimation tasks.

Time-Domain Data

Experiments are collections of signal data, specifically input and output signal data. After you create an experiment, as described in "Create Experiment" on page 1-5, you can import data into your experiment from various sources including MATLAB® variables, MAT-files, Excel® files, or comma-separated-value files.

To import data into your experiment right-click and select **Edit...**. This will launch the experiment editor. In the experiment editor, you can define the signals contained in the experiment.

Edit Experiment: Exp	×
Outputs	-
Define measured output signals for this experiment.	
engine idle speed/Sum:1 (Engine Speed)	
<1x1 Signal, 1 points> 🔹 🖶 🛃 🗙	
Select Meaured Output Signals	
Inputs	
Optionally define inputs signals for this experiment.	
engine idle speed/BPAV:1 (BPAV)	
<1x2 Signal> 🔹 🖶 🛃 🗶	Ξ
Select Inputs	
Initial States	
Optionally define initial states for this experiment.	
There are currently no initial states defined for this experiment.	
Select Initial States	
Parameters	
Optionally define parameters for this experiment.	
There are currently no parameters defined for this experiment.	
Select Parameters	-

For example, the rows in the **Inputs** panel of the editor correspond to Inport block BPAV in the engine_idle_speed model.

The rows in the **Outputs** panel correspond to Outport block **Engine Speed**. You can import signal data from files or MATLAB workspace.

Note The Simulink model must contain an Inport or Outport block or logged signals to enable importing data. For more information, see "Model Requirements for Importing Data" on page 1-2. To select more output signals to specify data for, click **Select Measured Output Signals** in the **Outputs** panel.

The idle-speed model of an automotive engine contains the measured data stored in the iodata array in the workspace. The array contains two columns: the first for input data, and the second for output data. The time data is in the time array in the workspace. You can import the input data by typing [time,iodata(:,1)] in the **Inputs** panel.

Inputs
Optionally define inputs signals for this experiment.
engine idle speed/BPAV:1 (BPAV)
[time,iodata(:,1)] 🔹 🕁 🗶 🗶
Select Inputs

You can import the output data by typing [time,iodata(:,2)] in the **Outputs** panel. You can view the data by clicking . The input data should look like this:

	7501x2 double			×
	1	2	3	4
1	0	0		
2	0.0200	0		
3	0.0400	0		
4	0.0600	0		
5	0.0800	0		
1	 0.1000 ₹ III 	0		•

Output data should look like this:

H 7	7501x2 double			• •
	1	2	3	4
1	0	713.2568		
2	0.0200	713.2568		
3	0.0400	709.0454		
4	0.0600	704.4067		
5	0.0800	699.8901		
-	 0.1000 ₹ III 	600 6460		•
				.:

After importing the data for NewData experiment, the experiment editor looks like this:

Edit Experiment: NewData	X
Outputs	
Define measured output signals for this experiment.	
engine idle speed/Sum:1 (Engine Speed)	
<1x1 Signal, 7501 points> 🔹 🖶 🐇 🗙	
Select Meaured Output Signals	
Inputs	
Optionally define inputs signals for this experiment.	
engine idle speed/BPAV:1 (BPAV)	
<1x1 Signal, 7501 points> 🔹 🖶 🐇 🗙	Ξ
Select Inputs	
Initial States	
Optionally define initial states for this experiment.	
There are currently no initial states defined for this experiment.	
Select Initial States	
Parameters	
Optionally define parameters for this experiment.	
There are currently no parameters defined for this experiment.	
	-

To import data from a file, click the 📥 button.

To learn more about the **Edit Experiment:** dialog box, see "Edit Experiment Data" on page 2-5.

Time-Series Data

Time-series data is stored in time-series objects. For more information, see "Time Series Objects and Collections".

When you import input data from a time-series object, t, for parameter estimation, you must specify the time vector and data as [t.time,t.inputdata] in the Inport signal dialog box. Similarly, to import output data, you must specify the time vector and data as [t.time,t.outputdata] in the Outport signal dialog box. For more information on how to import data into the experiment, see "Time-Domain Data" on page 1-6.

Complex Data

Complex-valued data is data whose value is a complex number. For example, a signal with the value 1+2j is complex. You can use complex data to estimate parameters of electrical systems, such as the magnitude and phase.

Note You must sample the real and imaginary parts of the data as a function of the same time vector.

To use complex data for parameter estimation:

- 1 Split the data into two data sets that contain the real and imaginary parts. To split the data, use the MATLAB functions real, and imag.
- 2 Create two signals, one for the real part and one for the imaginary part for the Inport or Outport block.
- **3** Select both signals in the experiment editor.
- 4 Import the data to the corresponding signal as described in "Time-Domain Data" on page 1-6.

See Also

Related Examples

- "Plot and Analyze Data" on page 1-11
- "Preprocess Data" on page 1-13

More About

• "Model Requirements for Importing Data" on page 1-2

Plot and Analyze Data

Why Plot the Data Before Parameter Estimation

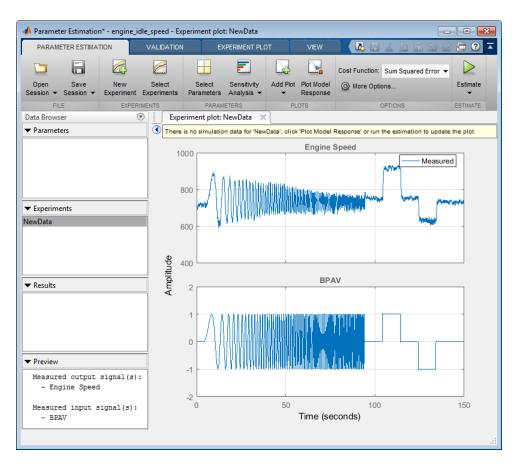
After you import the estimation data as described in "Import Data for Parameter Estimation" on page 1-5, you can remove outliers, smooth, detrend, or otherwise treat the data to prepare for analysis and estimation. To view and analyze data characteristics, plot the data on a time plot.

Plot Data

Use an experiment plot to visualize experiment data. First, create an experiment and import data as described in "Import Data for Parameter Estimation" on page 1-5. To create an experiment plot, in the **Parameter Estimator**, on the **Parameter Estimation** tab, click **Add Plot**, and select NewData under **Experiment Plots**.

ITERATION P	ITERATION PLOTS						
1 2 3	Parameter Trajectory Display the parameter values as they change during estimation.						
1 2 3	Estimation Cost Display the estimation cost as it changes during estimation.						
EXPERIMENT	PLOTS						
	NewData						

This creates plots of the input signal for the Inport block BPAV and output signal for the Outport block Engine Speed for the engine_idle_speed model (see "Create Experiment" on page 1-5).



You can also plot the experiment data by right-clicking NewData and selecting Plot measured experiment data from the list.

Using the time plot, you can examine the data characteristics such as noise, outliers and portions of the data to use for estimating parameters. After you analyze the data, you can preprocess it as described in "Preprocess Data" on page 1-13.

See Also

Related Examples

- "Import Data for Parameter Estimation" on page 1-5
- "Preprocess Data" on page 1-13

More About

• "Model Requirements for Importing Data" on page 1-2

Preprocess Data

Ways to Preprocess Data

In the **Parameter Estimator** and **Sensitivity Analyzer**, you can preprocess imported data before you use it for estimation or evaluation. After plotting the measured data, you have access to the **Experiment Plot** tab where you can preprocess the data.

				EXPERIMENT PLOT	VIEW		
Print	Grid Legend OProperties	• • •	Edit	Remove Offset	Low-Pass Filter High-Pass Filter Band-Pass Filter	Resample Data	Plot Model Response
PRINT	VIEW	ZOOM & PAN	EDIT		DATA PROCESSING		PLOT

For information about how to plot the imported data in the **Parameter Estimator**, see "Plot and Analyze Data" on page 1-11. For information about how to import and plot the data in the **Sensitivity Analyzer**, see "Match Model Outputs to Measured Signals" on page 4-21.

You can perform the following preprocessing operations:

- "Remove Offset" on page 1-13 Remove mean values, a constant value, or an initial value from the data.
- "Scale Data" on page 1-14 Scale data by a constant value, signal maximum value, or signal initial value.
- "Extract Data" on page 1-14 Select a subset of the data to use in the estimation or evaluation. You can graphically select the data to extract, or enter start and end times in the text boxes.
- "Filter Data" on page 1-15 Process data using a low-pass, high-pass, or band-pass filter.
- "Resample Data" on page 1-15 -- Resample data using zero-order hold or linear interpolation.
- "Replace Data" on page 1-16 -- Replace data with a constant value, region initial value, region final value, or a line. You can use this functionality to replace outliers.

You can perform as many preprocessing operations on your data as are required for your application. For instance, you can both filter the data and remove an offset.

Remove Offset

On the Experiment Plot tab, click Remove Offset.



It is important for good estimation results to remove data offsets. In the **Remove Offset** tab, you can remove offset from all signals at once or select a particular signal using the **Remove offset from signal** drop down list. Specify the value to remove using the **Offset to remove** drop down list. The options are:

- A constant value. Enter the value in the box. (Default: 0)
- Mean of the data, to create zero-mean data.
- Signal initial value.

As you change the offset value, the modified data is shown in preview in the plot.

After making choices, update the existing data with the preprocessed data by clicking \bigotimes .

Or, to save the modified data values in a new experiment (in the **Parameter Estimator**) or

requirement (in the Sensitivity Analyzer), click and select Save As: Create a new experiment from the modified data.

Scale Data

On the **Experiment Plot** tab, click **Scale Data**.

		EXPERIMENT PLOT	SCA	ALE DATA
O Scale all signals ○ Scale signal: Position ▼	Scale to use: 1	Apply	Close Scale Data	
SIGNAL	SCALE	APPLY	CLOSE	

In the **Scale Data** tab, you can choose to scale all signals or specify a signal to scale. Select the scaling value from the **Scale to use** drop-down list. The options are:

- A constant value. Enter the value in the box. (Default: 1)
- Signal maximum value.
- Signal initial value.

As you change the scaling, the modified data is shown in preview in the plot.

After making choices, update the existing data with the preprocessed data by clicking \bigotimes .

Or, to save the modified data values in a new experiment (in the **Parameter Estimator**) or

requirement (in the Sensitivity Analyzer), click and select Save As: Create a new experiment from the modified data.

Extract Data

To extract a portion of your data to use in the estimation or evaluation process, on the **Experiment Plot** tab, click **Extract Data**.

				EXPERIMENT PLOT	EXTRACT DATA
Start Time:	37.5		୲		
End Time: 1	112.5	Save As	Close Extract Data		
E	XTRACT DATA	SAVE AS	CLOSE		

Select a subset of data to use in **Extract Data** tab. You can extract data graphically or by specifying start time and end time. To extract data graphically, click and drag the vertical bars to select a region of the data to use.

After you choose the data to extract, you can save it in a new experiment (in the **Parameter Estimator**) or requirement (in the **Sensitivity Analyzer**) by clicking **Save As**.

Filter Data

You can filter your data using a low-pass, high-pass, or band-pass filter. A low-pass filter blocks high frequency signals, a high-pass filter blocks low frequency signals, and a band-pass filter combines the properties of both low- and high-pass filters. On the **Experiment Plot** tab click one of the **Low-Pass Filter**, **High-Pass Filter**, or **Band-Pass Filter** to open a new tab. For example, the low-pass filter tab appears as shown:

		EXPERIMEN	T PLOT	LO	W-PASS FILTER
O Filter all signals ○ Filter signal: Position ▼	0.016	toff frequency:	Options	Apply	Close Low-Pass Filter
SIGNAL	LO	W-PASS FILTER		APPLY	CLOSE

On the **Low-Pass Filter**, **High-Pass Filter**, or **Band-Pass Filter** tab, you can choose to filter all signals or specify a particular signal. For the low-pass and high-pass filtering, you can specify the normalized cutoff frequency of the signal. Where, a normalized frequency of 1 corresponds to half the sampling rate. For the band-pass filter, you can specify the normalized start and end frequencies. Specify the frequencies by either entering the value in the associated field on the tab. Alternatively, you can specify filter frequencies graphically, by dragging the vertical bars in the frequency-domain plot of your data.

Click **Options** to specify the filter order, and select zero-phase shift filter.

After making choices, update the existing data with the preprocessed data by clicking 💴.

Or, to save the modified data values in a new experiment (in the **Parameter Estimator**) or

requirement (in the Sensitivity Analyzer), click	÷.	and select	Save As: Create a new
experiment from the modified data.			

Apply

Resample Data

On the **Experiment Plot** tab, click the **Resample Data** button.

		EXPER	IMENT PLOT	RESAMPLE DATA
Resample with sample period: Resample Using:		Apply	Close Resample Data	
RESAMPLE DATA		APPLY	CLOSE	

In the **Resample Data** tab, specify the sampling period using the **Resample with sample period**: field. You can resample your data using one of the following interpolation methods:

- Zero-order hold Fill the missing data sample with the data value immediately preceding it.
- Linear interpolation Fill the missing data using a line that connects the two data points.

By default, the resampling method is set to zero-order hold. You can select the linear interpolation method from the **Resample Using** drop-down list.

The modified data is shown in preview in the plot.

After making choices, update the existing data with the preprocessed data by clicking \bigotimes .

Or, to save the modified data values in a new experiment (in the Parameter Estimator) or

```
requirement (in the Sensitivity Analyzer), click and select Save As: Create a new experiment from the modified data.
```

Replace Data

On the **Experiment Plot** tab click the **Replace Data** button.

			EXPERIM	IENT PLOT	REPLACE DATA
Replace selected data	Clear preview	Apply	Close Replace Data		
REPLACE D/	APPLY	CLOSE			

In the **Replace Data** tab, select data to replace by dragging across a region in the plot. Once you select data, choose how to replace it using the **Replace selected data** drop-down list. You can replace the data you select with one of these options:

- A constant value
- Region initial value
- Region final value
- A line

The replaced preview data changes color and the replacement data appears on the plot. At any time before updating, click **Clear preview** to clear the data you replaced and start over.

After making choices, update the existing data with the preprocessed data by clicking \bigotimes .

Or, to save the modified data values in a new experiment (in the **Parameter Estimator**) or

requirement (in the Sensitivity Analyzer), click and select Save As: Create a new experiment from the modified data.

Replace Data can be useful, for example, to replace outliers. Outliers can be defined as data values that deviate from the mean by more than three standard deviations. When estimating parameters from data containing outliers, the results may not be accurate. Hence, you might choose to replace the outliers in the data before you estimate the parameters.

See Also

More About

- "Import Data for Parameter Estimation" on page 1-5
- "Match Model Outputs to Measured Signals" on page 4-21

- "What Is an Experiment?" on page 2-3
- "Specify Estimation Data" on page 2-4
- "Specify Parameters for Estimation" on page 2-7
- "Specify Known Initial States" on page 2-14
- "Specify Estimation Options" on page 2-17
- "Estimate Parameters and States" on page 2-19
- "Validate Estimation Results" on page 2-25
- "Speed Up Parameter Estimation Using Parallel Computing" on page 2-30
- "Use Parallel Computing for Parameter Estimation" on page 2-33
- "Estimating Initial Conditions for Blocks with External Initial Conditions" on page 2-38
- "Save and Load Estimation Sessions" on page 2-39
- "How the Software Formulates Parameter Estimation as an Optimization Problem" on page 2-41
- "Specify Steady-State Operating Point for Parameter Estimation" on page 2-47
- "Write a Cost Function" on page 2-49
- "Gradient Computations" on page 2-57
- "Estimate Model Parameter Values (Code)" on page 2-58
- "Estimate Model Parameters and Initial States (Code)" on page 2-67
- "Estimate Model Parameters Using Multiple Experiments (Code)" on page 2-76
- "Estimate Model Parameters Per Experiment (Code)" on page 2-86
- "Set Model to Steady-State When Estimating Parameters (Code)" on page 2-97
- "Parameter Estimation for Power Plant Excitation System Starting at Steady-State (GUI)" on page 2-106
- "Set Model to Steady-State When Estimating Parameters (GUI)" on page 2-117
- "Estimate Model Parameters with Parameter Constraints (Code)" on page 2-124
- "Importing and Preprocessing Experiment Data (GUI)" on page 2-132
- "Estimate Model Parameter Values (GUI)" on page 2-143
- "Estimate Model Parameters Per Experiment (GUI)" on page 2-154
- "Estimate Model Parameters and Initial States (GUI)" on page 2-168
- "Generate MATLAB Code for Parameter Estimation Problems (GUI)" on page 2-178
- "Improving Optimization Performance Using Fast Restart (GUI)" on page 2-181
- "Improving Optimization Performance Using Fast Restart (Code)" on page 2-187
- "Deployed Application of Parameter Estimation" on page 2-191
- "Muscle Reflex Parameter Estimation" on page 2-198
- "DC Servo Motor Parameter Estimation" on page 2-205
- "Engine Speed Model Parameter Estimation" on page 2-212

- "Clutch Friction Coefficient Estimation" on page 2-214
- "Inverted Pendulum Parameter Estimation" on page 2-221
- "Simplified Alternator Parameter Estimation" on page 2-230

What Is an Experiment?

To estimate unknown parameter values of a Simulink model, first create an experiment. An experiment specifies measured input and output data. During estimation, the experiment input data is used to simulate the model and the model output is compared with the measured experiment output data. For more information about creating experiments and importing data, see "Specify Estimation Data" on page 2-4.

In an experiment, you can specify initial-state values. To do so, specify the model initial states for each experiment. You can optionally specify an initial guess for the initial state values for any experiment. For more information, see "Specify Known Initial States" on page 2-14.

To estimate a model parameter on a per-experiment basis, specify the model parameter for each experiment. You can specify the initial values and limits for the parameter value for any of the experiments. Alternatively, you can specify a parameter value as a known quantity, not to be estimated. For more information, see "Specify Parameters for Estimation" on page 2-7. You can choose to update experiments with estimated model initial states and parameter values, or save the results in a new experiment. For more information, see "Specify Estimation Options" on page 2-17.

To use experiments for validating the estimated parameter values, see "Validate Estimation Results" on page 2-25.

See Also

Related Examples

- "Specify Estimation Data" on page 2-4
- "Specify Parameters for Estimation" on page 2-7
- "Specify Known Initial States" on page 2-14
- "Estimate Parameters and States" on page 2-19

More About

• "Edit Experiment Data" on page 2-5

Specify Estimation Data

This topic shows how to specify estimation data for parameter estimation.

Create Experiment

Before you specify estimation data, create an experiment. At the MATLAB prompt, open the nonlinear idle speed model of an automotive engine by typing

engine_idle_speed

In the Simulink model window, open the **Parameter Estimator** by selecting **Analysis > Parameter Estimation**.

PARA	METER ESTIMA	TION	VALIDATI	ON	VIEW				2 🗖 🕐
					.			Cost Function: Sum Squared Error 👻	
Open Session 💌	Save Session 💌	New Experiment	Select Experiments	Select Parameters	Sensitivity Analysis 👻	Add Plot	Plot Model Response	More Options	Estimate •
	LE	EXPER	IMENTS	PARAM	ETERS	PL	OTS	OPTIONS	ESTIMATE
ata Brow	ser		$\overline{\mathbf{v}}$						
 Parame 	ters								
' Experim	ients								
 Results 									
Preview	,								
 Preview 	(
 Preview 	1								
• Preview	<u>, </u>								

In the **Parameter Estimator**, on the **Parameter Estimation** tab, click **New Experiment**.



This action creates an experiment called Exp in the **Experiments** list in the **Data Browser** panel and opens the experiment editor. To change the name of the experiment, right-click Exp and select **Rename**. If you rename it NewData, the **Experiments** list now looks like this:

▼ Experiments	
NewData	

Edit Experiment Data

After creating an experiment, launch the experiment editor by right-clicking on the experiment name and selecting **Edit...** from the list. The experiment editor resembles the following figure.

Edit Experiment: Exp	X
Outputs	*
Define measured output signals for this experiment.	
engine idle speed/Sum:1 (Engine Speed)	
<1x1 Signal, 1 points> 🔹 🕁 🗶 🗙	
Select Meaured Output Signals	
Inputs	
Optionally define inputs signals for this experiment.	
engine idle speed/BPAV:1 (BPAV)	
<1x2 Signal> 🔻 🖶 🐇 🗙	Ξ
Select Inputs	
Initial States	
Optionally define initial states for this experiment.	
There are currently no initial states defined for this experiment.	
Select Initial States	
Parameters	
Optionally define parameters for this experiment.	
There are currently no parameters defined for this experiment.	
	+

The experiment editor has four panels. You select output signals on page 1-3 and import output data on page 1-5 in the **Outputs** panel. You select input signals on page 1-2 and import input data on page 1-5 in the **Inputs** panel. You can specify model initial states on page 2-14 in the **Initial States** panel. You can specify parameters to estimate on page 2-11 in the **Parameters** panel.

The rows in the **Inputs** panel of the editor correspond to Inport block BPAV in the engine_idle_speed model. See "Import Data for Parameter Estimation" on page 1-5.

The rows in the **Outputs** panel correspond to Outport block **Engine Speed**. You can import signal data from files or MATLAB workspace.

The idle-speed model of an automotive engine contains the measured data stored in the iodata array in the workspace. The array contains two columns: the first for input data, and the second for output data. The time data is in the time array in the workspace. Import the input data by typing [time,iodata(:,1)] in the dialog box in the **Inputs** panel.

Inputs	
Optionally define inputs signals for this experiment.	
engine idle speed/BPAV:1 (BPAV)	
[time,iodata(:,1)]	🔻 🖶 🤟 💌
Select Inputs	

Import the output data by typing [time, iodata(:,2)] in the dialog box in the **Outputs** panel.

Note You can have more than one input or output signal, but you can have only one data set for a signal. If you have multiple data sets, create multiple experiments.

See Also

Related Examples

- "Model Requirements for Importing Data" on page 1-2
- "Specify Parameters for Estimation" on page 2-7
- "Specify Known Initial States" on page 2-14
- "Estimate Parameters and States" on page 2-19

More About

• "What Is an Experiment?" on page 2-3

Specify Parameters for Estimation

Choosing Which Parameters to Estimate First

Simulink Design Optimization software lets you estimate scalar, vector, and matrix parameters. You can take an iterative approach to estimating model parameters. For example, if you have a large number of parameters to estimate, start by estimating those that most influence the output. After you estimate a subset of parameters and validate the estimated parameters, you can select the remaining parameters for estimation.

You can also first use sensitivity analysis to identify the parameters that most influence the estimation, and then specify these parameters for estimation. To open the **Sensitivity Analyzer**, in

the **Parameter Estimation** tab, click **Sensitivity Analysis**. In the **Sensitivity Analyzer**, you can identify the model parameters that most influence the estimation problem and compute initial values for the estimation parameters.

Add Model Parameters as Variables for Estimation

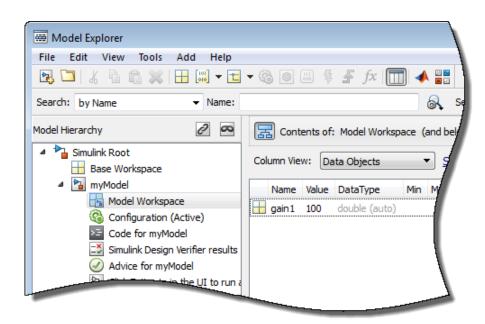
The software can only estimate variables that are in use by the model. Create variables for estimation in the MATLAB or model workspace, and specify your Simulink model or block parameters using these variables.

In this figure, the **Numerator coefficients** parameter of a Transfer Fcn block is specified as a numerical value.

	🔁 Function Block Parameters: Transfer Fcn	—				
	Transfer Fcn					
The numerator coefficient can be a vector or matrix expression. The denominator coefficient must be a vector. The output width equals the number of rows in the numerator coefficient. You should specify the coefficients in descending order of powers of s.						
	Parameters					
	Numerator coefficients:					
	[100]					
	Denominator coefficients:					
	p*damp1/freq1, 1]					

To estimate the **Numerator coefficients** parameter, specify it as variable gain1:

- 1 Create the variable gain1 in one of the following ways:
 - Add the variables to the model workspace, and specify initial values.



• Write initialization code in the **PreloadFcn** callback of the model. For more information, see "Model Callbacks".

gain1 = 100

2 Specify the block parameter as variable gain1 in the Transfer Fcn block dialog box.

coefficients in descending order of	fpowers
Parameters	
Numerator coefficients:	
[gain1]	
Denominator coefficients	

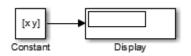
You can now select gain1 for estimation. See, "Specify Parameters for Estimation" on page 2-9.

Specify Independent Parameters for Estimation

You can also specify independent parameters that do not appear explicitly in the model as variables for estimation. However, you cannot use this workflow with Simulink fast restart.

Suppose that a model parameter Kint is related to independent parameters x and y such that Kint = x+y. To estimate x and y instead of Kint:

- Create the independent variables x and y by adding them to the model workspace and specifying initial values.
- The software only allows tuning of variables that are used by model blocks. To ensure that the software detects x and y for tuning, add a Constant block to your model, and specify the **Constant value** of the block as [x y]. Connect the block to a Display block.



• Write code in the **InitFcn** callback of the model that defines the relationship between Kint, x, and y. You must first use the get_param function to get the variables x and y from the model workspace before you can use them to define Kint.

```
wks = get_param(gcs, 'ModelWorkspace')
x = evalin(wks, 'x')
y = evalin(wks, 'y')
Kint = x+y;
```

You can now select x and y for estimation. Do not estimate the independent and dependent parameters simultaneously. Doing so can lead to incorrect results. For example, do not estimate Kint, x and y together.

Specify Parameters for Estimation

You can specify the parameters for estimation experiments using the **Estimated Parameters** editor. In the **Parameter Estimator**, on the **Parameter Estimation** tab, click **Select Parameters**.

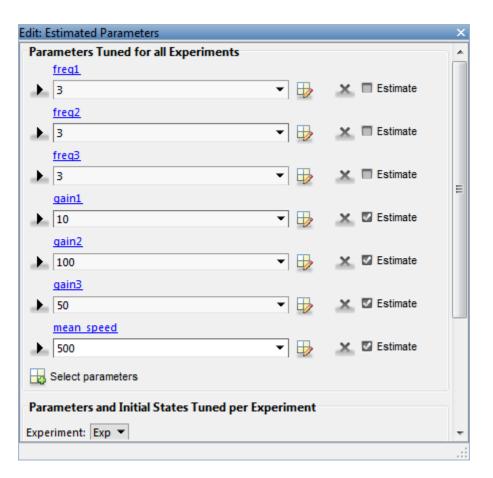


To select parameters for all experiments, click **Select Parameters** in the **Parameters Tuned for all Experiments** panel. This opens the **Select model variables** dialog. Here you can select the parameters you want to estimate by clicking the check box next to it or specifying an expression. For more information see "Select Parameters Using Select Model Variables Dialog Box" on page 2-11.

The editor looks like

Edit: Estimated Parameters			×
Parameters Tuned for all Experiments			
There are no parameters selected for estimation	tion.		
Belect parameters			
Parameters and Initial States Tuned per I	xperiment		
Experiment: NewData 💌			
Select experiment initial states for estimation			
There are no initial states defined for this ex	periment.		
Select experiment parameters for estimation.			
There are no parameters defined for this exp	periment.		
Edit experiment			
	🔈 Update Model	🖋 ок	(?) Help
			.:

For example, in the engine_idle_speed model, select freq1, freq2, freq3, gain1, gain2, gain3 and mean_speed for estimation. You do not need to estimate the parameters all at once. You can first select all the parameters you are interested in, and then later select a subset to estimate. By default, all the parameters are selected for estimation. To deselect the ones you do not want to estimate, clear the **Estimate** check box for a parameter. For this example, only estimate gain1, gain2, gain3 and mean_speed. Set their initial values 10, 100, 50, and 500, respectively, and then click **OK**. The **Edit: Estimated Parameters** dialog box looks like



To learn how to specify initial values and upper and lower bounds of the parameters, see "Specifying Initial Guesses and Upper/Lower Bounds" on page 2-13.

Select Parameters to Estimate for a Specific Experiment

To select the parameters to estimate in a specific experiment, first, select the experiment for estimation as described in "Estimate Parameters and States" on page 2-19. Then, you can use the **Edit:Estimated Parameters** dialog to select parameters to estimate for that experiment. Select the experiment name from the **Experiment:** combo box in the **Parameters and Initial States Tuned per Experiment** panel. Then click **Edit experiment** to launch the experiment editor for the experiment you select.

Alternatively, you can right click the experiment name in the **Experiments** list and select **Edit...** In the experiment editor, click the **Select parameters** button in the **Parameters** panel. In the Select model variables dialog box, you can select the parameters you want to estimate in this experiment by checking the box next to it or specifying an expression. For more information see "Select Parameters Using Select Model Variables Dialog Box" on page 2-11.

Select Parameters Using Select Model Variables Dialog Box

Use this dialog box to specify parameters to estimate. The table lists the variables that the model uses to set block parameter values. The variables can reside in the model workspace, the base workspace, or a data dictionary.

Select variables by clicking the check box next to each variable. If your model contains many variables, filter the list by typing in the **Filter by variable name** field. The **Used By** column lists all blocks in the model that use the variable. When a variable is used in more than one block, all blocks are listed. To highlight blocks in the model that use the variable, click the block name.

•	Variable	Current v	Used By	
	freq1	3	engine idle speed/Transfer Fcn	-
	freq2	3	engine idle speed/Transfer Fcn1	
	freq3	3	engine idle speed/Transfer Fcn2	
	gain1	100	engine idle speed/Transfer Fcn	=
	gain2	0	engine idle speed/Transfer Fcn1	
	gain3	0	engine idle speed/Transfer Fcn2	
Spo	ecify express	ion indexing	if necessary (e.g., a(3) or s.x.) V OK Cancel (?) Help

The variables that you select must have a numeric value that uses the data type double. If the value of a variable is not a double number, use these techniques:

• To select a single element or a subset of a matrix or array variable, click **Specify expression** indexing if necessary.

 Specify expression indexing if necessary (e.g., a(3) or s.x.) 	

Enter an expression such as myArray(2), which selects the second element of an array variable myArray.

After you type the expression, press the **Enter** key to add the variable to the list of model variables.

- To use a variable of a numeric data type other than double, convert the variable to a Simulink.Parameter object, which separates a parameter value from its data type. Set the Value property to a default double number, and use the DataType property to control the data type.
- To use the value of a Simulink.Parameter object, specify the Value property. Enter the expression myParamObj.Value.
- To use a numeric field of a structure, enter myStruct.PID.P1. If you store the structure in a Simulink.Parameter object, enter myStruct.Value.PID.P1.
- To use one cell of a cell array, enter myCells{3}.

You cannot use mathematical expressions such as a + b. Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters, see "Add Model Parameters as Variables for Estimation" on page 2-7.

Specifying Initial Guesses and Upper/Lower Bounds

After you select parameters, you can specify

- Initial guess The value the estimation uses to start the process.
- Minimum The smallest allowable parameter value. The default is Inf.
- **Maximum** The largest allowable parameter value. The default is +Inf.

You can enter the initial value in the dialog box below the parameter name. You can specify the minimum and maximum value fields by clicking the arrow . The default minimum and maximum values are -Inf and +Inf, respectively, but you can select any range you want.

<u>gain1</u>			
▼ 10		▼ 🖶	🗶 🗹 Estimate
Minimum:	-Inf	• 🖶	
Maximum:	Inf	▼ 🖶	
Scale:	10	-	

If you believe a parameter lies within a finite range, it is best not to use the default minimum and maximum values. Often, there are computational advantages in specifying finite bounds. It can be very important to specify lower and upper bounds. For example, if a parameter specifies the weight of a part, be sure to specify 0 as the absolute lower bound if better information is unavailable.

Note When you specify the minimum and maximum values for the parameters, it does not affect your settings in the **Parameters** list under **Data Browser** pane. You make these choices for each experiment.

• Scale — The scale value to use for normalization. The parameters are scaled, or normalized, by dividing their current value by the scale value. Scale is useful, in situations, for example, when parameters have different orders of magnitude.

The default scale value is the next power of 2 greater than the current value of the parameter. For example, if the current parameter value is 15, **Scale** is 16 ($=2^4$). You can edit this field to provide an alternate scaling factor.

See Also

Related Examples

• "Specify Known Initial States" on page 2-14

More About

• "What Is an Experiment?" on page 2-3

Specify Known Initial States

When to Specify Initial States Versus Estimate Initial States

Sets of measured data are often collected at various times and under different initial conditions. When you estimate model parameters using one data set and subsequently run another estimation with a second data set, your parameter values may not match.

You can use the **Parameter Estimator** to estimate the initial conditions using procedures that are similar to those you use to estimate parameters. You can then use these initial condition estimates as a basis for estimating parameters for your Simulink model.

Specify Model Initial States

After you select parameters for estimation, as described in "Specify Parameters for Estimation" on page 2-7, you can specify initial conditions of states in your model. By default, the estimation uses initial conditions specified in the Simulink model. If you want to specify initial conditions other than the defaults, use the **Initial States** panel in the experiment editor dialog. For this example, right click NewData and select Edit... from the list to open the experiment editor on page 2-5. Then, click Select Initial States button.

Initial States
Optionally define initial states for this experiment.
There are currently no initial states defined for this experiment.
Select Initial States

The Select Model States dialog for the engine_idle_speed model looks like

Select mod	el states			×
Filter by st	tate name			+ م
-	State		Curre	nt value
	engine idle speed/Transfer Fcn		[0 0]
	engine idle speed/Transfer Fcn1		[0 0]
	engine idle speed/Transfer Fcn2		[0 0]
L	√ 0	к 🖾	Cancel	(?) Help
		~~		

Click the check box next to the state you would like to modify. For example, if you select **engine_idle_speed/Transfer Fcn** and enter the initial values -0.2 and 0, the **Initial States** panel now looks like

Initial States	
Optionally define initial states for this experiment.	
engine idle speed/Transfer Fcn	
▶ [-0.2 0]	▼ 🖶 ×
Select Initial States	

Click **Select Parameters** in the **Parameter Estimation** tab. After you also select the parameters as described in "Specify Parameters for Estimation" on page 2-7, the Edit: Estimated Parameters dialog looks like the following figure.

Edit: Estimated Parameters		×
Parameters Tuned for all Experiments		
freq1		
3	• 6	🦆 🗶 🗖 Estimate
freq2		
3	•	Estimate
freq3		
3	•	Estimate
gain1		
10	• [🚽 🗶 🗹 Estimate
gain2		
100	- [🖌 🗶 🗹 Estimate
gain3		
50	- 6	🚽 🗶 🗹 Estimate
mean speed		
	•	🛃 🗶 🗹 Estimate
Select parameters		
Parameters and Initial States Tuned per Experime	ent	
Experiment: Exp 💌		
Select experiment initial states for estimation.		
engine idle speed/Transfer Fcn		
• [-0.2 0]	•	Estimate
Select experiment parameters for estimation.		
There are no parameters defined for this experiment.		
Edit experiment		
East oxperiment		
		🚂 Update Model 🛛 🖌 🕜 Help

See Also

Related Examples

- "Specify Estimation Data" on page 2-4
- "Specify Parameters for Estimation" on page 2-7

More About

• "Edit Experiment Data" on page 2-5

Specify Estimation Options

This topic shows how to specify estimation options in the **Parameter Estimator**.

After you have specified estimation data and parameters, specify the following estimation options:

1 Goodness of fit criteria (cost function)

Cost Function: Sum Squared Error 💌

The cost function is a function that estimation methods minimize. To specify the method for calculating the cost function, in the **Parameter Estimation** tab of the app, select one of the following from the **Cost Function** drop-down list:

- Sum Squared Error Uses a least-squares approach (default).
- Sum-Absolute Error Uses the sum of absolute errors.

If the experimental data has many outliers, you can select the robust cost option. The software uses a Huber loss function to handle the outliers in the cost function and improves the fit quality. This option reduces the influence of outliers on the estimation without you having to manually

modify your data. To select this option, in the **Parameter Estimation** tab, click **WMore Options** to open the Estimation Options dialog box. In the **Optimization Options** tab of the dialog box, select **Use robust cost**.

The software uses the error statistics to identify the outliers. The error, e, is calculated as the difference between measured and simulated output. The cost function used, F(x), depends on the method used.

Method Name	Cost Function for Nonoutliers	Cost Function for Outliers
Sum Squared Error	$F(x) = \sum_{t \in NOL} e(t) \times e(t)$	$F(x) = \sum_{t \in OL} w \times \left e(t) \right $
	<i>NOL</i> is the set of nonoutlier samples.	<i>w</i> is a linear weight. <i>OL</i> is the set of outlier samples.
Sum-Absolute Error	$F(x) = \sum_{t \in NOL} e(t)$	$F(x) = \sum_{t \in OL} w$
	<i>NOL</i> is the set of nonoutlier samples.	<i>w</i> is a constant value. <i>OL</i> is the set of outlier samples.

2 Estimation progress and result options for estimation task

To specify these options, in the **Parameter Estimation** tab, click **More Options** to open the Estimation Options dialog box. In the **General Options** tab, specify the estimation progress and result options. For details about the options, click the **Help** button.

Estimation Options	5			×
General Options	Optimization Options	Parallel Options		
Progress Option	s			
Show estimation	tion progress window d	uring estimation		
🔽 Create a para	ameter trajectory plot du	uring estimation		
Update plots	during estimation			
Result Options				
	el at end of estimation			
Estimation result				
	e estimation result with i	now actimated val	luer	
			lues	
Save estin	mated values as new est	imation result		
	C	K Cancel	Help	
				.::

3 Optimization options, such as optimization method and optimization termination options.

Specify these options in the **Optimization Options** tab of the Estimation Options dialog box. For details about the options, click the **Help** button.

4 Parallel computing options

Specify these options in the **Parallel Options** tab of the Estimation Options dialog box. For details about the options, see "Estimate Parameters Using Parallel Computing in the Parameter Estimator App" on page 2-34.

See Also

Related Examples

- "Specify Estimation Data" on page 2-4
- "Specify Parameters for Estimation" on page 2-7

More About

- "How the Software Formulates Parameter Estimation as an Optimization Problem" on page 2- 41

Estimate Parameters and States

This topic shows how to estimate parameters and states in the **Parameter Estimator**.

PARA	AMETER ESTIMA	ATION	VALIDATI	ON	VIEW	H	$\sim \sim \sim$	Seii A L L Se	ā ? 🔺
								Cost Function: Sum Squared Error 🕶	
Open Session 🔻	Save Session 👻	New Experiment	Select Experiments	Select Parameters	Sensitivity Analysis ▼	Add Plot	Plot Model Response	More Options	Estimate T
F	ILE	EXPER	RIMENTS	PARAM	ETERS	PL	.OTS	OPTIONS	ESTIMATE
Data Brow	vser)						
▼ Parame	eters								
			_						
 Experin 	nents								
			_						
 Results 	;								

To estimate parameters and states after you have specified estimation data on page 2-4, parameters on page 2-7, and estimation options on page 2-17:

1 Specify experiments for estimation.

In the **Parameter Estimator**, in the **Parameter Estimation** tab, click **Select Experiments**. In the Select Experiments dialog box, in the **Estimation** column, select the experiment to use.

Select Experime	ents		×
Select experim	ents to includ	e for estimation or validation	
Estimation	Validation	Experiment	
V		NewData	
			÷
		🖋 ок 🔅) Help

For information about the Validation column, see "Validate Estimation Results" on page 2-25.

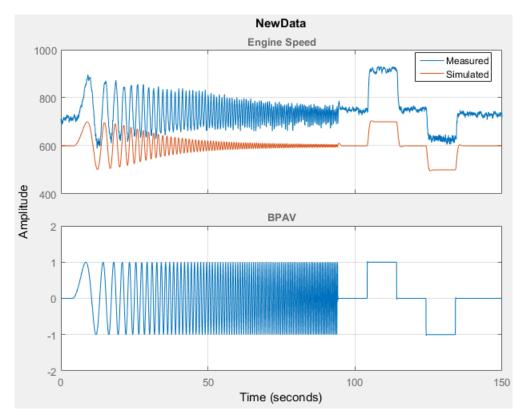
2 (Optional) Create progress plots to view estimation progress.

When you start the estimation, the app automatically displays a parameter trajectory plot that shows the change in the parameter values during estimation. You can create other plots for viewing the progress of the estimation before you begin estimating the parameters. During estimation, all these plots update with each iteration.

• To plot the measured data, on the **Parameter Estimation** tab, click **Add Plot**. Select the experiment to use for estimation under **Experiment Plots** of the drop-down list.

To add the simulated response to the measured data plot, on the **Parameter Estimation** tab, click **Plot Model Response**.

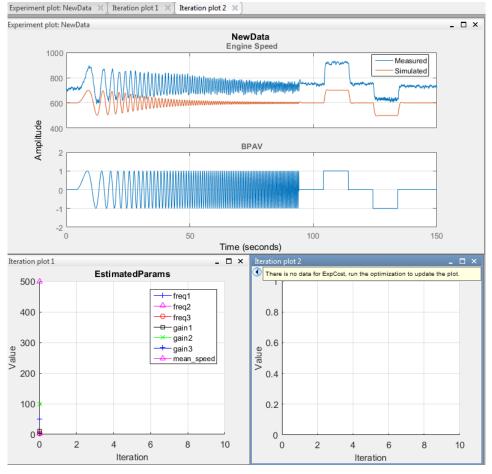
Alternatively, right-click the experiment name in the **Experiments** area of the app, and select **Plot measured & simulated data** from the menu.



You can edit the labels, adjust the limits, change the units, and the font style of the plot in the **Property Editor**. To open the editor, right-click the experiment plot and select **Properties** from the list.

- To plot the parameter values as they change, click **Add Plot**, and select **Parameter Trajectory**. To add the scaled values or save iteration data, right-click the plot.
- To add a plot for the estimation cost, click **Add Plot**, and select **Estimation Cost**. To add the scaled values or save iteration data, right-click the plot.

Use the **View** tab of the app to arrange the layout of the plots, so that all the progress plots you created are visible.



3 Estimate the parameters and states.

In the **Parameter Estimation** tab, click **Estimate**.

An Estimation Progress Report window opens at the start of estimation. The report and progress plots update with each iteration.

4 View the Estimation Progress Report after the estimation completes.

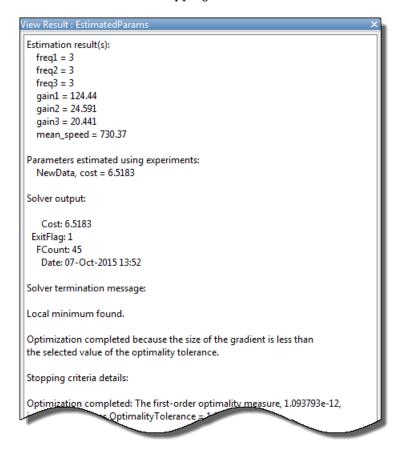
By default, the report displays the iteration number (**Iteration**), the number of times the objective function is calculated (**F-count**), and cost function value (**f(x)**, for example, **NewData(Minimize)**). You can change the display table by clicking **Display Options**. To learn more about the display table, see "Iterative Display".

Iteration	F-count	NewData (Minimize)
0	9	367.8139
1	18	20.5548
2	27	8.7363
3	36	6.6133
4	45	6.5183
Estimation	n convergeo	7-Oct-2015 13:51:34 , 07-Oct-2015 13:52:00

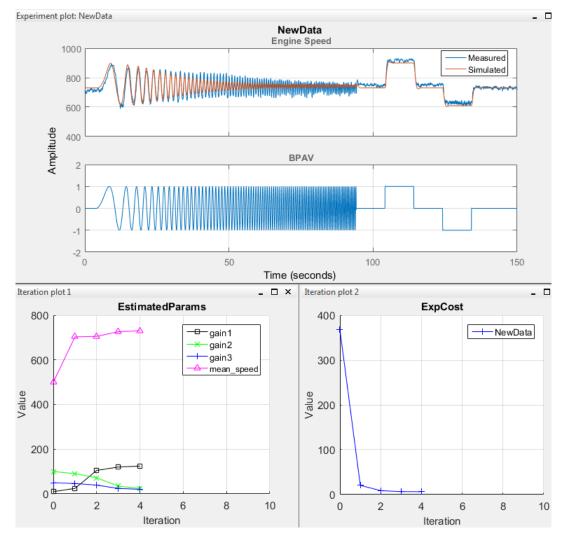
5 View the estimated results.

The estimation results are saved in a new variable, EstimatedParams, in the **Results** area of the app.

To view the contents of the variable, right-click EstimatedParams and select **Open...** from the menu. EstimatedParams includes the values of the parameters, the cost function value, and information about the stopping criteria for the estimation.



6 View the progress plots.



The measured versus simulated data plot shows how closely the simulated data matches the measured estimation data. The estimated parameters plot and cost function plots show the changes in the estimated value of the parameters and estimated cost function for each iteration.

Typically, a lower cost function value indicates the model simulation with the estimated parameters closely matches experimental data. If the optimization went well, you should see your cost function converge to a minimum value. The lower the cost, the more successful the estimation.

For information on types of problems you may encounter using optimization solvers, see "When the Solver Fails", "When the Solver Might Have Succeeded", and "When the Solver Succeeds".

See Also

Related Examples

- "Specify Estimation Data" on page 2-4
- "Specify Parameters for Estimation" on page 2-7

- "Specify Known Initial States" on page 2-14
- "Specify Estimation Options" on page 2-17
- "Validate Estimation Results" on page 2-25
- "Save and Load Estimation Sessions" on page 2-39

More About

• "What Is an Experiment?" on page 2-3

Validate Estimation Results

This topic shows how to validate estimation results in the Parameter Estimator. After you estimate parameters as described in "Estimate Parameters and States" on page 2-19, validate the estimation results using another data set.

Configure and Perform Validation

To validate a model using the **Parameter Estimator**:

1 Load the validation data set.

At the MATLAB prompt, load the validation data into the MATLAB workspace.

load iodataval;

2 Add a new experiment for validation.

In the **Parameter Estimator**, in the **Validation** tab, click **New Experiment**. A new experiment with default name appears in the **Experiments** area of the app. To rename the experiment ValidationData, right-click the experiment and select **Rename** from the drop-down menu.

PARAMETER ESTIMATION			V	ALIDATION	EXPERIMENT F	иот
New Experiment	Select Experiments	Select Re to Valid		Plot Measure	d & Simulated Data s	Validate ▼
EXPER	IMENTS	RESUL	TS	PLOT (PTIONS	VALIDATE

3 Import the validation data set into the validation experiment.

Right click the experiment name, and select **Edit**. Specify [time,iodataval(:,1)] in **Inputs**, and [time,iodataval(:,2)] in **Outputs**.

4 Specify the experiment for validation.

When you create an experiment, it is by default selected for estimation. To select another experiment for validation, in the **Validation** tab, click **Select Experiments**.

In the Select Experiments dialog box, clear **Estimation** and select **Validation** for the validation experiment.

elect experin	nents to includ	e for estimation or validation	
Estimation	Validation	Experiment	
V		NewData	₿ 🖶
	V	ValidationData	L.
			5
			 Help

5 Specify the estimation results to use in the validation.

After you import validation data, select the estimated parameter values to validate.

The estimation results are saved in the EstimatedParams variable in the **Results** area of the app. In the **Validation** tab, click **Select Results to Validate**.

To validate the estimated parameters, in the Select Results dialog box, select EstimatedParams and clear Use current parameter values.

Selec	t Res	sults	×
		Select results to include in validation	
		Results	
		Use current parameter values	
[1	EstimatedParams	
		🖋 ок	Help

6 Select the plots to display at the end of validation.

Parameter Estimator can display the measured and simulated responses and the residuals plot at the end of validation. Select the plots to display by checking the corresponding box on the **Validation Tab**.

 Plot Measured & Simulated Data Plot Residuals
PLOT OPTIONS

7 Validate the estimation results.

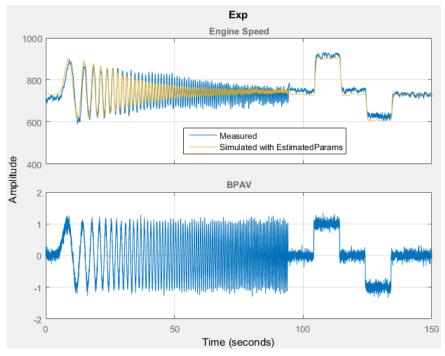
On the **Validation** tab, click **Validate**. The Validation Progress Report shows the status of the validation.

Experiment Name		Status	
/alidationExp	Completed		

Compare Measured and Simulated Responses

1 Compare measured validation data against the model output simulated with the estimated parameters.

The app displays the experiment plot for each experiment selected for validation. Each experiment plot shows the measured data, and data from simulation using each set of results selected. For example, the following figure shows the experiment plot for the ValidationExp data.

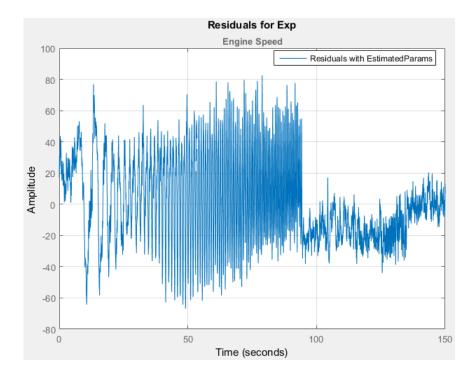


2 Examine the Residuals Plot.

The residuals plot shows the difference between simulated response and measured data. When there is a good fit between the simulated output and measured data, the residuals show the following behavior:

- Lie within a small percent of the maximum output variation.
- Do not display any systematic patterns.

For example, you can see from the following figure that the residuals for ValidationExp data satisfy both criteria.



See Also

Related Examples

• "Estimate Parameters and States" on page 2-19

More About

• "What Is an Experiment?" on page 2-3

Speed Up Parameter Estimation Using Parallel Computing

When to Use Parallel Computing for Parameter Estimation

You can use Simulink Design Optimization software with Parallel Computing Toolbox[™] software to speed up parameter estimation of Simulink models. Using parallel computing may reduce the estimation time in the following cases:

- The model contains a large number parameters to estimate, and the estimation method is specified as either Nonlinear least squares or Gradient descent.
- The Pattern search method is selected as the estimation method.
- The model is complex and takes a long time to simulate.

When you use parallel computing, the software distributes independent simulations to run them in parallel on multiple MATLAB sessions, also known as *workers*. The time required to simulate the model dominates the total estimation time. Therefore, distributing the simulations significantly reduces the estimation time.

For information on how the software distributes the simulations and the expected speedup, see "How Parallel Computing Speeds Up Estimation" on page 2-30.

For information on configuring your system and using parallel computing, see "Use Parallel Computing for Parameter Estimation" on page 2-33.

How Parallel Computing Speeds Up Estimation

You can enable parallel computing with the Nonlinear least squares, Gradient descent and Pattern search estimation methods.

Parallel Computing with Nonlinear least squares and Gradient descent Methods

When you select Gradient descent as the estimation method, the model is simulated during the following computations:

- Objective value computation One simulation per iteration
- Objective gradient computations Two simulations for every tuned parameter per iteration
- Line search computations Multiple simulations per iteration

The total time, *Ttotal*, taken per iteration to perform these simulations is given by the following equation:

$$Ttotal = T + (Np \times 2 \times T) + (Nls \times T) = T \times (1 + (2 \times Np) + Nls)$$

where T is the time taken to simulate the model and is assumed to be equal for all simulations, Np is the number of parameters to estimate, and Nls is the number of line searches. Nls is difficult to estimate and you generally assume it to be equal to one, two, or three.

When you use parallel computing, the software distributes the simulations required for objective gradient computations. The simulation time taken per iteration when the gradient computations are performed in parallel, *TtotalP*, is approximately given by the following equation:

$$TtotalP = T + (ceil\left(\frac{Np}{Nw}\right) \times 2 \times T) + (Nls \times T) = T \times (1 + 2 \times ceil\left(\frac{Np}{Nw}\right) + Nls)$$

where *Nw* is the number of MATLAB workers.

Note The equation does not include the time overheads associated with configuring the system for parallel computing and loading Simulink software on the remote MATLAB workers.

The expected reduction of the total estimation time is given by the following equation:

$$\frac{TtotalP}{Ttotal} = \frac{1 + 2 \times ceil\left(\frac{Np}{Nw}\right) + Nls}{1 + (2 \times Np) + Nls}$$

For example, for a model with $N_p=3$, $N_w=4$, and $N_{ls}=3$, the expected reduction of the total estimation time equals $\frac{1+2 \times ceil(\frac{3}{4})+3}{1+(2 \times 3)+3} = 0.6$.

Parallel Computing with the Pattern search Method

The Pattern search method uses search and poll sets to create and compute a set of candidate solutions at each estimation iteration.

The total time, *Ttotal*, taken per iteration to perform these simulations, is given by the following equation:

$$Ttotal = (T \times Np \times Nss) + (T \times Np \times Nps) = T \times Np \times (Nss + Nps)$$

where *T* is the time taken to simulate the model and is assumed to be equal for all simulations, *Np* is the number of parameters to estimate, *Nss* is a factor for the search set size, and *Nps* is a factor for the poll set size. *Nss* and *Nps* are typically proportional to *Np*.

When you use parallel computing, Simulink Design Optimization software distributes the simulations required for the search and poll set computations, which are evaluated in separate parfor loops. The simulation time taken per iteration when the search and poll sets are computed in parallel, *TtotalP*, is given by the following equation:

$$\begin{split} TtotalP &= (T \times ceil(Np \times \frac{Nss}{Nw})) + (T \times ceil(Np \times \frac{Nps}{Nw})) \\ &= T \times (ceil(Np \times \frac{Nss}{Nw}) + ceil(Np \times \frac{Nps}{Nw})) \end{split}$$

where *Nw* is the number of MATLAB workers.

Note The equation does not include the time overheads associated with configuring the system for parallel computing and loading Simulink software on the remote MATLAB workers.

The expected speed up for the total estimation time is given by the following equation:

$$\frac{TtotalP}{Ttotal} = \frac{ceil(Np \times \frac{Nss}{Nw}) + ceil(Np \times \frac{Nps}{Nw})}{Np \times (Nss + Nps)}$$

For example, for a model with N_p=3, N_w=4, N_{ss}=15, and N_{ps}=2, the expected speedup equals $\frac{ceil(3 \times \frac{15}{4}) + ceil(3 \times \frac{2}{4})}{3 \times (15 + 2)} = 0.27.$

Using the Pattern search method with parallel computing may not speed up the estimation time. When you do not use parallel computing, the method stops searching for a candidate solution at each iteration as soon as it finds a solution better than the current solution. When you use parallel computing, the candidate solution search is more comprehensive. Although the number of iterations may be larger, the estimation without using parallel computing may be faster.

See Also

Related Examples

• "Use Parallel Computing for Parameter Estimation" on page 2-33

Use Parallel Computing for Parameter Estimation

Configure Your System for Parallel Computing

You can speed up parameter estimation using parallel computing on multicore processors or multiprocessor networks. Use parallel computing with the **Parameter Estimator** and sdo.optimize to estimate parameters using the fmincon, lsqnonlin, and patternsearch methods. Parallel computing is not supported for the fminsearch (**Simplex search**) method.

When you estimate model parameters using parallel computing, the software uses the available parallel pool. If none is available, and you select **Automatically create a parallel pool** in your Parallel Computing Toolbox preferences, the software starts a parallel pool using the settings in those preferences. To open a parallel pool that uses a specific cluster profile, use:

parpool(MyProfile);

MyProfile is the name of a cluster profile.

For information regarding creating a cluster profile, see "Add and Modify Cluster Profiles" (Parallel Computing Toolbox).

Model Dependencies

Model dependencies are any referenced models, data such as model variables, S-functions, and additional files necessary to run the model. Before starting the optimization, verify that the model dependencies are complete. Otherwise, you may get unexpected results.

Making Model Dependencies Accessible to Remote Workers

When you use parallel computing, the Simulink Design Optimization software helps you identify model dependencies. To do so, the software uses the Dependency Analyzer. The dependency analysis may not find all the files required by your model. To learn more, see "Dependency Analyzer Scope and Limitations". If your model has dependencies that are undetected or inaccessible by the parallel pool workers, then add them to the list of model dependencies.

The dependencies are made accessible to the parallel pool workers by specifying one of the following:

- File dependencies: the model dependency files are copied to the parallel pool workers.
- Path dependencies: the paths to the model dependencies are added to the paths of the parallel pool workers. If you are working in a multi-platform scenario, ensure that the paths are compatible across platforms.

Using file dependencies is recommended, however, in some cases it can be better to choose path dependencies. For example, if parallel computing is set up on a local multi-core computer, using path dependencies is preferred as using file dependencies creates multiple copies of the dependent files on the local computer.

For more information, see:

- "Estimate Parameters Using Parallel Computing in the Parameter Estimator App" on page 2-34 (Not supported in Simulink Online™.)
- "Estimate Parameters Using Parallel Computing (Code)" on page 2-35

4

Estimate Parameters Using Parallel Computing in the Parameter Estimator App

To estimate model parameters using parallel computing in the **Parameter Estimator**:

1 Ensure that the software can access parallel pool workers that use the appropriate cluster profile.

For more information, see "Configure Your System for Parallel Computing" on page 2-33.

- 2 Open the **Parameter Estimator** for the Simulink model.
- **3** Configure the estimation data, estimation parameters and states, and, optionally, estimation settings.

For more information, see "Specify Estimation Data" on page 2-4, "Specify Parameters for Estimation" on page 2-7, and "Specify Estimation Options" on page 2-17.

- On the **Parameter Estimation** tab, click **More Options** to open the **Estimation Options** dialog box.
- 5 Select the **Parallel Options** tab.

Estimation Options	×
General Options Optimization Options Parallel Options	
Use the parallel pool during optimization	
Model file dependencies Model path dependencies	
No model file dependencies specified.	
Add file dependency Sync file dependencies from mo	odel
OK Cancel Help	

6 Select the **Use the parallel pool during optimization** check box.

This option checks for dependencies in your Simulink model. The file dependencies are displayed in the **Model file dependencies** list box, and corresponding path to the files in **Model path dependencies**. The files listed in **Model file dependencies** are copied to the remote workers.

Note The automatic dependencies check may not detect all the dependencies in your model.

For more information, see "Model Dependencies" on page 2-33. In this case, add the undetected dependencies manually.

7 Add any file dependencies that the automatic check does not detect.

Specify the files in the **Model file dependencies** list box separated by semicolons or on separate lines.

Estimation Options ×
General Options Optimization Options Parallel Options
✓ Use the parallel pool during optimization
Model file dependencies Model path dependencies
C:\myproject\file1.slx; C:\myproject2\file2.m C:\estim_common_dependencies\file3.mat
Add file dependency Sync file dependencies from model
OK Cancel Help

Alternatively, click Add file dependency to open a dialog box, and select the file to add.

Note If you do not want to copy the files to the remote workers, delete all entries in the **Model** file dependencies list box. Populate the **Model path dependencies** list box by clicking the **Sync path dependencies from model**, and add any undetected path dependencies. In addition, in the list box, update the paths on local drives to make them accessible to remote workers. For example, change C:\ to \\\hostname\\C\$\\.

8 If you modify the Simulink model, resync the dependencies to ensure that any new dependencies are detected. Click **Sync file dependencies from model** in the **Parallel Options** tab to rerun the automatic dependency check for your model.

This action updates the **Model file dependencies** list box with any new file dependency found in the model.

- 9 Click OK.
- **10** In the **Parameter Estimation** tab, click **Estimate** to estimate the model parameters using parallel computing.

For information on troubleshooting problems related to estimation using parallel computing, see "Troubleshooting" on page 2-36.

Estimate Parameters Using Parallel Computing (Code)

To use parallel computing for parameter estimation at the command line:

1 Ensure that the software can access parallel pool workers that use the appropriate cluster profile.

For more information, see "Configure Your System for Parallel Computing" on page 2-33.

2 Open the model.

- **3** Configure an estimation experiment. For example, see "Estimate Model Parameter Values (Code)" on page 2-58.
- 4 Enable parallel computing using an optimization option set, opt.

```
opt = sdo.OptimizeOptions;
opt.UseParallel = true;
```

5 Find the model dependencies.

```
[dirs,files] = sdo.getModelDependencies(modelname)
```

Note sdo.getModelDependencies may not detect all the dependencies in your model. For more information, see "Model Dependencies" on page 2-33. In this case, add the undetected dependencies manually.

6 Modify files to include any file dependencies that sdo.getModelDependencies does not detect.

```
files = vertcat(files, 'C:\matlab\work\filename.m')
```

Note If you do not want to copy the files to the remote workers, use the path dependencies. Add any undetected path dependencies to dirs and update the paths on local drives to make them accessible to remote workers. See sdo.getModelDependencies for more details.

7 Add the file dependencies for optimization.

opt.ParallelFileDependencies = files;

8 Run the optimization.

[pOpt,opt_info] = sdo.optimize(opt_fcn,param,opt);

For information on troubleshooting problems related to estimation using parallel computing, see "Troubleshooting" on page 2-36.

Troubleshooting

Why Are the Estimation Results With and Without Parallel Computing Different?

- Different numerical precision on the client and worker machines can produce marginally different simulation results. Thus, the optimization method can take a different solution path and produce a different result.
- When you use parallel computing with the Pattern search method, the search is more comprehensive and can result in a different solution. To learn more, see "Parallel Computing with the Pattern search Method" on page 2-31.

Why Didn't the Estimation Speed up Using Parallel Computing?

- When you estimate a few parameters or when the model does not take long to simulate, you do not see a speedup in the estimation time. In such cases, the overhead associated with creating and distributing the parallel tasks outweighs the benefits of running the estimation in parallel.
- Using the Pattern search method with parallel computing might not speed up the optimization time. Without parallel computing, the method stops the search at each iteration as soon as it finds a solution better than the current solution. The candidate solution search is more comprehensive

when you use parallel computing. Although the number of iterations might be larger, the optimization without using parallel computing might be faster.

To learn more about the expected speedup, see "Parallel Computing with the Pattern search Method" on page 2-31.

Why Doesn't the Estimation Using Parallel Computing Make Any Progress?

To troubleshoot the problem:

- **1** Run the optimization for a few iterations without parallel computing to see if the optimization progresses.
- 2 Check whether the remote workers have access to all model dependencies. Model dependencies include data variables and files required by the model to run.

To learn more, see "Model Dependencies" on page 2-33.

Why Does the Estimation Using Parallel Computing Continue When I Click Stop?

When you use parallel computing with the Pattern search method, the software must wait until the current optimization iteration completes before it notifies the workers to stop. The optimization does not terminate immediately when you click **Stop**, and, instead, appears to continue running.

See Also

sdo.optimize | sdo.OptimizeOptions | sdo.getModelDependencies | parpool

More About

- "Speed Up Parameter Estimation Using Parallel Computing" on page 2-30
- "Ways to Speed Up Design Optimization Tasks"

Estimating Initial Conditions for Blocks with External Initial Conditions

When an integrator block uses an initial-condition port, which you specify by an IC block, you cannot estimate the initial conditions of the integrator using Simulink Design Optimization software. Estimation is not possible because external initial conditions have priority over the initial conditions of a specific block to maintain the integrity of the model.

To tune the initial conditions of an integrator block with external initial conditions, you must modify the model to make the external signal into a tunable parameter. For example, you can set the IC block that feeds into the integrator to be a tunable variable and estimate it.

See Also

Save and Load Estimation Sessions

This topic shows how to save and load estimation sessions in the **Parameter Estimator**.

Structure of an Estimation Session

The **Parameter Estimator** stores and organizes data from a given Simulink model inside a *session*. An estimation session includes the following information:

- One or more estimation or validation experiments along with their configurations
- Parameter information
- Estimation results
- Estimation settings
- Plots Changes to plots layout and plot characteristics, such as axis limits, line colors, are not included.

The default session name is the same as the Simulink model name. The session name is shown on the title pane of the **Parameter Estimator**.

Save Parameter Estimator App Sessions

Saving a session lets you reuse your estimation settings and results later. You can save the session as a MAT-file or workspace variable:

- To save the session as a MAT-file, in the **Parameter Estimation** tab, in the **Save Session** dropdown list, click **Save to file**. A window opens where you specify the MAT-file name.
- To save the session as a model or MATLAB workspace variable, select **Save to model workspace** or **Save to MATLAB workspace** in the **Save Session** drop-down list.

Load Parameter Estimator App Sessions

To load a previously saved MAT-file or workspace sessions:

- **1** Open a **Parameter Estimator** for the model.
- **2** To load a MAT-file, in the **Parameter Estimation** tab click the **Open Session** drop-down list, and select **Open from file**. A window opens where you select the MAT-file to load.

To load a workspace variable, select **Open from model workspace** or **Open from MATLAB workspace** in the **Open Session** drop-down list.

Load Legacy Projects

Open legacy projects that are in MAT-files by selecting Open from file from the **Open Session** drop-down list. The **Parameter Estimator** recognizes and converts them into the new session format.

See Also

More About

• "What Is an Experiment?" on page 2-3

How the Software Formulates Parameter Estimation as an Optimization Problem

Overview of Parameter Estimation as an Optimization Problem

When you perform parameter estimation, the software formulates an optimization problem. The optimization problem solution is the estimated parameter values set. This optimization problem consists of:

- x Design variables. The model parameters and initial states to be estimated.
- F(x) Objective function. A function that calculates a measure of the difference between the simulated and measured responses. Also called cost function or estimation error.
- (Optional) $\underline{x} \le x \le \overline{x}$ Bounds. Limits on the estimated parameter values.
- (Optional) C(x) Constraint function. A function that specifies restrictions on the design variables.

The optimization solver tunes the values of the design variables to satisfy the specified objectives and constraints. The exact formulation of the optimization depends on the optimization method that you use.

Cost Function

The software tunes the model parameters to obtain a simulated response (y_{sim}) that tracks the measured response or reference signal (y_{ref}) . To do so, the solver minimizes the *cost function* or *estimation error*, a measure of the difference between the simulated and measured responses. The cost function, F(x), is the objective function of the optimization problem.

Types

The raw estimation error, e(t), is defined as:

 $e(t) = y_{ref}(t) - y_{sim}(t)$

e(t) is also referred to as the error residuals or, simply, residuals.

Simulink Design Optimization software provides you the following cost functions to process e(t):

Cost Function		Option Name in GUI or Command Line
Sum squared error (default)	$F(x) = \sum_{t=0}^{t_N} e(t) \times e(t)$ <i>N</i> is the number of samples.	'SSE'
Sum absolute error	$F(x) = \sum_{t=0}^{t_N} \left e(t) \right $ N is the number of samples.	'SAE'

Cost Function	Formulation	Option Name in GUI or Command Line
Raw error	$F(x) = \begin{bmatrix} e(0) \\ \vdots \\ e(N) \end{bmatrix}$ N is the number of samples.	'Residuals' This option is available only at the command line.
Custom function	N/A	This option is available only at the command line.

Time Base

The software evaluates the cost function for a specific time interval. This interval is dependent on the measured signal time base and the simulated signal time base.

- The measured signal time base consists of all the time points for which the measured signal is specified. In case of multiple measured signals, this time base is the union of the time points of all the measured signals.
- The simulated signal time base consists of all the time points for which the model is simulated.

If the model uses a variable-step solver, then the simulated signal time base can change from one optimization iteration to another. The simulated and measured signal time bases can be different. The software evaluates the cost function for only the time interval that is common to both. By default, the software uses only the time points specified by the measured signal in the common time interval.

- In the GUI, you can specify the simulation start and stop times in the **Simulation time** area of the **Simulation Options** dialog box.
- At the command line, the software specifies the simulation stop time as the last point of the measured signal time base. For example, the following code simulates the model until the end time of the longest running output signal of exp, an sdo.Experiment object:

```
sim_obj = createSimulator(exp);
sim_obj = sim(sim_obj);
```

sim_obj contains the simulated response for the model associated with exp.

Bounds and Constraints

You can specify bounds for the design variables (estimated model parameters), based on your knowledge of the system. Bounds are expressed as:

 $\underline{x} \leq x \leq \overline{x}$

 \underline{x} and \overline{x} are the lower and upper bounds for the design variables.

For example, in a battery discharging experiment, the estimated battery initial charge must be greater than zero and less than Inf. These bounds are expressed as:

 $0 < x < \infty$

For an example of how to specify these types of bounds, see "Estimate Model Parameters and Initial States (Code)" on page 2-67.

You can also specify other constraints, C(x), on the design variables at the command line. C(x) can be linear or nonlinear and can describe equalities or inequalities. C(x) can also specify multiparameter constraints. For example, for a simple friction model, C(x) can specify that the static friction coefficient must be greater than or equal to the dynamic friction coefficient. One way of expressing this constraint is:

$$C(x): x_1 - x_2$$
$$C(x) \le 0$$

 x_1 and x_2 are the dynamic and static friction coefficients, respectively.

For an example of how to specify a constraint, see "Estimate Model Parameters with Parameter Constraints (Code)" on page 2-124.

Optimization Methods and Problem Formulations

An optimization problem can be one of the following types:

- Minimization problem Minimizes an objective function, F(x). You specify the measured signal that you want the model output to track. You can optionally specify bounds for the estimated parameters.
- Mixed minimization and feasibility problem Minimizes an objective function, F(x), subject to specified bounds and constraints, C(x). You specify the measured signal that you want the model to track and bounds and constraints for the estimated parameters.
- Feasibility problem Finds a solution that satisfies the specified constraints, C(x). You specify only bounds and constraints for the estimated parameters. This type of problem is not common in parameter estimation.

The optimization method that you specify determines the formulation of the estimation problem. The software provides the following optimization methods:

Optimization Method Name	Description	Optimization Problem Formulation
 User interface: Nonlinear Least Squares Command line: 'lsqnonlin' 	Minimizes the squares of the residuals, recommended method for parameter estimation. This method requires a vector of error residuals, computed using a fixed time base. Do not use this approach if you have a scalar cost function or if the number of error residuals can change from one iteration to another. This method uses the Optimization Toolbox [™] function, lsqnonlin.	Minimization Problem $ \begin{array}{l} \min_{x} \ F(x)\ _{2}^{2} = \min_{x} \left(f_{1}(x)^{2} + f_{2}(x)^{2} + \ldots + f_{n}(x)^{2}\right) \\ s.t. \underline{x} \leq x \leq \overline{x} \\ f_{1}(x), f_{2}(x), \ldots, f_{n}(x) \text{ represent residuals. } n \text{ is the number of samples.} \\ \begin{array}{l} \text{Mixed Minimization and Feasibility Problem} \\ \text{Not supported.} \\ \end{array} $ Not supported.

Optimization Method Name	Description	Optimization Problem Formulation
 User interface: Gradient Descent Command line: 'fmincon' 	 General nonlinear solver, uses the cost function gradient. Use this approach if you want to specify one or any combination of the following: Custom cost functions Parameter-based constraints Signal-based constraints This method uses the Optimization Toolbox function, fmincon. For information on how the gradient is computed, see "Gradient Computations" on page 2-57. 	Minimization Problem $\min_{x} F(x)$ s.t. $\underline{x} \le x \le \overline{x}$ Mixed Minimization and Feasibility Problem $\min_{x} F(x)$ s.t. $C(x) \le 0$ $\underline{x} \le x \le \overline{x}$ Note When tracking a reference signal, the software ignores the maximally feasible solution option. Feasibility Problem • If you select the maximally feasible solution option (i.e., the optimization continues after an initial feasible solution is found), the software uses the following problem formulation: $\min_{\substack{[x,y]\\ s.t. \ C(x) \le \gamma\\ \underline{x} \le x \le \overline{x}\\ \gamma \le 0}$ γ is a slack variable that permits a feasible solution with $C(x) \le \gamma$ rather than $C(x) \le 0$. • If you do not select the maximally feasible solution option (i.e., the optimization terminates as soon as a feasible solution is found), the software uses the following problem formulation: $\min_{x} 0$ x $s.t. \ C(x) \le 0$ $\underline{x} \le x \le \overline{x}$

Optimization Method Name	Description	Optimization Problem Formulation
 User interface: Simplex Search Command line: 'fminsearch' 	Based on the Nelder-Mead algorithm, this approach does not use the cost function gradient. Use this approach if your cost function or constraints are not continuous or differentiable. This method uses the Optimization Toolbox functions, fminsearch and fminbnd. fminbnd is used if one scalar parameter is being optimized. Otherwise, fminsearch is used. You cannot specify parameter bounds, $\underline{x} \le x \le \overline{x}$, with fminsearch.	Minimization Problem $\min_{x} F(x)$ Mixed Minimization and Feasibility Problem The software formulates the problem in two steps: 1 Finds a feasible solution. $\min_{x} \max(C(x))$ 2 Minimizes the objective. The software uses the results from step 1 as initial guesses. It maintains feasibility by introducing a discontinuous barrier in the optimization objective. $\min_{x} \Gamma(x)$ where $\Gamma(x) = \begin{cases} \infty & \text{ifmax}(C(x)) > 0 \\ F(x) & \text{otherwise.} \end{cases}$ Feasibility Problem $\min_{x} \max(C(x))$

Optimization Method Name	Description	Optimization Problem Formulation
 User interface: Pattern Search Command line: 'patternsearch' 	Direct search method, based on the generalized pattern search algorithm, this method does not use the cost function gradient. Use this approach if your cost function or constraints are not continuous or differentiable. This method uses the Global Optimization Toolbox function, patternsearch.	Minimization Problem $\min_{x} F(x)$ s.t. $\underline{x} \le x \le \overline{x}$ Mixed Minimization and Feasibility Problem The software formulates the problem in two steps: 1 Finds a feasible solution. $\min_{x} \max(C(x))$ s.t. $\underline{x} \le x \le \overline{x}$ 2 Minimizes the objective. The software uses the results from step 1 as initial guesses. It maintains feasibility by introducing a discontinuous barrier in the optimization objective. $\min_{x} \Gamma(x)$ s.t. $\underline{x} \le x \le \overline{x}$ where $\Gamma(x) = \begin{cases} \infty & \text{ifmax}(C(x)) > 0 \\ F(x) & \text{otherwise.} \end{cases}$ Feasibility Problem $\min_{x} \max(C(x))$ s.t. $\underline{x} \le x \le \overline{x}$

See Also

sdo.SimulationTest|sdo.Experiment|sdo.requirements.SignalTracking| evalRequirement|lsqnonlin|fmincon|fminsearch|fminbnd|patternsearch

Related Examples

- "Estimate Model Parameter Values (Code)" on page 2-58
- "Estimate Model Parameters with Parameter Constraints (Code)" on page 2-124
- "Estimate Parameters from Measured Data"

More About

• "Write a Cost Function" on page 2-49

Specify Steady-State Operating Point for Parameter Estimation

What is a Steady-State Operating Point?

An *operating point* of a dynamic system defines the states and root-level input signals of the model at a specific time. For example, in a car engine model, variables such as engine speed, throttle angle, engine temperature, and surrounding atmospheric conditions typically describe the operating point.

A *steady-state operating point* of a model, also called an equilibrium or *trim* condition, includes state variables that do not change with time.

A model can have several steady-state operating points. For example, a hanging damped pendulum has two steady-state operating points at which the pendulum position does not change with time. A *stable steady-state operating point* occurs when a pendulum hangs straight down. When the pendulum position deviates slightly, the pendulum always returns to equilibrium. In other words, small changes in the operating point do not cause the system to leave the region of good approximation around the equilibrium value.

When using optimization search to compute operating points for nonlinear systems, your initial guesses for the states and input levels must be near the desired operating point to ensure convergence.

When linearizing a model with multiple steady-state operating points, it is important to have the right operating point. For example, linearizing a pendulum model around the stable steady-state operating point produces a stable linear model, whereas linearizing around the unstable steady-state operating point produces an unstable linear model.

For more information on operating points, see "What Is an Operating Point?" (Simulink Control Design) and "What Is a Steady-State Operating Point?" (Simulink Control Design).

Setting up a Steady-State Operating Point

This topic shows how to setup a steady-state operating point in **Parameter Estimator**. To improve the fit between the model and measured data, the model must be set to steady-state when parameters are estimated.

- **1** Open the **Parameter Estimator** and setup your experiment using the steps outlined in "Estimate Model Parameter Values (GUI)" on page 2-143.
- 2 In the toolstrip, click **More Options** and select **Operating Point Options** from the drop down menu.



3 The following **Operating Point** dialog box opens.



The **Estimate at steady-state** option is checked by default when you open the operating point dialog. Select the appropriate experiment to change the parameters for from the Experiment: drop down menu. Use the **States**, **Inputs** and **Outputs** tabs to specify the known parameters, bounds and deviations. For instance, there is one state in the above figure. Use the operating point dialog to specify that this state should be treated as an unknown, and it should be set to steady state. During parameter estimation, the operating point computation will vary this state to set it at steady-state.



You can also sync operating point specifications from your Simulink model or another experiment using the Sync with specification from: drop-down list. After you make your selection,

click on the subtraction to copy the parameters.

The Simulink Design Optimization software uses optimization methods to search for operating 4 points in a model. Use the **Options** tab of the dialog to specify these optimization methods. These options specify the optimization algorithm, tolerances, and stopping conditions. For instance, the option Gradient descent with projection is often used to find the operating point for systems that use physical modeling. For more information, click on the - button.

Delevate at steady at						
Speinest Exp	×	Spec with specification for	et Sinulink &	Nodel ~	5	
Spectrations Option						
Optimization Method						
Optimization Mathematica	e Brebert ber	et all giving of	Algorithms	Active-Set.		
Optimization Options						
Mainus charge:	M		Main	un fun evelu	2000	
Minimum change			Macim	un bestiens	400	
Function Inference	74-06		Passes	ar bie aver	1+01	
Contraint Talwarrow	14-06					
Carlos analysis a	notian					
Deploy results	CW		e.			
Cutton Optimization	-					
Objective function						
Constraint Function						
Mapping function						

5 Having specified the operating point parameters, continue with the estimation workflow as described in "Estimate Model Parameter Values (GUI)" on page 2-143.

See Also

More About

- "What Is an Operating Point?" (Simulink Control Design)
- "What Is a Steady-State Operating Point?" (Simulink Control Design) .
- "Set Model to Steady-State When Estimating Parameters (GUI)" on page 2-117 .
- "Set Model to Steady-State When Estimating Parameters (Code)" on page 2-97 .

Write a Cost Function

A cost function is a MATLAB function that evaluates your design requirements using design variable values. After writing and saving the cost function, you can use it for estimation, optimization, or sensitivity analysis at the command line.

When you optimize or estimate model parameters, you provide the saved cost function as an input to sdo.optimize. At every optimization iteration, sdo.optimize calls this function and uses the function output to decide the optimization direction. When you perform sensitivity analysis using sdo.evaluate, you generate sample values of the design variables and evaluate the cost function for each sample value using sdo.evaluate.

Anatomy of a Cost Function

To understand the parts of a cost function, consider the following sample function myCostFunc. For a design variable x, myCostFunc evaluates the objective x^2 and the nonlinearity constraint $x^2-4x+1 \le 0$.

```
function [vals,derivs] = myCostFunc(params)
% Extract the current design variable values from the parameter object, params.
x = params.Value;
% Compute the requirements (objective and constraint violations) and
% assign them to vals, the output of the cost function.
vals.F = x.^2;
vals.Cleq = x.^2-4*x+1;
% Compute the cost and constraint derivatives.
derivs.F = 2*x;
derivs.Cleq = 2*x-4;
end
```

This cost function performs the following tasks:

1 Specifies the inputs of the cost function.

A cost function must have as input, params, a vector of the design variables to be estimated, optimized, or used for sensitivity analysis. Design variables are model parameter objects (param.Continuous objects) or model initial states (param.State objects).

Since the cost function is called repeatedly during estimation, optimization, or evaluation, you can specify additional inputs to the cost function to help reduce code redundancy and computation cost. For more information, see "Specify Inputs of the Cost Function" on page 2-50.

2 Computes the requirements.

Requirements can be objectives and constraints based on model parameters, model signals, or linearized models. In this sample cost function, the requirements are based on the design variable x, a model parameter. The cost function first extracts the current values of the design variables and then computes the requirements.

For information about computing requirements based on model parameters, model signals, or linearized models, see "Compute Requirements" on page 2-51.

3 Specifies the requirement values as outputs, vals and derivs, of the cost function.

A cost function must return vals, a structure with one or more fields that specify the values of the objective and constraint violations.

The output can optionally include derivs, a structure with one or more fields that specify the values of the gradients of the objective and constraint violations. For more information, see "Specify Outputs of the Cost Function" on page 2-52.

After saving the cost function as a MATLAB file myCostFunc.m, to perform the optimization, use the cost function as an input to sdo.optimize.

[param_opt,opt_info] = sdo.optimize(@myCostFunc,params)

When performing sensitivity analysis, to compute the requirements in the cost function for a range of design variable sample values paramsamples, use the cost function as an input to sdo.evaluate.

[y,info] = sdo.evaluate(@myCostFunc,paramsamples)

Specify Inputs of the Cost Function

The sample cost function myCostFunc takes one input, params.

```
function [vals,derivs] = myCostFunc(params)
```

A cost function must have as input, params, a vector of the design variables to be estimated, optimized, or used for sensitivity analysis. Design variables are model parameter objects (param.Continuous objects) or model initial states (param.State objects). You obtain params by using the sdo.getParameterFromModel and sdo.getStateFromModel commands.

Specify Multiple Inputs

Because the cost function is called repeatedly during estimation, optimization, or evaluation, you can specify additional inputs to the cost function to help reduce code redundancy and computation cost. However, sdo.optimize and sdo.evaluate accept a cost function with only one input argument. To use a cost function that accepts more than one input argument, you use an anonymous function. Suppose that the myCostFunc_multi_inputs.m file specifies a cost function that takes params and argl as inputs. For example, you can make the model name an input argument, argl, and configure the cost function to be used for multiple models. Then, assuming that all input arguments are variables in the workspace, specify an anonymous function myCostFunc2, and use it as an input to sdo.optimize or sdo.evaluate.

```
myCostFunc2 = @(params) myCostFunc_multi_inputs(params,arg1);
[param_opt,opt_info] = sdo.optimize(@myCostFunc2,params);
```

You can also specify additional inputs using convenience objects provided by Simulink Design Optimization software. You create convenience objects once and pass them as an input to the cost function to reduce code redundancy and computation cost.

For example, you can create a simulator (sdo.SimulationTest object) to simulate your model using alternative model parameters without modifying the model, and pass the simulator to your cost function.

```
simulator = sdo.SimulationTest(model)
myCostFunc2 = @(params) myCostFunc_multi_inputs(params,arg1,arg2,simulator);
[param_opt,opt_info] = sdo.optimize(@myCostFunc2,params);
```

For more information about the available convenience objects, see "Convenience Objects as Additional Inputs" on page 2-54. For an example, see "Design Optimization to Meet a Custom Objective (Code)" on page 3-118.

Compute Requirements

The sample cost function myCostFunc computes the requirements based on a model parameter x. In general, requirements can be objectives or constraints based on model parameters, model signals, or linearized models. As seen in myCostFunc, you can use MATLAB functions to compute the requirements. You can also use the requirements objects that Simulink Design Optimization software provides. These objects enable you to specify requirements such as step-response characteristics, gain and phase margin bounds, and Bode magnitude bounds. You can use the evalRequirement method of these objects to evaluate the objective and constraint violations. For a list of available requirement objects, see "Convenience Objects as Additional Inputs" on page 2-54.

Parameter-Based Requirements

If you have requirements on model parameters, in the cost function you first extract the current parameter values, and then compute the requirements.

1 Extract the current parameter value from params.

x = params.Value;

2 Compute the requirement, and specify it as vals, the output of the cost function.

Suppose that the objective to be computed is x^2 and the constraint is the nonlinearity constraint x^2-4x+1 .

vals.F = x.^2; vals.Cleq = x.^2-4*x+1;

In the context of optimization, x^2 is minimized subject to satisfying the constraints. For sensitivity analysis, the cost and constraints are evaluated for all values of the parameter params.

For more information about the output of a cost function, see "Specify Outputs of the Cost Function" on page 2-52.

For an example of a cost function with a parameter-based requirement, see "Design Optimization to Meet a Custom Objective (Code)" on page 3-118. In this example, you minimize the cylinder cross-sectional area, a design variable in a hydraulic cylinder.

Model Signal Requirements

If you have requirements on model signals, in the cost function you simulate the model using current design variable values, extract the signal of interest, and compute the requirement on the signal.

- 1 Simulate the model using the current design variable values in param. There are multiple ways to simulate your model:
 - Using sdo.SimulationTest object If an sdo.SimulationTest object, simulator, is a cost function input, you update the model parameter values using the Parameters property of the simulator. Then use sim to simulate the model.

```
simulator.Parameters = params;
simulator = sim(simulator);
```

For an example, see "Design Optimization to Meet a Custom Objective (Code)" on page 3-118.

• Using sdo.Experiment object — If you are performing parameter estimation based on input-output data defined in an sdo.Experiment object, exp, update the design variable

values associated with the experiment using the setEstimatedValues method. Create a simulator using the createSimulator method, and simulate the model using the updated model configuration.

```
exp = setEstimatedValues(exp,params);
simulator = createSimulator(exp,simulator);
simulator = sim(simulator);
```

For an example, see "Estimate Model Parameters Per Experiment (Code)" on page 2-86.

• Using sim command — If you are not using sdo.SimulationTest or sdo.Experiment objects, use sdo.setValueInModel to update the model parameter values, and then call sim to simulate the model.

```
sdo.setValueInModel('model_name',param);
LoggedData = sim('model_name');
```

2 Extract the logged signal of interest, SignalOfInterest.

Use the SignalLoggingName model parameter to get the simulation log name.

logName = get_param(simulator.ModelName,'SignalLoggingName'); simLog = get(simulator.LoggedData,logName); Sig = get(simLog,'SignalOfInterest')

3 Evaluate the requirement, and specify it as the output of the cost function.

For example, if you specified a step-response bound on a signal using a sdo.requirements.StepResponseEnvelope object, StepResp, you can use the evalRequirement method of the object to evaluate the objective and constraint violations.

vals.Cleq = evalRequirement(StepResp,SignalOfInterest.Values);

For an example, see "Design Optimization to Meet Step Response Requirements (Code)". For more information about the output of a cost function, see "Specify Outputs of the Cost Function" on page 2-52.

Linearization-Based Requirements

If you are optimizing or evaluating frequency-domain requirements, in the cost function you linearize the model, and compute the requirement values. Linearizing the model requires Simulink Control Design[™] software.

Use the SystemLoggingInfo property of sdo.SimulationTest to specify linear systems to log when simulating the model. For an example, see "Design Optimization to Meet Frequency-Domain Requirements (Code)" on page 3-216. Alternatively, use linearize to linearize the model.

Note For models in Simulink fast restart mode, you cannot use the linearize command.

Specify Outputs of the Cost Function

The sample cost function myCostFunc outputs vals, a structure with fields that specify the values of the objective and constraint violations. The second output is derivs, a structure with fields that specify the derivatives of the objective and constraint.

function [vals,derivs] = myCostFunc(params)

A cost function must output vals, a structure with one or more of the following fields that specify the values of the objective and constraint violations:

- F Value of the cost or objective evaluated at param.
- Cleq Value of the nonlinear inequality constraint violations evaluated at param. For optimization, the solver ensures Cleq ≤ 0.
- Ceq Value of the nonlinear equality constraint violations evaluated at param. For optimization, the solver ensures Ceq = 0.
- leq Value of the linear inequality constraint violations evaluated at param. For optimization, the solver ensures leq ≤ 0.
- eq Value of the linear equality constraint violations evaluated at param. For optimization, the solver ensures eq = 0.
- Log Additional optional information from evaluation.

If you have multiple constraints of one type, concatenate the values into a vector, and specify this vector as the corresponding field value. For instance, if you have a hydraulic cylinder, you can specify nonlinear inequality constraints on the piston position (Cleq1) and cylinder pressure (Cleq2). In this case, specify the Cleq field of the output structure vals as:

vals.Cleq = [Cleq1; Cleq2];

For an example, see "Design Optimization to Meet a Custom Objective (Code)" on page 3-118.

By default, the sdo.optimize command computes the objective and constraint gradients using numeric perturbation. You can also optionally return the gradients as an additional cost function output, derivs. Where derivs must contain the derivatives of all applicable objective and constraint violations and is specified as a structure with one or more of the following fields:

- F Derivatives of the cost or objective.
- Cleq Derivatives of the nonlinear inequality constraints.
- Ceq Derivatives of the nonlinear equality constraints.

The derivatives are not required for sensitivity analysis. For estimation or optimization, specify the GradFcn property of sdo.OptimizeOptions as 'on'.

Multiple Objectives

Simulink Design Optimization software does not support multi-objective optimization. However, you can return the objective value (vals.F) as a vector that represents the multiple objective values. The software sums the elements of the vector and minimizes this sum. The exception to this behavior is in the use of the nonlinear least squares (lsqnonlin) optimization method. The nonlinear least squares method, used for parameter estimation, requires that you return the error residuals as a vector. In this case, the software minimizes the sum square of this vector. If you are tracking multiple signals and using lsqnonlin, then concatenate the error residuals for the different signals into one vector. Specify this vector as the F field value.

For an example of single-objective optimization using the gradient descent method, see "Design Optimization to Meet a Custom Objective (Code)" on page 3-118.

For an example of multiple-objective optimization using the nonlinear least squares method, see "Estimate Model Parameters Per Experiment (Code)" on page 2-86.

Convenience Objects as Additional Inputs

A cost function must have as input, params, a vector of the design variables to be estimated, optimized, or used for sensitivity analysis. You can specify additional inputs to the cost function using convenience objects provided by the Simulink Design Optimization software. You create convenience objects once and pass them as an input to the cost function to reduce code redundancy and computation cost. For information about specifying additional inputs to the cost function, see "Specify Multiple Inputs" on page 2-50.

Convenience Object	Class Name	Description
Simulator objects	sdo.SimulationTest	Use the simulator object to simulate the model using alternative inputs, model parameters, and initial-state values without modifying the model. Use the SystemLoggingInfo property of sdo.SimulationTest to specify linear systems to log when you have frequency-domain requirements. In the cost function, use the sim method to simulate the model. Then extract the model response from the object, and evaluate the requirements. For an example, see "Design Optimization to Meet a Custom
		Objective (Code)" on page 3-118. Note To perform estimation, optimization, or evaluation using Simulink fast restart, it is necessary to create the simulator before the cost function, and then pass the simulator to the cost function.

Convenience Object	Class Name	Description
Requirements objects	 Available Requirements Objects Time-domain requirements: sdo.requirements.SignalBo 	Use these objects to specify time- domain and frequency-domain costs or constraints that depend on the design variable values.
	<pre>und sdo.requirements.StepResp onseEnvelope sdo.requirements.SignalTr acking</pre>	In the cost function, use the evalRequirement method of the object to evaluate how closely the current design variables satisfy your design requirement. For an example, see "Design Optimization to Meet Step Response Requirements (Code)".
	neRegion Parameter requirements:	
	 sdo.requirements.Function Matching sdo.requirements Monotoni 	
	 sdo.requirements.Monotoni cVariable 	
	 sdo.requirements.Relation alConstraint 	
	 sdo.requirements.Smoothne ssConstraint 	
	Frequency-domain requirements:	
	• sdo.requirements.GainPhas eMargin	
	 sdo.requirements.BodeMagn itude 	
	 sdo.requirements.ClosedLo opPeakGain 	
	 sdo.requirements.PZDampin gRatio 	
	• sdo.requirements.PZNatura lFrequency	
	 sdo.requirements.PZSettli ngTime 	
	 sdo.requirements.SignalTr acking 	
	 sdo.requirements.StepResp onseEnvelope 	
	• sdo.requirements.OpenLoop GainPhase	

Convenience Object	Class Name	Description
Experiment objects	sdo.Experiment	Use an experiment object to specify the input-output data, model parameters, and initial-state values for parameter estimation. In the cost function, update the design variable values associated with the experiment using the setEstimatedValues method. Then, to simulate the model using the updated model configuration, create a simulator using the createSimulator method.
		For an example, see "Estimate Model Parameters Per Experiment (Code)" on page 2-86.

See Also

sdo.optimize|sdo.OptimizeOptions|sdo.evaluate|sdo.setValueInModel|
param.Continuous|sdo.SimulationTest|sdo.Experiment

Related Examples

- "How the Optimization Algorithm Formulates Minimization Problems" on page 3-3
- "Design Optimization to Meet a Custom Objective (Code)" on page 3-118
- "How the Software Formulates Parameter Estimation as an Optimization Problem" on page 2-41
- "Estimate Model Parameter Values (Code)" on page 2-58
- "What is Sensitivity Analysis?" on page 4-2
- "Identify Key Parameters for Estimation (Code)" on page 4-169

Gradient Computations

For the Gradient descent (fmincon) optimization solver, the gradients are computed using numerical perturbation:

$$dx = \sqrt[3]{eps} \times \max(\left|x\right|, \frac{1}{10}x_{typical})$$
$$dL = \max(x - dx, x_{min})$$
$$dR = \min(x + dx, x_{max})$$
$$F_L = opt_fcn(dL)$$
$$F_R = opt_fcn(dR)$$
$$\frac{dF}{dx} = \frac{(F_L - F_R)}{(dL - dR)}$$

- *x* is a scalar design variable.
- x_{min} is the lower bound of *x*.
- x_{max} is the upper bound of *x*.
- $x_{typical}$ is the scaled value of *x*.
- *opt fcn* is the objective function.

dx is relatively large to accommodate simulation solver tolerances.

If you want to compute the gradients in any other way, you can do so in the cost function you write for performing design optimization programmatically. See sdo.optimize and GradFcn of sdo.OptimizeOptions for more information.

See Also

fmincon

More About

- "How the Software Formulates Parameter Estimation as an Optimization Problem" on page 2-41
- "How the Optimization Algorithm Formulates Minimization Problems" on page 3-3

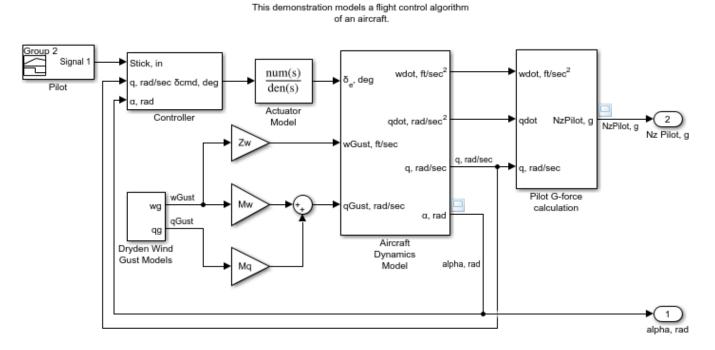
Estimate Model Parameter Values (Code)

This example shows how to use experimental data to estimate model parameter values.

Aircraft Model

The Simulink® model, sdoAircraftEstimation, models the longitudinal flight control system of an aircraft.

open_system('sdoAircraftEstimation')



Aircraft Longitudinal Flight Control

Copyright 1990-2012 The MathWorks, Inc.

Estimation Problem

You use measured data to estimate the aircraft model parameters and states.

Measured output data:

- Pilot G force, output of the Pilot G-force calculation block
- Angle of attack, fourth output of the Aircraft Dynamics Model block

Parameters:

• Actuator time constant, Ta, used by the Actuator Model block

- Vertical velocity, Zd, used by the Aircraft Dynamics Model block
- Pitch rate gains, Md, used by the Aircraft Dynamics Model block

State:

• Initial state of the first-order actuator model, sdoAircraftEstimation/Actuator Model

Define the Estimation Experiment

Get the measured data.

[time,iodata] = sdoAircraftEstimation_Experiment;

The sdoAircraftEstimation_Experiment function returns the measured output data, iodata, and the corresponding time vector. The first column of iodata is the pilot G force and the second column is the angle of attack.

To see the code for this function, type edit sdoAircraftEstimation_Experiment.

Create an experiment object to store the measured input/output data.

Exp = sdo.Experiment('sdoAircraftEstimation');

Create an object to store the measured pilot G-Force output.

```
PilotG = Simulink.SimulationData.Signal;
PilotG.Name = 'PilotG';
PilotG.BlockPath = 'sdoAircraftEstimation/Pilot G-force calculation';
PilotG.PortType = 'outport';
PilotG.PortIndex = 1;
PilotG.Values = timeseries(iodata(:,2),time);
```

Create an object to store the measured angle of attack (alpha) output.

```
AoA = Simulink.SimulationData.Signal;
AoA.Name = 'AngleOfAttack';
AoA.BlockPath = 'sdoAircraftEstimation/Aircraft Dynamics Model';
AoA.PortType = 'outport';
AoA.PortIndex = 4;
AoA.Values = timeseries(iodata(:,1),time);
```

Add the measured pilot G-Force and angle of attack data to the experiment as the expected output data.

```
Exp.OutputData = [...
PilotG; ...
AoA];
```

Add the initial state for the Actuator Model block to the experiment. Set its Free field to true so that it is estimated.

```
Exp.InitialStates = sdo.getStateFromModel('sdoAircraftEstimation','Actuator Model');
Exp.InitialStates.Minimum = 0;
Exp.InitialStates.Free = true;
```

Compare the Measured Output and the Initial Simulated Output

Create a simulation scenario using the experiment and obtain the simulated output.

```
Simulator = createSimulator(Exp);
Simulator = sim(Simulator);
```

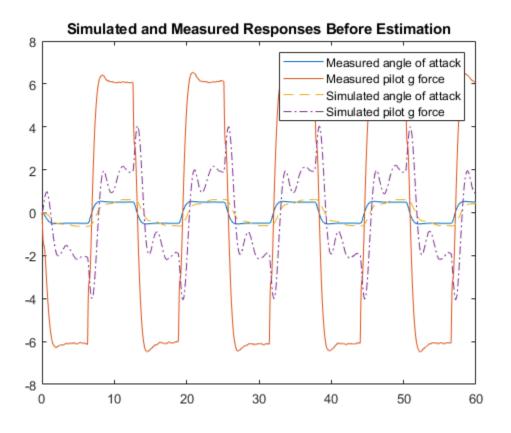
Search for the pilot G-Force and angle of attack signals in the logged simulation data.

```
SimLog = find(Simulator.LoggedData,get_param('sdoAircraftEstimation','SignalLoggingName'))
PilotGSignal = find(SimLog,'PilotG');
AoASignal = find(SimLog,'AngleOfAttack');
```

Plot the measured and simulated data.

As expected, the model response does not match the experimental output data.

```
plot(time, iodata, ...
AoASignal.Values.Time,AoASignal.Values.Data,'--', ...
PilotGSignal.Values.Time,PilotGSignal.Values.Data,'-.');
title('Simulated and Measured Responses Before Estimation')
legend('Measured angle of attack', 'Measured pilot g force', ...
'Simulated angle of attack', 'Simulated pilot g force');
```



Specify the Parameters to Estimate

Select the model parameters that describe the flight control actuation system. Specify bounds for the estimated parameter values based on the understanding of the actuation system.

```
p = sdo.getParameterFromModel('sdoAircraftEstimation',{'Ta','Md','Zd'});
p(1).Minimum = 0.01; %Ta
p(1).Maximum = 1;
```

p(2).Minimum = -10; %Md
p(2).Maximum = 0;
p(3).Minimum = -100; %Zd
p(3).Maximum = 0;

Get the actuator initial state value that is to be estimated from the experiment.

s = getValuesToEstimate(Exp);

Group the model parameters and initial states to be estimated together.

```
v = [p;s]
v(1,1) =
      Name: 'Ta'
      Value: 0.5000
   Minimum: 0.0100
   Maximum: 1
       Free: 1
      Scale: 0.5000
      Info: [1x1 struct]
v(2,1) =
      Name: 'Md'
      Value: -1
   Minimum: -10
   Maximum: 0
      Free: 1
      Scale: 1
      Info: [1x1 struct]
v(3,1) =
      Name: 'Zd'
      Value: -80
   Minimum: -100
   Maximum: 0
      Free: 1
      Scale: 128
      Info: [1x1 struct]
v(4,1) =
      Name: 'sdoAircraftEstimation/Actuator...'
      Value: 0
   Minimum: 0
   Maximum: Inf
       Free: 1
      Scale: 1
   dxValue: 0
    dxFree: 1
       Info: [1x1 struct]
```

4x1 param.Continuous

Define the Estimation Objective Function

Create an estimation objective function to evaluate how closely the simulation output, generated using the estimated parameter values, matches the measured data.

Use an anonymous function with one input argument that calls the sdoAircraftEstimation_Objective function. Pass the anonymous function to sdo.optimize, which evaluates the function at each optimization iteration.

```
estFcn = @(v) sdoAircraftEstimation_Objective(v,Simulator,Exp);
```

The sdoAircraftEstimation_Objective function:

- Has one input argument that specifies the actuator parameter values and the actuator initial state.
- Has one input argument that specifies the experiment object containing the measured data.
- Returns a vector of errors between simulated and experimental outputs.

The sdoAircraftEstimation_Objective function requires two inputs, but sdo.optimize requires a function with one input argument. To work around this, estFcn is an anonymous function with one input argument, v, but it calls sdoAircraftEstimation_Objective using two input arguments, v and Exp.

For more information regarding anonymous functions, see "Anonymous Functions".

The sdo.optimize command minimizes the return argument of the anonymous function estFcn, that is, the residual errors returned by sdoAircraftEstimation_Objective. For more details on how to write an objective/constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB® command prompt.

To examine the estimation objective function in more detail, type edit sdoAircraftEstimation_Objective at the MATLAB command prompt.

type sdoAircraftEstimation_Objective

```
function vals = sdoAircraftEstimation Objective(v,Simulator,Exp)
%SDOAIRCRAFTESTIMATION OBJECTIVE
%
     The sdoAircraftEstimation Objective function is used to compare model
%
%
     outputs against experimental data.
%
%
     vals = sdoAircraftEstimation_Objective(v,Exp)
%
%
     The |v| input argument is a vector of estimated model parameter values
%
     and initial states.
%
%
     The [Simulator] input argument is a simulation object used
%
     simulate the model with the estimated parameter values.
%
%
     The |Exp| input argument contains the estimation experiment data.
```

```
%
%
     The |vals| return argument contains information about how well the
%
     model simulation results match the experimental data and is used by
0/0
     the |sdo.optimize| function to estimate the model parameters.
%
%
     See also sdo.optimize, sdoExampleCostFunction,
     sdoAircraftEstimation cmddemo
%
%
% Copyright 2012-2015 The MathWorks, Inc.
%%
% Define a signal tracking requirement to compute how well the model output
% matches the experiment data. Configure the tracking requirement so that
% it returns the tracking error residuals (rather than the
% sum-squared-error) and does not normalize the errors.
%
r = sdo.requirements.SignalTracking;
           = '==';
r.Type
r.Method = 'Residuals';
r.Normalize = 'off';
<u>%</u>%
% Update the experiments with the estimated parameter values.
0/
Exp = setEstimatedValues(Exp,v);
%%
% Simulate the model and compare model outputs with measured experiment
% data.
%
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
             = find(Simulator.LoggedData,get_param('sdoAircraftEstimation','SignalLoggingName'))
SimLoa
PilotGSignal = find(SimLog, 'PilotG');
AoASignal = find(SimLog, 'AngleOfAttack');
PilotGError = evalRequirement(r,PilotGSignal.Values,Exp.OutputData(1).Values);
           = evalRequirement(r,AoASignal.Values,Exp.OutputData(2).Values);
AoAError
200
% Return the residual errors to the optimization solver.
%
vals.F = [PilotGError(:); AoAError(:)];
end
```

Estimate the Parameters

Use the sdo.optimize function to estimate the actuator parameter values and initial state.

Specify the optimization options. The estimation function sdoAircraftEstimation_Objective returns the error residuals between simulated and experimental data and does not include any constraints, making this problem ideal for the lsqnonlin solver.

```
opt = sdo.OptimizeOptions;
opt.Method = 'lsqnonlin';
```

Estimate the parameters.

```
v0pt = sdo.optimize(estFcn,v,opt)
```

Optimization started 02-Sep-2021 00:30:03

				First-order
Iter	F-count	f(x)	Step-size	optimality
0	8	27955.2	1	
1	17	10121.6	0.4744	5.68e+04
2	26	3127.5	0.3845	1.24e+04
3	35	872.577	0.4293	2.81e+03
4	44	238.548	0.5156	618
5	53	71.5731	0.4939	147
6	62	16.9301	0.4222	44.3
7	71	1.82188	0.3019	11.5
8	80	0.0426814	0.1356	1.35
9	89	0.00113417	0.02555	0.243
10	98	0.000247412	0.008296	0.00764
Local	minimum	possible.		

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

```
vOpt(1,1) =
```

```
Name: 'Ta'
      Value: 0.0500
    Minimum: 0.0100
    Maximum: 1
       Free: 1
      Scale: 0.5000
       Info: [1x1 struct]
vOpt(2,1) =
       Name: 'Md'
      Value: -6.8849
   Minimum: -10
    Maximum: 0
       Free: 1
      Scale: 1
       Info: [1x1 struct]
vOpt(3,1) =
      Name: 'Zd'
Value: -63.9989
    Minimum: -100
    Maximum: 0
       Free: 1
      Scale: 128
       Info: [1x1 struct]
```

```
v0pt(4,1) =
    Name: 'sdoAircraftEstimation/Actuator...'
    Value: 1.3254e-04
    Minimum: 0
    Maximum: Inf
        Free: 1
        Scale: 1
        dxValue: 0
        dxFree: 1
        Info: [1x1 struct]
```

```
4x1 param.Continuous
```

Compare the Measured Output and the Final Simulated Output

Update the experiments with the estimated parameter values.

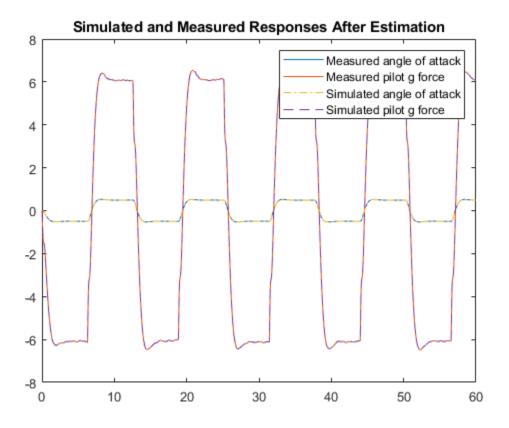
Exp = setEstimatedValues(Exp,v0pt);

Simulate the model using the updated experiment and compare the simulated output with the experimental data.

The model response using the estimated parameter values closely matches the experiment output data.

```
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoAircraftEstimation','SignalLoggingName'))
PilotGSignal = find(SimLog,'PilotG');
AoASignal = find(SimLog,'AngleOfAttack');
```

```
plot(time, iodata, ...
AoASignal.Values.Time,AoASignal.Values.Data,'-.', ...
PilotGSignal.Values.Time,PilotGSignal.Values.Data,'--')
title('Simulated and Measured Responses After Estimation')
legend('Measured angle of attack', 'Measured pilot g force', ...
'Simulated angle of attack', 'Simulated pilot g force');
```



Update the Model Parameter Values

Update the model with the estimated actuator parameter values. Do not update the model actuator initial state (fourth element of vOpt) as it is dependent on the experiment.

sdo.setValueInModel('sdoAircraftEstimation',vOpt(1:3));

Related Examples

To learn how to estimate model parameters using the **Parameter Estimator** app, see "Estimate Model Parameter Values (GUI)" on page 2-143.

Close the model.

bdclose('sdoAircraftEstimation')

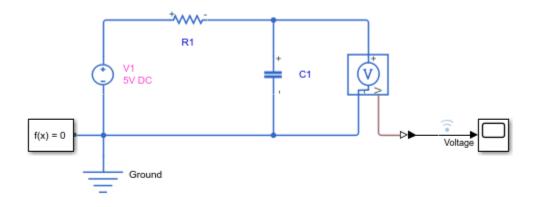
Estimate Model Parameters and Initial States (Code)

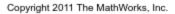
This example shows how to estimate the initial state and parameters of a model.

This example requires Simscape[™] software.

RC Circuit Model

The Simulink® model, sdoRCCircuit, models a simple resistor-capacitor (RC) circuit. open_system('sdoRCCircuit');





Estimation Problem

You use the measured data to estimate the RC model parameter and state values.

Measured output data:

• Capacitor voltage, output of the PS-Simulink Converter block

Parameter:

• Capacitance, C1, used by the C1 block

State:

• Initial voltage of the capacitor

Define the Estimation Experiment

Get the measured data.

load sdoRCCircuit_ExperimentData

The variables time and data are loaded into the workspace, where data is the measured capacitor voltage for times time.

Create an experiment object to store the experimental voltage data.

```
Exp = sdo.Experiment('sdoRCCircuit');
```

Create an object to store the measured capacitor voltage output.

```
Voltage = Simulink.SimulationData.Signal;
Voltage.Name = 'Voltage';
Voltage.BlockPath = 'sdoRCCircuit/PS-Simulink Converter';
Voltage.PortType = 'outport';
Voltage.PortIndex = 1;
Voltage.Values = timeseries(data,time);
```

Add the measured capacitor data to the experiment as the expected output data.

Exp.OutputData = Voltage;

Compare the Measured Output and the Initial Simulated Output

Create a simulation scenario using the experiment and obtain the simulated output.

Simulator	<pre>= createSimulator(Exp);</pre>
Simulator	<pre>= sim(Simulator);</pre>

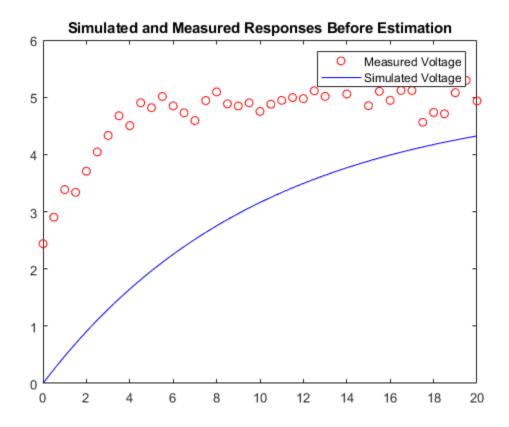
Search for the voltage signal in the logged simulation data.

SimLog	=	<pre>find(Simulator.LoggedData,get_param('sdoRCCircuit','SignalLoggingName'));</pre>
Voltage	=	<pre>find(SimLog,'Voltage');</pre>

Plot the measured and simulated data.

The model response does not match the experimental output data.

```
plot(time,data,'ro',Voltage.Values.Time,Voltage.Values.Data,'b')
title('Simulated and Measured Responses Before Estimation')
legend('Measured Voltage','Simulated Voltage')
```



Specify the Parameters to Estimate

Select the capacitance parameter from the model. Specify an initial guess for the capacitance value (460 uF) and a minimum bound (0 F).

```
p = sdo.getParameterFromModel('sdoRCCircuit','C1');
p.Value = 460e-6;
p.Minimum = 0;
```

Define the Estimation Objective Function

Create an estimation objective function to evaluate how closely the simulation output, generated using the estimated parameter value, matches the measured data.

Use an anonymous function with one input argument that calls the sdoRCCircuit_Objective function. Pass the anonymous function to sdo.optimize, which evaluates the function at each optimization iteration.

estFcn = @(v) sdoRCCircuit_Objective(v,Simulator,Exp);

The sdoRCCircuit_Objective function:

- Has one input argument that specifies the estimated circuit capacitance value.
- Has one input argument that specifies the experiment object containing the measured data.
- Returns a vector of errors between simulated and experimental outputs.

The sdoRCCircuit_Objective function requires two inputs, but sdo.optimize requires a function with one input argument. To work around this, estFcn is an anonymous function with one input argument, v, but it calls sdoRCCircuit_Objective using two input arguments, v and Exp.

For more information regarding anonymous functions, see "Anonymous Functions".

The optimization solver minimizes the residual errors. For more details on how to write an objective/ constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB® command prompt.

To examine the estimation object function in more detail, type edit sdoRCCircuit_Objective at the MATLAB command prompt.

```
type sdoRCCircuit_Objective
```

```
function vals = sdoRCCircuit Objective(v,Simulator,Exp)
%SDORCCIRCUIT OBJECTIVE
%
%
     The sdoRCCircuit Objective function is used to compare model
%
     outputs against experimental data.
%
     vals = sdoRCCircuit_Objective(v,Exp)
%
%
    The |v| input argument is a vector of estimated model parameter values
%
%
     and initial states.
%
%
    The |Simulator| input argument is a simulation object used
%
     simulate the model with the estimated parameter values.
%
%
    The [Exp] input argument contains the estimation experiment data.
%
%
    The |vals| return argument contains information about how well the
     model simulation results match the experimental data and is used by
%
%
     the [sdo.optimize] function to estimate the model parameters.
%
%
    See also sdo.optimize, sdoExampleCostFunction, sdoRCCircuit cmddemo
%
% Copyright 2012-2015 The MathWorks, Inc.
%%
% Define a signal tracking requirement to compute how well the model output
% matches the experiment data. Configure the tracking requirement so that
% it returns the tracking error residuals (rather than the
% sum-squared-error) and does not normalize the errors.
%
r = sdo.requirements.SignalTracking;
          = '==';
r.Type
          = 'Residuals';
r.Method
r.Normalize = 'off';
%%
% Update the experiments with the estimated parameter values.
%
Exp = setEstimatedValues(Exp,v);
%%
```

```
% Simulate the model and compare model outputs with measured experiment
% data.
%
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoRCCircuit','SignalLoggingName'));
Voltage = find(SimLog,'Voltage');
VoltageError = evalRequirement(r,Voltage.Values,Exp.OutputData(1).Values);
%%
% Return the residual errors to the optimization solver.
% vals.F = VoltageError(:);
end
```

Estimate the Parameters

Use the sdo.optimize function to estimate the capacitance value.

Specify the optimization options. The estimation function sdoRCCircuit_Objective returns the error residuals between simulated and experimental data and does not include any constraints, making this problem ideal for the |lsqnonlin|solver.

```
opt = sdo.OptimizeOptions;
opt.Method = 'lsqnonlin';
```

Estimate the parameters.

```
p0pt = sdo.optimize(estFcn,p,opt)
```

Optimization started 02-Sep-2021 00:31:33

				First-order
Iter F	-count	f(x)	Step-size	optimality
\odot	3	55.0017	1	
1	6	21.0148	0.2124	17.3
2	9	11.5069	0.1272	6.1
3	12	9.56554	0.06554	2
4	15	9.27804	0.0275	0.43
5	18	9.27316	0.006987	0.0788
1	A A			

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

```
pOpt =
```

Name: 'C1' Value: 1.1346e-04 Minimum: 0 Maximum: Inf Free: 1 Scale: 0.0020 Info: [1x1 struct] 1x1 param.Continuous

Compare the Measured Output and the Simulated Output

Update the experiment with the estimated capacitance value.

Exp = setEstimatedValues(Exp,pOpt);

Create a simulation scenario using the experiment and obtain the simulated output.

```
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
```

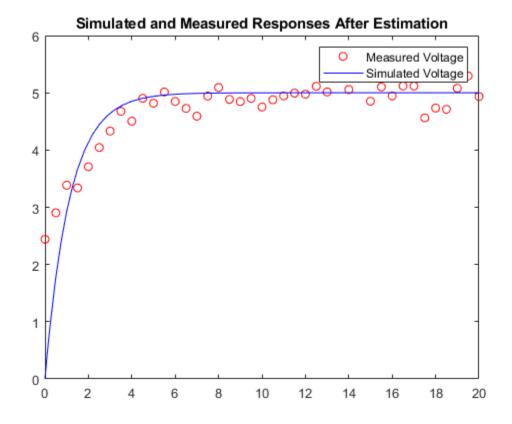
Search for the voltage signal in the logged simulation data.

```
SimLog = find(Simulator.LoggedData,get_param('sdoRCCircuit','SignalLoggingName'));
Voltage = find(SimLog,'Voltage');
```

Plot the measured and simulated data.

The simulated and measured signals match well, except for near time zero. This mismatch is because the capacitor initial voltage defined in the model does not match the initial voltage from the experiment.

```
plot(time,data,'ro',Voltage.Values.Time,Voltage.Values.Data,'b')
title('Simulated and Measured Responses After Estimation')
legend('Measured Voltage','Simulated Voltage')
```



2-72

Estimate the Initial State

Add the capacitor initial voltage for the C1 block to the experiment. Set its initial guess value to 1 V.

```
Exp.InitialStates = sdo.getStateFromModel('sdoRCCircuit','C1');
Exp.InitialStates.Value = 1;
```

Recreate the estimation function to use the experiment with initial state estimation

estFcn = @(v) sdoRCCircuit_Objective(v,Simulator,Exp);

Get the initial state and capacitance value that is to be estimated from the experiment.

```
v = getValuesToEstimate(Exp);
```

Estimate the parameters.

```
v0pt = sdo.optimize(estFcn,v,opt)
```

Optimization started 02-Sep-2021 00:31:50

					First-order
Iter F-count			f(x)	Step-size	optimality
	0	5	4.74867	1	
	1	10	2.1196	1.537	22.7
	2	15	1.34958	0.1262	0.0713
	3	20	1.34365	0.05718	0.129
	4	25	1.34363	0.001378	0.000765

Local minimum found.

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

vOpt(1,1) =

```
Name: 'sdoRCCircuit/C1:sdoRCCircuit.C1.vc'
Value: 2.3597
Minimum: -Inf
Maximum: Inf
Free: 1
Scale: 1
dxValue: 0
dxFree: 1
Info: [1x1 struct]
```

```
vOpt(2,1) =
```

Name: 'C1' Value: 2.2638e-04 Minimum: 0 Maximum: Inf Free: 1 Scale: 0.0020 Info: [1x1 struct] 2x1 param.Continuous

Compare the Measured Output and the Final Simulated Output

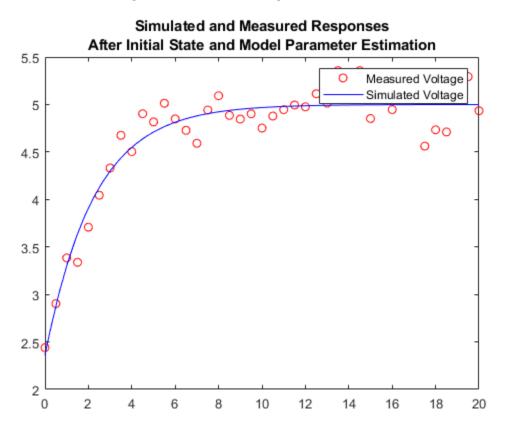
Update the experiment with the estimated capacitance and capacitor initial voltage values.

```
Exp = setEstimatedValues(Exp,vOpt);
```

Simulate the model with the estimated initial-state and parameter values and compare the simulated output with the experiment data.

```
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoRCCircuit','SignalLoggingName'));
Voltage = find(SimLog,'Voltage');
plot(time,data,'ro',Voltage.Values.Time,Voltage.Values.Data,'b')
title({'Simulated and Measured Responses';...
```

```
'After Initial State and Model Parameter Estimation'})
legend('Measured Voltage', 'Simulated Voltage')
```



Update the Model Parameter Values

Update the model with the estimated capacitance value. Do not update the model capacitor initial voltage (first element of vOpt) as it is dependent on the experiment.

```
sdo.setValueInModel('sdoRCCircuit',v0pt(2));
```

Related Examples

To learn how to estimate model parameters using the sdo.optimize command, see "Estimate Model Parameters and Initial States (GUI)" on page 2-168.

Close the model

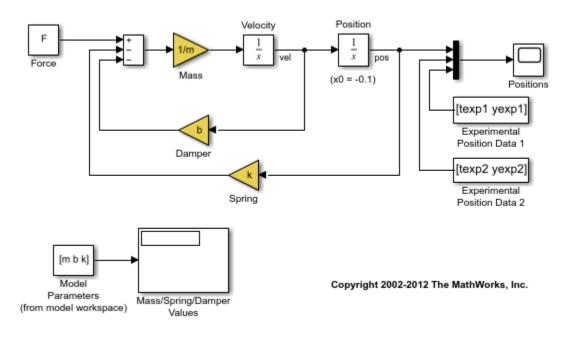
bdclose('sdoRCCircuit')

Estimate Model Parameters Using Multiple Experiments (Code)

This example shows how to estimate model parameters from multiple sets of experimental data. You estimate the parameters of a mass-spring-damper system.

Open the Model and Get Experimental Data

This example uses the sdoMassSpringDamper model. The model includes two integrators to model the velocity and position of a mass in a mass-spring-damper system.



open_system('sdoMassSpringDamper');

Load the experiment data.

load sdoMassSpringDamper_ExperimentData

The variables texp1, yexp1, texp2, and yexp2 are loaded into the workspace. yexp1 and yexp2 describe the mass position for times texp1 and texp2, respectively.

Define the Estimation Experiments

Create a 2-element array of experiment objects to store the measured data for the two experiments.

Create an experiment object for the first experiment.

Exp = sdo.Experiment('sdoMassSpringDamper');

Create an object to store the measured mass position output.

```
MeasuredPos = Simulink.SimulationData.Signal;
MeasuredPos.Values = timeseries(yexp1,texp1);
MeasuredPos.BlockPath = 'sdoMassSpringDamper/Position';
MeasuredPos.PortType = 'outport';
```

MeasuredPos.PortIndex = 1; MeasuredPos.Name = 'Position';

Add the measured mass position data to the experiment as the expected output data.

Exp.OutputData = MeasuredPos;

Create an object to specify the initial state for the Velocity block. The initial velocity of the mass is 0 m/s.

```
sVel = sdo.getStateFromModel('sdoMassSpringDamper','Velocity');
sVel.Value = 0;
sVel.Free = false;
```

sVel.Free is set to false because the initial velocity is known and does not need to be estimated.

Create an object to specify the initial state for the **Position** block. Specify a guess for the initial mass position. Set the **Free** field of the initial position object to **true** so that it is estimated.

```
sPos = sdo.getStateFromModel('sdoMassSpringDamper','Position');
sPos.Free = true;
sPos.Value = -0.1;
```

Add the initial states to the experiment.

Exp.InitialStates = [sVel;sPos];

Create a two-element array of experiments. As the two experiments are identical except for the expected output data, copy the first experiment twice.

Exp = [Exp; Exp];

Modify the expected output data of the second experiment object in Exp.

```
Exp(2).OutputData.Values = timeseries(yexp2,texp2);
```

Compare the Measured Output and the Initial Simulated Output

Create a simulation scenario using the first experiment and obtain the simulated output.

```
Simulator = createSimulator(Exp(1));
Simulator = sim(Simulator);
```

Search for the position signal in the logged simulation data.

```
SimLog = find(Simulator.LoggedData,get_param('sdoMassSpringDamper','SignalLoggingName'));
Position = find(SimLog,'Position');
```

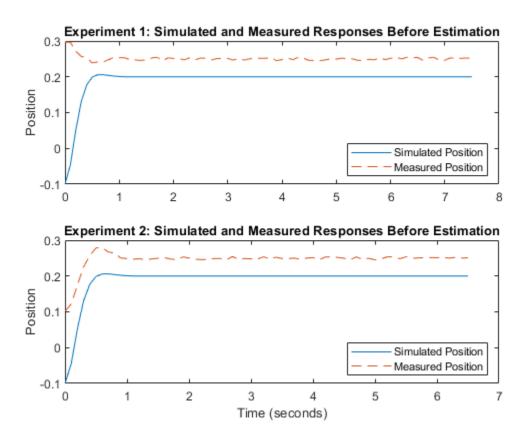
Obtain the simulated position signal for the second experiment.

```
Simulator = createSimulator(Exp(2),Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoMassSpringDamper','SignalLoggingName'));
Position(2) = find(SimLog,'Position');
```

Plot the measured and simulated data.

The model response does not match the experimental output data.

```
subplot(211)
plot(...
    Position(1).Values.Time,Position(1).Values.Data, ...
    Exp(1).OutputData.Values.Time, Exp(1).OutputData.Values.Data,'--')
title('Experiment 1: Simulated and Measured Responses Before Estimation')
ylabel('Position')
legend('Simulated Position','Measured Position','Location','SouthEast')
subplot(212)
plot(...
    Position(2).Values.Time,Position(2).Values.Data, ...
    Exp(2).OutputData.Values.Time, Exp(2).OutputData.Values.Data,'--')
title('Experiment 2: Simulated and Measured Responses Before Estimation')
xlabel('Time (seconds)')
ylabel('Position')
legend('Simulated Position','Measured Position','Location','SouthEast')
```



Specify Parameters to Estimate

Select the mass m, spring constant k, and damping coefficient b parameters from the model. Specify that the estimated values for these parameters must be positive.

```
p = sdo.getParameterFromModel('sdoMassSpringDamper', {'b', 'k', 'm'});
p(1).Minimum = 0;
p(2).Minimum = 0;
p(3).Minimum = 0;
```

Get the position initial state values to be estimated from the experiment.

s = getValuesToEstimate(Exp);

s contains two initial state objects, both for the **Position** block. Each object corresponds to an experiment in Exp.

Group the model parameters and initial states to be estimated together.

```
v = [p;s]
v(1,1) =
      Name: 'b'
      Value: 100
   Minimum: 0
   Maximum: Inf
       Free: 1
      Scale: 128
      Info: [1x1 struct]
v(2,1) =
      Name: 'k'
      Value: 500
   Minimum: 0
   Maximum: Inf
      Free: 1
      Scale: 512
      Info: [1x1 struct]
v(3,1) =
      Name: 'm'
      Value: 8
   Minimum: 0
   Maximum: Inf
      Free: 1
      Scale: 8
      Info: [1x1 struct]
v(4,1) =
      Name: 'sdoMassSpringDamper/Position'
      Value: -0.1000
   Minimum: -Inf
   Maximum: Inf
      Free: 1
      Scale: 0.1250
   dxValue: 0
    dxFree: 1
       Info: [1x1 struct]
v(5,1) =
```

```
Name: 'sdoMassSpringDamper/Position'
Value: -0.1000
Minimum: -Inf
Maximum: Inf
Free: 1
Scale: 0.1250
dxValue: 0
dxFree: 1
Info: [1x1 struct]
```

5x1 param.Continuous

Define the Estimation Objective

Create an estimation objective function to evaluate how closely the simulation output, generated using the estimated parameter values, matches the measured data.

Use an anonymous function with one input argument that calls the sdoMassSpringDamper_Objective function. P ass the anonymous function to sdo.optimize, which evaluates the function at each optimization iteration.

estFcn = @(v) sdoMassSpringDamper_Objective(v,Simulator,Exp);

The sdoMassSpringDamper_Objective function:

- Has one input argument that specifies the mass, spring constant and damper values as well as the initial mass position.
- Has one input argument that specifies the experiment object containing the measured data.
- Returns a vector of errors between simulated and experimental outputs.

The sdoMassSpringDamper_Objective function requires two inputs, but sdo.optimize requires a function with one input argument. To work around this, estFcn is an anonymous function with one input argument, v, but it calls sdoMassSpringDamper_Objective using two input arguments, v and Exp.

For more information regarding anonymous functions, see "Anonymous Functions".

The sdo.optimize command minimizes the return argument of the anonymous function estFcn, that is, the residual errors returned by sdoMassSpringDamper_Objective. For more details on how to write an objective/constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB® command prompt.

To examine the estimation objective function in more detail, type edit sdoMassSpringDamper_Objective at the MATLAB command prompt.

```
type sdoMassSpringDamper_Objective
```

```
function vals = sdoMassSpringDamper_Objective(v,Simulator,Exp)
%SDOMASSSPRINGDAMPER_OBJECTIVE
%
%
% The sdoMassSpringDamper_Objective function is used to compare model
% outputs against experimental data.
```

```
%
%
     vals = sdoMassSpringDamper Objective(v,Exp)
%
%
     The |v| input argument is a vector of estimated model parameter values
%
     and initial states.
%
%
     The |Simulator| input argument is a simulation object used
%
     simulate the model with the estimated parameter values.
%
%
     The [Exp] input argument contains the estimation experiment data.
%
%
     The |vals| return argument contains information about how well the
%
     model simulation results match the experimental data and is used by
%
     the |sdo.optimize| function to estimate the model parameters.
%
%
     see also sdo.optimize, sdoExampleCostFunction
%
% Copyright 2012-2015 The MathWorks, Inc.
%%
% Define a signal tracking requirement to compute how well the model output
% matches the experiment data. Configure the tracking requirement so that
% it returns the tracking error residuals (rather than the
% sum-squared-error) and does not normalize the errors.
%
r = sdo.requirements.SignalTracking;
           = '==';
r.Type
            = 'Residuals';
r.Method
r.Normalize = 'off';
%%
% Update the experiments with the estimated parameter values.
%
Exp = setEstimatedValues(Exp,v);
200
% Simulate the model and compare model outputs with measured experiment
% data.
%
Error = [];
for ct=1:numel(Exp)
    Simulator = createSimulator(Exp(ct),Simulator);
    Simulator = sim(Simulator);
    SimLog = find(Simulator.LoggedData,get param('sdoMassSpringDamper','SignalLoggingName'));
    Position = find(SimLog, 'Position');
    PositionError = evalRequirement(r,Position.Values,Exp(ct).OutputData.Values);
    Error = [Error; PositionError(:)];
end
%%
% Return the residual errors to the optimization solver.
%
```

```
vals.F = Error(:);
end
```

Estimate the Parameters

Use the sdo.optimize function to estimate the actuator parameter values and initial state.

Specify the optimization options. The estimation function sdoMassSpringDamper_Objective returns the error residuals between simulated and experimental data and does not include any constraints, making this problem ideal for the lsqnonlin solver.

```
opt = sdo.OptimizeOptions;
opt.Method = 'lsqnonlin';
```

Estimate the parameters. Notice that the initial mass position is estimated twice, once for each experiment.

```
v0pt = sdo.optimize(estFcn,v,opt)
```

Optimization started 02-Sep-2021 00:32:38

				First-order
Iter	F-count	f(x)	Step-size	optimality
Θ	11	0.777696	1	
1	22	0.00413099	3.696	0.00648
2	33	0.00118327	0.3194	0.00243
3	44	0.0011106	0.06718	5.09e-05
Local	minimum	found.		

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

```
vOpt(1,1) =
```

```
Name: 'b'
Value: 58.1959
Minimum: 0
Maximum: Inf
Free: 1
Scale: 128
Info: [1x1 struct]
```

```
vOpt(2,1) =
```

Name: 'k'
Value: 399.9452
Minimum: 0
Maximum: Inf
Free: 1
Scale: 512
Info: [1x1 struct]
v0pt(3,1) =

Name: 'm'

```
Value: 9.7225
   Minimum: 0
   Maximum: Inf
      Free: 1
     Scale: 8
      Info: [1x1 struct]
vOpt(4,1) =
      Name: 'sdoMassSpringDamper/Position'
     Value: 0.2995
   Minimum: -Inf
   Maximum: Inf
      Free: 1
      Scale: 0.1250
   dxValue: 0
    dxFree: 1
      Info: [1x1 struct]
vOpt(5,1) =
      Name: 'sdoMassSpringDamper/Position'
     Value: 0.0994
   Minimum: - Inf
   Maximum: Inf
      Free: 1
     Scale: 0.1250
   dxValue: 0
    dxFree: 1
      Info: [1x1 struct]
```

```
5x1 param.Continuous
```

Compare the Measured Output and the Final Simulated Output

Update the experiments with the estimated parameter values.

```
Exp = setEstimatedValues(Exp,v0pt);
```

Obtain the simulated output for the first experiment.

```
Simulator = createSimulator(Exp(1),Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoMassSpringDamper','SignalLoggingName'));
Position(1) = find(SimLog,'Position');
```

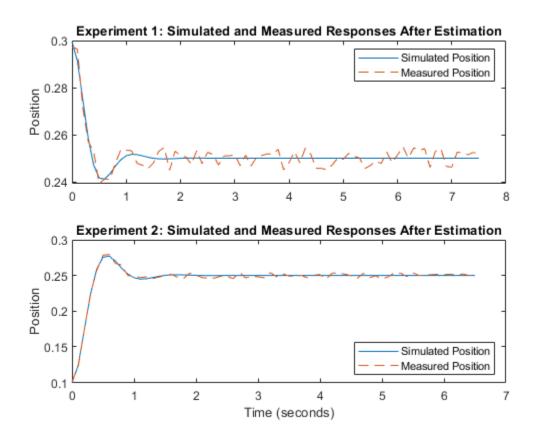
Obtain the simulated output for the second experiment.

```
Simulator = createSimulator(Exp(2),Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoMassSpringDamper','SignalLoggingName'));
Position(2) = find(SimLog,'Position');
```

Plot the measured and simulated data.

The model response using the estimated parameter values matches the output data for the experiments.

```
subplot(211)
plot(...
    Position(1).Values.Time,Position(1).Values.Data, ...
    Exp(1).OutputData.Values.Time, Exp(1).OutputData.Values.Data,'--')
title('Experiment 1: Simulated and Measured Responses After Estimation')
ylabel('Position')
legend('Simulated Position', 'Measured Position', 'Location', 'NorthEast')
subplot(212)
plot(...
    Position(2).Values.Time,Position(2).Values.Data, ...
    Exp(2).OutputData.Values.Time, Exp(2).OutputData.Values.Data,'--')
title('Experiment 2: Simulated and Measured Responses After Estimation')
xlabel('Time (seconds)')
ylabel('Position')
legend('Simulated Position', 'Measured Position', 'Location', 'SouthEast')
```



Update the Model Parameter Values

Update the model m, k, and b parameter values. Do not update the model initial position value as this is dependent on the experiment.

sdo.setValueInModel('sdoMassSpringDamper',v0pt(1:3));

Close the model

bdclose('sdoMassSpringDamper')

Estimate Model Parameters Per Experiment (Code)

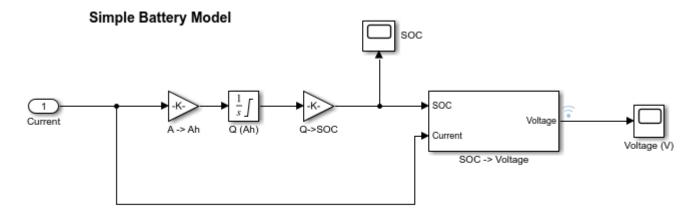
This example shows how to use multiple experiments to estimate a mix of model parameter values; some that are estimated using all the experiments and others that are estimated using individual experiments. The example also shows how to configure estimation experiments with experiment dependent parameter values.

You estimate the parameters of a rechargeable battery based on data collected in experiments that discharge and charge the battery.

Open the Model and Get Experimental Data

This example estimates parameters of a simple, rechargeable battery model, sdoBattery. The model input is the battery current and the model output, the battery terminal voltage, is computed from the battery state-of-charge.

```
open_system('sdoBattery');
```



Copyright 2012-2020 The MathWorks, Inc.

The model is based on the equation

$$E = (1 - \text{Loss})V - KQ_{\max}\frac{1 - s}{s}$$

In the equation:

E is the battery terminal voltage in Volts.

V is the battery constant voltage in Volts.

K is the battery polarization resistance in Ohms.

 Q_{max} is the maximum battery capacity in Ampere-hours.

s is the battery charge state, with 1 being fully charged and 0 discharged. The battery state-of-charge is computed from the integral of the battery current with a positive current indicating discharge and

a negative current indicating charging. The battery initial state-of-charge is specified by Q_0 in Ampere-hours.

Loss is the voltage drop when charging, expressed as a fraction of the battery constant voltage. When the battery is discharging this value is zero.

V, K, Qmax, Q0, and Loss are variables defined in the model workspace.

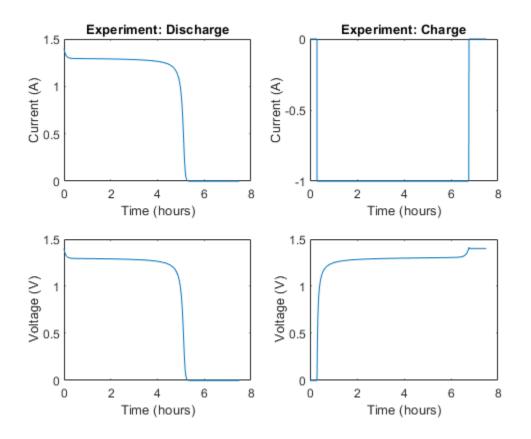
Load the experiment data. A 1.2V (6500mAh) battery was subjected to a discharge experiment and a charging experiment.

load sdoBattery_ExperimentData

The variables Charge_Data and DCharge_Data are loaded into the workspace. The first column of Charge_Data contains time data. The second and third columns of Charge_Data describe the current and voltage during a battery charging experiment. DCharge_Data is similarly structured and contains data for a battery discharging experiment.

Plot the Experiment Data

```
subplot(221),
plot(DCharge_Data(:,1)/3600,DCharge Data(:,2))
title('Experiment: Discharge')
xlabel('Time (hours)')
ylabel('Current (A)')
subplot(223)
plot(DCharge Data(:,1)/3600,DCharge Data(:,3))
xlabel('Time (hours)')
ylabel('Voltage (V)')
subplot(222),
plot(Charge_Data(:,1)/3600,Charge_Data(:,2))
title('Experiment: Charge')
xlabel('Time (hours)')
ylabel('Current (A)')
subplot(224)
plot(Charge_Data(:,1)/3600,Charge_Data(:,3))
xlabel('Time (hours)')
ylabel('Voltage (V)')
```



Define the Estimation Experiments

Create a 2-element array of experiment objects to specify the measured data for the two experiments.

Create an experiment object for the battery discharge experiment. The measured current data is specified as a timeseries in the experiment object.

DCharge_Exp = sdo.Experiment('sdoBattery');

Specify the input data (current) as a timeseries object.

DCharge_Exp.InputData = timeseries(DCharge_Data(:,2),DCharge_Data(:,1));

Create an object to specify the measured voltage output data.

```
VoltageSig = Simulink.SimulationData.Signal;
VoltageSig.Name = 'Voltage';
VoltageSig.BlockPath = 'sdoBattery/SOC -> Voltage';
VoltageSig.PortType = 'outport';
VoltageSig.PortIndex = 1;
VoltageSig.Values = timeseries(DCharge_Data(:,3),DCharge_Data(:,1));
```

Add the voltage signal to the discharge experiment as the expected output data.

DCharge_Exp.OutputData = VoltageSig;

Specify the battery initial charge state for the experiment. The battery charge state is modeled by the Q (Ah) block and it's initial value is specified by the variable Q0. Create a parameter for the Q0

variable and add the parameter to the experiment. Q0 is experiment dependent and assumes different values in the discharging and charging experiments.

```
Q0 = sdo.getParameterFromModel('sdoBattery','Q0');
Q0.Value = 6.5;
Q0.Free = false;
```

Q0.Free is set to false because the initial battery charge is known and does not need to be estimated.

Add the QO parameter to the experiment.

DCharge_Exp.Parameters = Q0;

Create an experiment object to store the charging experiment data. Add the measured current input and measured voltage output data to the object.

```
Charge_Exp = sdo.Experiment('sdoBattery');
Charge_Exp.InputData = timeseries(Charge_Data(:,2),Charge_Data(:,1));
VoltageSig.Values = timeseries(Charge_Data(:,3),Charge_Data(:,1));
Charge_Exp.OutputData = VoltageSig;
```

Add the battery initial charge and charging loss fraction parameters to the experiment. For this experiment, the initial charge (Q0) is known (0 Ah), but the value of the charging loss fraction (Loss) is not known.

```
Q0.Value = 0;
Loss = sdo.getParameterFromModel('sdoBattery','Loss');
Loss.Free = true;
Loss.Minimum = 0;
Loss.Maximum = 0.5;
Charge Exp.Parameters = [Q0;Loss];
```

Loss.Free is set to true so that the value of Loss is estimated.

Collect both experiments into one vector.

Exp = [DCharge_Exp; Charge_Exp];

Compare the Measured Output and the Initial Simulated Output

Create a simulation scenario using the first (discharging) experiment and obtain the simulated output.

```
Simulator = createSimulator(Exp(1));
Simulator = sim(Simulator);
```

Search for the voltage signal in the logged simulation data.

```
SimLog = find(Simulator.LoggedData,get_param('sdoBattery','SignalLoggingName'));
Voltage(1) = find(SimLog,'Voltage');
```

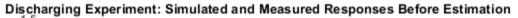
Obtain the simulated voltage signal for the second (charging) experiment.

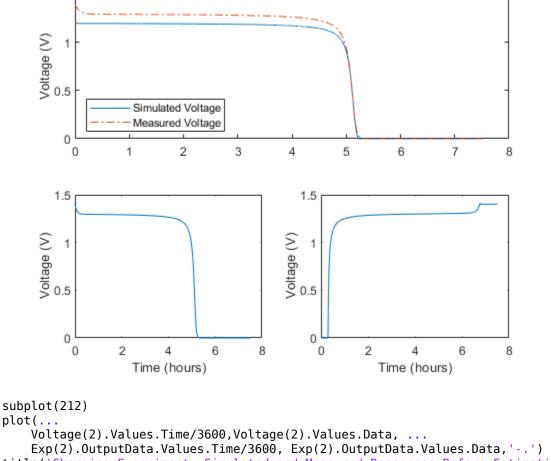
```
Simulator = createSimulator(Exp(2),Simulator);
Simulator = sim(Simulator);
```

```
SimLog = find(Simulator.LoggedData,get_param('sdoBattery','SignalLoggingName'));
Voltage(2) = find(SimLog,'Voltage');
```

Plot the measured and simulated data. The model response does not match the experimental output data.

```
subplot(211)
plot(...
    Voltage(1).Values.Time/3600,Voltage(1).Values.Data, ...
    Exp(1).OutputData.Values.Time/3600, Exp(1).OutputData.Values.Data,'-.')
title('Discharging Experiment: Simulated and Measured Responses Before Estimation')
ylabel('Voltage (V)')
legend('Simulated Voltage','Measured Voltage','Location','SouthWest')
```





title('Charging Experiment: Simulated and Measured Responses Before Estimation')
xlabel('Time (hours)')
ylabel('Voltage (V)')

legend('Simulated Voltage','Measured Voltage','Location','SouthEast')

Specify Parameters to Estimate

Estimate the values of the battery voltage V, the battery polarization resistance K, and the charging loss fraction Loss. The V and K parameters are estimated using all the experiment data while the Loss parameter is estimated using only the charging data.

Select the battery voltage V and the battery polarization resistance K parameters from the model. Specify minimum and maximum bounds for these parameters.

p = sdo.getParameterFromModel('sdoBattery', {'V', 'K'});

p(1).Minimum = 0; p(1).Maximum = 2; p(2).Minimum = 1e-6;

p(2).Maximum = 1e-1;

Get the experiment-specific Loss parameter from the experiment.

```
s = getValuesToEstimate(Exp);
```

Group all the parameters to be estimated.

```
v = [p;s]
v(1,1) =
      Name: 'V'
      Value: 1.2000
   Minimum: 0
   Maximum: 2
       Free: 1
      Scale: 2
      Info: [1x1 struct]
v(2,1) =
      Name: 'K'
      Value: 1.0000e-03
   Minimum: 1.0000e-06
   Maximum: 0.1000
       Free: 1
      Scale: 0.0020
      Info: [1x1 struct]
v(3,1) =
      Name: 'Loss'
      Value: 0.0100
   Minimum: 0
   Maximum: 0.5000
      Free: 1
      Scale: 0.0156
      Info: [1x1 struct]
```

3x1 param.Continuous

Define the Estimation Objective

Create an estimation objective function to evaluate how closely the simulation output, generated using the estimated parameter values, matches the measured data.

Use an anonymous function with one input argument that calls the sdoBattery_Objective function. We pass the anonymous function to sdo.optimize, which evaluates the function at each optimization iteration.

```
estFcn = @(v) sdoBattery_Objective(v,Simulator,Exp);
```

The sdoBattery_Objective function:

- Has one input argument that specifies the estimated battery parameter values.
- Has one input argument that specifies the experiment object containing the measured data.
- Returns a vector of errors between simulated and experimental outputs.

The sdoBattery_Objective function requires two inputs, but sdo.optimize requires a function with one input argument. To work around this, estFcn is an anonymous function with one input argument, v, but it calls sdoBattery_Objective using two input arguments, v and Exp.

For more information regarding anonymous functions, see "Anonymous Functions".

The sdo.optimize command minimizes the return argument of the anonymous function estFcn, that is, the residual errors returned by sdoBattery_Objective. For more details on how to write an objective/constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB® command prompt.

To examine the estimation objective function in more detail, type edit sdoBattery_Objective at the MATLAB command prompt.

type sdoBattery_Objective

```
function vals = sdoBattery_Objective(v,Simulator,Exp)
%SDOBATTERY OBJECTIVE
%
%
    The sdoBattery Objective function is used to compare model
%
    outputs against experimental data.
%
    vals = sdoBattery Objective(v,Exp)
%
%
    The |v| input argument is a vector of estimated model parameter values
%
%
    and initial states.
%
%
    The [Simulator] input argument is a simulation object used
%
    simulate the model with the estimated parameter values.
%
%
    The |Exp| input argument contains the estimation experiment data.
%
%
    The |vals| return argument contains information about how well the
%
    model simulation results match the experimental data and is used by
%
    the |sdo.optimize| function to estimate the model parameters.
%
%
    See also sdo.optimize, sdoExampleCostFunction
%
```

```
% Copyright 2012-2015 The MathWorks, Inc.
%%
% Define a signal tracking requirement to compute how well the model output
% matches the experiment data. Configure the tracking requirement so that
% it returns the tracking error residuals (rather than the
% sum-squared-error) and does not normalize the errors.
%
r = sdo.reguirements.SignalTracking;
           = '==';
r.Type
           = 'Residuals';
r.Method
r.Normalize = 'off';
%%
% Update the experiments with the estimated parameter values.
%
Exp = setEstimatedValues(Exp,v);
%%
% Simulate the model and compare model outputs with measured experiment
% data.
%
Error = [];
for ct=1:numel(Exp)
    Simulator = createSimulator(Exp(ct),Simulator);
    Simulator = sim(Simulator);
    SimLog = find(Simulator.LoggedData,get param('sdoBattery','SignalLoggingName'));
    Voltage = find(SimLog, 'Voltage');
   VoltageError = evalRequirement(r,Voltage.Values,Exp(ct).OutputData(1).Values);
    Error = [Error; VoltageError(:)];
end
202
% Return the residual errors to the optimization solver.
%
vals.F = Error(:);
end
```

Estimate the Parameters

Use the sdo.optimize function to estimate the battery parameter values.

Specify the optimization options. The estimation function sdoBattery_Objective returns the error residuals between simulated and experimental data and does not include any constraints, making this problem ideal for the 'lsqnonlin' solver.

```
opt = sdo.OptimizeOptions;
opt.Method = 'lsqnonlin';
```

Estimate the parameters.

```
v0pt = sdo.optimize(estFcn,v,opt)
```

```
Optimization started 02-Sep-2021 00:07:31
```

				First-order
Iter	F-count	f(x)	Step-size	optimality
\odot	7	3272.22	1	
1	14	619.356	0.1634	3.15e+05
2	21	411.131	0.2175	28.7
3	28	405.529	0.3838	2.16e+03
4	35	403.727	0.2767	15.2
5	42	403.379	0.1645	1.14e+03
Local	minimum	possible.		

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

v0pt(1,1) =

```
Name: 'V'
Value: 1.3083
Minimum: 0
Maximum: 2
Free: 1
Scale: 2
Info: [1x1 struct]
```

```
vOpt(2,1) =
```

```
Name: 'K'
Value: 0.0010
Minimum: 1.0000e-06
Maximum: 0.1000
Free: 1
Scale: 0.0020
Info: [1x1 struct]
```

vOpt(3,1) =

```
Name: 'Loss'
Value: 5.1801e-05
Minimum: 0
Maximum: 0.5000
Free: 1
Scale: 0.0156
Info: [1x1 struct]
```

```
3x1 param.Continuous
```

Compare the Measured Output and the Final Simulated Output

Update the experiments with the estimated parameter values.

Exp = setEstimatedValues(Exp,v0pt);

Obtain the simulated output for the first (discharging) experiment.

```
Simulator = createSimulator(Exp(1),Simulator);
Simulator = sim(Simulator);
```

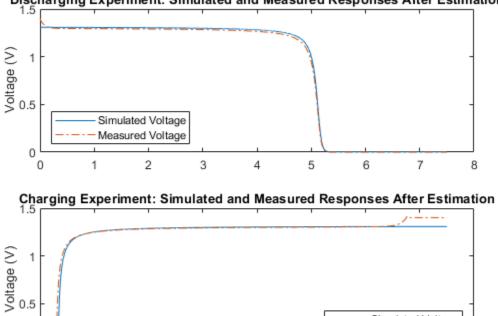
```
SimLog = find(Simulator.LoggedData,get_param('sdoBattery','SignalLoggingName'));
Voltage(1) = find(SimLog,'Voltage');
```

Obtain the simulated output for the second (charging) experiment.

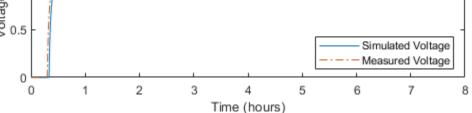
```
Simulator = createSimulator(Exp(2),Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoBattery','SignalLoggingName'));
Voltage(2) = find(SimLog,'Voltage');
```

Plot the measured and simulated data. The simulation results match the experimental data well except in the regions when the battery is fully charged. This is not unexpected as the simple battery model does not model the exponential voltage drop when the battery is fully charged.

```
subplot(211)
plot(...
    Voltage(1).Values.Time/3600,Voltage(1).Values.Data, ...
    Exp(1).OutputData.Values.Time/3600, Exp(1).OutputData.Values.Data,'-.')
title('Discharging Experiment: Simulated and Measured Responses After Estimation')
ylabel('Voltage (V)')
legend('Simulated Voltage','Measured Voltage','Location','SouthWest')
subplot(212)
plot(...
    Voltage(2).Values.Time/3600,Voltage(2).Values.Data, ...
    Exp(2).OutputData.Values.Time/3600, Exp(2).OutputData.Values.Data,'-.')
title('Charging Experiment: Simulated and Measured Responses After Estimation')
xlabel('Time (hours)')
ylabel('Voltage (V)')
legend('Simulated Voltage','Measured Voltage','Location','SouthEast')
```



Discharging Experiment: Simulated and Measured Responses After Estimation



Update the Model Parameter Values

Update the model V, K, and Loss parameter values.

sdo.setValueInModel('sdoBattery',v0pt);

Related Examples

To learn how to estimate the battery parameters using the **Parameter Estimator**, see "Estimate Model Parameters Per Experiment (GUI)" on page 2-154.

Close the model

bdclose('sdoBattery')

Set Model to Steady-State When Estimating Parameters (Code)

This example shows how to set a model to steady-state in the process of parameter estimation. Setting a model to steady-state is important in many applications such as power systems and aircraft dynamics. This example uses a population dynamics model.

This example requires Simulink[®] Control Design[™] software.

Model Description

The Simulink model sdoPopulationInflux models a simple ecology where an organism population growth is limited by the carrying capacity of the environment.

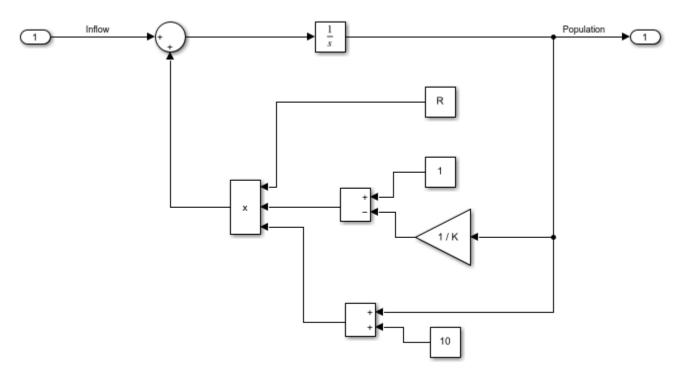
$$\frac{dy}{dt} = R(1 - \frac{y}{K})(y + 10)$$

- *R* is the inherent growth rate of the organism population.
- *K* is the carrying capacity of the environment.

There is also an influx of other members of the organism from a neighboring environment. The model uses normalized units.

Open the model.

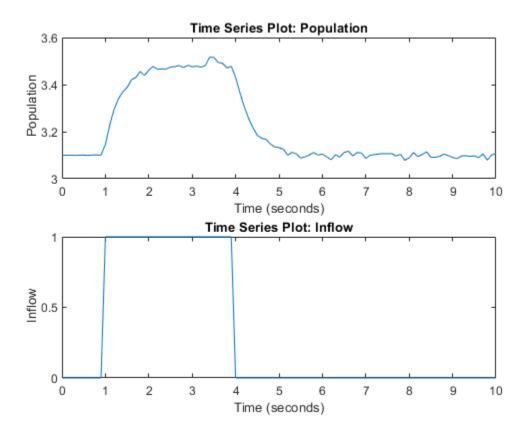
```
modelName = 'sdoPopulationInflux';
open_system(modelName)
```



Copyright 2018 The MathWorks, Inc.

The file sdoPopulationInflux_Data.mat contains data of the population in the environment as well as the influx of additional organisms from the neighboring environment.

```
load sdoPopulationInflux_Data.mat; % Time series: Population_ts Inflow_ts
hFig = figure;
subplot(2,1,1);
plot(Population_ts)
subplot(2,1,2);
plot(Inflow_ts)
```



The population starts in a steady state. After some time, there is an influx of organisms from the neighboring environment. Based on the measured data, you can estimate values for the model parameters.

The parameter R represents the inherent growth rate of the organism. Use 1 as the initial guess for this parameter. It is non-negative.

R = sdo.getParameterFromModel(modelName, 'R'); R.Value = 1; R.Minimum = 0;

The parameter K represents the carrying capacity of the environment. Use 2 as the initial guess for this parameter. It is known to be at least 0.1.

```
K = sdo.getParameterFromModel(modelName, 'K');
K.Value = 2;
K.Minimum = 0.1;
```

Collect these parameters into a vector.

v = [R K];

Compare Measured Data to Initial Simulated Output

Create an Experiment object.

Exp = sdo.Experiment(modelName);

Associate Population_ts with model output.

```
Population = Simulink.SimulationData.Signal;
Population.Name = 'Population';
Population.BlockPath = [modelName '/Integrator'];
Population.PortType = 'outport';
Population.PortIndex = 1;
Population.Values = Population_ts;
```

Add Population to the experiment.

Exp.OutputData = Population;

Associate Inflow ts with model input.

```
Inflow = Simulink.SimulationData.Signal;
Inflow.Name = 'Population';
Inflow.BlockPath = [modelName '/In1'];
Inflow.PortType = 'outport';
Inflow.PortIndex = 1;
Inflow.Values = Inflow_ts;
```

Add Inflow to the experiment.

Exp.InputData = Inflow;

Create a simulation scenario using the experiment, and obtain the simulated output.

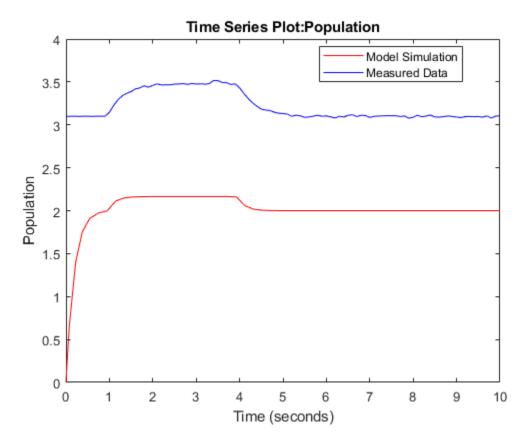
```
Exp = setEstimatedValues(Exp, v); % use vector of parameters/states
Simulator = createSimulator(Exp);
Simulator = sim(Simulator);
```

Search for the model output signal in the logged simulation data.

```
SimLog = find(Simulator.LoggedData, ...
get_param(modelName, 'SignalLoggingName') );
PopulationSim = find(SimLog, 'Population');
```

The model output does not match the data very well, indicating a need to compute better estimates of the model parameters.

```
clf;
plot(PopulationSim.Values, 'r');
hold on;
plot(Population_ts, 'b');
legend('Model Simulation', 'Measured Data', 'Location', 'best');
```



Estimate Parameters

To estimate parameters, define an objective function to compute the sum squared error between model simulation and measured data.

```
estFcn = @(v) sdoPopulationInflux_Objective(v, Simulator, Exp);
type sdoPopulationInflux_Objective.m
```

```
function vals = sdoPopulationInflux Objective(v, Simulator, Exp, OpPointSetup)
% Compare model output with data
%
%
     Inputs:
%
                     - vector of parameters and/or states
        V
%
        Simulator
                     - used to simulate the model
%
                     - Experiment object
        Exp
        OpPointSetup - Object to set up computation of steady-state
%
%
                       operating point
% Copyright 2018 The MathWorks, Inc.
%Parse inputs
if nargin < 4
    OpPointSetup = [];
end
% Requirement setup
req = sdo.requirements.SignalTracking;
```

```
req.Type = '==';
req.Method = 'Residuals';
% Simulate the model
Exp = setEstimatedValues(Exp, v); % use vector of parameters/states
Simulator = createSimulator(Exp,Simulator);
strOT = mat2str(Exp.OutputData(1).Values.Time);
if isempty(OpPointSetup)
    Simulator = sim(Simulator, 'OutputOption', 'AdditionalOutputTimes', 'OutputTimes', strOT);
else
    Simulator = sim(Simulator, 'OutputOption', 'AdditionalOutputTimes', 'OutputTimes', strOT, ...
        'OperatingPointSetup', OpPointSetup);
end
% Compare model output with data
SimLog = find(Simulator.LoggedData, ...
    get_param(Exp.ModelName, 'SignalLoggingName') );
OutputModel = find(SimLog, 'Population');
Model Error = evalRequirement(req, OutputModel.Values, Exp.OutputData.Values);
vals.F = Model Error;
%Define options for the optimization.
%
opts = sdo.OptimizeOptions;
opts.Method = 'lsqnonlin';
```

Estimate the parameters.

```
vOpt = sdo.optimize(estFcn, v, opts);
disp(vOpt)
```

Optimization started 02-Sep-2021 00:41:15

				First-order
Iter F	-count	f(x)	Step-size	optimality
0	5	12.485	1	
1	10	1.09824	1.184	0.244
2	15	0.9873	1.088	0.0259
3	20	0.952948	1.217	0.00624
4	25	0.946892	0.9151	0.00197
5	30	0.946484	0.3541	0.00153
		a a a i h 1 a		

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

(1,1) =

Name: 'R' Value: 5.5942 Minimum: 0 Maximum: Inf Free: 1 Scale: 1 Info: [1x1 struct]

```
(1,2) =
    Name: 'K'
    Value: 3.2061
    Minimum: 0.1000
    Maximum: Inf
        Free: 1
        Scale: 2
        Info: [1x1 struct]
```

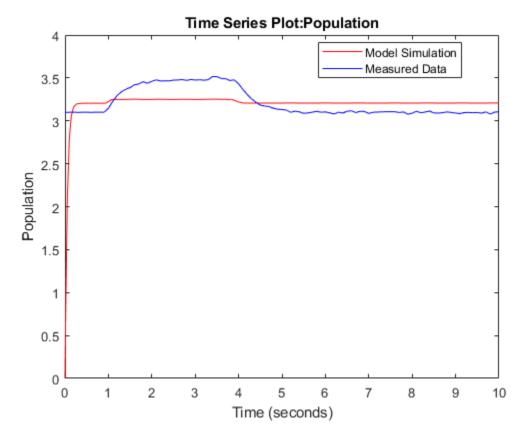
```
1x2 param.Continuous
```

Use the estimated parameter values in the model, and obtain the model response. Search for the model output signal in the logged simulation data.

```
Exp = setEstimatedValues(Exp, vOpt);
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData, ...
get_param(modelName, 'SignalLoggingName') );
PopulationSim = find(SimLog, 'Population');
```

Comparing the measured population data with the optimized model response shows that they still do not match well. There is a transient at the beginning of the model response, where it is markedly different from the measured data.

```
clf;
plot(PopulationSim.Values, 'r');
hold on;
plot(Population_ts, 'b');
legend('Model Simulation', 'Measured Data', 'Location', 'best');
```



Put Model in Steady-State During Estimation

To improve the fit between the model and measured data, the model needs to be set to steady-state when parameters are estimated. Define an operating point specification. The input is known from experimental data. Therefore, (1) it should not be treated as a free variable when computing the steady-state operating point, and (2) after the operating point is found, its input should not be used when simulating the model. On the other hand, all the states found when computing the operating point should be used when simulating the model. Create an sdo.OperatingPointSetup object to collect the operating point specification, inputs to use, and states to use, so this information can be passed to the objective function and used when simulating the model. You can also provide a fourth argument to sdo.OperatingPointSetup to specify options for computing the operating point. For example, the option 'graddescent-proj' is often used to find the operating point for systems that use physical modeling.

```
opSpec = operspec(modelName);
opSpec.Inputs(1).Known = true;
inputsToUse = [];
statesToUse = 1:numel(opSpec.States);
OpPointSetup = sdo.OperatingPointSetup(opSpec, inputsToUse, statesToUse);
```

Estimate the parameters, setting the model to steady-state in the process.

```
estFcn = @(v) sdoPopulationInflux_Objective(v, Simulator, Exp, OpPointSetup);
vOpt = sdo.optimize(estFcn, v, opts);
disp(vOpt)
```

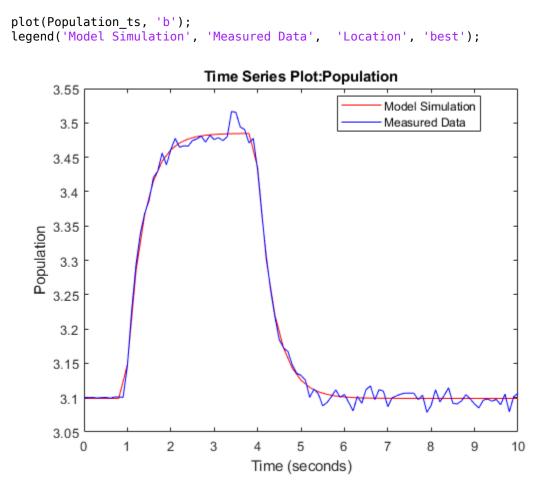
```
Optimization started 02-Sep-2021 00:41:29
                                          First-order
 Iter F-count
                     f(x)
                               Step-size optimality
   0
         5
                  11.1517
                                    1
    1
          10
                 0.025674
                                0.5732
                                              0.045
    2
             0.00239293
         15
                                0.3451
                                              0.357
    3
          20
              0.000692478
                                0.0148
                                            0.00301
    4
          25
             0.00069236
                             6.539e-05
                                           1.16e-07
Local minimum found.
Optimization completed because the size of the gradient is less than
the value of the optimality tolerance.
(1,1) =
      Name: 'R'
      Value: 0.5953
   Minimum: 0
   Maximum: Inf
       Free: 1
      Scale: 1
       Info: [1x1 struct]
(1,2) =
      Name: 'K'
      Value: 3.0988
   Minimum: 0.1000
   Maximum: Inf
      Free: 1
      Scale: 2
       Info: [1x1 struct]
1x2 param.Continuous
```

Use the estimated parameter values in the model, and obtain the model response. Search for the model output signal in the logged simulation data.

```
Exp = setEstimatedValues(Exp, v0pt);
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator, 'OperatingPointSetup', OpPointSetup);
SimLog = find(Simulator.LoggedData, ...
get_param(modelName, 'SignalLoggingName') );
PopulationSim = find(SimLog, 'Population');
```

There is no more transient at the beginning of the model response, and there is a much better match between the model response and measured data, which is also reflected by the lower objective/cost function value in the second optimization. All these indicate that a good set of parameter values was found.

```
clf;
plot(PopulationSim.Values, 'r');
hold on;
```



Related Examples

To learn how to put models in a steady state using the **Parameter Estimator** app, see "Set Model to Steady-State When Estimating Parameters (GUI)" on page 2-117.

Close the model and figure.

bdclose(modelName)
close(hFig)

Parameter Estimation for Power Plant Excitation System Starting at Steady-State (GUI)

This example shows how to perform parameter estimation while starting the system in steady state using the example of an excitation system model for a power plant electric generator.

Need for Parameter Estimation in Power Systems

Parameter estimation is a powerful tool for power system operations where the accuracy of models is critical and may be required by regulation. There are several reasons why one might need to perform parameter estimation in power systems, including:

- The system parameters may have been unknown from the start. For instance, if some or all of the parameters were not provided by the supplier.
- Even if the system parameters were known in the past, these parameters may drift with time due to wear on components in the system.
- Some settings may be changed for the system causing unknown effects on the system parameters. Parameter estimation can be used to account for these settings changes.
- The system may need to be fit to some standardized model. For instance, in this example we are fitting the IEEE DC1A standard model for excitation systems to our system.

Excitation System Model Description

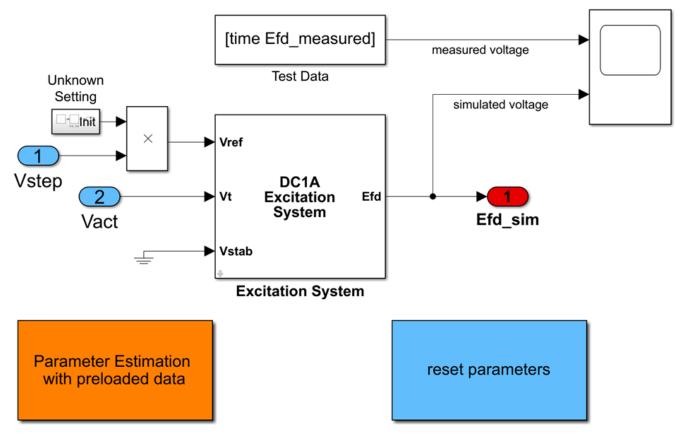
Generators create power by rotating a magnetic field and coiled wires relative to one another to induce an electric current. For generators that use electromagnets, an excitation system supplies current to the generator's field coils to create the magnetic field. By controlling the strength of the magnetic field inside the generator, the exciter system can control the generator's output voltage.

The Simulink® model spe_exciter models an excitation system in an offline step test. In this test, the generator is taken offline, then a step voltage input is applied to the exciter, and the output voltage is measured for system characterization purposes. This model includes the subsystem labeled "DC1A Excitation System" which follows the model structure for an excitation system outlined in the IEEE DC1A standard. The block contains several parameters, such as gains and time constants, that define the system's behavior and need to be fit to our system. The voltage inputs and outputs are in p.u. (per unit).

You can open the model with the following command:

```
open_system('spe_exciter');
```

Exciter Offline Step Test Model

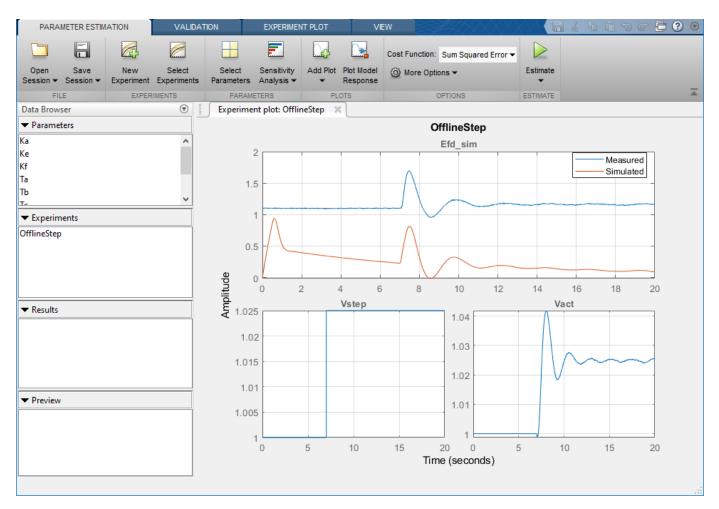


Open Parameter Estimator App

Double-click the orange block labeled Parameter Estimation with preloaded data in the lower left corner of the model. This will launch a Parameter Estimation session pre-loaded with data for this project, including the experimental data from the offline step test.

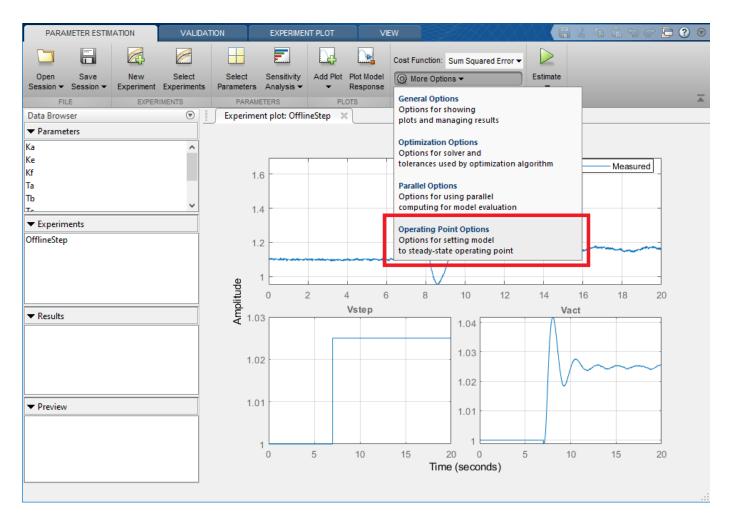
The Parameter Estimation session is loaded with the system parameters which were determined to need tuning due to any of the reasons noted previously. These parameters include the gains Ka, Ke, and Kf; and time constants Ta, Tb, Tc, Te, Tf, and Tr. These parameters are bound to use only positive values during estimation.

To plot the model's response against the experimental data, click the **Plot Model Response** button in the toolbar. Notice that the initial conditions for the states in our model are currently incorrect, which causes the initial dynamic in the simulated response and the offset between the simulated and measured response. In the next step we will update the options in the **Parameter Estimator** app to solve for the correct initial conditions in our model.



Compute Steady-State Operating Point During Parameter Estimation

In the experimental step test that produced the measured response, the excitation system was in steady state and outputting around 1.1 p.u. before the test measurements began. To match these conditions in our parameter estimation, we will specify that the model should start at a steady-state operating point during parameter estimation. Click **More Options** and select **Operating Point Options**.



This shows a dialog where you can specify how steady-state operating points should be computed during parameter estimation. Open the dialog and select the check box Estimate at steady-state so that the **Parameter Estimator** will put the model into steady-state each time it varies parameters and runs the model. There are seven states in this model, by default they will be set to unknown and marked as states to be set to steady-state. This matches our system, so we will keep these options unchanged.

Operating) Point							>
🗹 Estim	nate at	steady-state						
Experime	ent: 0	fflineSten	 Sync w 	ith specification from	Simulink Mor	lel V (<u>c</u>	
Experiment: OfflineStep \checkmark Sync with specification from: Simulink Model \checkmark								
Specifications Options								
States	Input	s Outputs						
			State Specifications					
Sta	ate	Value	Known	Steady State	Minimum	Maximum	dx Minimum	dx Maximum
spe_ex	citer/E	xcitation System	n/(Tc.s+1)//(Tb.s+1)/	/Integrator				
State - 1	1	0			-Inf	Inf	-Inf	Inf
spe_ex	citer/E	xcitation System	n/Integrator					
State - 1	1	0		\checkmark	-Inf	Inf	-Inf	Inf
spe_exciter/Excitation System/k.s//(tau.s+1)/Integrator								
State - 1	1	0		\checkmark	-Inf	Inf	-Inf	Inf
spe_exciter/Excitation System/k//(tau.s+1)/1//(taus+1)/Integrator								
State - 1	1	0		\checkmark	-Inf	Inf	-Inf	Inf
spe_ex	citer/E	xcitation System	n/k//(tau.s+1)2/1//(t	taus+1)/Integrator				
State - 1	1	0		\checkmark	-0.9	1	-Inf	Inf
spe_ex	citer/l	Inknown Setting	g/Init_Val					
State - 1	1	1		\checkmark	-Inf	Inf	-Inf	Inf
Help								Close

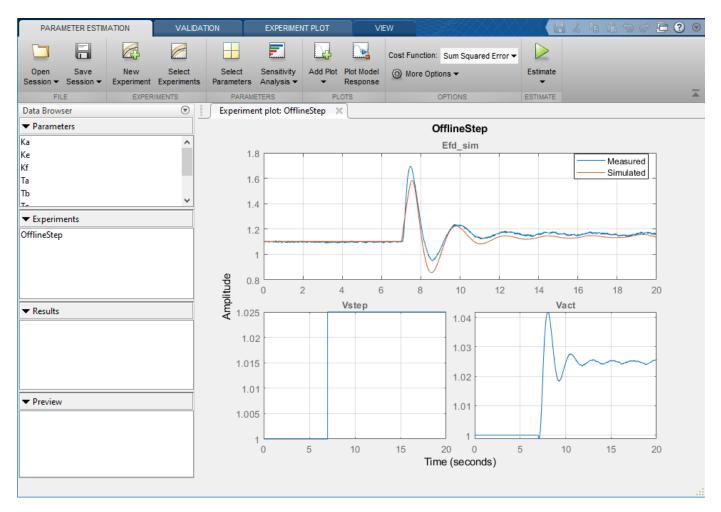
The inputs to the model (terminal voltage and reference voltage) are known from the offline step test. Switching to the **Inputs** tab under **Specifications**, we can specify these conditions. We can see that the inputs are marked as known by default with a value of one. These come from the starting value in the measured data, and we will leave these values unchanged.

Operating Point				
Estimate at steady-state				
Experiment: OfflineStep	 Sync with 	h specification from: Simul	ink Model 🗸 😪	1
Specifications Options				
States Inputs Outputs				
States mpan outputs			Input Specifications	
Input	Value	Known	Minimum	Maximum
spe_exciter/Vstep		·		
Channel - 1	1		-Inf	Inf
spe_exciter/Vact				
Channel - 1	1		-Inf	Inf
Help				Close

Switching to the **Outputs** tab under **Specifications**, we mark the output (field voltage) of our system as known by checking the "Known" checkbox, and set its "Value" to 1.1028, which is the first value of our measured field voltage test data.

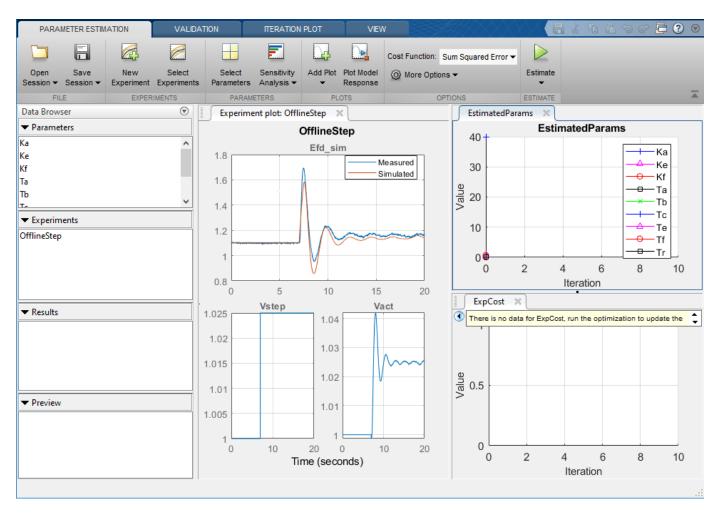
Operating Poi	int						×
🗹 Estimate	at steady-	state					
Experiment:	OfflineSte	ep ∨	Syne	c with specification from:	Simulink Model	< C	
Specificatio	ons Ontio	ns					
States Inp							
				1	Output Specifica	ations	
Ou	tput	Value		Known	Minimum	Maximum	
spe_excite						-	
Channel - 1		1.1028			-Inf	Inf	
Help							Close

With the options we have now set up, before running each model simulation, Parameter Estimation will solve for a set of initial conditions that will place all the specified states in steady-state at the specified input and output levels. To see the result of these changes, click **Plot Model Response** again and see that the simulated response is now in steady state at the expected initial output.



Set up Parameter Estimation View

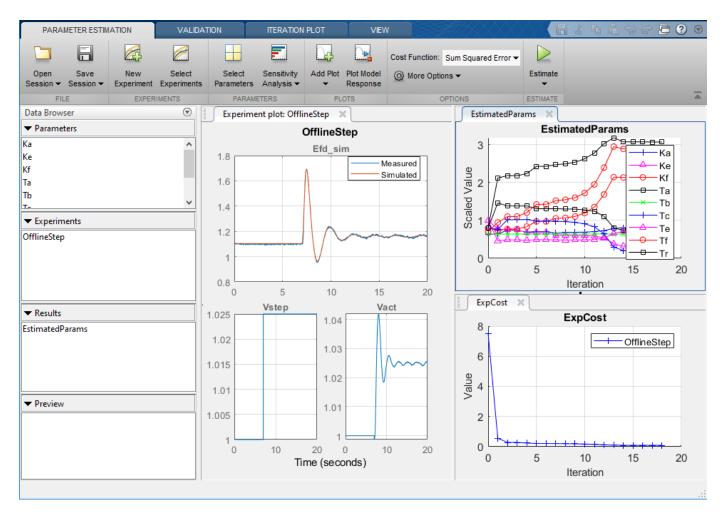
Before estimating parameters, we can use the toolbar to customize the view of Parameter Estimation to display the information we are interested in. Use the **Add Plot** button on the toolbar to add a **Parameter Trajectory** plot and an **Estimation Cost** plot. You can use the **View** tab to adjust the layout and make all plots visible.



Perform Parameter Estimation

Now we are ready to perform parameter estimation. In the **Parameter Estimation** tab, click **Estimate**.. Due to the large number of parameters being estimated in this example, this process may take several minutes.

Once the estimation process has converged, the new model response is shown in the **Experiment Plot**. We see a better match between the model and the measured data, and the error in the **ExpCost** plot decreased significantly. These indicate that a good set of parameters was found. The **EstimatedParams** plot shows how each parameter changed at each iteration. To more clearly see how much each parameter changed relative to its initial value, right click the **EstimatedParams** plot and select Show scaled values.



Speed Up Estimation Using Parallel Pool Options

Because of the large number of parameters being estimated, the parameter estimation can take a long time. As the number of parameters increases, the number of times the model must run at each iteration also increases. This leads to an increase in the total computation time required for the parameter estimation to converge.

To speed up our parameter estimation we can set up our options to use a parallel pool. Then our parallel workers can run simulations simultaneously to speed up the parameter estimation process.

To do this you will need **MATLAB Parallel Computing Toolbox**. Before performing parameter estimation, go to **More Options>Parallel Options** in the Parameter Estimation toolbar. Then select Use parallel pool during estimation. Click OK, then click **Estimate** in the toolbar.

For a parallel pool with 8 workers, the estimation process for this example was 3.5 times faster to complete. For access to options related to parallel computing like number of workers and cluster setup, see "Specify Your Parallel Preferences" (Parallel Computing Toolbox).

See Also Parameter Estimator

Related Examples

- "Set Model to Steady-State When Estimating Parameters (GUI)" on page 2-117
- "Set Model to Steady-State When Estimating Parameters (Code)" on page 2-97

More About

- "Specify Steady-State Operating Point for Parameter Estimation" on page 2-47
- "What Is an Operating Point?" (Simulink Control Design)
- "What Is a Steady-State Operating Point?" (Simulink Control Design)

Set Model to Steady-State When Estimating Parameters (GUI)

This example shows how to set a model to steady-state in the process of parameter estimation. Setting a model to steady-state is important in many applications such as power systems and aircraft dynamics. This example uses a population dynamics model.

Model Description

The Simulink® model sdoPopulationInflux models a simple ecology where an organism population growth is limited by the carrying capacity of the environment.

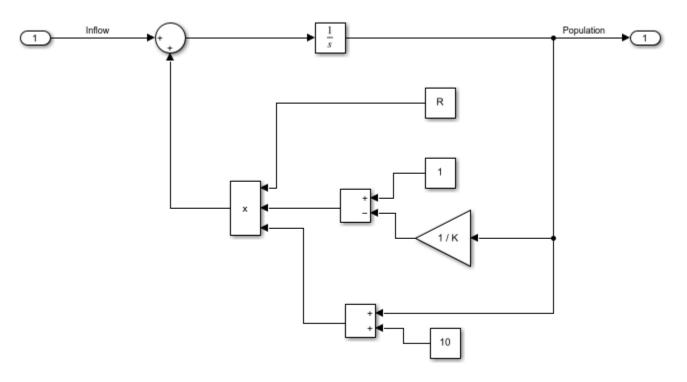
$$\frac{dy}{dt} = R(1 - \frac{y}{K})(y + 10)$$

- *R* is the inherent growth rate of the organism population.
- *K* is the carrying capacity of the environment.

There is also an influx of other members of the organism from a neighboring environment. The model uses normalized units.

Open the model.

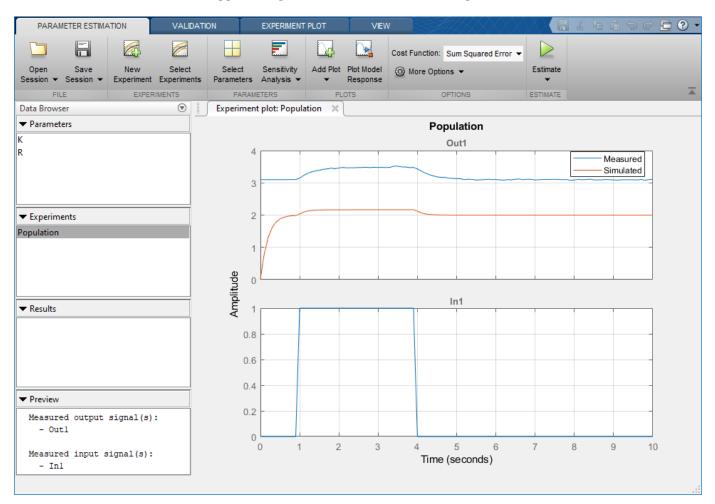
```
open_system('sdoPopulationInflux')
```



Copyright 2018 The MathWorks, Inc.

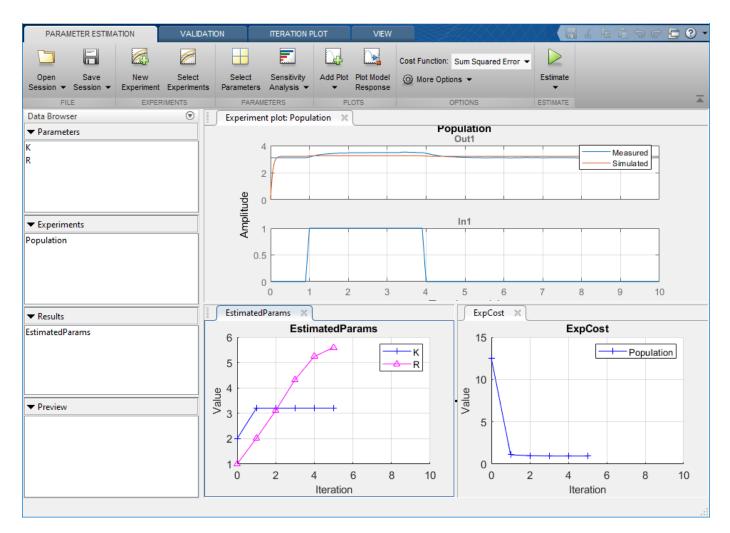
Open Parameter Estimator App

To open the **Parameter Estimator**, in the Simulink model window, in the **Apps** gallery, under **Control Systems**, click **Parameter Estimator**. In **Parameter Estimator**, click **Open Session** and select **Open from model workspace**, and then select sdoPopulationInflux_spesession to load a session with population experiment data already loaded. In the toolstrip, click **Plot Model Response** to plot the model response with the model's initial parameter values for R and K. The plot shows that with the model's initial parameter values, the model output is not close to the measured data. You can use the app to compute better estimates of these parameters.



Estimate Parameters

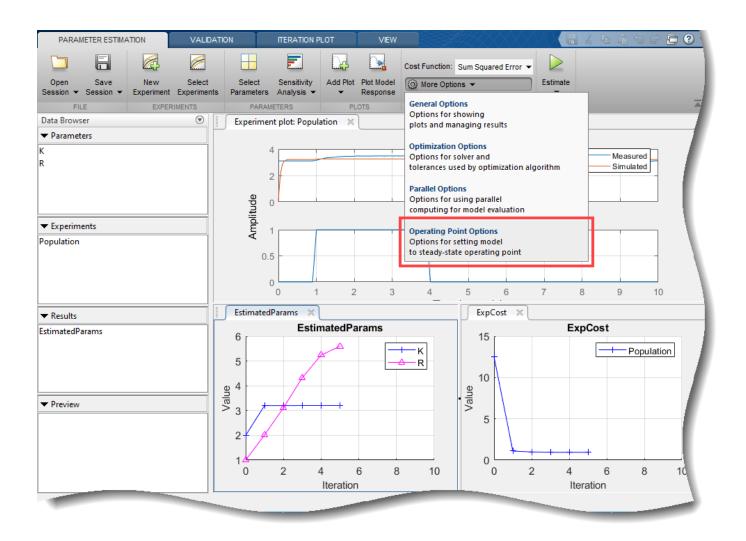
The pre-configured parameter estimation session also specifies that R and K are to be estimated, and that their lower bounds are 0 since the inherent growth rate and environment carrying capacity are not negative. In the toolstrip, click **Add Plot** and add a plot to show the parameter trajectories during estimation, and another plot to show the estimation cost. Use the **View** tab to lay out the plots in a convenient format. Click **Estimate** to estimate parameters R and K. The optimization goes through several iterations, changing the values of the parameters to improve the fit between model response and data.



Comparing the measured population data with the optimized model response shows that they still do not match well. There is a transient at the beginning of the model response, where it is markedly different from the measured data.

Compute Steady-State Operating Point During Parameter Estimation

To improve the fit between the model and measured data, the model needs to be set to steady-state when parameters are estimated. In the toolstrip click **More Options** and select **Operating Point Options**.



This shows a dialog where you can specify how to compute steady-state operating points during parameter estimation. There is one state in this model, namely the initial condition of the integrator. Use the operating point dialog to specify that this state should be treated as an unknown, and it should be set to steady state. During parameter estimation, the operating point computation will vary this state to set it at steady-state.

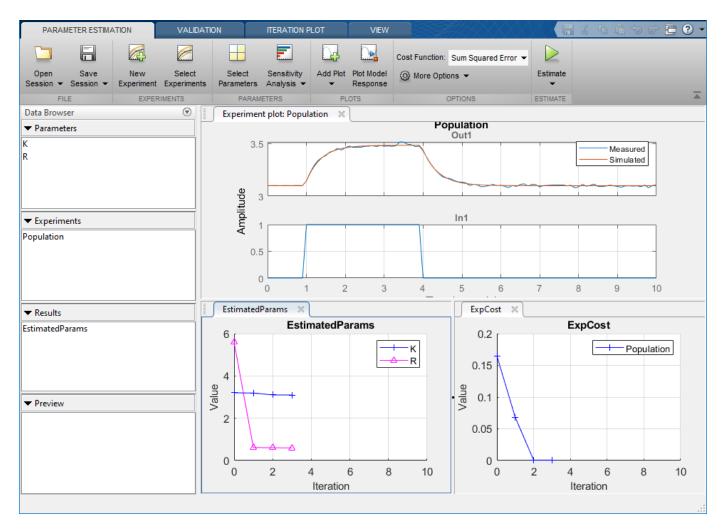
Operating Point)
☑ Estimate at s	steady-state						
Experiment: Po	pulation	 ✓ Sync w 	ith specification from	: Simulink Mod	lel 🗸 🤅	<u>e</u>	
Specifications	Options						
States Inputs							
				State Specificat	ions		
State	Value	Known	✓ Steady State	Minimum	Maximum	dx Minimum	dx Maximum
sdoPopulation	Influx/Integrat						
State - 1	0		\checkmark	-Inf	Inf	-Inf	Inf
Help							Close

The input to the model is known from the experimental data for the population influx. Use the operating point dialog to specify that this input is known. This input will not be varied by the operating point computation during parameter estimation.

Operating	Point						×
🗹 Estim	nate at s	teady-state					
Experime	_		~	Sync with	specification from: Simul	ink Model 🗸 🎯]
				-,		0	1
		Options					
States	inputs	Outputs	1			land Caral Carting	
	Inpu	t		Value		Input Specifications	
					🗹 Known	Minimum	Maximum
sdoPop Channe		Influx/In1		0		-Inf	Inf
Channe	9-1			U		-INT	Int
Help							Close

You can also specify options for computing the operating point, by using the options tab in the dialog. For example, the option Gradient descent with projection is often used to find the operating point for systems that use physical modeling.

After specifying to compute the operating point, click **Estimate** and perform parameter estimation again. There is no more transient at the beginning of the model response, and there is a much better match between the model response and measured data, which is also reflected by the lower objective/ cost function value in the second optimization. This indicates that a good set of parameter values was found.



Related Examples

To learn how to put models in a steady state using the **sdo.optimize** command, see "Set Model to Steady-State When Estimating Parameters (Code)" on page 2-97.

Close the model.

bdclose('sdoPopulationInflux')

Estimate Model Parameters with Parameter Constraints (Code)

This example shows how to estimate model parameters while imposing constraints on the parameter values.

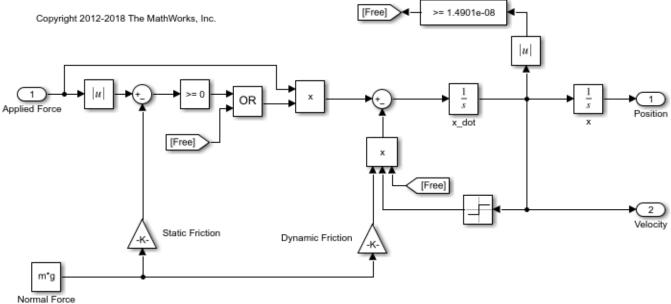
You estimate dynamic and static friction coefficients of a simple friction system.

Open the Model and Get Experimental Data

This example estimates parameters for a simple friction system, **sdoFriction**. The model input is the force applied to a mass and the model outputs are the mass position and velocity.

open_system('sdoFriction');

Friction Model



The model is based on a mass sliding on a surface. The mass is subject to a static friction that must be overcome before the mass moves and a dynamic friction once the mass moves. The static friction, u_static, is a fraction of the mass normal force; similarly the dynamic friction, u_dynamic, is a fraction of the mass normal force.

Load the experiment data. The mass was subjected to an applied force and its position recorded.

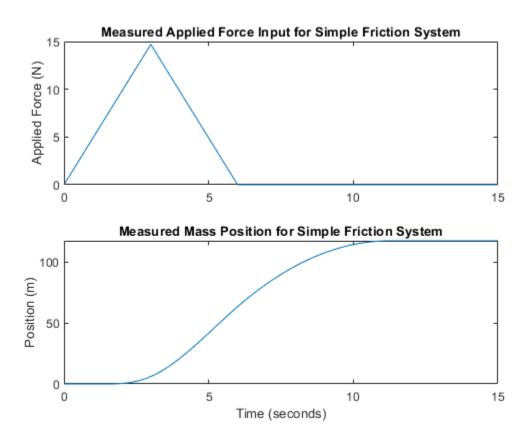
load sdoFriction_ExperimentData

The variables AppliedForce, Position, and Velocity are loaded into the workspace. The first column of each of these variables represents time and the second column represents the measured data. Because velocity is the first derivative of position, only use the position measurements for this example.

Plot the Experiment Data

```
subplot(211),
plot(AppliedForce(:,1),AppliedForce(:,2))
```

```
title('Measured Applied Force Input for Simple Friction System');
ylabel('Applied Force (N)')
subplot(212)
plot(Position(:,1),Position(:,2))
title('Measured Mass Position for Simple Friction System');
xlabel('Time (seconds)')
ylabel('Position (m)')
```



Define the Estimation Experiment

Create an experiment object to specify the experiment data.

```
Exp = sdo.Experiment('sdoFriction');
```

Specify the input data (applied force) as a timeseries object.

```
Exp.InputData = timeseries(AppliedForce(:,2),AppliedForce(:,1));
```

Create an object to specify the measured mass position output.

```
PositionSig = Simulink.SimulationData.Signal;
PositionSig.Name = 'Position';
PositionSig.BlockPath = 'sdoFriction/x';
PositionSig.PortType = 'outport';
PositionSig.PortIndex = 1;
PositionSig.Values = timeseries(Position(:,2),Position(:,1));
```

Add the measured mass position data to the experiment as the expected output data.

Exp.OutputData = PositionSig;

Compare the Measured Output and the Initial Simulated Output

Create a simulation scenario using the experiment and obtain the simulated output.

```
Simulator = createSimulator(Exp);
Simulator = sim(Simulator);
```

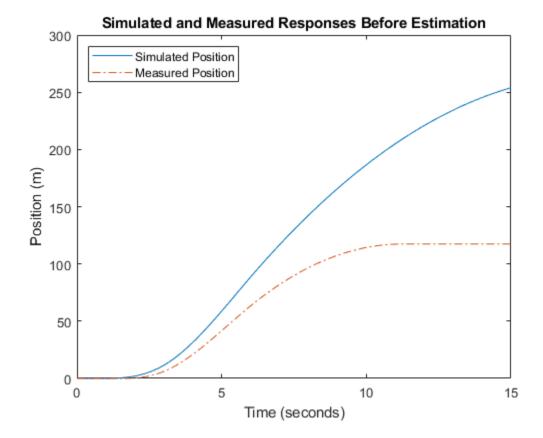
Search for the position signal in the logged simulation data.

```
SimLog = find(Simulator.LoggedData,get_param('sdoFriction','SignalLoggingName'));
Position = find(SimLog,'Position');
```

Plot the measured and simulated data.

As expected, the model response does not match the experimental output data.

```
figure
plot(...
    Position.Values.Time,Position.Values.Data, ...
    Exp.OutputData.Values.Time, Exp.OutputData.Values.Data,'-.')
title('Simulated and Measured Responses Before Estimation')
ylabel('Position (m)')
xlabel('Time (seconds)')
legend('Simulated Position','Measured Position','Location','NorthWest')
```



Specify Parameters to Estimate

Estimate the u_static and u_dynamic friction coefficients using the experiment data. These coefficients are used as gains in the Static Friction and Dynamic Friction blocks, respectively. Physics indicates that friction coefficients should be constrained so that u_static \geq u_dynamic; this parameter constraint is implemented in the estimation objective function.

Select the u_static and u_dynamic model parameters. Specify bounds for the estimated parameter values. Both coefficients are limited to the range [0 1].

```
p = sdo.getParameterFromModel('sdoFriction', {'u_static', 'u_dynamic'});
```

```
p(1).Minimum = 0;
p(1).Maximum = 1;
p(2).Minimum = 0;
p(2).Maximum = 1;
```

Define the Estimation Objective

Create an estimation objective function to evaluate how closely the simulation output, generated using the estimated parameter values, matches the measured data.

Use an anonymous function with one input argument that calls the sdoFriction_Objective function. We pass the anonymous function to sdo.optimize, which evaluates the function at each optimization iteration.

estFcn = @(v) sdoFriction_Objective(v,Simulator,Exp);

The sdoFriction_Objective function:

- Has one input argument that specifies the estimated friction coefficients.
- Has one input argument that specifies the experiment object containing the measured data.
- Returns the sum-squared-error errors between simulated and experimental outputs, and returns the parameter constraint.

The sdoFriction_Objective function requires two inputs, but sdo.optimize requires a function with one input argument. To work around this, estFcn is an anonymous function with one input argument, v, but it calls sdoFriction_Objective using two input arguments, v and Exp.

For more information regarding anonymous functions, see "Anonymous Functions".

The sdo.optimize command minimizes the return argument of the anonymous function estFcn, that is, the residual errors returned by sdoFriction_Objective. For more details on how to write an objective/constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB® command prompt.

To examine the estimation objective function in more detail, type edit sdoFriction_Objective at the MATLAB command prompt.

type sdoFriction_Objective

```
function vals = sdoFriction_Objective(p,Simulator,Exp)
%SD0FRICTION_OBJECTIVE
```

```
%
%
     The sdoFriction Objective function is used to compare model
%
     outputs against experimental data and measure how well constraints are
%
     satisfied.
%
%
     vals = sdoFriction Objective(p,Exp)
%
%
     The |p| input argument is a vector of estimated model parameter values.
%
%
     The [Simulator] input argument is a simulation object used
%
     simulate the model with the estimated parameter values.
%
%
     The [Exp] input argument contains the estimation experiment data.
%
%
     The |vals| return argument contains information about how well the
%
     model simulation results match the experimental data and how well
%
     constraints are satisfied. The |vals| argument is used by the
%
     |sdo.optimize| function to estimate the model parameters.
%
%
     See also sdo.optimize, sdoExampleCostFunction, sdoFriction cmddemo
%
% Copyright 2012-2015 The MathWorks, Inc.
% Define a signal tracking requirement to compute how well the model output
% matches the experiment data. Configure the tracking requirement so that
% it returns the sum-squared-error.
%
r = sdo.requirements.SignalTracking;
r.Type = '==';
r.Method = 'SSE';
0,0
% Update the experiments with the estimated parameter values.
%
Exp = setEstimatedValues(Exp,p);
%%
% Simulate the model and compare model outputs with measured experiment
% data.
%
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get param('sdoFriction','SignalLoggingName'));
Position = find(SimLog, 'Position');
PositionError = evalRequirement(r,Position.Values,Exp.OutputData(1).Values);
%%
\% Measure how well the parameters satisfy the friction coefficient constraint,
\% |u static| >= |u dynamic|. Note that constraints are returned to the
% optimizer in a c <=0 format. The friction coefficient constraint is
% rewritten accordingly.
PConstr = p(2).Value - p(1).Value; % u_dynamic - u_static <= 0</pre>
```

%%

```
% Return the sum-squared-error and constraint violation to the optimization
% solver.
%
vals.F = PositionError(:);
vals.Cleq = PConstr;
end
```

The friction coefficient constraint, u_static \geq u_dynamic, is implemented in the sdoFriction_Objective function as u_dynamic - u_static \leq 0. This is because the optimizer requires constraint values in a $c \leq 0$ format. For more information, type help sdo.optimize at the MATLAB command prompt.

Estimate the Parameters

Use the sdo.optimize function to estimate the friction model parameter values.

Specify the optimization options. The estimation function sdoFriction_Objective returns the sum-squared-error between simulated and experimental data and includes a parameter constraint. The default 'fmincon' solver is ideal for this type of problem.

Estimate the parameters.

```
p0pt = sdo.optimize(estFcn,p)
```

Optimization started 02-Sep-2021 00:33:28

			max			First-order
Iter	F-count	f(x)	constra	int	Step-size	optimality
0	5	27.7267	7	0		
1	11	22.5643	3	0	2.21	72.9
2	15	17.4771	_	0	0.51	16
3	22	0.76336	5	0	1.33	10.7
4	29	0.408381	_	0	0.263	3.15
5	34	0.0255292)	0	0.0897	1.22
6	39	0.00527178	3	0	0.0295	0.271
7	44	0.00405706	5	0	0.02	0.177
8	49	0.00111788	3	0	0.109	0.176
9	63	0.00111788	3	0	0.00176	0.176
Local	minimum	possible. C	Constraints	sat	isfied.	

fmincon stopped because the size of the current step is less than the value of the step size tolerance and constraints are satisfied to within the value of the constraint tolerance.

pOpt(1,1) =

```
Name: 'u_static'
Value: 0.8224
Minimum: 0
Maximum: 1
Free: 1
Scale: 0.5000
Info: [1x1 struct]
```

pOpt(2,1) =

```
Name: 'u_dynamic'
Value: 0.3973
Minimum: 0
Maximum: 1
Free: 1
Scale: 0.2500
Info: [1x1 struct]
```

2x1 param.Continuous

Compare the Measured Output and the Final Simulated Output

Update the experiments with the estimated parameter values.

Exp = setEstimatedValues(Exp,p0pt);

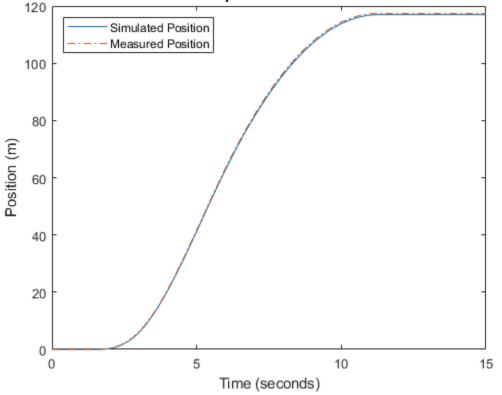
Obtain the simulated output for the experiment.

```
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData,get_param('sdoFriction','SignalLoggingName'));
Position = find(SimLog,'Position');
```

Plot the measured and simulated data.

It can be seen that the model response using the estimated parameter values nicely matches the experiment output data.

```
plot(...
    Position.Values.Time,Position.Values.Data, ...
    Exp.OutputData.Values.Time, Exp.OutputData.Values.Data,'-.')
title('Simulated and Measured Responses After Model Parameter Estimation')
ylabel('Position (m)')
xlabel('Time (seconds)')
legend('Simulated Position','Measured Position','Location','NorthWest')
```



Simulated and Measured Responses After Model Parameter Estimation

Update the Model Parameter Values

Update the model u_static and u_dynamic parameter values.

sdo.setValueInModel('sdoFriction',pOpt);

Close the model.

bdclose('sdoFriction')

Importing and Preprocessing Experiment Data (GUI)

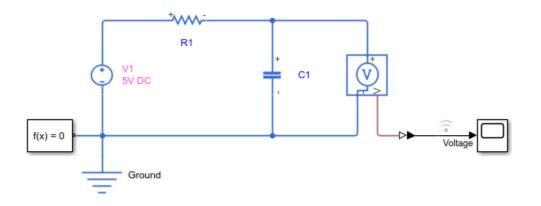
This example shows how to create an estimation experiment from measured data stored in a file and how to preprocess the measured data. You use the imported data to estimate the parameters of a simple RC circuit.

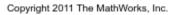
This example requires Simscape[™] software.

RC Circuit Model

The Simulink® model, sdoRCCircuit, models a simple resistor-capacitor (RC) circuit.

```
open_system('sdoRCCircuit');
```





You use measured data to estimate the RC model parameter and state values.

Measured output data:

• Capacitor voltage, output of the PS-Simulink Converter block

Parameter:

• Capacitance, C1, used by the C1 block

State:

• Initial voltage of the capacitor

Define the Estimation Experiment

In this example, you load the measured data from a saved MATLAB® file. The data is also stored in a comma separated variable (csv) text file. You can also load the measured data directly from text or Excel® files.

First load the measured data from the MATLAB file, the file defines two variables, time and data that specify the measured capacitor voltage.

load sdoRCCircuit_ExperimentData

To launch the **Parameter Estimator**, in the Simulink model window, in the **Apps** gallery, under **Control Systems**, click **Parameter Estimator**.

PARA	METER ESTIMA		VALIDATI	ON	VIEW						1 h i b c f ? -
	-							Cost Function:	Sum Squared Error 💌		
Open Session 👻	Save Session 💌	New Experiment	Select Experiments	Select Parameters	Sensitivity Analysis 💌	Add Plot	Plot Model Response	More Option		Estimate	
F	ILE	EXPER	MENTS	PARAM	ETERS	PL	.OTS		OPTIONS	ESTIMATE	
Data Brow	/ser										
▼ Parame	eters										
Experiment	nents										
- 0 - 11											
 Results 											
Preview	v										
	-										

Click **New Experiment** to create an estimation experiment that contains the measured data. A variable Exp is created in the **Parameter Estimator** and a dialog to edit the experiment opens.

PARAMETER ESTIMA	TION	VALIDATI	011	VIEW		2000 S	NA SS
PARAMETER ESTIMA	ATION	VALIDATI	UN	VIEW		2444 	
							Cost Func
Open Save Session ▼ Session ▼	New Experiment	Select Experiments	Select Parameters	Sensitivity Analysis 💌		Plot Model Response	More
FILE	Edit Experin	nent: Exp					x
Data Browser	Outputs						
▼ Parameters		easured output	-	s experiment.			
	/PS-Sim	ulink Converter	:1 (Voltage)				
	<1x1 Sig	gnal, 1 points>			-	🦻 🛃 🗡	
	<mark>, Se</mark> le	ct Measured O	utput Signals				
	Initial St	ates					51
	Optionally	define initial sta	ates for this ex	periment.			
 Experiments 	There are	currently no ini	tial states defir	ned for this exp	periment.		
Exp	Selec	ct Initial States					
-1	Paramet	ore					1
		define paramet	ers for this ex	periment.			
	There are	currently no pa	rameters defin	ed for this exp	periment.		
		ct Parameters					
▼ Results							-
• Results	=						
	-		Not & Sin	nulate 🔀 P	Plot 🛷 OF	K 🕐 Hel	In
▼ Preview			in the dealer				·

The experiment editor contains sections to specify measured output data and sections to optionally specify experiment initial states and parameters.

The experiment editor automatically adds measured output signals for model root level ports and logged model signals. Click **Select Measured Output Signals** to add additional measured outputs if needed. For this example the capacitor voltage signal is logged in the model and already added to the experiment.

Specify the measured capacitor voltage by typing [time data] in the edit field. This uses the MATLAB variables time and data loaded from file earlier to specify the measured capacitor voltage. Measured data is specified as a matrix where the 1st column is time and subsequent columns signal data.

Edit Experiment: Exp 🛛 🗙
Outputs Specify measured output signals for this experiment. /PS-Simulink Converter:1 (Voltage)
[time data] 🔹 🖶 🛓 🗶
Select Measured Output Signals
Initial States Optionally define initial states for this experiment.
There are currently no initial states defined for this experiment.
Parameters Optionally define parameters for this experiment.
There are currently no parameters defined for this experiment.
💽 Plot & Simulate 🛛 Plot 🖌 OK 🕜 Help

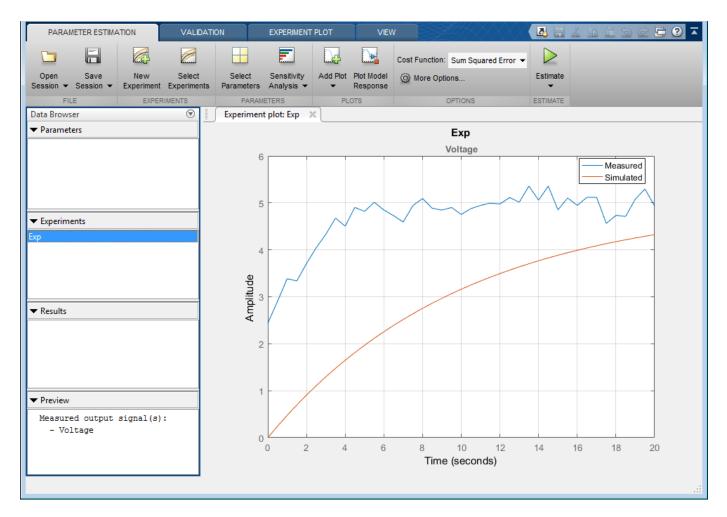
Alternatively, you can specify the measured capacitor voltage variables by loading the measured data directly from text or Excel files. Click the import button to open a file browser and select sdoRCCircuit_ExperimentData.csv file.

Outputs	
Specify measured output signals for this experiment.	
/PS-Simulink Converter:1 (Voltage)	-
[time data]	╼ ┣ (┺)×
Select Measured Output Signals	$\mathbf{}$

A tool for importing column data from a file opens. The first column selected for import is used to specify the signal time, subsequent columns selected for import are used to specify the signal data. Select the time and data columns and click **Import Selection**.

Rango Die Names Rov		Matrix	Replace	▼ unimportable cells	with 🔻 NaN	<u>-+</u>	Import Selection
SELEC		IMPORTED DATA		UNIMPORTABLE CELLS		V	IMPORT
	ExperimentData.	xlsx ×					
A		B					
	'S-Simulink Con ata	verter:1 (Voltage)					
0 0	ata	2.4398					
0.5000		2.9028					
1		3.3851					
1.5000		3.3380					
2		3.7081					
2.5000		4.0439					
3		4.3315					
3.5000		4.6756					
4.5000		4.5037 4.9031					
4.5000		4.9031					
5.5000		5.0132					
6		4.8488					
6.5000		4.7280					
7		4.5916					
7.5000		4.9418					
8		5.0926					
8.5000		4.8848					
9.5000		4.8467					
9.5000		4.9023					
10.5000							
10.5000			4.7520 4.8786 4.9457	4.8786	4.8786	4.8786	4.8786

In the experiment editor click ${\bf Plot}~\&~{\bf Simulate}$ to plot the measured experiment data and the simulated model response.



The experiment plot shows that the simulated data does not match the measured data. The plot also shows that the model initial state is not correct and needs to be estimated (the measured and simulated voltages at time 0 are significantly different). From the experiment editor, click **Select Initial States** to open a dialog to select model initial states; select the sdoRCCircuit.Cl.vc state and click **Ok** to add the state to the experiment.

Select mod	lel states			×
Filter by s	tate name			+ م
-		State		Current value
	sdoRCCircuit.C1.vc			0
			🖌 ок 🖇	Cancel 🕐 Help
				.:

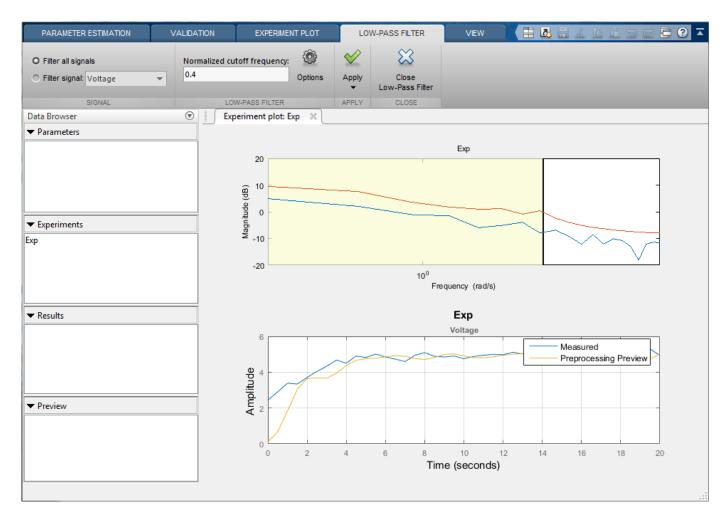
dit Experiment: Exp ×
Outputs
Specify measured output signals for this experiment.
/PS-Simulink Converter:1 (Voltage)
<1x1 Signal, 41 points> 🔹 🖶 🐇
Select Measured Output Signals
Initial States
Optionally define initial states for this experiment.
sdoRCCircuit/C1:sdoRCCircuit.C1.vc
▶ 0 - - - - - -
Select Initial States
Parameters
Optionally define parameters for this experiment.
There are currently no parameters defined for this experiment.
Select Parameters
Plot & Simulate 🔀 Plot ✔ OK 🔇 Help

Preprocess the Experiment Data

The measured data contains high frequency noise that you can remove using a low-pass filter. Click the **Experiment Plot** tab and select **Low Pass Filter**.

PARAMETER ESTIMATION		VALIDATION		EX	(PERIMENT F	PLOT	
Remove Offset	Low-Pass Filter Resample Data High-Pass Filter Replace Data Band-Pass Filter Replace Data		a P	Plot Model Response			
	DATA PROCE	SSING			PLOT		
Data Browser		\odot	Experim	nent plot	t: Exp 🚿		

This opens the Low-Pass Filter tool. The upper axis shows the signal FFT, the lower axis shows the signals. The original signal is shown in blue and the filtered signal in red. Adjust the filter bandwidth by either typing a value in the **Normalized cutoff frequency** edit field or clicking and dragging the yellow patch edge. Drag the filter cutoff to 0.4. Click **Options** and select **Zero-phase shift filter** to avoid introducing the filter phase shift into the measured data.



Click **Apply** and **Close Low-Pass Filter** to complete low-pass filtering of the data. The experiment is updated with the filtered signal. You can use other preprocessing tools such as remove offset, scale, and resample to further process the measured data. For this example low-pass filtering is sufficient.

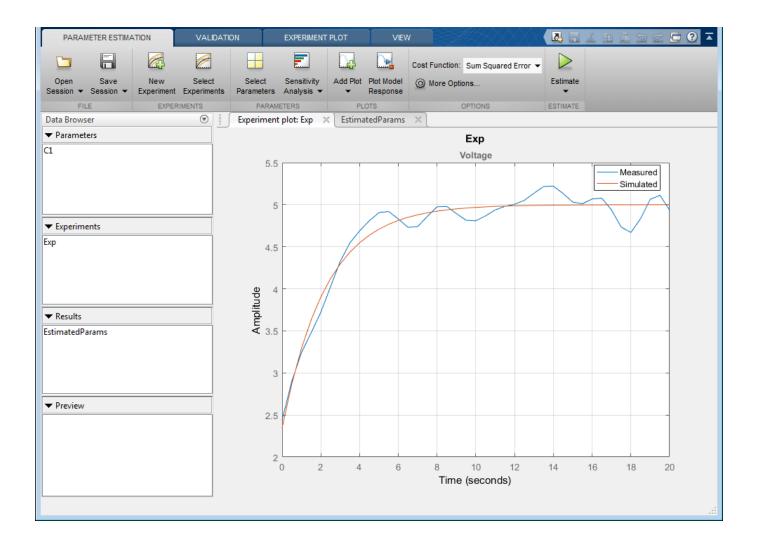
Estimate Model Parameter Values

With the experiment data configured and preprocessed you can now run an estimation. First select parameters to estimate. Click the **Parameter Estimation** tab and select **Select Parameters**. A dialog to specify model parameters for estimation opens. Click **Select Parameters** and select, C1, the circuit capacitor value. Set the capacitor minimum value to 0 and the initial guess to 460e-6.

it: Estimated Parameters						
Parameters Tuned for all Experiments						
<u>a</u>						
▼ 0.00046						
Minimum: 0						
Maximum: Inf 🔹 🕁						
Scale: 0.001953125 🔹 🖶						
Select parameters						
Parameters and Initial States Tuned per Experiment						
xperiment: Exp 💌						
elect experiment initial states for estimation.						
sdoRCCircuit/C1:sdoRCCircuit.C1.vc						
▶ 0 ► Estimate						
elect experiment parameters for estimation.						
There are no parameters defined for this experiment.						

Click **Estimate** to start the estimation. You can modify estimation options by setting the **Cost Function** combobox and clicking **More Options**.

While the estimation is running, the plots update and a dialog showing estimation progress appears. The progress dialog shows the estimation iterations, the number of times the model has been evaluated (F-count), and the estimation cost at each iteration.



Iteration	F-count	Exp			
		(Minimize)			
0	5	1.9855			
1	10	0.1005			
2	15	0.0380			
3	20	0.026			
4	25	0.025			
		4 4 2014 20:00:50			
optimizati	on started 2	4-Apr-2014 09:02:56			
Estimation	converged	. 24-Apr-2014 09:03:12			
stimated	experiment	values written to the workspace			

After a number of iterations the estimation converges and terminates. The model is updated with the estimated parameters and the estimation results are saved in the data browser.

Related Examples

To learn how to estimate model parameters using the sdo.optimize command, see "Estimate Model Parameters and Initial States (Code)" on page 2-67.

Close the model.

bdclose('sdoRCCircuit')

Estimate Model Parameter Values (GUI)

This example shows how to use experiment data to estimate model parameters. You estimate the parameters of an engine throttle system.

Engine Throttle Model

Engine Throttle System Model

Open the Simulink® model.

open_system('spe_engine_throttle')

1 U T Motor Image: Constraint of the UI to run an estimation. Image: Constraint of the UI to run an estimation.

Copyright (c) 2002-2014 The MathWorks, Inc.

Throttle Model Description

The throttle controls the air mass flow into the intake manifold of an engine. The throttle body contains a butterfly valve that opens when the driver presses down on the accelerator pedal. This lets more air enter the cylinders and causes the engine to produce more torque.

A DC motor controls the opening angle of the butterfly valve. There is also a spring attached to the valve to return it to its closed position when the DC motor is de-energized. The amount of rotation of the valve is limited to approximately 90 degrees. Therefore, if a large command input is applied to the motor, the valve hits the hard stops preventing it from rotating further.

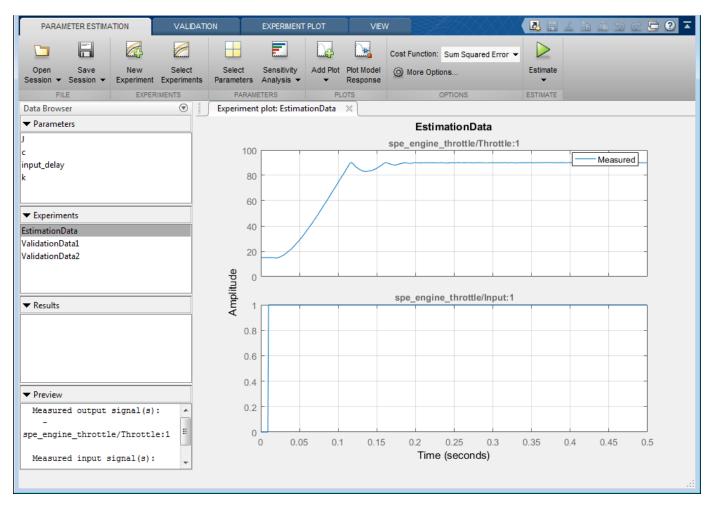
The motor is modeled as a torque gain and a time-delay input with parameters Kt and input_delay. The butterfly valve is modeled as a mass-spring-damper system with parameters J, c and k. This system is augmented with hard stops to limit the valve opening to 90 degrees. The model components are known, however, the parameter values of the system are not known accurately.

Estimation Experiment Data

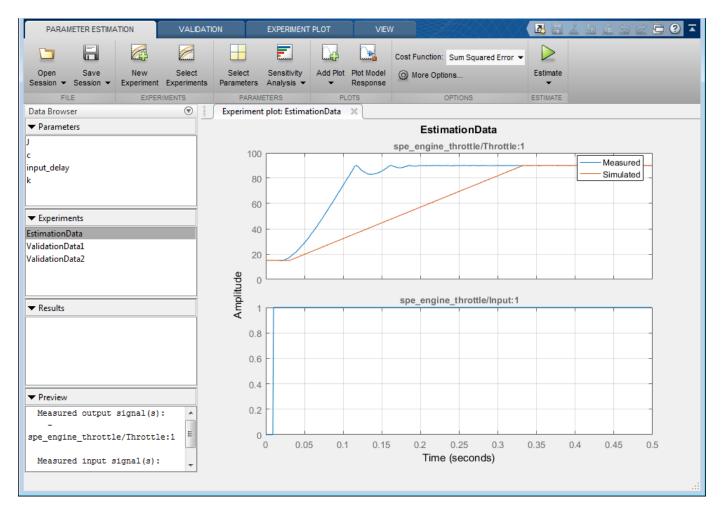
Double-click the Parameter Estimation with preloaded data block in the model to open a preconfigured estimation GUI session.

The saved estimation project defines three experiments; the EstimationData experiment is used for parameter estimation, while ValidationData1, ValidationData2 are used for validating the estimated parameters. The EstimateData experiment is plotted.

The signal data for the experiments can be imported from various sources including MATLAB® variables, MAT files, Excel® files, or comma-separated-value files. See "Importing and Preprocessing Experiment Data (GUI)" on page 2-132 for more information.



The experiment plot is also used to see how well the measured data matches the current model. Click **Plot Model Response** to display simulated signal data on the experiment plots.



The simulation results show that the model does not match the measured data and that model parameters need to be estimated.

Estimated Parameters

The next step is to define the parameter to estimate. Click **Select Parameters** to open a dialog to select model parameters to estimate. This example contains four unknown parameters; the butterfly valve inertia, J; the damping coefficient, C; the return spring constant, k; the time lag in motor response, input_delay.

Edit: Estimated Parameters	×					
Parameters Tuned for all Experiments						
1						
▶ 0.05 ♥	🗙 🖾 Estimate					
2						
🕨 40 👻 🚽	🗙 🖾 Estimate					
input delay						
▶ 0.02 ▼ 🕀	🗙 🖾 Estimate					
<u>k</u>						
▶ 1 · · · ·	🗶 🖾 Estimate					
Select parameters						
Parameters and Initial States Tuned per Experiment						
Experiment: EstimationData 💌						
Select experiment initial states for estimation.						
There are no initial states defined for this experiment.						
Select experiment parameters for estimation.						
There are no parameters defined for this experiment.						
Edit experiment						
📠 Update Model	쑫 OK (🧿 Help					

Since it is known from physical insight that all of these parameters have positive values, set their lower limits to zero. Also, put an upper bound of 0.1 sec on the input_delay parameter. You can also select an initial value for the parameters. These can come from some quick calculations of some formulas that determine the parameters.

Click the right arrow toggle button to modify the parameter minimum and maximum bounds.

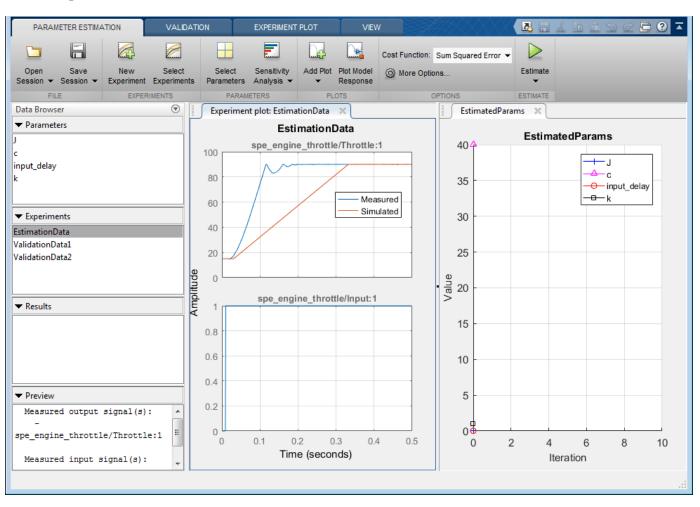
Edit: Estimated Parameters ×						
Parameters Tuned for all Experiments						
1						
▼ 0.05	•	1	🗙 🖾 Estimate			
Minimum:	0 -	· 🕂				
Maximum:	Inf	· 🖶				
Scale:	0.05	· 🖶				
<u>c</u>						
40	•	1	🗶 🖾 Estimate			
input delay		_				
• 0.02	-	- -	🗶 🗹 Estimate			
<u>k</u>		_				
▶ 1	-	- ₽	🗶 🖾 Estimate			
📙 Select param	neters					
Parameters an	d Initial States Tuned per Exp	eriment				
Experiment: EstimationData 💌						
Select experiment initial states for estimation.						
There are no initial states defined for this experiment.						
Select experiment parameters for estimation.						
There are no parameters defined for this experiment.						

The Estimation Task

After selecting the parameters for estimation, select the experiments to use for estimation. Click **Select Experiments** and select EstimationData for estimation.

Select Experim	ents		X
Select experim	nents to includ	le for estimation or validation	
Estimation	Validation	Experiment	
V		EstimationData	1
		ValidationData1	
		ValidationData2	
父 ОК		Help	
		-	.:

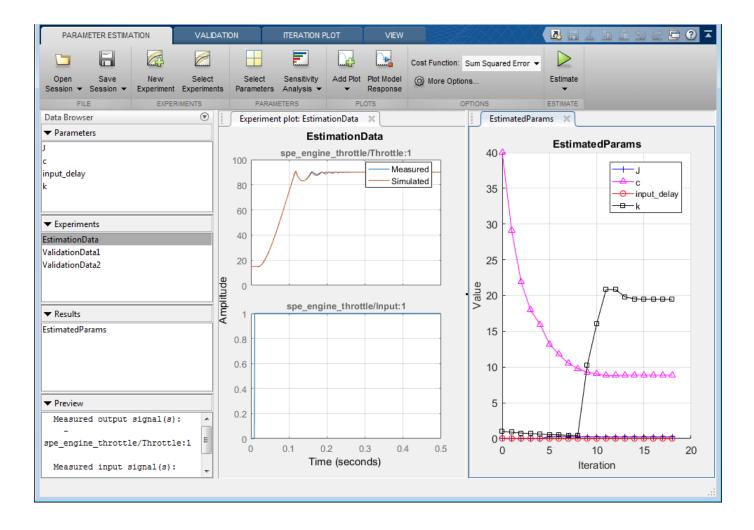
You can now start the estimation but first create plots to monitor the estimation progress. Click **Add Plot** and select **Parameter Trajectory**. This creates a plot that shows how the estimated parameter



values change during estimation. Click the **View** tab to layout the plots so that **Experiment plot:EstimationData** and **EstimatedParams** are both visible.

Click the **Estimate** button to start the estimation. You can modify estimation options by setting the **Cost Function** combobox and clicking **More Options**.

While the estimation is running, the plots update and a dialog showing estimation progress appears. The progress dialog shows the estimation iterations, the number of times the model has been evaluated (F-count), and the estimation cost at each iteration.



teration	F-count	EstimationData (Minimize)	
0	9		32.04
1	18		12.30
2	27		3.65
3	36		1.13
4	45		0.66
5	54		0.32
6	63		0.11
7	72		0.02
8	81		0.00
9	90		0.00
10	99		0.00
11	108		0.00 •
		III	4
		Apr-2014 16:50:09 21-Apr-2014 16:52:25	
stimated e	experiment v	alues written to the workspace	
		Save Iteration Display Option	ns Estimate

After a number of iterations the estimation converges and terminates. The model is updated with the estimated parameters and the estimation results are saved in the data browser. To see the details of the estimation result, right-click EstimatedParams and select **Open**.

View Result : EstimatedParams	×
Estimation result(s): J = 0.22774 c = 8.9489	*
input_delay = 0.0097738 k = 17.661	
Parameters estimated using experiments: - EstimationData, cost = 0.0065549	E
Solver output: Cost: 0.0065549	
ExitFlag: 3 FCount: 307 Date: 21-Apr-2014 17:03	
Solver termination message:	
Local minimum possible.	
Isqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the selected value of the function tolerance.	-
Use as initial guess 🔚 Update Model 😒	🖉 ок

Validation

It is important to validate the estimation results against other experiments. A successful estimation will not only match the experimental data that was used for estimation but also other independent measured data that were collected in experiments.

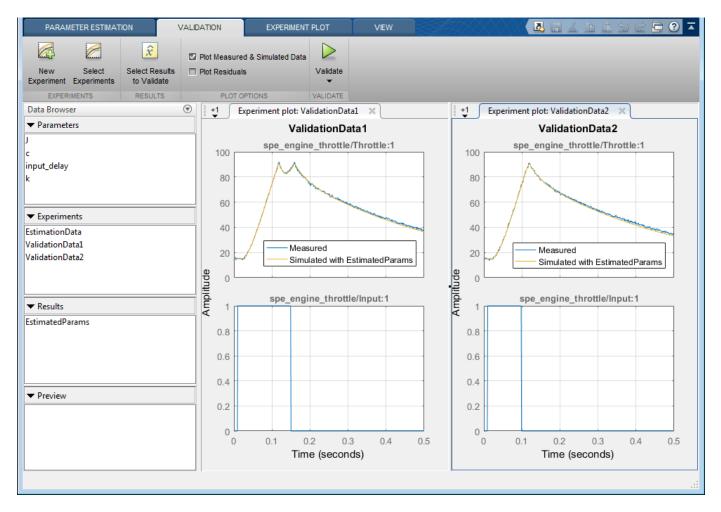
To select experiments for validation, click the **Validation** tab and click **Select Experiments**. Select both ValidationData1 and ValidationData2 for validation.

Select Experim Select experim		le for estimation or validation	>
Estimation	Validation	Experiment	
V		EstimationData	1
	v	ValidationData1	
	V	ValidationData2	<u>~</u>
🖌 ок		(?) Help	
VOR			

Click **Select Results** to select the estimation result(s) to use for validation. Select EstimatedParams and deselect Use current parameter values.

Select Re	sults	×
	Select results to include in validation	
	Results	
	Use current parameter values	
V	EstimatedParams	

Click **Validate** to validate the estimation result against the validation experiments. Validation simulates the model using the estimated parameters and selected experiments and creates plots showing the measured and simulation data. Use the **View** tab to layout the plots so that the **Experiment plot:ValidationData1** and **Experiment plot:ValidationData2** are both visible.



The validation plots confirm that the estimation was successful. The plots also show that the estimated parameters are robust enough to handle a variety of inputs.

Related Examples

To learn how to estimate model parameters using the sdo.optimize command, see "Estimate Model Parameter Values (Code)" on page 2-58.

Close the model.

```
bdclose('spe_engine_throttle')
```

Estimate Model Parameters Per Experiment (GUI)

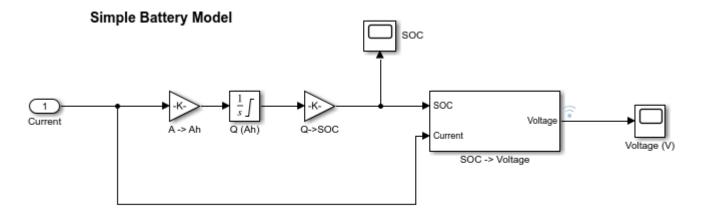
This example shows how to use multiple experiments to estimate a mix of model parameter values; some that are estimated using all the experiments and others that are estimated using individual experiments. The example also shows how to configure estimation experiments with experiment dependent parameter values.

You estimate the parameters of a rechargeable battery based on data collected in experiments that discharge and charge the battery.

Open the Model and Get Experimental Data

This example estimates parameters of a simple, rechargeable battery model, sdoBattery. The model input is the battery current and the model output, the battery terminal voltage, is computed from the battery state-of-charge.

```
open_system('sdoBattery')
```



Copyright 2012-2020 The MathWorks, Inc.

The model is based on the equation

$$E = (1 - \text{Loss})V - KQ_{\max}\frac{1 - s}{s}$$

In the equation:

E is the battery terminal voltage in Volts.

V is the battery constant voltage in Volts.

K is the battery polarization resistance in Ohms.

 Q_{max} is the maximum battery capacity in Ampere-hours.

s is the battery charge state, with 1 being fully charged and 0 discharged. The battery state-of-charge is computed from the integral of the battery current with a positive current indicating discharge and

a negative current indicating charging. The battery initial state-of-charge is specified by Q_0 in Ampere-hours.

Loss is the voltage drop when charging, expressed as a fraction of the battery constant voltage. When the battery is discharging this value is zero.

V, K, Qmax, Q0, and Loss are variables defined in the model workspace.

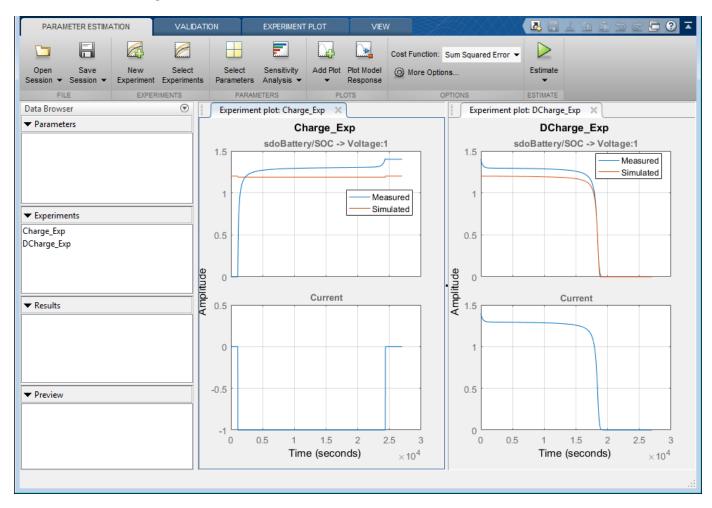
Estimation Experiment Data

A 1.2V (6500mAh) battery was subjected to a discharge and a charging experiment. This experiment data has been loaded into a preconfigured estimation session.

Use the following commands to load the pre-configured estimation session.

load sdoBattery_spesession
spetool(SD0SessionData)

The measured charge and discharge experiment data are loaded and plotted. Click the **View** tab to layout the plots so that the Experiment plot:Charge_Exp and Experiment plot:DCharge_Exp are both visible. Click **Plot Model Response** to see how well the model simulation matches the measured experiment data.



The plots show that the battery initial charge Q0 is not set correctly for the Charge_Exp experiment and that the model V, K, and Loss parameters need to be estimated.

Setting Experiment Parameter Values

The previous plot indicates that the Charge_Exp battery initial charge, Q0, is not set correctly. Add the initial charge to both experiments. Right click Charge_Exp and select Edit. A dialog to edit the experiment opens.

dit Experiment: Charge_Exp ×					
Outputs					
Define measured output signals for this experiment.					
<pre>sdoBattery/SOC -> Voltage:1 (sdoBattery/SOC -> Voltage:1)</pre>					
<1x1 Signal, 1351 points> 💌 🖶 🛓 🗶					
Select Meaured Output Signals					
Inputs					
Optionally define inputs signals for this experiment.					
sdoBattery/Current:1 (Current)					
<1x1 Signal, 1351 points> 🔹 🖶 🛃 🗶					
Select Inputs					
Initial States					
Optionally define initial states for this experiment.					
There are currently no initial states defined for this experiment.					
Select Initial States					
Parameters					
Optionally define parameters for this experiment.					
There are currently no parameters defined for this experiment.					
Select Parameters					
🙀 Plot & Simulate 🛛 Plot 🖌 OK 🕐 Help					

Click **Select Parameters** to open a dialog to add model parameters to the experiment. Select Loss and Q0 to add to the experiment. Select Loss as we need to estimate this parameter using only the Charge_Exp experiment. Click **Ok** to add the Q0 and Loss parameters to the experiment.

S	elect	model variat	oles		X
ſ	Filter	by variable i	name	L. L	ρ-
▼ Variable Current v Used By					
		К	0.001	sdoBattery/SOC -> Voltage	
	1	Loss	0.01	sdoBattery/SOC -> Voltage	
	V	Q0	6.5	sdoBattery/Q (Ah)	Ξ
		Qmax	6.5	<u>sdoBattery/Q (Ah)</u> <u>sdoBattery/Q->SOC</u> sdoBattery/SOC -> Voltage/Qmax*(1-s)//s	-
	▶ Sp	ecify express	ion indexing	if necessary (e.g., a(3) or s.x) 🖌 OK 🔀 Cancel ③	Help
					1.1

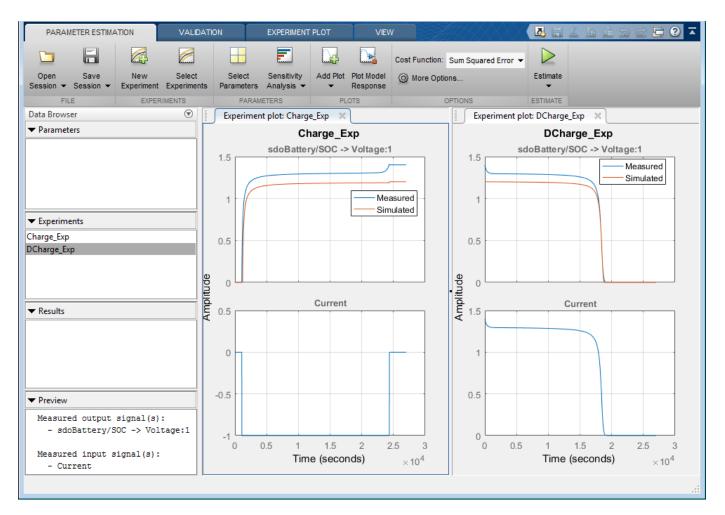
Set the battery initial charge Q0 in the $\mbox{Charge}_\mbox{Exp}$ to 0, i.e. there is no initial charge.

Edit Experiment: Charge_Exp ×					
Outputs Define measured output signals for this experiment. sdoBattery/SOC -> Voltage:1 (sdoBattery/SOC -> Voltage:1) <1x1 Signal, 1351 points> Image: Select Meaured Output Signals					
Inputs Optionally define inputs signals for this experiment. sdoBattery/Current:1 (Current) <1x1 Signal, 1351 points> Select Inputs					
Initial States Optionally define initial states for this experiment. There are currently no initial states defined for this experiment. Select Initial States					
Parameters Optionally define parameters for this experiment. Loss ▶ 0.01 ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ ● ♀ > ♀ > ♀ > ♀ > ♀ > ♀ ● ♀ ● ♀ ● ♀ > ♀ > ♀ > ♀ > ♀ >					
🙀 Plot & Simulate 🛛 Plot 🖌 OK 🕜 Help					

Similarly add the battery initial charge Q0 to the DCharge_Exp experiment and set the initial charge to 6.5., i.e. for this experiment there is an initial charge.

Edit Experiment: DCharge_Exp 🛛 🗙						
Outputs Define measured output signals for this experiment. sdoBattery/SOC -> Voltage:1 (sdoBattery/SOC -> Voltage:1) <1x1 Signal, 1351 points> Image: Select Meaured Output Signals						
Inputs Optionally define inputs signals for this experiment. sdoBattery/Current:1 (Current) (1x1 Signal, 1351 points> Select Inputs						
Initial States Optionally define initial states for this experiment. There are currently no initial states defined for this experiment.						
Parameters Optionally define parameters for this experiment. Q0 6.5 Select Parameters						
Plot & Simulate 😡 Plot 🖋 OK 🕐 Help						

Now that the experiments are updated with the correct initial battery charge click **Plot Model Response** to simulate the model and compare measured and simulated data.



The experiment plots show that the experiment initial conditions match but the battery response does not. The next step is to estimate the K and V model parameters.

Select Estimation Parameters

The previous plot showed that the model response does not match the measured data and we need to estimate the model V and K parameters.

Click **Select Parameters** to open a dialog to select model parameters.

Edit: Estimated Parameters ×					
Parameters Tuned for all Experiments					
There are no parameters selected for estimation.					
Select parameters					
Parameters and Initial States Tuned per Experiment					
Experiment: Charge_Exp 💌					
Select experiment initial states for estimation.					
There are no initial states defined for this experiment.					
Select experiment parameters for estimation.					
Loss					
▶ 0.01					
<u>Q0</u>					
0 Settimate					
Edit experiment					
🔀 Update Model 🛛 🔗 Help					

The upper portion of the select parameters dialog has a section for parameters that are tuned using all experiments. Click **Select Parameters** and add the V and K model parameters to the estimated parameters. Set the V minimum to 0 and the maximum to 2, similarly set the K minimum to 1e-6 and maximum to 0.1.

Edit: Estimated I	Parameters		×
	uned for all Experiment	s	
к		-	
▼ 0.001		▼ 🖶	🗙 🗹 Estimate
Minimum	1e-06		
Maximum	0.1	▼ 🖶	
Scale	0.001953125	-	
v	1	~	
▼ 1.2		▼ 🖶	🗙 🗹 Estimate
Minimum	0	- 🖶	
Maximum	2	- 🖶	
Scale	2		
🛃 Select para	meters		
-			
	nd Initial States Tuned p	er Experiment	
Experiment: Cl	harge_Exp 💌		
Select experime	ent initial states for estima	tion.	
There are no ir	nitial states defined for th	is experiment.	
Select experime	ent parameters for estimat	ion.	
Loss			
• 0.01		◄ 🖶	Estimate
<u>Q0</u>			_
• 0		▼ 🚽	Estimate
🧾 Edit experir	nent		
		logate Model	🖋 OK (🤉 Help

The lower section of the dialog has a section for initial states and parameters that are tuned using individual experiments.

For the Charge_Exp we tune the Loss parameter and set its minimum to 0 maximum to 0.5. The battery initial charge Q0 is fixed to 0 and should not be estimated; uncheck Estimate.

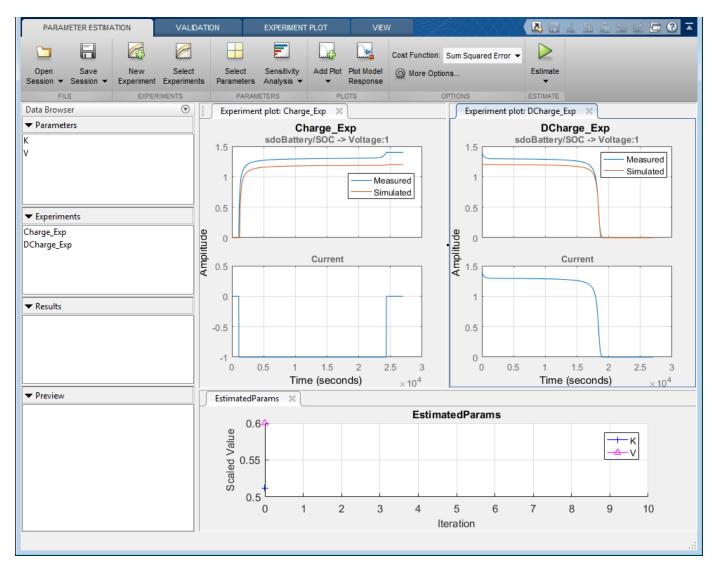
Edit: Estimated Parameters		×
Parameters Tuned for all Experime	ents	
<u>K</u>		
• 0.001	- 🕁	🗙 🖾 Estimate
<u>v</u>		
1.2	▼ 🖶	🗶 🗹 Estimate
Select parameters		
Parameters and Initial States Tune	ed per Experiment	
Experiment: Charge_Exp 💌		
Select experiment initial states for est	imation.	
There are no initial states defined fo	r this experiment.	
Select experiment parameters for esti	mation.	
Loss		
▼ 0.01	▼ 🖶	Estimate
Minimum: 0	▼ 🖶	
Maximum: 0.5	-	
Scale: 0.015625	◄ 🖶	
<u>Q0</u>		
	▼ 🖶	Estimate
Edit experiment		
	脑 Update Model	쑫 OK (🧿 Help

Select DCharge_Exp from the **Experiment** combobox to view the parameter settings for the DCharge_Exp experiment. The battery initial charge Q0 is fixed to 6.5 and should not be estimated; uncheck **Estimate**.

Edit: Estimated Parameters			×
Parameters Tuned for all Experim	ents		
ĸ			
• 0.001	- 🕁	X 🛛 🖬	Estimate
¥			
▶ 1.2	- 🖶	X 🛛 🖬	stimate
Belect parameters			
Parameters and Initial States Tune	ed per Experiment		
Experiment: DCharge_Exp 💌			
Select experiment initial states for estim	ation.		
There are no initial states defined for the	nis experiment.		
Select experiment parameters for estim	ation.		
<u>Q0</u>			
6.5	- 🕁	🔲 Est	imate
Edit experiment			
	🔈 Update Model	🖋 ок	(?) Help

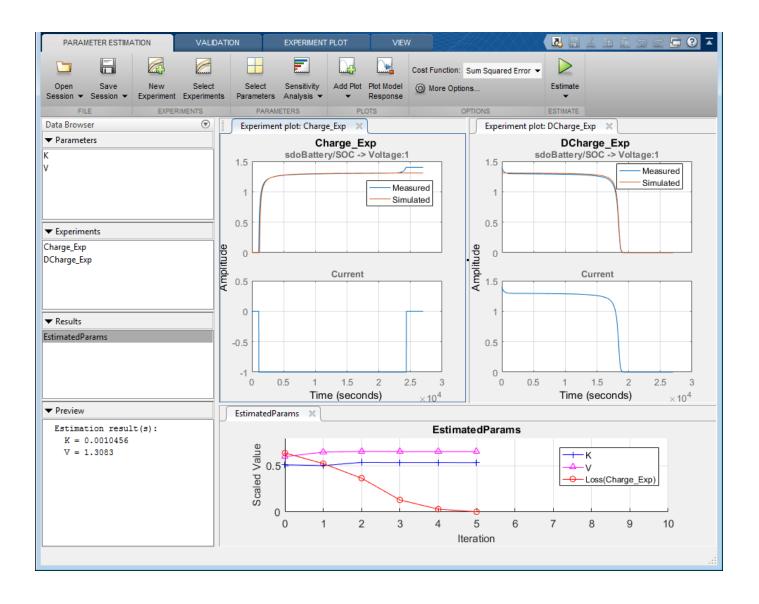
Estimate Parameter Values

The experiments and estimated parameters are configured and we a ready to run the estimation. First create a plot to monitor the estimation progress. Click **Add Plot** and select **Parameter Trajectory**. This creates a plot that shows how the estimated parameter values change during estimation. Click the **View** tab to layout the plots so that the experiment and parameter trajectory plots are all visible.



Click the **Estimate** button to start the estimation. You can modify estimation options by setting the **Cost Function** combobox and clicking **More Options**.

While the estimation is running the plots update and a dialog showing estimation progress appears. The progress dialog shows the estimation iterations, the number of times the model has been evaluated (**F-count**), and the estimation cost at each iteration.



Iteration	F-count	Charge_Exp (Minimize)	DCharge_Exp (Minimize)
0	7	12.6204	4.1010
1	14	2.2115	1.3692
2	21	1.9613	0.2716
3	28	1.9663	0.2372
4	35	1.9765	0.2169
5	42	1.9756	0.2164
Optimizati	on started 23	3-Apr-2014 15:10:19	
Estimation	converged,	23-Apr-2014 15:11:15	
stimated	experiment	values written to the workspace	-

After a number of iterations the estimation converges and terminates. The experiment plots show the measured and simulation data matching well. The EstimatedParams plot shows the V, K, and Loss parameters changing during the estimation; the scale of V, K, and Loss are different, right click on the plot and select **Show scaled values** to see how all the parameters changed from their original values.

Related Examples

To learn how to estimate parameters per experiment using the sdo.optimize command, see "Estimate Model Parameters Per Experiment (Code)" on page 2-86.

Close the model

bdclose('sdoBattery')

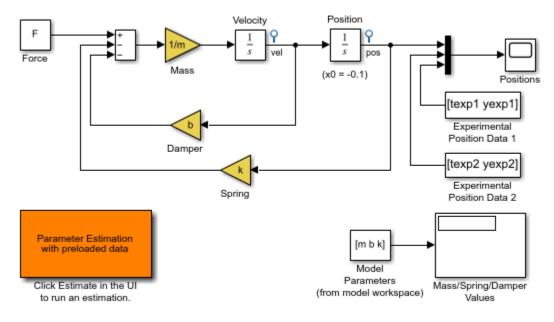
Estimate Model Parameters and Initial States (GUI)

This example shows how to estimate the physical parameters - mass (m), spring constant (k) and damping (b) of a simple mass-spring-damper model. This example illustrates the significance of initial state estimation.

Mass Spring Damper System Model

Open the Simulink® model.

open_system('msd_system')



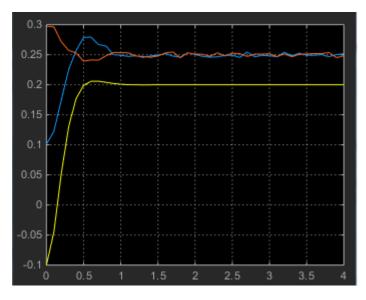
Mass-Spring-Damper System

Copyright 2002-2014 The MathWorks, Inc.

The model's output is the displacement response (position) of the mass in a mass-spring-damper system, subject to a constant force (F), and an initial displacement ($\times 0$). $\times 0$ is the initial condition of the Position integrator block. Run the simulation once to observe the response of the model to a nominal set of parameter values.

Experimental Data Sets

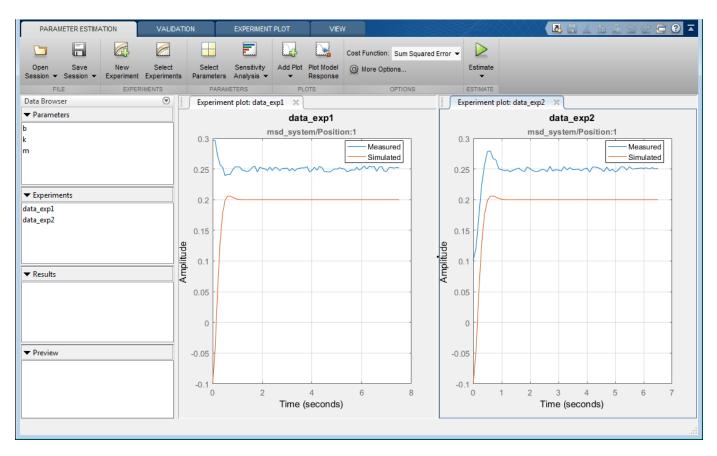
For estimation of the model parameters (m, b, and k), two sets of experimental data are used. These data sets were obtained using two different initial positions (0.1 and 0.3), and contain additive noise. Following is the plot of these data sets (orange and cyan curves), along with the simulated response (yellow curve) of the Simulink model for x0=-0.1 and a nominal set of parameter values (m=8, k=500, b=100).



Estimation of Model Parameters

The model has three parameters (k, b, m) that appear in the Gain blocks of the Simulink model msd_system. We estimate these parameters using Parameter Estimation.

Double-click the Parameter Estimation GUI with preloaded data block in the model to open a preconfigured estimation GUI session. The experimental data sets are already loaded in the project (data_exp1 and data_exp2). Click the View tab to lay out the plots so that the Experiment plot:data_exp1 and Experiment plot:data_exp2 are both visible. Click Plot Model Response to simulate the model for the two experiments. The plots show that the model simulation does not match the experiment data.



Parameter Estimation with No State Estimation

The app is configured to estimate the model parameters using both data_exp1 and data_exp2 experiments; click **Select Parameters** to see the selected parameters and **Select Experiments** to see the experiments selected for estimation.

Click **Estimate** to start the estimation. You can modify estimation options by setting the **Cost Function** combobox and clicking **More Options**.

While the estimation is running, the plots update and a dialog showing estimation progress appears. The progress dialog shows the estimation iterations, the number of times the model has been evaluated (F-count), and the estimation cost at each iteration.

Image: Section Service Open Save Rew Select Select Sensitivity Add Plot Plot Model Parameters Analysis Parameters Parameters Image: Select Sensitivity Parameters Analysis Parameters Parameters Image: Select Sensitivity Parameters Parameters Image: Select Sensitivity Parameters Parameters Parameters Image: Select Sensitivity Parameters Parameters Image: Select Sensitivity Parameters Parameters Image: Select Sensitivity Parameters Image: Select Sensitivity Parameters Image: Select Sensitivity Parameters Image: Select Sensitivity Image: S	PARAMETER ESTIMATION	VALIDATION	EXPERIMENT PLOT	VIEW		1 / h i b c i 0 7
Session * Experiments Experiments PARAMETERS PLOTS OPTIONS ESTIMATE Plate EXPERIMENTS PARAMETERS PLOTS OPTIONS ESTIMATE Parameters Image: System/Position:1 Image: System/Position:1 Image: System/Position:1 Image: System/Position:1 Image: System/Position:1 * Experiments 0.25 0.2 Image: System/Position:1 Image: System/Position:1 * Experiments 0.25 0.15 Image: System/Position:1 Image: System/Position:1 * Experiments 0.15 0.15 Image: System/Position:1 Image: System/Position:1 * Results Image: System/Position:1 Image: System/Position:1 Image: System/Position:1 Image: System/Position:1 * Preview 0.15 Image: System/Position:1 Image				Cost Function:	Sum Squared Error 👻 >	
Data Browser ●					///	
Parameters data_exp1 mm	FILE EXPE		RAMETERS	LOTS C	OPTIONS ESTIMATE	
• review • outdate_exp2 • review • outdate_exp2 • review • outdate_exp2 • review • outdate_exp2 • review • outdate_exp1 • review • outdate_exp2 • review • outdate_exp2 • review • outdate_exp2 • review • outdate_exp2	Data Browser	Experiment	nent plot: data_exp1 🛛 🛛		Experiment plot: data_e	xp2 🗙 EstimatedParams 🗙
wm 0.3 imsd_system/Position:1 v Experiments 0.2 0.2 data_exp1 0.1 data_exp2 0.15 0.1 0.1 v Results EstimatedParams 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.01 0.02 0.01 0.02 0.01 0.02 0.03 0.04 0.05	▼ Parameters		data_ex	p1	d	ata_exp2
m ✓ Experiments data_exp2			msd_system/P	osition:1	msd_sy	/stem/Position:1
✓ Experiments Jata_expl Jata_exp2 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.16 0.17 0.18 0.19 0.10 • Results 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.1 0.2 0.1 0.2 0.1 0.2 0.1 0.2			~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~			
• 0.15 • 0.15 • 0.15 • Results • 0.05 • 0.05 • Preview • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05 • 0.05	data_exp1					
✓ Results EstimatedParams 0.05 0 0.05 0 0.05 0 0.05 0 0.05 0 0.05 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	lata_expz	3				
EstimatedParams	▼ Results	0.1			0.1	
Preview -0.05 -0.1 0 2 4 6 8 -0.1 0 1 2 3 4 5 6 7		0.05			0.05	
		0			0	
0 2 4 6 8 0 1 2 3 4 5 6 7	▼ Preview	-0.05			-0.05	
Time (seconds) Time (seconds)			_		0 1 2	
			nine (seco	inusj		

eration	F-count	data_exp1 (Minimize)	data_exp2 (Minimize)
0	7	4.8737	2.49
1	14	4.8737	2.49
2	21	3.7331	1.18
3	28	3.2630	0.87
4	35	3.2630	0.87
5	42	3.0230	0.74
6	49	2.8380	0.70
7	56	2.7386	0.66
8	63	2.3614	0.64
9	70	2.3129	0.56
10	77	2.0777	0.59
11	84	1.9252	0.67 -
		III	4
stimation	converged	2-Apr-2014 13:19:47 , 22-Apr-2014 13:21:29 values written to the workspace	

After a number of iterations the estimation converges and terminates. The model is updated with the estimated parameters and the estimation results are saved in the data browser.

The data_exp1 and data_exp2 experiment plots show that the model parameters have been tuned to match the measured experiment data as closely as possible. The simulated measured signals match well from the 2 second mark onward but don't match well before 2 seconds. The simulation results for both experiments start at -0.1. This is the initial condition of the model which was not estimated; these plots show that the initial condition should also be estimated.

Parameter Estimation with Initial State Estimation

The data_expl and data_exp2 experiments specify the measured output data but as seen above must also specify the model initial state. Add the initial states to the experiments and estimate them.

Right-click data_exp1 and select Edit to open a dialog to configure the experiment.

Edit Experiment: data_exp1 ×
Outputs Define measured output signals for this experiment.
msd_system/Position:1 (msd_system/Position:1)
<1x1 Signal, 76 points>
Initial States Optionally define initial states for this experiment. There are currently no initial states defined for this experiment. Select Initial States
Parameters Optionally define parameters for this experiment.
There are currently no parameters defined for this experiment. Relect Parameters
Plot & Simulate 🛛 Plot 🖌 OK 📀 Help

Click **Select Initial States** and select the position state. Click OK to close the state selector and add the selected state to the experiment.

Select mod	lel states				×
Filter by s	tate name				• م
-		State		Curre	nt value
	msd_system/Position				0.1
	msd_system/Velocity				0
			🖋 ок	🔀 Cancel	(?) Help

Edit Experiment: data_exp1 ×
Outputs
Define measured output signals for this experiment.
msd_system/Position:1 (msd_system/Position:1)
<1x1 Signal, 76 points> 💌 🖶 🚣 🗶
Select Meaured Output Signals
Initial States
Optionally define initial states for this experiment.
msd_system/Position
▶ -0.1 ▼ 🖶 🗙
Select Initial States
Parameters
Optionally define parameters for this experiment.
There are currently no parameters defined for this experiment.
🛃 Select Parameters
. —
🕞 Plot & Simulate 🛛 Plot 🖌 OK 🕜 Help

Right-click data_exp2 and select Edit and add the position state to the experiment.

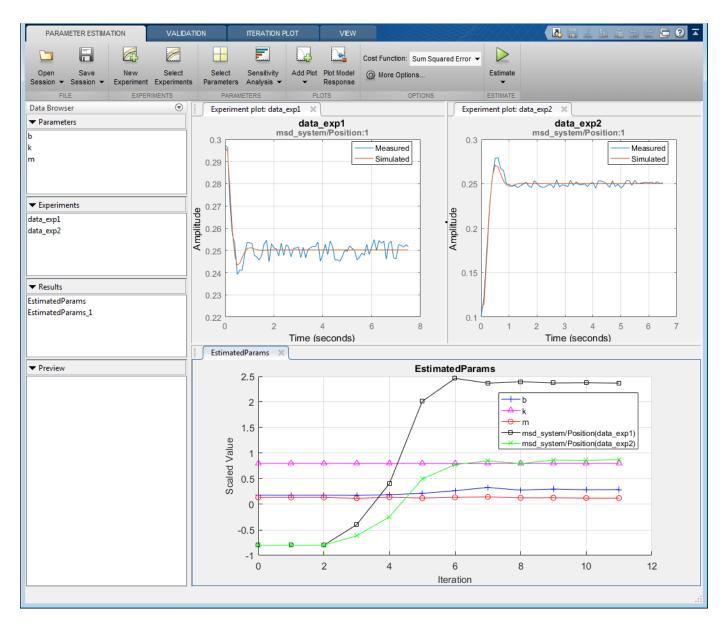
The experiments are now configured to include initial states that can be estimated. Click **Select Parameters**.

Edit: Estimated Parameters		×
Parameters Tuned for all Experiments		
<u>b</u>		
100	- 🖶	🗙 🖾 Estimate
<u>k</u>		
500	- 🗗	🗶 🖾 Estimate
<u>m</u>		_
▶ 8	- 🕁	🗙 🗹 Estimate
Select parameters		
Parameters and Initial States Tuned per Ex	periment	
Experiment: data_exp1 💌		
Select experiment initial states for estimation.		
msd_system/Position		
-0.1	-	Estimate
Select experiment parameters for estimation.		
There are no parameters defined for this expe	riment.	
Edit experiment		
📠 Update	e Model	🖋 ОК 🥘 Неір

The upper portion of the select parameters dialog has a section for parameters that are tuned using all experiments selected for estimation. The lower section of the dialog has a combo-box to select an experiment and widgets to specify initial states and parameters that are tuned using only the selected experiment. For this problem, the data_exp1 and data_exp2 experiments estimate the model initial state for each experiment.

You can now start the estimation, but first create plots to monitor the estimation progress. Click Add Plot and select **Parameter Trajectory**, right click the plot and select **Show scaled values**. This creates a plot that shows how the estimated parameter values change during estimation. Click the **View** tab to lay out the plots so that the **Experiment plot:data_exp1**, **Experiment plot:data_exp2**, and **Iteration plot 1** are both visible.

Click the **Estimate** button to start the estimation.



After a number of iterations the estimation converges and terminates. The data_exp1 and data_exp2 experiment plots show how estimating the initial value improves the estimation fit. The EstimatedParams plot shows the estimated initial state for the two experiments, the plot also shows that the estimated k value did not change while b and m changed slightly. You can confirm this by clicking EstimatedParams and examining the preview pane and then clicking EstimatedParams1 and examining the preview pane. Alternatively right click EstimatedParams and select **Open** to open a dialog to view the results.

▼ Results	▼ Results
EstimatedParams	EstimatedParams
EstimatedParams1	EstimatedParams1
▼ Preview	▼ Preview
Estimation result(s):	Estimation result(s):
b = 17.559	b = 28.593
k = 400.07	k = 399.79
m = 1.0676	m = 0.97266

This example shows that it is important to independently estimate initial states for each experiment in order to obtain the correct estimates of the model parameters.

Related Examples

To learn how to estimate model parameters and initial states using the sdo.optimize command, see "Estimate Model Parameters and Initial States (Code)" on page 2-67.

Close the model

bdclose('msd_system')

Generate MATLAB Code for Parameter Estimation Problems (GUI)

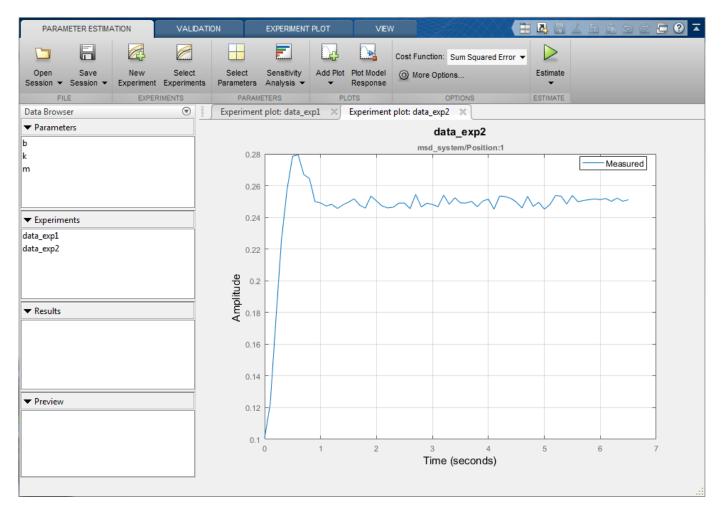
This example shows how to automatically generate a MATLAB® function to solve a parameter estimation problem. You use the **Parameter Estimator** to define an estimation problem for a mass-spring-damper and generate MATLAB code to solve this estimation problem.

Mass-Spring-Damper Estimation Problem

The "Estimate Model Parameters and Initial States (GUI)" on page 2-168 example shows how to use the **Parameter Estimator** to estimate parameters of a mass-spring-damper model. In this example, you load a pre-configured **Parameter Estimator** session based on that example.

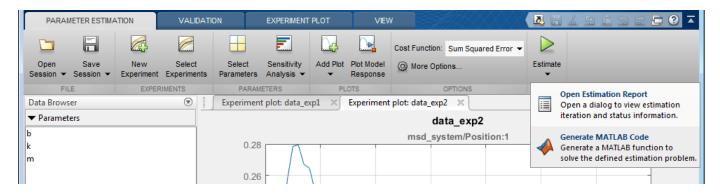
Use the following commands to load the pre-configured estimation session.

load sdoMassSpringDamper_sdosession
spetool(SD0SessionData)

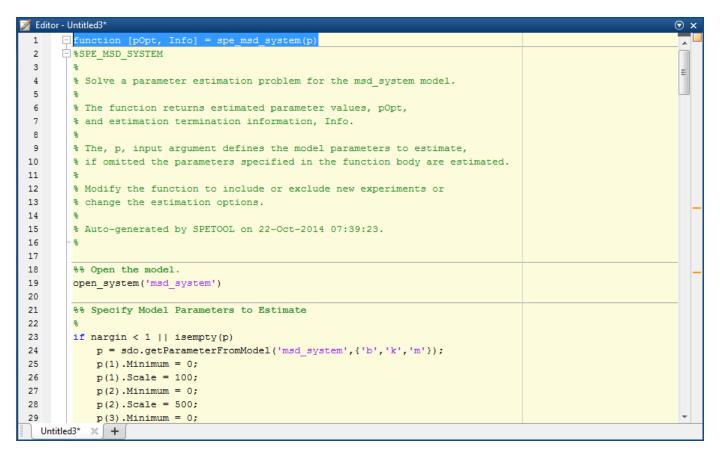


Generate MATLAB Code

From the Estimate list, select Generate MATLAB Code.



The generated code is added to the MATLAB editor as an unsaved MATLAB function.



Examine the generated code. Significant code portions are:

- Specify Model Parameters to Estimate Definition of the model parameters being estimated.
- **Define the Estimation Experiments** Definition of the measured and expected signal data to use for estimation.
- **Create Estimation Objective Function** Creation of an anonymous function that calls the subfunction msd_system_optFcn, which evaluates the model using each experiment and compares simulation and measured experiment outputs. This anonymous function is called by sdo.optimize at each iteration of the optimization problem to solve the estimation problem.

• Estimate the Parameters - Solve the estimation problem using the sdo.optimize command.

Select Save from the MATLAB editor to save the generated function.

Run Generated Code

Run the generated function.

Command \	Window					C
New to MA	TLAB? See re	sources for <u>Getting St</u>	arted.			>
>> [pH	St,estIn	fo] = spe_msd_	system;			4
Optin	nization :	started 22-Oct	-2014 07:42:3	6		
			Step-size	First-order		
Iter	F-count	f(x)		optimality		
0	11	9.18411	1			
1	22	0.0502487	3.697	0.0807		=
2	33	0.0140152	0.3313	0.0262		
3	44	0.0132182	0.06352	0.00053		
Local	minimum	found.				
Optimi	ization c	ompleted becau	se the size o	f the gradient is les	ss than	
the se	elected v	alue of the fu	nction tolera	nce.		
$f_{\star} >>$						•

The first output argument, pOpt, contains the optimized parameter values and the second output argument, optInfo, contains optimization information.

Modify the Generated Code

You can:

- Modify the generated spe_msd_system function to include or exclude new experiments or change estimation options.
- Call the generated spe_msd_system function with a different set of parameters to estimate.

For details on how to write an objective/constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB command prompt.

Close the model.

Improving Optimization Performance Using Fast Restart (GUI)

This example shows how to use the **Fast Restart** feature of Simulink® to speed up optimization of a model. You use fast restart to estimate the parameters of an engine throttle model.

How Fast Restart Speeds up the Optimization

Simulation of Simulink models requires that the model be compiled before it is simulated. In this context compilation of a model means analyzing and formatting the model so that it can be simulated. The idea of fast restart is to perform the model compilation once and reuse the compiled information for subsequent simulations. For more information about when to use fast restart, see "How Fast Restart Improves Iterative Simulations".

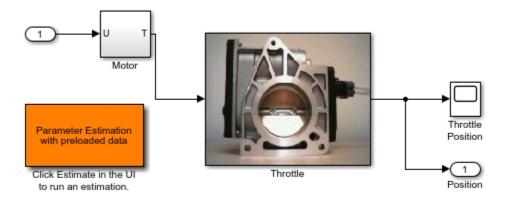
During optimization the model is repeatedly simulated (often tens or hundreds of times) Fast Restart means that the model is only compiled once for these simulation in comparison to non-fast restart where the model is recompiled each time.

Models where compilation is a significant portion of overall simulation time benefit the most from Fast Restart. Further once a model is compiled not all model parameters can be changed, specifically only tunable parameters can be changed. For more information, see "Get Started with Fast Restart".

Open Model and Parameter Estimator

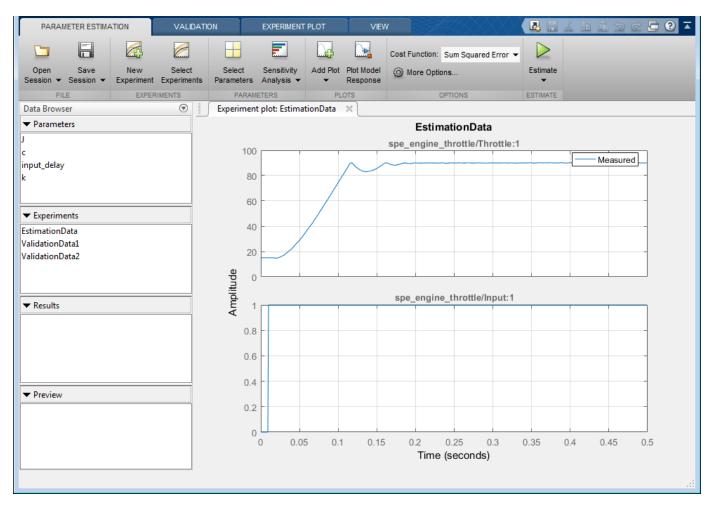
Load the model and click the "Parameter Estimation with preloaded data" block to load a preconfigured parameter estimation problem. The goal is to tune the parameters of an engine throttle model to match measured data. For details on the problem setup see the "Estimate Model Parameter Values (GUI)" on page 2-143 example.

open_system('spe_engine_throttle')



Engine Throttle Model

Copyright (c) 2002-2014 The MathWorks, Inc.



Estimate Without Using Fast Restart

To compare the estimation with and without fast restart, change the estimation options in the app to not update the model with estimated values.

Click **More Options** in the **Parameter Estimator** and click General Options. Clear **Update model at end of estimation**, and select **Save estimated values as new estimation result**.

Estimation Options	;			×
General Options	Optimization Options	Parallel Options		
Progress Option	5			h
Show estima	tion progress window d	uring estimation		
🔽 Create a para	ameter trajectory plot du	iring estimation		
🔽 Update plots	during estimation			
Result Options				h
🔲 Update mod	el at end of estimation			
Estimation result	S:			
Overwrite	e estimation result with r	new estimated val	lues	
Save estin	mated values as new esti	mation result		
	0	K Cancel	Help	
				:

Click **Estimate** to estimate the model parameter values. The estimation progress report shows the estimation start and end time.

🗼 Estimati	on Progre	ss Report
Iteration	F-count	EstimationData (Minimize)
0	9	32.04 🔺
1	18	12.30
2	27	3.65
3	36	1.15
4	45	0.6E
5	54	0.32
6	63	0.11
7	72	0.02
8	81	0.00
9	90	0.00
10	99	0.00
4	400	
Estimation	converted	7-Apr-2015 13:56:33 , 17-Apr-2015 13:57:43 values written to 'EstimatedParams' Save Iteration Display Options Estimate

Estimate Using Fast Restart

To configure the model to use Fast Restart during simulation, click **Enable Fast Restart** in the Simulink model.



Click **Estimate** in the **Parameter Estimator**. The estimation progress report shows the estimation start and end time. Note the reduction in total estimation time compared to the estimation without using fast restart, in this case around 28 seconds or 45% of the original estimation time.

承 Esti	matio	on Progre	ss Report
Iterat	ion	F-count	EstimationData
			(Minimize)
	0	9	32.04 🔺
	1	18	12.30
	2	27	3.65 =
	3	36	1.13
	4	45	0.66
	5	54	0.32
	6	63	0.11
	7	72	0.02
	8	81	0.00
	9	90	0.00
	10	99	0.00
		400	
			III
Estima	ation	conversed	7-Apr-2015 14:40:58 , 17-Apr-2015 14:41:40 values written to 'EstimatedParams_1'
			Save Iteration Display Options Estimate

Related Examples

The Generate MATLAB Code feature of the **Parameter Estimator** and **Response Optimizer** will generate the MATLAB® code to configure the model for fast restart if the app is configured to use fast restart.

To learn how to use Fast Restart at the command line see "Improving Optimization Performance Using Fast Restart (Code)" on page 2-187.

Close the model.

bdclose('spe_engine_throttle')

See Also

Related Examples

- "Improving Optimization Performance Using Fast Restart (Code)" on page 2-187
- "Use Fast Restart Mode During Response Optimization" on page 3-189
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109

More About

"Ways to Speed Up Design Optimization Tasks"

Improving Optimization Performance Using Fast Restart (Code)

This example shows how to use the **Fast Restart** feature of Simulink® to speed up optimization of a model. You use fast restart to estimate the parameters of an engine throttle model.

How Fast Restart Speeds up the Optimization

Simulation of Simulink models requires that the model be compiled before it is simulated. In this context compilation of a model means analyzing and formatting the model so that it can be simulated. The idea of fast restart is to perform the model compilation once and reuse the compiled information for subsequent simulations. For more information about when to use fast restart, see "How Fast Restart Improves Iterative Simulations".

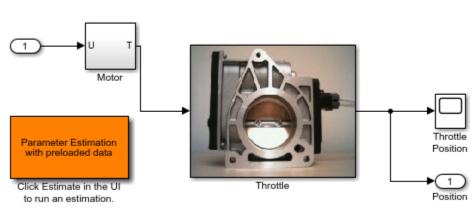
During optimization the model is repeatedly simulated (often tens or hundreds of times) Fast Restart means that the model is only compiled once for these simulation in comparison to non-fast restart where the model is recompiled each time.

Models where compilation is a significant portion of overall simulation time benefit the most from fast restart. Further once a model is compiled not all model parameters can be changed, specifically only tunable parameters can be changed. For more information, see "Get Started with Fast Restart".

Open Model

Load the engine throttle model. The goal is to tune the parameters of the model to match measured data. For details on the problem setup see the "Estimate Model Parameter Values (GUI)" on page 2-143 example.

open_system('spe_engine_throttle')



Engine Throttle Model

Copyright (c) 2002-2014 The MathWorks, Inc.

Define the Estimation Problem Data

This examples focuses on the command line interface for using Fast Restart during estimation. For a detailed description of the estimation command line interface see "Estimate Model Parameter Values (Code)" on page 2-58.

Specify the model parameter values to estimate and any parameter bounds.

```
p = sdo.getParameterFromModel('spe_engine_throttle',{'J','c','input_delay','k'});
p(1).Minimum = 0;
p(2).Minimum = 0;
p(3).Minimum = 0;
p(3).Maximum = 0.1;
p(4).Minimum = 0;
```

Define the estimation experiment. The measured experiment data is loaded from the spe_engine_throttle_ExperimentData MATLAB file. The MATLAB file contains a
Input_SignalData and Output_SignalData variable specifying the experiment input and output
signal data.

```
load spe_engine_throttle_ExperimentData
```

```
Exp = sdo.Experiment('spe engine throttle');
Input = Simulink.SimulationData.Signal;
               = Input_SignalData;
Input.Values
Input.BlockPath = 'spe engine throttle/Input';
Input.PortType = 'inport';
Input.PortIndex = 1;
               = 'spe engine_throttle/Input:1';
Input.Name
Exp.InputData = Input;
Output = Simulink.SimulationData.Signal;
Output.Values = Output_SignalData;
Output.BlockPath = 'spe engine throttle/Throttle';
Output.PortType = 'outport';
Output.PortIndex = 1;
             = 'spe_engine_throttle/Throttle:1';
Output.Name
Exp.OutputData = Output;
```

Create a model simulator from the experiment

Simulator = createSimulator(Exp);

Configure the Simulator for Fast Restart

The simulator controls whether the model is simulated using fast restart or not. The fastRestart command is used to configure the simulator to use Fast Restart.

The spe_engine_throttle model uses a variable-step solver, and may not output values at the times in the measured experiment data. To output values at the times of the measured data, use the set_param command to specify the model logging output times as a workspace variable. In the estimation objective function, the variable is then used to specify output times to be the same as the measured experiment data. The model OutputTimes is set before configuring the simulator for fast restart, as once the model is configured for fast restart, the model logging configuration can not change.

```
set_param('spe_engine_throttle','OutputOption','SpecifiedOutputTimes','OutputTimes','OutputTimes'
Simulator = fastRestart(Simulator,'on');
```

The simulator can now be used during estimation, and the model will be simulated using fast restart.

Run the Estimation

Create an estimation objective function to evaluate how closely the simulation output, generated using the estimated parameter values, matches the measured data. Use an anonymous function with

one argument that calls the spe_engine_throttle_Objective function. The spe_engine_throttle_Objective function includes the Simulator argument that has been configured to use fast restart.

optimfcn = @(P) spe_engine_throttle_Objective(P,Simulator,Exp);

Set the optimization options, and run the optimization.

```
Options = sdo.OptimizeOptions;
Options.Method = 'lsqnonlin';
[pOpt,Info] = sdo.optimize(optimfcn,p,Options);
```

Optimization started 02-Sep-2021 00:39:24

			Firs	t-order
Iter F-co	ount f	(x) Step	-size opti	mality
Θ	9 32.	048	1	
1	18 12	.24 0.6	495	18
2	27 3.5	944 0.3	919	8.65
3	36 1.11	.908 0.1	881	3.11
4	45 0.648	845 0.	197	1.25
5	54 0.288	719 1.	218	1.15
6	63 0.147	572 0.2	943	0.42
7	72 0.0874	598 0.5	345	0.102
8	81 0.0668	259 0.4	315	0.169
9	90 0.0668	259 9.	907	0.169
10	99 0.0662	283 2	.33	0.27
11 1	LO8 0.0491	.157 0.	209 0	.0989
12 1	117 0.0491	.157 0.2	997 0	.0989
13 1	126 0.0490	206 0.07	492 0	.0663
local mini	imum nocciblo			

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

Restore the Model

Restore the simulator fast restart settings. This clears the logging and other settings used for the optimization problem.

Simulator = fastRestart(Simulator, 'off');
set_param('spe_engine_throttle', 'OutputOption', 'RefineOutputTimes', 'OutputTimes', '[]');

Related Examples

You can also generate code to configure you model for fast restart in the **Parameter Estimator** and the **Response Optimizer**. Configure the model for fast restart as described in "Improving Optimization Performance Using Fast Restart (GUI)" on page 2-181. Then use the Generate MATLAB Code feature of the app.

Close the model.

bdclose('spe_engine_throttle')

See Also

sdo.SimulationTest | fastRestart

Related Examples

- "Improving Optimization Performance Using Fast Restart (GUI)" on page 2-181
- "Use Fast Restart Mode During Response Optimization" on page 3-189
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109

More About

• "Ways to Speed Up Design Optimization Tasks"

Deployed Application of Parameter Estimation

This example shows how to monitor the condition of an electric vehicle battery in the field, with a deployed version of parameter estimation in Simulink® Design Optimization^m, together with Simulink Compiler^m.

Battery Monitoring

Batteries in electric vehicles are expensive to replace and need to be monitored and maintained carefully, to ensure they function well for their intended lifetime. In this example, an electric car is driven to work and back on a daily commute. At home, the car is plugged in to a smart charger that monitors both the current and the battery voltage. The charger analyzes the battery data to estimate the battery parameters, using a deployed version of parameter estimation in Simulink Design Optimization, together with Simulink Compiler. The charger relays these parameters to the car manufacturer through an Internet of Things (IoT) connection, so that the manufacturer can monitor the battery health over time.

Battery Model

This example estimates parameters of a simple, rechargeable battery model, sdoBattery. The input to sdoBattery is the battery current, and the model output is the battery terminal voltage which is computed from the battery state of charge.

The battery model is based on the equation:

$$E = (1 - \text{Loss})V - KQ_{\max}\frac{1 - s}{s}$$

where:

- *E* is the battery terminal voltage in Volts.
- V is the battery constant voltage in Volts.
- *K* is the battery polarization resistance in Ohms.
- Q_{max} is the maximum battery capacity in Ampere-hours. Q_0 is the battery initial state of charge in Ampere-hours.
- *s* is the battery charge state, with 1 being fully charged and 0 discharged. The battery state of charge is computed from the integral of the battery current with a positive current indicating discharge and a negative current indicating charging.
- Loss is the voltage drop when charging, expressed as a fraction of the battery constant voltage.

Use the following command to open the model.

open_system('sdoBattery')

Battery Characteristics

The following battery characteristics are known:

- Voltage, V = 400V
- Loss factor, Loss = 0.012
- Resistance, K = 0.32 Ohms.

 Q_{max} is known to be 250 Ampere-hours (100 kWh) when the battery is new. As the battery ages, Q_{max} is expected to decrease, and this is monitored to track the health of the battery. The initial state of charge Q_0 and the new battery capacity Q_{max} need to be estimated.

Steps for Deployed Parameter Estimation

There are two main steps to run parameter estimation in deployed mode:

- 1 Make a *setup* file, to set up parameter estimation objects for use in deployed mode
- **2** Make a *run* file, which is a MATLAB function for parameter estimation that can be compiled and run in deployed mode

It is recommended to create the *setup* and *run* files by starting with MATLAB code generated from the **Parameter Estimator**. Copy, split, and modify the generated code to make the *setup* and *run* files as demonstrated in the following section.

Parameter Estimation in Non-Deployed Mode

First generate MATLAB code to estimate Q_0 and Q_{max} in non-deployed mode. Use the following commands to load the pre-configured estimation session:

load sdoBattery_spesession_forDeployment
spetool(SD0SessionData)

This step loads and plots the experiment with measured voltage and current data and configures the **Parameter Estimator** to estimate Q_0 and Q_{max} .

Navigate to the **Estimate** button in the toolstrip and from the dropdown list, select **Generate MATLAB Function** (see "Generate MATLAB Code for Parameter Estimation Problems (GUI)" on page 2-178). This step generates a MATLAB function which is added to the MATLAB editor, and a MAT-file parameterEstimation_sdoBattery_Data.mat. The generated code is available for you in file parameterEstimationSdoBattery.m. You can use the generated code to estimate parameters in non-deployed mode.

It is recommended to start with this generated code and copy, split and modify the code to create the *setup* and *run* files described in the following sections.

Setup-File for Deployed Parameter Estimation

To estimate parameters in deployed mode, the code for non-deployed parameter estimation can be split into a *setup* file to use in non-deployed mode, and a *run* file to use in deployed mode. The setup file is available as parameterEstimationSdoBattery_setup.m and the main parts are:

- **1** Define parameters
- 2 Define experiments
- **3** Prepare for deployment and save

Define Parameters

Parameters are defined in parameterEstimationSdoBattery_setup.m in the same way as the generated MATLAB code, parameterEstimationSdoBattery.m. Use the sdo.getParameterFromModel command to create a parameter object, containing fields for parameter value, minimum and maximum, and a field ("Free") indicating whether the parameter will be tuned during estimation.

In this example, parameter information is also stored in a database in which cars are identified by a code akin to a pseudo vehicle identification number (VIN). The car manufacturer can use this to monitor the health of the battery over time. The parameterEstimationSdoBattery_setup.m file uses the VIN database to update battery parameter values. See the parameterEstimationSdoBattery_setup.m file for more details.

The initial database is loaded from the MATLAB file sdoBatteryVinDatabase.mat which has the VIN database stored in variable vinDatabase. This is a containers.Map object, and the VIN key 4DEF is used to look up parameters for the battery in this example.

Run

vinDatabase("4DEF")

to display the following table:

	ĸ	Loss	Qmax	QO	v
CurrentValue	0.32	0.012	250	150	400
PreviousValue	0.32	0.012	250	150	400

Define Experiments

Experiments are defined in parameterEstimationSdoBattery_setup.m in the same way as the generated MATLAB code, parameterEstimationSdoBattery.m. Experiments have measured data and information about specific ports or signals in the model that are associated with the data.

Prepare for Deployment and Save

At the end of the parameterEstimationSdoBattery_setup.m file, define a simulator which can run the model and compare model output to measured data. Use the prepareToDeploy command to configure the experiments and simulator so they can be used in deployed mode. Save these prepared objects to a MAT-file.

```
84 %% Configure objects for deployment and save to a MATLAB file
85 %
86 - Exper = prepareToDeploy(Exper);
87 %
88 - Simulator = createSimulator(Exper);
89 - Simulator = prepareToDeploy(Simulator,p);
90 - save sdoBatteryObjectsToDeploy Exper Simulator p vinDatabase
```

When running these steps on another model and preparing for deployment, you may be prompted to save the model to continue after running the *setup* function. Save the model to preserve logging settings that need to be in place for deployed mode.

The *run* file parameterEstimationSdoBattery_run.m uses the objects saved in sdoBatteryObjectsToDeploy.mat for parameter estimation in deployed mode.

Run-File for Deployed Parameter Estimation

The run file is available as parameterEstimationSdoBattery_run.m and the main parts are:

- **1** Load preconfigured deployment objects
- 2 Update experiments and parameters
- 3 Run optimization
- 4 Update Parameter Database

Load Preconfigured Deployment Objects

The parameterEstimationSdoBattery_run.m needs a pragma so that the Simulink Compiler includes the model in the compiled code as follows:

```
20 %% Ensure model is compiled
21 %#function sdoBattery.slx
```

Load the preconfigured objects that were saved at the end of parameterEstimationSdoBattery_setup.m file as follows:

 24
 %% Load configured experiment and simulator objects

 25 load sdoBatteryObjectsToDeploy Exper Simulator p vinDatabase

Update Experiments and Parameters

The parameterEstimationSdoBattery_run.m file takes two input arguments:

- dataFilename a data file name for experiment data
- vin a vehicle identification number for parameter values

Read the data from the comma-separated-values (CSV) text file specified by dataFilename. Use the *updateIOData* command to update the deployed experiments with new input and output data (current and voltage data for this model). Since the data is from a CSV file, you do not need the getData function that is present in the generated MATLAB code, parameterEstimationSdoBattery.m.

31 -	<pre>data = readtable(dataFilename);</pre>
32	%Update experiment with input current data
33 -	<pre>current = timeseries(data.Current,data.Time);</pre>
34 -	<pre>Exper = updateIOData(Exper,'sdoBattery/Current',current);</pre>
35	%Update experiment with output voltage data
36 -	<pre>voltage = timeseries(data.Voltage,data.Time);</pre>
37 -	<pre>Exper = updateIOData(Exper,'sdoBattery/SOC -> Voltage',voltage);</pre>

Use the VIN as a key to look up this car's battery parameters in the parameter database. Use the current value from the database to update the initial parameter values prior to running the new estimation. See the parameterEstimationSdoBattery_run.m file for more details.

Run Optimization

The next several steps in parameterEstimationSdoBattery_run.m are very similar to the code in parameterEstimationSdoBattery.m (for non-deployed estimation). Define a handle to the estimation objective function, specify optimization options, and use the sdo.optimize function. This step runs the model and compares model output to experiment data. Parameters are tuned to achieve a close match between the model and data.

The objective function is defined in the subfunction sdoBattery_optFcn which is also like the objective function in parameterEstimationSdoBattery.m. However, the name of the signal logging variable needs to be specified since it cannot be queried from the model in deployed mode.

```
117 - SimLog = find(Simulator.LoggedData,'logsout');
```

To determine the name of the variable ('logsout' in this case), query the model from MATLAB in non-deployed mode:

```
get_param('sdoBattery','SignalLoggingName')
```

Alternatively, in Simulink use the **Modeling** tab in the toolstrip and click **Model Settings**. In the configuration dialog, select **Data Import/Export** and find the variable name in the Signal logging box.

Update Parameter Database

After calling sdo.optimize in the main function of parameterEstimationSdoBattery_run.m, update the VIN database. For each parameter that is estimated, copy the CurrentValue to the PreviousValue and then use the new parameter estimate to update the CurrentValue. See parameterEstimationSdoBattery_run.m for more details.

```
40
       %% Update Parameters for this Vehicle from the Database
41 -
       if nargin < 2
42 -
            vin = "4DEF";
43 -
       end
44 -
       carData = vinDatabase(vin);
45 -
     for ct = 1:width(carData)
46 -
            variableName = carData.Properties.VariableNames{ct};
47 -
           bIdx = strcmp(variableName, {p.Name});
48 -
            p(bIdx).Value = carData{'CurrentValue',variableName};
49 -
      - end
```

Running Parameter Estimation in Deployed Mode

Use the mcc command to compile the parameterEstimationSdoBattery_run.m function from either the MATLAB command window or the DOS or UNIX command prompt. You need to have MATLAB Runtime installed to complete the following steps. For more information, see "Install and Configure MATLAB Runtime" (MATLAB Compiler).

```
Command Window
>> mcc -m parameterEstimationSdoBattery_run
### Building the rapid accelerator target for model: sdoBattery
### Successfully built the rapid accelerator target for model: sdoBattery
```

Run parameter estimation in deployed mode.

🚾 Command Prompt

```
C:\Users\msaginaw\CarBattery> parameterEstimationSdoBattery run sdoBattery Data1.csv 4DEF
Optimization started 17-Jan-2020 13:43:31
                                          First-order
Iter F-count
                               Step-size optimality
                     f(x)
             0.00467119
   0
         10
             2.04632e-06
                               0.01003
                                            0.00712
         15 1.63009e-07
                              0.001241
                                            0.00108
ocal minimum possible.
lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the value of the function tolerance.
```

In MATLAB, run

vinDatabase("4DEF")

to display the following result:

	ĸ	Loss	Qmax	QO	v
					—
CurrentValue	0.32	0.012	250.35	152.81	400
PreviousValue	0.32	0.012	250	150	400

Tracking Battery Parameters over Time

The table below shows estimates of battery parameters Q_0 and Q_{max} over time. The file sdoBattery_Datal.csv contains data for the battery when it was new, sdoBattery_Data2.csv contains data for the battery when it was 1 year old and sdoBattery_Data3.csv contains data for the battery when it was 2 years old.

Data File	Qo	Qmax	Ratio
sdoBattery_Data1.csv	152.81	250.35	61%
sdoBattery_Data2.csv	101.50	199.01	51%
sdoBattery_Data3.csv	86.91	184.45	47%

Observe that there is degradation in battery capacity over time. There is a high rate of degradation in the first year after which the rate of degradation reduces. When the battery was new, the round-trip commute left the battery state of charge at 61% while after 2 years, the commute left the battery state of charge at 61% while after 2 years, the commute left the battery state of charge falls below 40%, this condition reduces the number of

times the battery can be recharged. By tracking battery parameters over time, the manufacturer can monitor the battery health, and determine if the car needs a new battery.

See Also

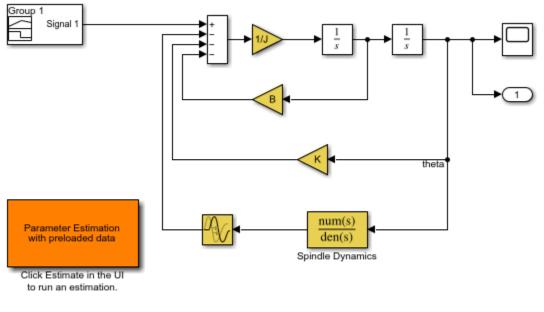
prepareToDeploy(Experiment) | prepareToDeploy(SimulationTest) |
updateIOData(Experiment) | sdo.Experiment | sdo.SimulationTest

Muscle Reflex Parameter Estimation

This example shows how to estimate parameters of a muscle reflex model.

Simulink Model of the Muscle Reflex

The Simulink® model for the muscle reflex system, spe_muscle, is shown below.



Muscle Reflex Model

Copyright (c) 2002-2014 The MathWorks, Inc.

Muscle Reflex Model Description

For this example a simple knee reflex action of humans is modeled. When the patellar tendon is excited, for example when a doctor strikes it with the nub of a small rubber hammer, the tendon reacts with a small but quick reflex force. This in turn pulls the muscle and we observe that the leg jerks forward slightly at the knee.

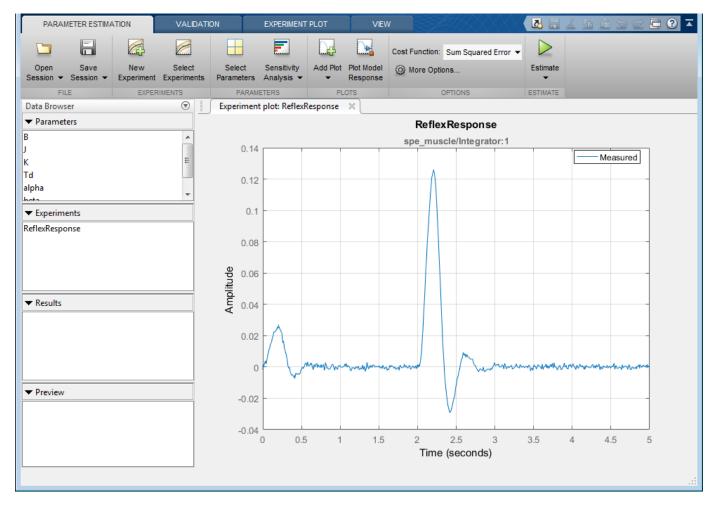
For this model we treat the tendon itself as a small torsional spring damper with inertia (J), stiffness (K) and damping (B). When the tendon is excited a signal is sent through the nervous system to the spinal cord reporting a structural change (i.e. tendon length). The nervous system then sends a signal back to the tendon to produce a reflex. There are receptors on the muscle called spindles that have their own dynamics, shown in the model as a transfer function in the feedback path. The spindles are modeled as a spring (Kpe) and damper (Bpe) in parallel, and then with the pair in series with another spring (Kse). The differential equation describing these dynamics is given by

$$T' = (Kse/b) * [Bpe * x' + Kpe * x)] - [(Kse + Kpe)/b] * T$$

For this model we supply two brief pulses, one stronger than the other, as input. This is similar to what one might experience in a doctor's office.

Estimation Data

There is a project already associated with this model. You can access it by double-clicking the orange block in the lower left corner of the model. This opens the **Parameter Estimator** configured with measured experiment data ReflexResponse and parameters J, B, K, Td, beta, alpha, and tau selected for estimation. The measured data in the ReflexResponse experiment is shown in the plot. There is only one data set used for this example.



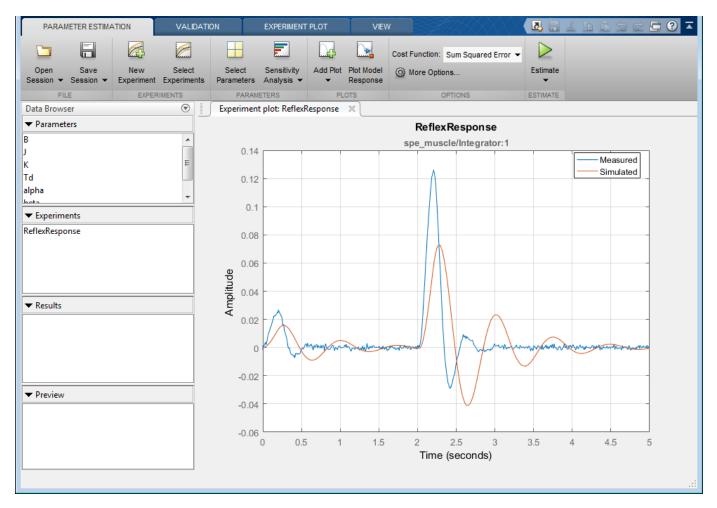
The experiment data can be imported from various sources including MATLAB® variables, MAT files, Excel® files, or comma-separated-value files.

Estimated Parameters

The estimation parameters are selected by clicking on **Select Parameters** in the **Parameter Estimation** tab. We have already loaded the parameters for this model. These parameters are the inertia, J; damping coefficient, B; the return spring constant, K; the neural transmission delay, Td as well as the spindle dynamics parameters beta, alpha, and tau. Since we know from our physical insight that none of these parameters can be negative we set their lower limits to zero. Based on known neural transmission times, we set the lower limit of Td to 10 microseconds.

Edit: Estimated Parameters		X
Parameters Tuned for all Experiments		
<u>B</u>		
▼ 3 ×	-	🗶 🗹 Estimate
Minimum: 0		
Maximum: Inf	4	
Scale: 3		
1		
▶ 1	-	K 🗹 Estimate
<u>K</u>		
▶ 75	-	🗶 🗹 Estimate
	_	_
• 0.01 •		🗶 🖾 Estimate
alpha		
		🗶 🖾 Estimate
beta		🗶 🖾 Estimate
	_ //	Listinate
tau ▶ 0.005 ▼	B :	🗶 🖾 Estimate
Belect parameters		
Parameters and Initial States Tuned per Exper	iment	
Experiment: ReflexResponse 💌		
Select experiment initial states for estimation.		
There are no initial states defined for this experim	ient.	
Select experiment parameters for estimation.		
There are no parameters defined for this experime	ent.	

The experiment plot is also used to see how well the measured data matches the current model. Click **Plot Model Response** to display simulated signal data on the experiment plots. The simulation results show that the model does not match the measured data and that model parameters need to be estimated.

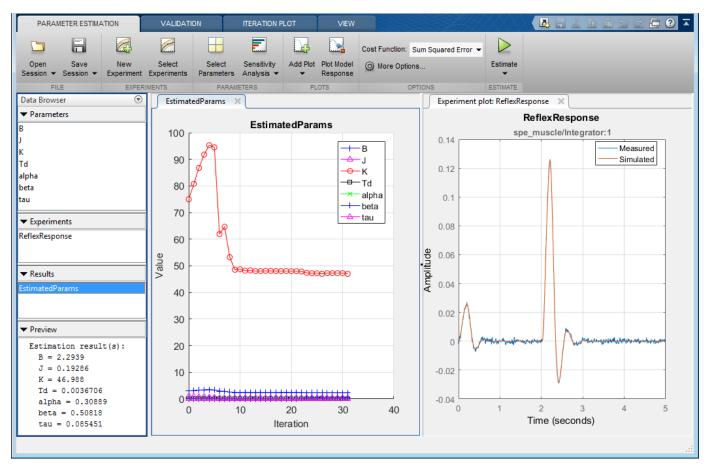


Estimation

With the parameters for estimation specified, we select experiments to use for estimation. Click **Select Experiments** in the **Parameter Estimation** tab and select ReflexResponse for estimation.

We are now almost ready to start our estimation but first create another plot to monitor the estimation progress. Click **Add Plot** and select **Parameter Trajectory**. This creates a plot that shows how the parameter values change during estimation. Click the **View** tab to lay out the plots so that the experiment plot and trajectory iteration plot are both visible.

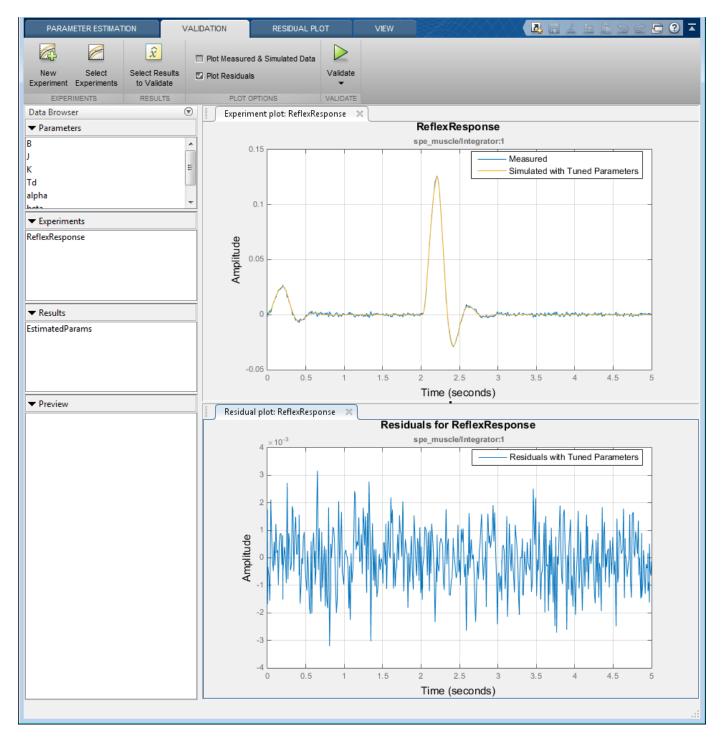
Click the **Estimate** button in the **Parameter Estimation** tab to start the estimation. The estimation will keep iterating the parameter values until the estimation converges and terminates. The plot below shows the measured data overlaid with the simulated data. The simulated data comes from the model with the estimated parameters. The results of the estimation appear satisfactory, the simulated curve closely matches the measured results.



Validation

We can also view residuals of the estimation. Residuals are the error between the measured response and simulated response at each time step.

Click the **Validation** tab and click **Select Experiments**. Select the ReflexResponse experiment for validation. In the **Validation** tab, select **Plot Residuals** and click **Validate**. The plot below shows that the residuals do not exhibit a correlation pattern. They are one or two orders of magnitude smaller than the measured data and are essentially the noise from the experimental data, so we are again satisfied that parameters in the model were estimated well.



The parameters of the model have been tuned to match the experimental results very well and our estimation error is only the original noise in the results. We can conclude that the parameters in the model have been successfully estimated.

Close the model.

See Also

More About

- "Specify Parameters for Estimation" on page 2-7
- "Validate Estimation Results" on page 2-25

Click Estimate in the U

to run an estimation.

DC Servo Motor Parameter Estimation

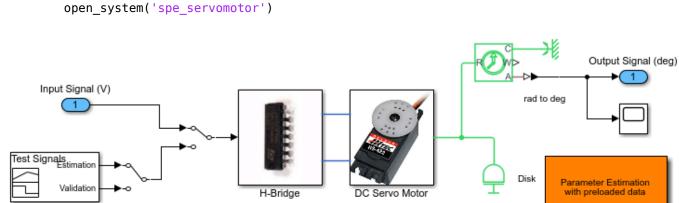
This example shows how to estimate the parameters of a multi-domain DC servo motor model constructed using various physical modeling products.

This example requires Simscape[™] Driveline[™] and Simscape[™] Electrical[™].

Description of the DC Servo Motor System

A DC servo motor, with its electrical and mechanical components, provides a great example to illustrate multi-domain modeling using first principles.

The DC servo motor is part of a larger system that contains the control electronics (H-Bridge) and a disk attached to the motor shaft. The overall model, spe_servomotor, is shown below, where the Input Signal (V) is the voltage signal applied to the H-bridge circuit, and the Output Signal (deg) is the angular position of the motor shaft.

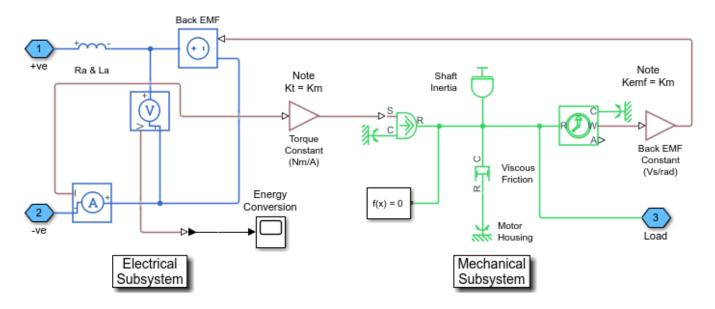


Motor Test Signals Signal Selector

Copyright 2009-2017 The MathWorks, Inc.

We developed a first-principles model of the DC motor within the DC servo motor subsystem. We used Simscape Electrical to model the electrical components and Simscape Driveline to model the mechanical components of the motor. The figure below shows the content of the servo motor subsystem.

open_system('spe_servomotor/DC Servo Motor')



The DC motor model shows a relationship from current to torque (the green line on the left). The torque causes the shaft of the motor to spin and we have a relationship between this spinning to the Back EMF (electromotive force). The rest of the parameters include a shaft inertia, viscous friction (damping), armature resistance, and armature inductance.

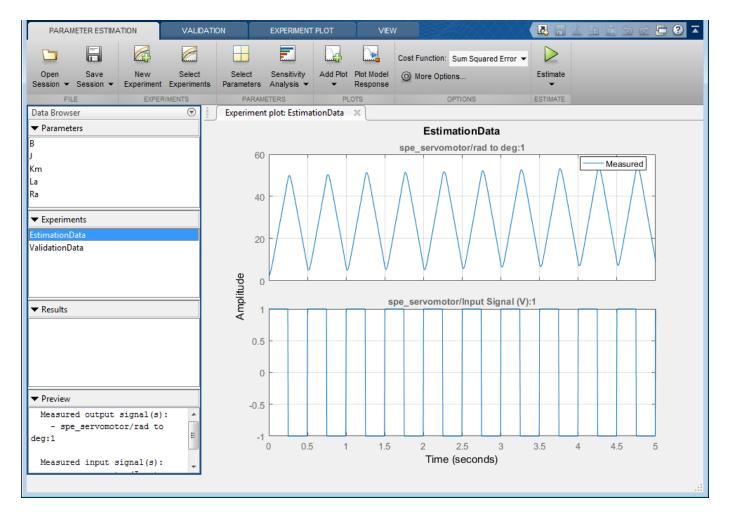
While manufacturers may provide values for some of these quantities, they are only approximate. We want to estimate these parameters as precisely as possible for our model to ascertain whether it is an accurate representation of the actual DC servo motor system.

When we apply a series of voltage pulses to the motor input, the motor shaft turns in response. However, if the model parameters do not match those of the physical system, the model response will not match that of the actual system, either. This is where Simulink® Design Optimization[™] plays a pivotal role in estimating parameter values. A parameter estimation process consists of a number of well-defined steps:

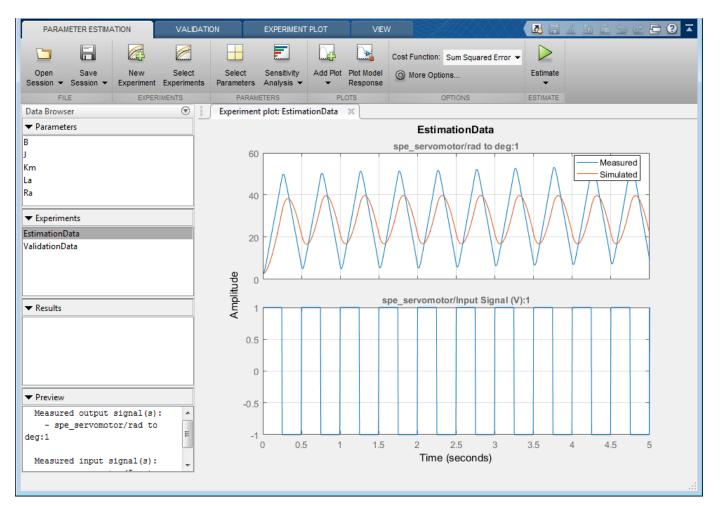
- Collect test data from your system (experiment).
- Specify the parameters to estimate (including initial guesses, parameter bounds, etc.).
- Configure your estimation and run a suitable estimation algorithm.
- Validate the results against other test data sets and repeat above steps if necessary.

Simulink Design Optimization provides the **Parameter Estimator** app which is a user interface to help you perform parameter estimation, organize your estimation project, and save it for future work.

Double-click the orange block in the lower right corner of the servomotor model to launch the **Parameter Estimator**, pre-loaded with data for this project. This is configured with measured experiment data EstimationData. For other uses you can import experimental data sets from various sources including MATLAB® variables, MAT files, Excel® files, or comma-separated-value files. The **Parameter Estimator** is also pre-loaded with parameters for the servomotor subsystem selected for estimation: B, J, Km, La, and Ra. It is also configured with validation data ValidationData which we will use later, after estimation. The measured data in EstimationData is shown in the experiment plot. There is only one data set used for estimation in this example.



The experiment plot is also used to see how well the measured data matches the current model. Click **Plot Model Response** in the **Parameter Estimation** tab to display simulated signal data on the experiment plots. The simulation does not match the measured data, showing that the model parameters need to be estimated.



Selecting Parameters for Estimation

Simulink Design Optimization lets you estimate some or all of the parameters in your model in a manner that best suits your application. The estimation parameters are selected by clicking **Select Parameters** in the **Parameter Estimation** tab. For our DC motor example, we have already loaded the five parameters of the motor model: B, J, Km, La, and Ra. Since we know from our physical insight that none of these parameters can be negative we set their lower limits to zero.

Edit: Estimated P	arameters		×
Parameters Tu	ned for all Experiments		
<u>B</u>			
▼ 0.008		•	🗶 🖾 Estimate
Minimum:	0	•	
Maximum:	Inf	•	
Scale:	0.008	•	
1			
▶ 5.7e-07		•	🗶 🖾 Estimate
<u>Km</u>			
• 0.0134	•	•	🗶 🗹 Estimate
<u>La</u>			
▶ 6.5e-05	Y	- 🖶	🗶 🗹 Estimate
Ra N I O			🗙 🖾 Estimata
1.9	· · · · · · · · · · · · · · · · · · ·		🗶 🗹 Estimate
🛃 Select para	neters		
Parameters an	d Initial States Tuned per Ex	perimen	t
Experiment: Est	imationData 💌		
Select experime	nt initial states for estimation.		
There are no in	itial states defined for this expe	eriment.	
Select experime	nt parameters for estimation.		
			.::

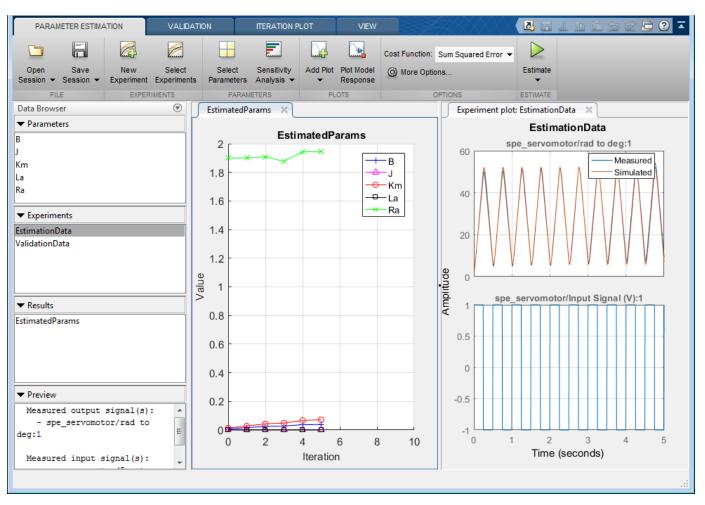
Estimating Parameters of the DC Motor Model

With the parameters for estimation specified, we select experiments to use for estimation. Click **Select Experiments** in the **Parameter Estimation** tab and select EstimationData for estimation.

We are now almost ready to start our estimation but first create another plot to monitor the estimation progress. Click **Add Plot** and select **Parameter Trajectory**. This creates a plot that shows how the parameter values change during estimation. Click the **View** tab to lay out the plots so that the experiment plot and trajectory iteration plot are both visible.

Click the **Estimate** button in the **Parameter Estimation** tab to start the estimation. The estimation will keep iterating the parameter values until the estimation converges and terminates. Parameter Estimation provides various state-of-the-art estimation methods. The most common selections include the nonlinear least-squares and Nelder-Mead optimization methods. More information on these methods is available in the Optimization Toolbox[™] documentation. You can also use the pattern search method in the Global Optimization Toolbox for parameter estimation.

The plot below shows the measured data overlaid with the simulated data. The simulated data comes from the model with the estimated parameters. Comparing the response of the system before and

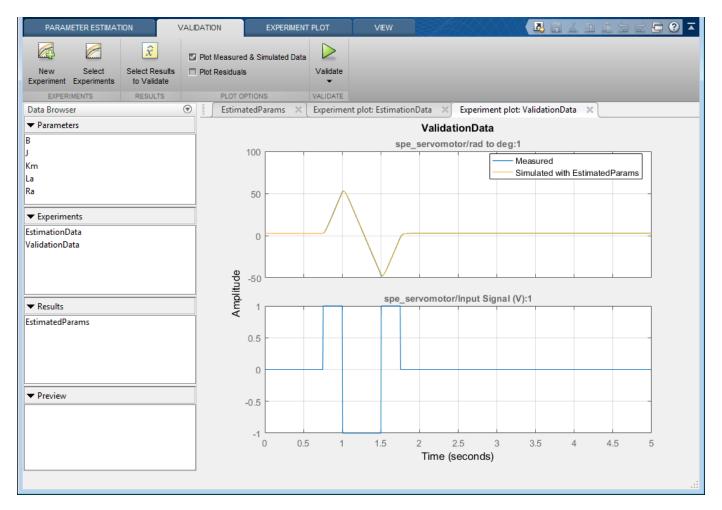


after the estimation process clearly shows that the estimation successfully identified the model parameters and the simulated response accurately matches the experimental data.

Validation

After completing the estimation, it is important to validate the results against other data sets. A successful estimation should match not only the experimental data that we used for estimation, but also the other data sets that we collected in our experiments.

We used our second data set in ValidationData for validating the estimation results. As the next figure shows, the match between the model response and the data set is very good. In fact, the two curves are almost identical for this example.



Summary

Engineers and scientists across disciplines and industries are well acquainted with the benefits of modeling dynamic systems. They may use either first-principles mathematics or test-data based methods. First-principles models provide important insight into system behavior, but may lack accuracy. Data-driven models provide accurate results, but tend to offer limited understanding of the physics of the system. This article showed the use of Parameter Estimation to improve the accuracy of a DC Servo Motor model by estimating the model parameters using experimental data.

Close the model

bdclose('spe_servomotor')

Engine Speed Model Parameter Estimation

This example shows how to estimate the coefficients of a nonlinear (quadratic) function to approximate the dynamic behavior of a system component.

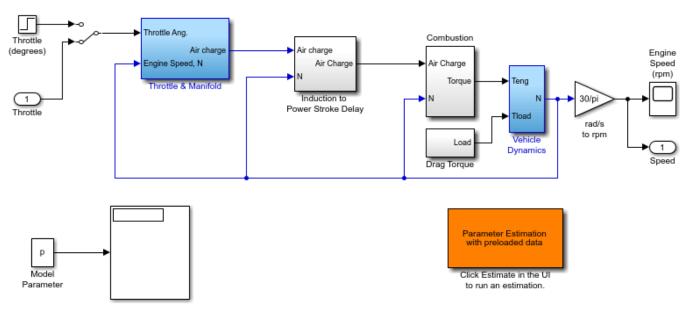
Description of the Engine Speed Model

The Simulink® model for the engine system, spe_speed, is shown below.

The throttle angle from the block labeled "Throttle" on the left side of the diagram drives the simulation. The output of interest in the model is the engine speed, which can be monitored by opening the Scope block labeled "Engine Speed (rpm)".

Open the engine speed model.

open_system('spe_speed')



Engine Speed Model

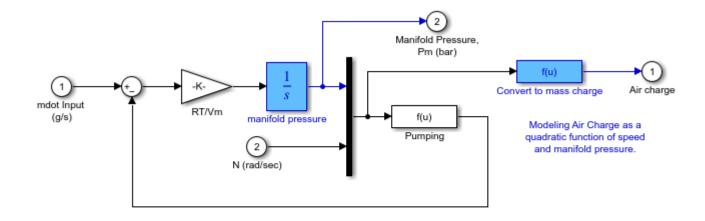
Copyright (c) 2005-2014 The MathWorks, Inc.

Modeling Air Charge Using a Nonlinear Function

Among other dynamic components in the model, the "Intake Manifold" subsystem is used to model the dynamics of the air intake manifold in the engine.

Open the Intake Manifold subsystem.

open_system('spe_speed/Throttle & Manifold/Intake Manifold')



Intake Manifold Vacuum

In particular, the "Convert to mass charge" block above defines a quadratic multi-variable polynomial to approximate the relationship between the Air Charge, the Manifold Pressure, and the Engine Speed. This approximation has the following form:

$$AirCharge = p(1) \times EngineSpeed + p(2) \times ManifoldPressure$$

 $+p(3) \times (ManifoldPressure)^{2} + p(4) \times EngineSpeed \times ManifoldPressure + p(5)$

The Parameter Estimation Problem

When measured data for various signals in your model are available, you can use Simulink® Design Optimization[™] to compute the unknown parameters.

The parameter estimation problem in our case is to compute the coefficients

p(1), p(2), p(3), p(4), p(5)

using measured data.

You can launch a pre-configured parameter estimation task in the **Parameter Estimator** by first opening the model and by double-clicking on the orange block in the lower corner of the model.

Close the model

bdclose('spe_speed')

Clutch Friction Coefficient Estimation

This example shows how to use Simulink® Design Optimization^m to estimate parameters of a clutch model created using Simscape^m Driveline^m library blocks.

Requires Simscape Driveline.

Description of Clutch Model

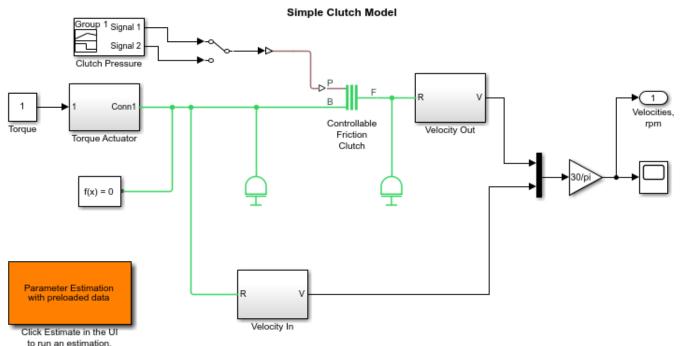
The Simulink® model of the clutch system, simple_clutch, is shown below.

This model consists of two inertias coupled by a clutch. Initially, the pressure applied to the clutch plates is zero and Inertia 2 has zero velocity. A constant torque is also applied to Inertia 1. Once the clutch pressure starts increasing, Inertia 2 starts rotating. However, the friction between the clutch plates causes slippage so that the two inertias accelerate at different rates and have different velocities.

The clutch system consists of two rotational inertias and a clutch. Pressure is applied to the clutch plates, which then couples the two inertias. A **Simscape Driveline** block is used to model the clutch, which has a speed-dependent friction coefficient linearly varying from C1 at 0 rad/s to C2 at 10 rad/s.

The coefficients of friction (C1, C2) in the **Controllable Friction Clutch** block are unknown and are estimated using experimental data for the output velocities of Inertia 1 and Inertia 2.

```
open_system('simple_clutch')
set( find_system(gcs, 'FindAll', 'on', 'BlockType', 'Scope'), 'Open', 'off' )
```





Using Simulink Design Optimization

In the **Apps** tab, click **Parameter Estimator** under **Control Systems** to launch the **Parameter Estimator** app.

The launched Parameter Estimation UI consists of projects where we store our experimental data sets and estimation results. These projects can be saved and reused later.

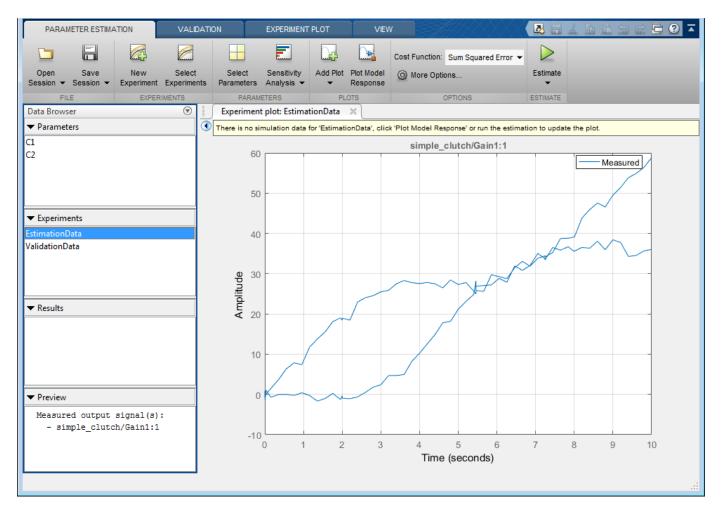
Alternatively, you can double-click the orange colored block at the lower left corner of the Simulink diagram. This will reload a project that has already been saved.

In general, estimating model parameters consists of three main steps: importing experimental data sets into the project, selecting the model parameters for estimation, and running an estimation and analyzing the results.

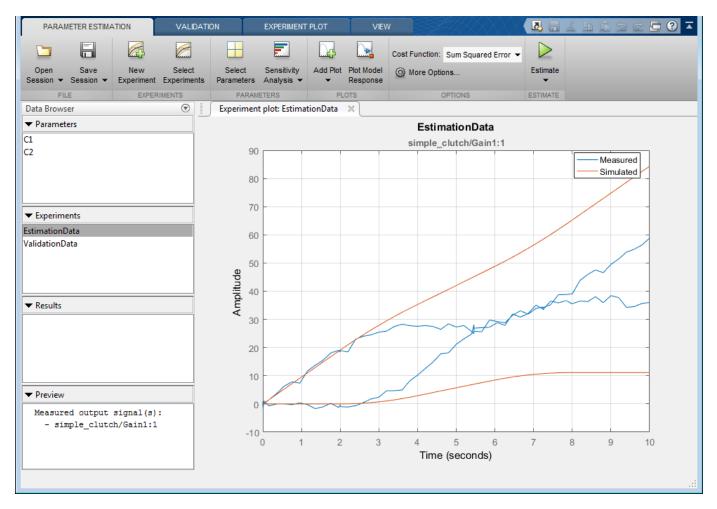
Experimental Data for Estimation

We have two sets of output data on this clutch system. The first one, **EstimationData**, will be used for parameter estimation and the other one, **ValidationData**, for validating the response of the Simulink model with the estimated parameters.

In the first experiment the clutch pressure follows the profile of **Signal 1** supplied by the **Clutch Pressure** block in the Simulink model. This signal applies a ramp-up and a ramp-down pressure on the clutch plates. Click **Add Plot** in the Parameter Estimation UI, and select **EstimationData** to view the output velocities of the inertias in response to this input. Such data sets could also be imported from various sources including MATLAB® variables, MAT files, Excel® files, or comma-separatedvalue files.



The parameter values for the friction coefficients are not known accurately. Clicking **Plot Model Response** provides a look at the response of this system, and shows that it does not match the experimental data, hence the parameters need to be estimated for a better fit.



Setting up and Running the Estimation

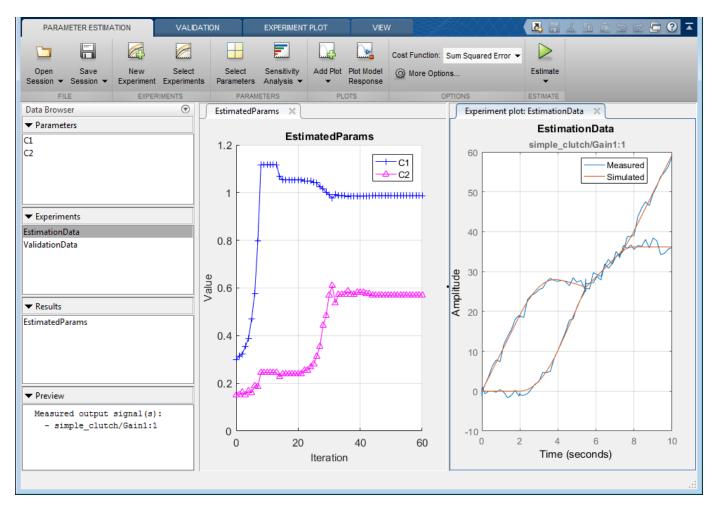
We will use the experimental data set **EstimationData** to estimate friction parameters of the clutch system.

The first step is to define the variables to be estimated. This establishes which parameters of the simulation can be adjusted, and any rules governing their values. Click **Select Parameters** to specify parameters to be estimated. Here we wish to estimate the friction coefficients C1 and C2 in the **Controllable Friction Clutch** block of the Simulink model. In the pre-loaded parameter estimation example, these parameters have already been specified for estimation. If there are known bounds on the parameter values, they can be set in the minimum and maximum fields.

Edit: Estimated Pa	arameters			×
Parameters Tu	ned for all Experiments			-
<u>C1</u>				
▼ 0.3	- 🖶	× I	🖉 Estim	ate
Minimum:	0 🔹 🗗			
Maximum:	1 👻 🕁			
Scale:	0.5 👻 🕁			
<u>C2</u>				
• 0.15	- 🚽	x I	🖉 Estim	ate
😼 Select paran	neters			E
Parameters an Experiment: Est	d Initial States Tuned per Experiment imationData 💌			
Select experimer	nt initial states for estimation.			
There are no ini	tial states defined for this experiment.			
Calact avparimen	t parameters for estimation			
	it parameters for estimation.			
There are no pa	rameters defined for this experiment.			-
🛃 Edit experim	ent			-
•				Þ.
		_		

Next, click **Select Experiments** to specify which experiments are to be used for estimation. It is possible to use one or more data sets at once in a given estimation. For our example, we will use the data set called **EstimationData**.

You are now ready to run the estimation. Click **Estimate** to start the estimation process. We provide a number of estimation methods, including nonlinear least-squares minimization, gradient descent, pattern search, or simplex search. Running estimation will vary the model parameters in order to reduce the error between the simulation outputs and the experimental data. During estimation, the experiment plot showing measured data and simulation response will updated. As the parameter values improve, the simulation curve should get closer to the experimental data curve. Also, a trajectory plot will show the parameter values at each iteration. These curves should reach steady-state as the parameter values get closer to their physical values.

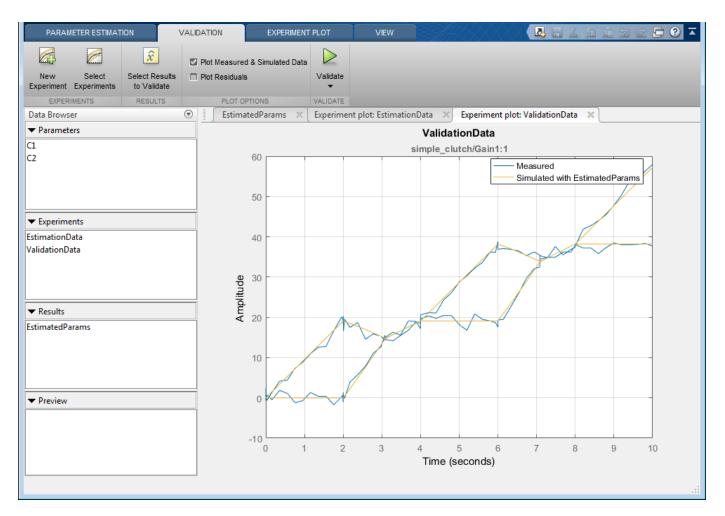


In addition, the table in the Estimation Progress Report will show data regarding the estimation process such as the number of iterations, the number of simulations, and the cost function. The cost function value represents the degree of fit between the simulation response and the estimation data. This value would decrease at each iteration, indicating the amount of improvement in the fit.

Validation

Once we complete the estimation, it is important to validate the results against other data sets. A successful estimation should be able to not only match the experimental data that we used for estimation, but also the other data sets that we collected in our experiments.

In the second set of experimental data we have for the clutch system, the clutch pressure follows the profile of **Signal 2** supplied by the **Clutch Pressure** block in the Simulink model. This signal applies a periodic pressure on the clutch plates. To use this, **first double-click on the Manual Switch block to change the input signal to the one used for validation data (Signal 2)**. Then in the Parameter Estimation UI, click the **Validation** tab, click **Select Experiments** and select the experiment **ValidationData** for validation. This contains output data corresponding to input from **Signal 2**. Finally, click **Validate** to carry out the validation. An experiment plot will compare the simulation response against experimental data. We see that the match is very good.



In summary, we have carried out estimation by specifying an experiment with measured output data, and designating certain parameters to be estimated. We then checked the parameter values by validating with a different data set, giving confidence in the parameter values.

Close the model

bdclose('simple_clutch')

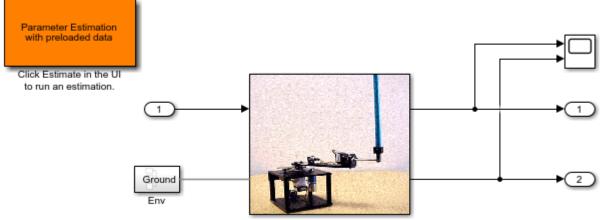
Inverted Pendulum Parameter Estimation

This example shows how to use Simulink® Design Optimization^m to estimate multiple parameters of a model by iterated estimations.

This example requires Simscape[™] Multibody[™] software.

Simscape Multibody Model of the Inverted Pendulum System

The Simulink® model for the inverted pendulum, spe_mech_invpend, is shown below.

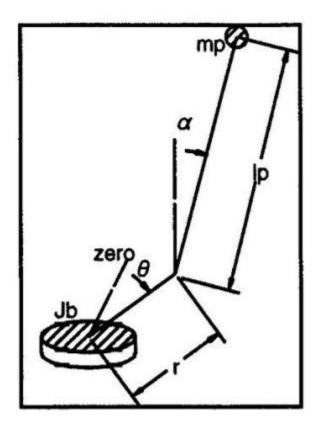


Rotational Inverted Pendulum

Copyright 2002-2014 The MathWorks, Inc.

Inverted Pendulum Model Description

The pendulum system has an arm that swings in the horizontal plane, driven by a DC motor. The purpose of the arm is to provide a balancing torque to a swinging pendulum, to keep the pendulum in an upright position. The angle of both the arm and pendulum is monitored and used as feedback to control the motion of the system. For this example we will only concentrate on estimating parameters of the uncontrolled system shown below.



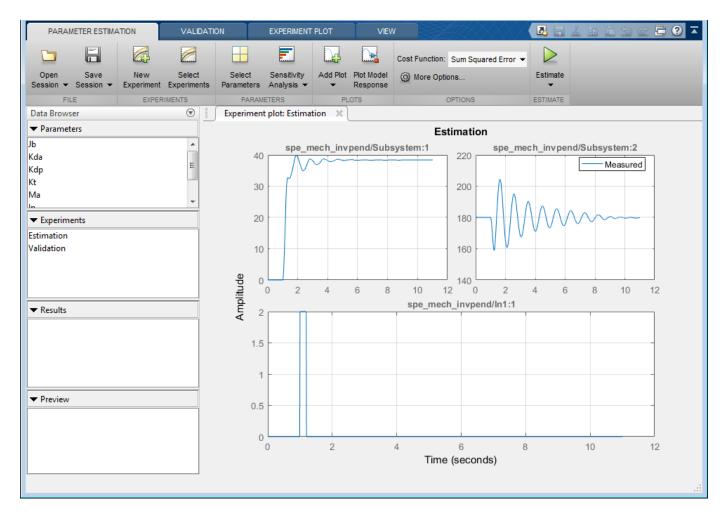
The system is modeled using Simscape Multibody. There are two bodies modeled in this system. The machine consists of one body representing the rotational arm and the other representing the pendulum. The bodies are connected by revolute joints that constrain the motion of the bodies relative to each other. An input voltage is delivered to a DC motor that provides the torque to the rotational arm.

The motor is modeled as a torque gain Kt. The arm of the pendulum has mass Ma, inertia Jb and length r. The pendulum has length lp and mass mp. For this example damping is modeled in the revolute joints using gains Kda and Kdp. The outputs of the system are the angles of the arm and the pendulum.

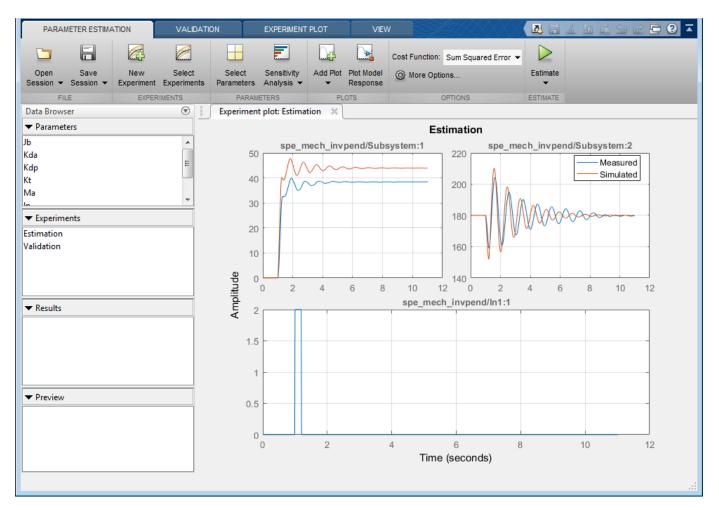
For this example we will run two estimations using different parameter sets for each estimation. This allows us to customize our estimation and can result in a more efficient solution.

Estimation Data

Double-click the orange block in the upper left corner of the inverted pendulum model to launch the **Parameter Estimator**, pre-loaded with data for this project. This is configured with measured experiment data Estimation. For other uses you can import experimental data sets from various sources including MATLAB® variables, MAT files, Excel® files, or comma-separated-value files. It is also configured with validation data Validation which we will use later, after estimation. The measured data in Estimation is shown in the experiment plot. There is only one data set used for estimation in this example.



The experiment plot is also used to see how well the measured data matches the current model. Click **Plot Model Response** in the **Parameter Estimation** tab to display simulated signal data on the experiment plots. The simulation does not match the measured data, showing that the model parameters need to be estimated.



Define Variables

The next step is to define the variables for the estimation. This establishes which parameters of the simulation can be adjusted, and any rules governing their values. Click **Select Parameters** in the **Parameter Estimation** tab. For our inverted pendulum example, we have already selected the torque gain parameter, Kt, for estimation. Since we know from our physical insight that this parameter can not be negative we set its lower limit to zero.

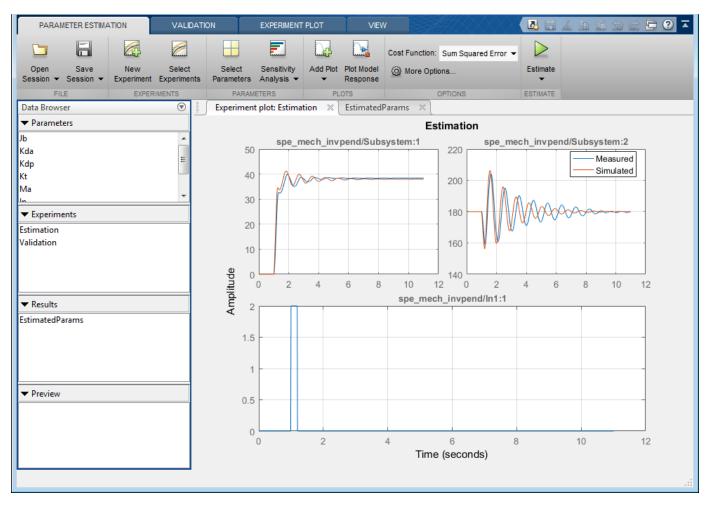
Edit: Estimated Parameters			×	
Parameters Tuned for all Experiments				
		_		
• 0.0019 •		<u>х</u> Пе	stimate	
<u>Kda</u>		_		
• 0.07 •	-//	<u>х</u> Пе	stimate	
		_		
		X 🗆 E	stimate	
<u>Kt</u>				
		X 🖾 E	stimate	
Minimum: 0	4			
Maximum: Inf	-			
Scale: 0.1342 💌	-			
Ma				
• 0.01 •		X III	stimate	
<u>Ip</u>				
▶ -0.192 ▼	-	X 🗆 E	stimate	
<u>mp</u>				
▶ 0.04 ▼	-	X DE	stimate	
1	_			
• 0.192 •	-//	X 🗆 E	stimate	
🛃 Select parameters				
Parameters and Initial States Tuned per Expe	riment			
Experiment: Estimation 💌				
Experimenta Estimation				
Select experiment initial states for estimation.				
There are no initial states defined for this experiment.				
Select experiment parameters for estimation.				
There are no parameters defined for this experim	ent.			

First Estimation

With the parameters for estimation specified, we select experiments to use for estimation. Click **Select Experiments** in the **Parameter Estimation** tab and select the experiment named **Estimation** for estimation.

We are now ready to start our estimation. Click **Estimate** in the **Parameter Estimation** tab to start the estimation. The estimation will keep iterating the parameter value until the estimation converges and terminates.

The plot below shows the experimental data overlaid with the simulated data. The simulated data comes from the model with the estimated parameter Kt. The results of the estimation show that the first output (the position of the arm) matches, however we can see that the second output (the position of the pendulum) does not show very satisfactory results. It is clear that additional estimation is needed to obtain better results.

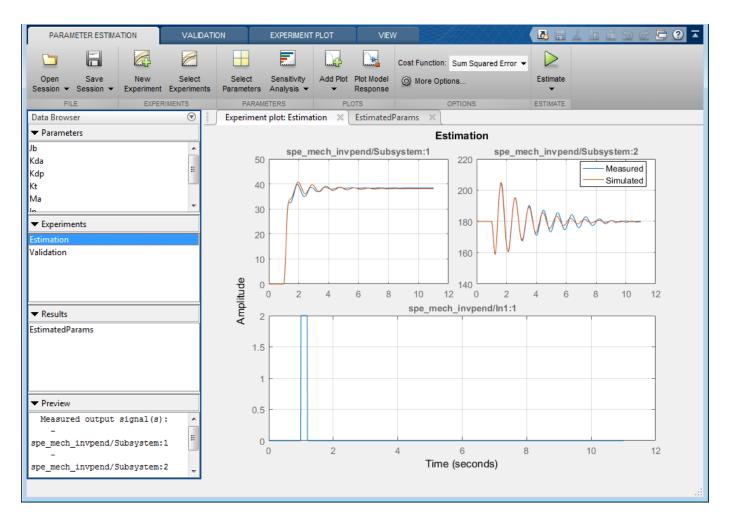


Running an Additional Estimation

This time we will leave the torque gain, Kt, constant and estimate the other parameters of the model. Click **Select Parameters** in the **Parameter Estimation** tab. Uncheck Kt, and check the other parameters as shown below.

Edit: Estimated Parameters	X				
Parameters Tuned for all Experiments					
• 0.0019 • 🕀	🗶 🖾 Estimate				
<u>Kda</u>	_				
• 0.07 • 🚽	🗶 🖾 Estimate				
Kdp					
▶ 0.0002 ▼ 🕀	🗶 🖾 Estimate				
<u>Kt</u>					
▶ 0.115901185063082 ▼ 🕀	🗙 🔲 Estimate				
<u>Ma</u>					
▶ 0.01	🗙 🖾 Estimate				
<u>lp</u>					
• -0.192 👻 🖶	🗙 🖾 Estimate				
<u>mp</u>					
▶ 0.04 ▼ 🕀	🗙 🖾 Estimate				
1					
• 0.192 • 🖶	🗙 🖾 Estimate				
Select parameters					
Parameters and Initial States Tuned per Experime	ent				
Experiment: Estimation 💌					
Select experiment initial states for estimation.					
There are no initial states defined for this experiment.					
Select experiment parameters for estimation.					
There are no parameters defined for this experiment.					
Edit experiment					
🔚 Update Model	쑫 OK (🥑 Help				

Click **Estimate** to start the new estimation. The results of the second estimation are shown below.

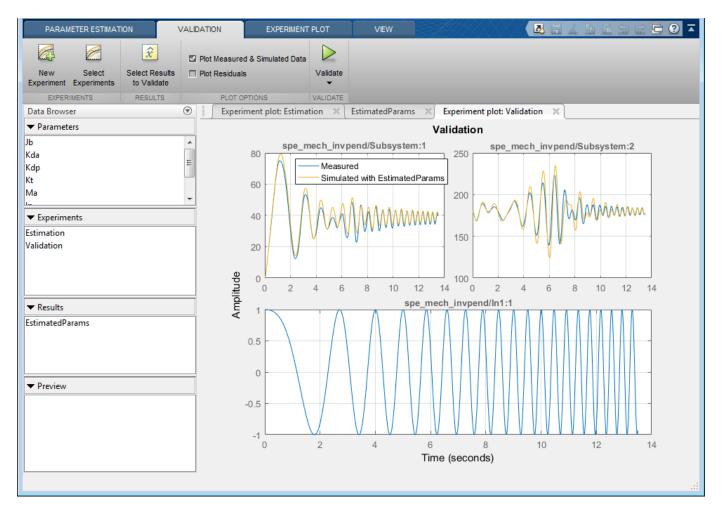


This is clearly a better result. This shows that in order to complete an estimation task it is not necessary to estimate all the parameters in the model at the same time. We can run multiple estimations keeping some parameters constant while varying others.

Validation

It is important to validate the results against other data sets. A successful estimation will not only match the experimental data that was used for estimation, but also the other data sets that were collected in experiments. An experiment named Validation has already been created for this project. Click Add Plot in the Parameter Estimation tab and select Validation to view the data.

Click **Plot Model Response** to see the simulation output overlaid on the data. The figure below shows how the inverted pendulum system responds to the validation input data. The validation shows that this model does handle the lower frequencies of the input validation data well and the model parameters were successfully estimated.



Conclusion

This example shows the flexibility of Parameter Estimation for segmenting an estimation task into multiple estimations. This allows for estimations to be run on different parameter sets which can help in the speed of estimating a given model.

Close the model.

Simplified Alternator Parameter Estimation

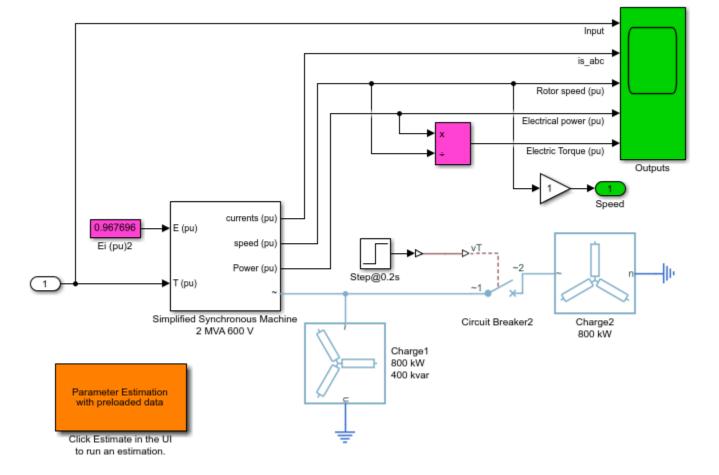
This example shows how to use parameter bounds to improve estimation performance. This is illustrated by estimating the power rating, P, of a synchronous machine.

This example requires Simscape[™] and Simscape[™] Electrical[™].

Simulink Model of the Simplified Alternator

The Simulink® model for the alternator system, spe_psbloadshed_machine, is shown below.

Speed Regulation of a Simple Alternator



Copyright 2002-2017 The MathWorks, Inc.

Model Description

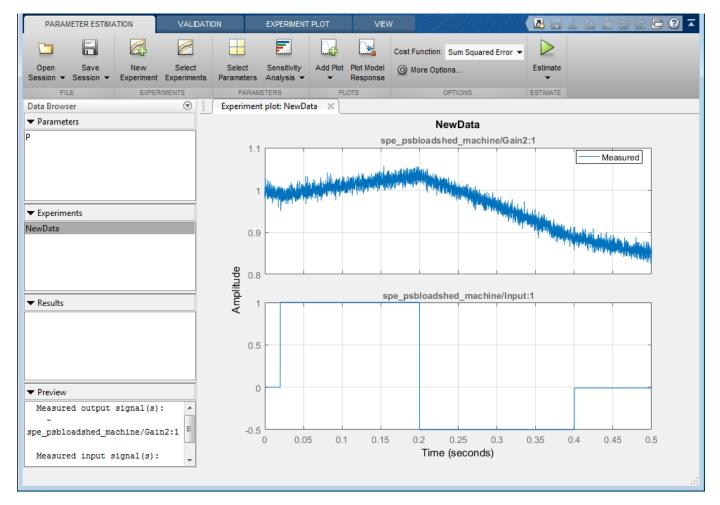
A three-phase, four-wire alternator rated 2000 kVA, 1600 kW, 0.8 power factor, 600 V, 1800 rpm is connected to a 1600 kW, 400 kvar inductive load. The stator neutral point is grounded. The internal impedance of the generator (Zg = 0.0036 + j*0.16 pu) represents the armature winding resistance Ra and direct axis transient reactance X'd. The total inertia constant of the generator and prime mover is H = 0.6 s, corresponding to $J = 67.5 kg.m^2$.

A three-phase breaker is used to switch out a 800 kW resistive load. The breaker is initially closed and it is opened at t = 0.2 s, resulting in a 50% load shedding.

The machine is excited with a constant voltage. The mechanical torque is modeled as a two-step signal. The first step has a magnitude of 1.0 and a duration of 0.18s. The second step has a magnitude of -0.5 for a duration of 0.2s.

Estimation Data

Double-click the orange block on the left side of the model to open the **Parameter Estimator** already loaded with experimental data, NewData. The input and output data from this experiment are shown in the plot. There is only one data set used for this example.



Define Variables

Click **Select Parameters** in the **Parameter Estimation** tab to specify parameters for estimation. Here, we have already loaded the parameters for this model. There is only one parameter in this model that we are interested in estimating, the nominal power, P, of the Simplified Synchronous Machine.

We know from the specs that this value should be around 2000kW but here we assume that we only have an initial guess of 3000kW. We also set the minimum value of the nominal power to be 3000kW, and the maximum value to be 4000kW.

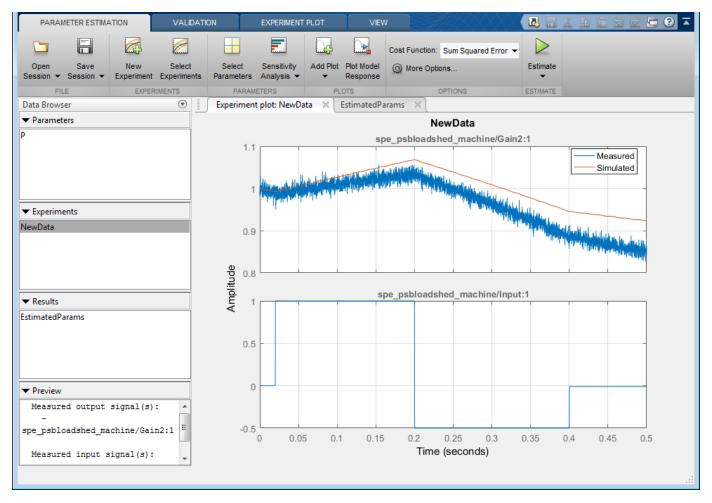
Edit: Estimated Parameters	×
Parameters Tuned for all Experiments	
<u>P</u>	
💌 3000000 👻 🖶 🗶 🖾	Estimate
Minimum: 3e+06 💌 🕁	
Maximum: 4e+06 💌 🕁	
Scale: 3e+06 💌 🕁	
Belect parameters	
Parameters and Initial States Tuned per Experiment	
Experiment: NewData 💌	
Select experiment initial states for estimation.	
There are no initial states defined for this experiment.	
Select experiment parameters for estimation.	
There are no parameters defined for this experiment.	
Edit experiment	
📠 Update Model 🛛 🖌 OK	Help

The Estimation Task

With the parameters for estimation specified, we select experiments to use for estimation. Click **Select Experiments** in the **Parameter Estimation** tab and select NewData for estimation.

Click **Estimate** to start the estimation. The estimation will keep iterating the parameter value and the experiment plot will continue to update, until the estimation converges and terminates.

The plot below shows the experimental data overlaid with the simulated data. The simulated data comes from the model with the estimated parameters. As expected the estimation terminated quickly and the results were not so good because of our initial guess and bounds of the parameter P.

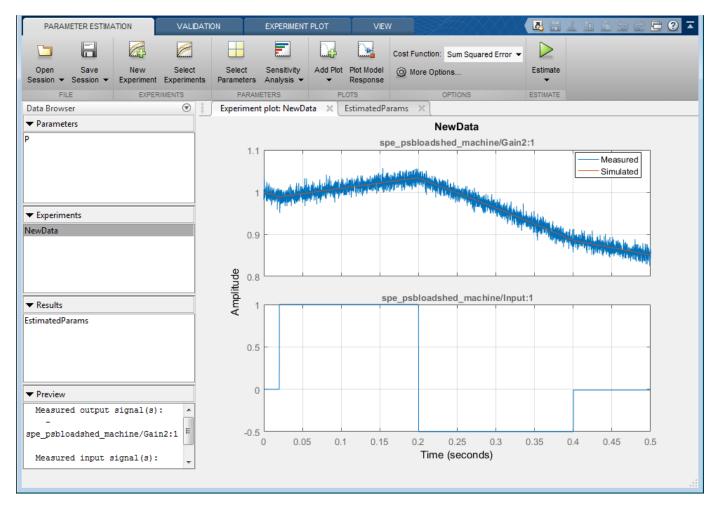


Running the Estimation with a Different Initial Guess

Since we are not satisfied with our results, we will choose a different initial guess. This is quite simple to do with the Simulink® Design Optimization[™] **Parameter Estimator**. Click **Select Parameters** in the **Parameter Estimation** tab and change the value to 1000 kV and the **Minimum** and **Maximum** bounds to 0 kV and 3000 kV respectively.

Edit: Estimated Pa	arameters	×
Parameters Tu	ned for all Experiments	
<u>P</u>		
▼ 1e+06	- 🚽 🚽 🗶 🖾	Estimate
Minimum:	0 🗸	
Maximum:	3e+06 💌 🖶	
Scale:	3e+06 💌 🖶	
📙 Select paran	neters	
Parameters and	d Initial States Tuned per Experiment	
Experiment: Nev	wData 💌	
Select experimen	t initial states for estimation.	
There are no ini	tial states defined for this experiment.	
Select experimen	t parameters for estimation.	
There are no pa	rameters defined for this experiment.	
Edit experime	ent	
	📙 Update Model 🛛 🖌 OK	Help

Click **Estimate** to run another estimation. Once the estimation is complete we can verify that our results are more accurate by looking once again at the measured vs. simulated response.



Conclusion

Placing inappropriate bounds on your parameters can lead to inaccurate results and most often the estimation will not converge successfully. However, with the parameter estimation GUI, it is simple to change these bounds to run another estimation without creating a new estimation project.

Close the model.

Response Optimization

- "How the Optimization Algorithm Formulates Minimization Problems" on page 3-3
- "Specify Signals to Log" on page 3-10
- "Specify Custom Requirements in the App" on page 3-11
- "Move Constraints" on page 3-14
- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Edit Design Requirements" on page 3-29
- "Specify Variable Requirements in the App" on page 3-31
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43
- "Specify Design Variables" on page 3-56
- "Update Model with Design Variables Set" on page 3-60
- "Specify Optimization Options" on page 3-62
- "Create Linearization I/O Sets" on page 3-64
- "Interact with Plots" on page 3-67
- "Compare Requirements and Design Variables Using Spider Plot" on page 3-71
- "Save Design Variable Values for Specific Iteration" on page 3-74
- "Design Optimization to Meet Time-Domain and Frequency-Domain Requirements (GUI)" on page 3-76
- "Design Optimization Tuning Parameters in Referenced Models (GUI)" on page 3-88
- "Design Optimization Tuning Parameters in Referenced Models (Code)" on page 3-95
- "Specify Steady-State Operating Point for Response Optimization" on page 3-101
- "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103
- "Design Optimization to Meet a Custom Objective (Code)" on page 3-118
- "Design Optimization to Meet Custom Signal Requirements (GUI)" on page 3-125
- "Design Optimization to Meet Frequency-Domain Requirements (GUI)" on page 3-129
- "Specify Custom Signal Objective with Uncertain Variable (GUI)" on page 3-143
- "Design Optimization with Uncertain Variables (Code)" on page 3-152
- "Generate MATLAB Code for Design Optimization Problems (GUI)" on page 3-160
- "Skip Model Simulation Based on Parameter Constraint Violation (GUI)" on page 3-163
- "Optimizing Parameters for Robustness" on page 3-170
- "Use Accelerator Mode During Simulations" on page 3-179
- "Speed Up Response Optimization Using Parallel Computing" on page 3-180
- "Use Parallel Computing for Response Optimization" on page 3-183
- "Use Fast Restart Mode During Response Optimization" on page 3-189
- "Optimization Does Not Make Progress" on page 3-191
- "Optimization Convergence" on page 3-192

- "Optimization Speed and Parallel Computing" on page 3-194
- "Undesirable Parameter Values" on page 3-196
- "Reverting to Initial Parameter Values" on page 3-198
- "Save and Load Optimization Sessions" on page 3-199
- "Improving Optimization Performance Using Parallel Computing" on page 3-200
- "Optimizing Time-Domain Response of Simulink Models Using Parallel Computing" on page 3-209
- "Design Optimization to Meet Frequency-Domain Requirements (Code)" on page 3-216
- "PID Tuning with Actuator Constraints" on page 3-222
- "PID Tuning with Reference Tracking and Plant Uncertainty" on page 3-227
- "Engine Design and Cost Tradeoffs" on page 3-232
- "Magnetic Levitation Controller Tuning" on page 3-240
- "LQG Controller Tuning" on page 3-248
- "Inverted Pendulum Controller Tuning" on page 3-252
- "Pitch Rate Controller Tuning" on page 3-256
- "Tuning of Airframe Autopilot Gains" on page 3-260
- "Distillation Controller Tuning" on page 3-264
- "Heat Exchanger Controller Tuning" on page 3-268
- "Power Converter Tuning" on page 3-272
- "Servomechanism Tuning" on page 3-276
- "Stewart Platform Controller Tuning" on page 3-280
- "Phase Lock Loop Tuning" on page 3-284
- "Surrogate Optimization in Simulink Design Optimization" on page 3-288

How the Optimization Algorithm Formulates Minimization Problems

When you optimize parameters of a Simulink model to meet design requirements, Simulink Design Optimization software automatically converts the requirements into a constrained optimization problem and then solves the problem using optimization techniques. The constrained optimization problem iteratively simulates the Simulink model, compares the results of the simulations with the constraint objectives, and uses optimization methods to adjust tuned parameters to better meet the objectives.

This topic describes how the software formulates the constrained optimization problem used by the optimization algorithms. For each optimization algorithm, the software formulates one of the following types of minimization problems:

- Feasibility on page 3-3
- Tracking on page 3-5
- Mixed feasibility and tracking

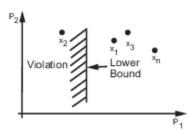
For more information on how each optimization algorithm formulates these problems, see:

- "Gradient Descent Method Problem Formulations" on page 3-6
- "Simplex Search Method Problem Formulations" on page 3-7
- "Pattern Search Method Problem Formulations" on page 3-7
- "Gradient Computations" on page 3-8

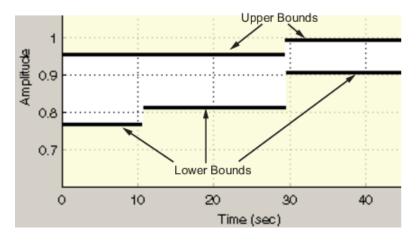
Feasibility Problem and Constraint Formulation

Feasibility means that the optimization algorithm finds parameter values that satisfy all constraints to within specified tolerances but does not minimize any objective or cost function in doing so.

In the following figure, x_1 , x_3 , and x_n represent a combination of parameter values P_1 and P_2 and are feasible solutions because they do not violate the lower bound constraint.



In a Simulink model, you constrain a signal by specifying lower and upper bounds in a Check block (Check Step Response Characteristics, ...) or a requirement object (sdo.requirements.StepResponseEnvelope, ...), as shown in the following figure.



These constraints are piecewise linear bounds. A piecewise linear bound y_{bnd} with *n* edges can be represented as:

$$y_{bnd}(t) = \begin{cases} y_1(t) & t_1 \le t \le t_2 \\ y_2(t) & t_2 \le t \le t_3 \\ \vdots & \vdots \\ y_n(t) & t_n \le t \le t_{n+1} \end{cases},$$

The software computes the signed distance between the simulated response and the edge. The signed distance for lower bounds is:

$$c = \begin{bmatrix} \max_{t_1 \le t \le t_2} y_{bnd} - y_{sim} \\ \max_{t_2 \le t \le t_3} y_{bnd} - y_{sim} \\ \max_{t_2 \le t \le t_3} y_{bnd} - y_{sim} \\ t_n \le t \le t_{n+1} \end{bmatrix}$$

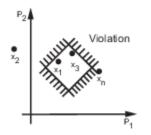
where y_{sim} is the simulated response and is a function of the parameters being optimized.

The signed distance for upper bounds is:

$$c = \begin{bmatrix} \max_{t_1 \le t \le t_2} y_{sim} - y_{bnd} \\ \max_{t_1 \le t \le t_2} y_{sim} - y_{bnd} \\ t_2 \le t \le t_3 \end{bmatrix}$$
$$\max_{t_n \le t \le t_n + 1} y_{sim} - y_{bnd}$$

At the command line, "optimFcn" supplies c directly from the Cleq field of vals.

If *all* the constraints are met ($c \le 0$) for some combination of parameter values, then that solution is said to be feasible. In the following figure, x_1 and x_3 are feasible solutions.



When your model has multiple requirements or vector signals feeding a requirement, the constraint vector is extended with the constraint violations for each signal and bound:

$$C = [c_1; c_2; \cdots; c_n].$$

Tracking Problem

In addition to lower and upper bounds, you can specify a reference signal in a Check Against Reference block or sdo.requirements.SignalTracking object, which the Simulink model output can track. The tracking objective is a sum-squared-error tracking objective.

You specify the reference signal as a sequence of time-amplitude pairs:

$$y_{ref}(t_{ref}), t_{ref} \in \left\{T_{ref0}, T_{ref1}, \cdots, T_{refN}\right\}.$$

The software computes the simulated response as a sequence of time-amplitude pairs:

$$y_{sim}(t_{sim}), t_{sim} \in \{T_{sim0}, T_{sim1}, \cdots, T_{simN}\}$$

where some values of t_{sim} may match the values of t_{ref} .

A new time base, t_{new} , is formed from the union of the elements of t_{ref} and t_{sim} . Elements that are not within the minimum-maximum range of both t_{ref} and t_{sim} are omitted:

$$t_{new} = \left\{ t : t_{sim} \cup t_{ref} \right\}$$

Using linear interpolation, the software computes the values of y_{ref} and y_{sim} at the time points in t_{new} and then computes the scaled error:

$$e(t_{new}) = \frac{\left(y_{sim}(t_{new}) - y_{ref}(t_{new})\right)}{\max_{t_{new}}|y_{ref}|}$$

Finally, the software computes the weighted, integral square error:

$$f=\int w(t)e(t)^2 dt\,.$$

Note The weight w(t) is 1 by default. You can specify a different value of weight only at the command line.

When your model has requirements or vector signals feeding a requirement, the tracking objective equals the sum of the individual tracking integral errors for each signal:

$F=\sum f_i.$

Gradient Descent Method Problem Formulations

The Gradient Descent method uses the function fmincon to optimize model parameters to meet design requirements.

Problem Type	Problem Formulation
Feasibility Problem	The software formulates the constraint $C(x)$ as described in "Feasibility Problem and Constraint Formulation" on page 3-3.
	• If you select the maximally feasible solution option (i.e., the optimization continues after an initial feasible solution is found), the software uses the following problem formulation:
	$ \begin{array}{l} \min_{[x, y]} \gamma \\ s.t. C(x) \leq \gamma \end{array} $
	$\frac{x \le x \le \overline{x}}{\overline{x}}$
	$\frac{\lambda}{\gamma} = \lambda = \lambda$ $\gamma \le 0$
	γ is a slack variable that permits a feasible solution with $C(x) \leq \gamma$ rather than $C(x) \leq 0$.
	• If you do not select the maximally feasible solution option (i.e., the optimization terminates as soon as a feasible solution is found), the software uses the following problem formulation:
	min 0 x
	$s.t. C(x) \leq 0$
	$\underline{x} \le x \le \overline{x}$
Tracking Problem	The software formulates the tracking objective $F(x)$ as described in "Tracking Problem" on page 3-5 and minimizes the tracking objective:
	$\min_{x} F(x)$
	$s.t. \ \underline{x} \le x \le \overline{x}$
Mixed Feasibility and	The software minimizes following problem formulation:
Tracking Problem	min $F(x)$
	X
	$s.t. C(x) \leq 0$
	$\underline{x} \le x \le \overline{x}$
	Note When tracking a reference signal, the software ignores the maximally feasible solution option.

Simplex Search Method Problem Formulations

The Simplex Search method uses the function fminsearch and fminbnd to optimize model parameters to meet design requirements. fminbnd is used if one scalar parameter is being optimized, otherwise fminsearch is used. You cannot use parameter bounds $\underline{x} \le x \le \overline{x}$ with fminsearch.

Problem Type	Problem Formulation
Feasibility Problem	The software formulates the constraint $C(x)$ as described in "Feasibility Problem and Constraint Formulation" on page 3-3 and then minimizes the maximum constraint violation: $\min_{x} \max(C(x))$
Tracking Problem	The software formulates the tracking objective $F(x)$ as described in "Tracking Problem" on page 3-5 and then minimizes the tracking objective: $\min_{x} F(x)$
Mixed Feasibility and Tracking Problem	The software formulates the problem in two steps: 1 Finds a feasible solution. min max($C(x)$) 2 Minimizes the tracking objective. The software uses the results from step 1 as initial guesses and maintains feasibility by introducing a discontinuous barrier in the optimization objective. min $\Gamma(x)$ where $\Gamma(x) = \begin{cases} \infty & \text{ifmax}(C(x)) > 0 \\ F(x) & \text{otherwise.} \end{cases}$

Pattern Search Method Problem Formulations

The Pattern Search method uses the function patternsearch to optimize model parameters to meet design requirements.

Problem Type	Problem Formulation
Feasibility Problem	The software formulates the constraint $C(x)$ as described in "Feasibility Problem and Constraint Formulation" on page 3-3 and then minimizes the maximum constraint violation: $\min_{x} \max(C(x))$ $s.t. \underline{x} \le x \le \overline{x}$

Problem Type	Problem Formulation				
Tracking Problem	The software formulates the tracking objective $F(x)$ as described in "Tracking Problem" on page 3-5 and then minimizes the tracking objective:				
	$\min_{x} F(x)$				
	$s.t. \underline{x} \le x \le \overline{x}$				
Mixed Feasibility and	The software formulates the problem in two steps:				
Tracking Problem	Finds a feasible solution.min max(C(x))				
	 x x ≤ x ≤ x x ≤ x ≤ x Minimizes the tracking objective. The software uses the results from step 1 as initial guesses and maintains feasibility by introducing a discontinuous barrier in the optimization objective. 				
	$\begin{split} \min_{x} \Gamma(x) \\ s.t. \ \underline{x} &\leq x \leq \overline{x} \\ \text{where} \\ \Gamma(x) &= \begin{cases} \infty & \text{ifmax}(C(x)) > 0 \\ F(x) & \text{otherwise.} \end{cases} \end{split}$				

Gradient Computations

For the Gradient descent (fmincon) optimization solver, the gradients are computed using numerical perturbation:

$$dx = \sqrt[3]{eps} \times \max\left(\left|x\right|, \frac{1}{10}x_{typical}\right)$$
$$dL = \max(x - dx, x_{\min})$$
$$dR = \min(x + dx, x_{\max})$$
$$F_L = opt_fcn(dL)$$
$$F_R = opt_fcn(dR)$$
$$\frac{dF}{dx} = \frac{(F_L - F_R)}{(dL - dR)}$$

- *x* is a scalar design variable.
- x_{min} is the lower bound of *x*.
- x_{max} is the upper bound of *x*.
- $x_{typical}$ is the scaled value of x.
- *opt_fcn* is the objective function.

dx is relatively large to accommodate simulation solver tolerances.

If you want to compute the gradients in any other way, you can do so in the cost function you write for performing design optimization programmatically. See sdo.optimize and GradFcn of sdo.OptimizeOptions for more information.

Specify Signals to Log

Design requirements require logged model signals. During optimization, the model is simulated using the current value of the design variables and the logged signal is used to evaluate the design requirements.

- 1 In the **Response Optimizer**, select **Signal** in the **New** drop-down list. A window opens where you select a signal to log.
- 2 In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set		×
Signal set: Sig		
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal
		OK Cancel Help
		.:

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- 3
- Select the signal and click 😥 to add it to the signal set.
- 4 In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

See Also

- "Design Optimization to Track Reference Signal (GUI)"
- sdo.SimulationTest

Specify Custom Requirements in the App

This topic shows how to specify custom requirements in the **Response Optimizer**.

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

- 1 In the **Response Optimizer**, in **New** drop-down menu, select **Custom Requirement**. The Create Requirement dialog box opens where you specify the custom requirement.
- 2 Specify a requirement name in Name.
- **3** Specify the requirement type in the **Type** drop-down menu.
- 4 Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click



to review or edit the function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

5 (Optional) To prevent the solver from considering specific parameter combinations, select Error if constraint is violated. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next iteration.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current iteration. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

For more information, see "Skip Model Simulation Based on Parameter Constraint Violation (GUI)" on page 3-163.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

6 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

- a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**. A graphical display of the requirement also appears in the **Response Optimizer** app window.

See Also

Related Examples

"Design Optimization to Meet a Custom Objective (GUI)" on page 3-103

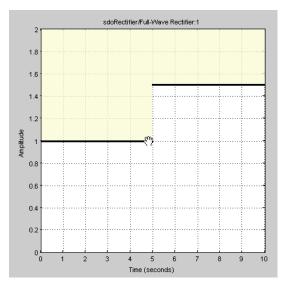
- "Design Optimization to Meet Custom Signal Requirements (GUI)" on page 3-125
- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43

Move Constraints

Constraint-bound edges define time-domain constraints you would like to place on a particular signal in your model. You can position these edges, which appear as a yellow shaded region bordered by a black line, graphically on page 3-14 or exactly on page 3-15.

Move Constraints Graphically

Use the mouse to click and drag edges in the amplitude versus time plot, as shown in the following figure.



- To move a constraint edge boundary or to change the slope of a constraint edge, position the pointer over a constraint edge endpoint, and press and hold down the left mouse button. The pointer should change to a hand symbol. While still holding the button down, drag the pointer to the target location, and release the mouse button. Note that the edges on either side of the boundary might not maintain their slopes.
- To move an entire constraint edge up, down, left, or right, position the mouse pointer over the edge and press and hold down the left mouse button. The pointer should change to a four-way arrow. While still holding the button down, drag the pointer to the target location, and release the mouse button. Note that the edges on either side of the boundary might not maintain their slopes.

To move a constraint edge to a perfectly horizontal or vertical position, hold down the **Shift** key while clicking and dragging the constraint edge. This causes the constraint edge to *snap* to a horizontal or vertical position.

When moving constraint bound edges, it is sometimes helpful to display gridlines on the axes for careful alignment of the constraint bound edges. To turn the gridlines on or off, right-click within the axes and select **Grid**.

Note You can move a lower bound constraint edge above an upper bound constraint edge, or vice versa, but this produces an error when you attempt to run the optimization.

Position Constraints Exactly

To position a constraint edge exactly:

1 Position the pointer over the edge you want to move and right-click. Select **Edit** to open the Edit Design Requirement dialog box.

4	Edit Desig	n Requirem	ent				
0	Design requirement: Lower time response bound from 0 to 10 sec						
٦	Design require	ement parame	ters				
S	egments:						
	Sta	art	Er	ıd			
	Time	Amplitude	Time	Amplitude	Slope	Weight	
	0	-0.01	1	-0.01	0	1	
Į	1	0.9	3	0.9	0	1	
Į.	3	0.99	10	0.99	0	1 💌	
					Insert	Delete	
						K Help	

2 Specify the position of each constraint edge in the **Time** and **Amplitude** columns.

See Also

More About

- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43

Specify Time-Domain Design Requirements in the App

In the **Response Optimizer**, you can specify the following time-domain requirements:

- Signal Bound "Specify Piecewise-Linear Lower and Upper Bounds" on page 3-16
- Signal Property "Specify Signal Property Requirements" on page 3-17
- Step Response Envelope "Specify Step Response Characteristics" on page 3-19
- Signal Tracking "Track Reference Signals" on page 3-21
- Ellipse Region Constraint "Impose Elliptic Bound on Phase Plane Trajectory of Two Signals" on page 3-22
- **Custom Requirement** "Specify Custom Requirements" on page 3-24

After you specify the constraints, you can see if the requirements are satisfied by optimizing the design variables. For more information, see "Specify Optimization Options" on page 3-62.

Specify Piecewise-Linear Lower and Upper Bounds

To specify upper and lower bounds on a signal:

- 1 In the **Response Optimizer**, select **Signal Bound** in the **New** drop-down list. A window opens where you specify upper or lower bounds on a signal.
- 2 Specify a requirement name in the **Name** box.
- **3** Select the requirement type using the **Type** list.
- 4 Specify the edge start and end times and corresponding amplitude in the **Time (s)** and **Amplitude** columns.
- 5 Click to specify additional bound edges.

Select a row and click to delete a bound edge.

6 In the **Select Signals to Bound** area, select a logged signal to apply the requirement to.

If you have already selected signals, as described in "Specify Signals to Log" on page 3-10, they appear in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		1
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

7 Click OK.

A variable with the specified requirement name appears in the **Data** area of the app. A graphical display of the requirement also appears in the **Response Optimizer** app window.

- 8 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can add a Check Custom Bounds block to your model to specify piecewise-linear bounds.

Specify Signal Property Requirements

To specify signal property requirements:

- 1 In the **Response Optimizer**, select **Signal Property** in the **New** drop-down list. A Create Requirement window opens where you specify signal property requirements.
- 2 In the **Name** box, specify a requirement name.
- **3** In the **Specify Property** area, specify a signal property requirement using the **Property** and **Type** lists and the **Bound** box.

Property List

Property		Time weighting available
Signal minimum	Minimum of the signal	No
Signal maximum	Maximum of the signal	No

Property	Description	Time weighting available
Signal final value	Last signal value	No
Signal mean	Average of signal value	Yes
Signal median	Middle value of signal	Yes
Signal variance	Variance of signal	Yes
Signal interquartile range	Difference between the 75th and 25th percentiles of signal values	No
Signal sum	$\sum_{i=t_0}^{t_N} S(i)$, where $S(t_0),, S(t_N)$ is the signal to constrain.	Yes
Signal sum square	$\sum_{i=t_0}^{t_N} S(i)^2$	Yes
Signal sum absolute	$\sum_{i=t_0}^{t_N} S(i) $	Yes

For signal properties where the **Time-Weighted** option is available, you can select it to weight the property computation by the time intervals between samples.

Custom Signal Property

You can add a custom signal property to the **Property** list by editing the function sdo.requirements.signalPropertyFcns.

- a At the MATLAB command prompt, enter edit sdo.requirements.signalPropertyFcns.
- **b** Add your signal property function to the FcnData cell array.

Your signal property function must be on the path.

4 In the **Select Signals to Bound** area, select the logged signal to which you want to apply the requirement.

The signal selected must have numeric type data (either floating-point or integer). Also, if the property selected is Signal median, Signal variance, or Signal interquartile range, then the signal data must be floating-point (either double or single).

If you have already selected a signal, as described in "Specify Signals to Log" on page 3-10, the signal appears in the list. Select the corresponding check box for that signal.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal	
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

5 Click OK.

A variable with the specified requirement name appears in the **Data** area of the app. An iteration plot depicting the signal property for each iteration also appears in the **Response Optimizer** app window.

Specify Step Response Characteristics

To apply a step response requirement to a signal in your model, specify the step response characteristics as follows:

1 Select a step response requirement from the **Response Optimizer**.

In the **New** drop-down menu of the app, in the **New Time Domain Requirement** section, select **Step Response Envelope**.

A Create Requirement dialog box opens where you specify the step response requirements on a signal.

- 2 Specify a requirement name in the **Name** field of the dialog box.
- **3** Specify the step response characteristics:
 - Initial value Input level before the step occurs
 - Step time Time at which the step takes place
 - Final value Input level after the step occurs
 - **Rise time** The time taken for the response signal to reach a specified percentage of the step range. The step range is the difference between the final and initial values.
 - % **Rise** The percentage of the step range used with **Rise time** to define the overall rise time characteristics.

- **Settling time** Time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
- % **Settling** The percentage of the final value that defines the settling range of settling time characteristic specified in **Settling time**.
- **% Overshoot** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **% Undershoot** The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- 4 Specify the signal to be bound.

To apply this requirement to a model signal, in the **Select Signals to Bound** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . The Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Siq			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- **d** In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

Alternatively, you can use the Check Step Response Characteristics block to specify step response bounds for a signal.

See Also

"Design Optimization to Meet Step Response Requirements (GUI)"

Track Reference Signals

Use reference tracking to force a model signal to match a desired signal. To track a reference signal:

- 1 In the **Response Optimizer**, select **Signal Tracking** in the **New** drop-down list. A window opens where you specify the reference signal to track.
- 2 Specify a requirement name in the **Name** box.
- **3** Define the reference signal by entering vectors, or variables from the workspace, in the **Time vector** and **Amplitude** fields.

Click **Update reference signal data** to use the new amplitude and time vector as the reference signal.

- 4 Specify how the optimization solver minimizes the error between the reference and model signals using the **Tracking Method** list:
 - SSE Reduces the sum of squared errors
 - SAE Reduces the sum of absolute errors
- 5 In the **Specify Signal to Track Reference Signal** area, select a logged signal to apply the requirement to.

If you already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, they appear in the list. Select the corresponding check-box.

If you have not selected a signal to log:

а

- Click _____. A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Siq			
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal	1
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- **d** In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

e Select the check-box corresponding to the signal and click **OK**.

A variable with the specified requirement name appears in the **Data** area of the app. A graphical display of the signal bound also appears in the **Response Optimizer** app window.

Note When tracking a reference signal, the software ignores the maximally feasible solution option. For more information on this option, in the **Response Optimization** tab, click **Options** > **Optimization Options**, and click **Help**.

Alternatively, you can use the Check Against Reference block to specify a reference signal to track.

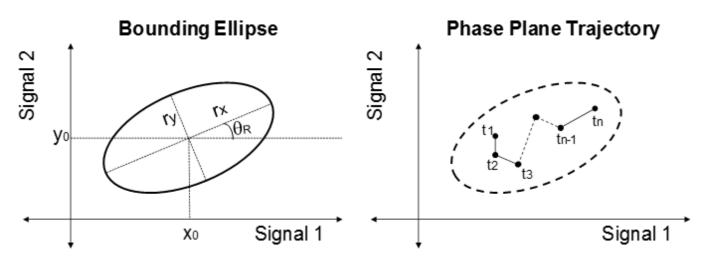
See Also

"Design Optimization to Track Reference Signal (GUI)"

Impose Elliptic Bound on Phase Plane Trajectory of Two Signals

You can impose an elliptic bound on the phase plane trajectory of two signals in your Simulink model. The phase plane trajectory is a plot of the two signals against each other. You specify the radii, center, and rotation of the bounding ellipse. You also specify whether you require the trajectory of the two signals to lie inside or outside the ellipse.

The following image shows the bounding ellipse and an example of the phase plane trajectory of two signals.



The X-Y plane is the phase plane defined by the two signals. r_x and r_y are the radii of the bounding ellipse along the x and y axes, and θ_R is the rotation of the ellipse about the center. The ellipse center is at (x_0, y_0) . In the image, the phase plane trajectory of the signals lies within the bounding ellipse for all time points t_1 to t_n .

To specify the elliptical bound requirement:

1 In the **Response Optimizer**, in **New** drop-down list, select **Ellipse Region Constraint**.



In the Create Requirement dialog box, specify the requirement.

Create Requirement	×		
Ellipse Region Constraint			
Constrain a pair of signals to an elli	ptical region in signal phase plane.		
Name: PhasePlaneEllipse			
 Specify ellipse bound 			
Signal	Semi-axis length Center		
Select Click to select signal.	1 0		
Select Click to select signal.	0.5 0		
Select Click to select signal.	0.5		
	Angle (rad): 0		
Type: Signal <= bound 👻			
Land and the second			

- 2 Specify a requirement name in Name.
- **1** Specify the two signals that you want to impose the requirement on. The signals define the X-Y plane of the bounding ellipse. To specify the signals, click the corresponding **Select** buttons.

✓ Specify ellipse bound	
Signal	Semi-axis length Center
Select/Controller/Derivative with LPF	1 r x 0 X 0
Select sdoShipSteering/Controller/Speed	0.5 ry 0 y o
	Angle (rad): 0

When you click **Select**, the Create Signal Set dialog box opens.

Create Signal Set		×
Signal set: Sig		
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal
		OK Cancel Help

In the Simulink model window, click the signal to which you want to add the requirement. The Create Signal Set dialog box updates with the name of the block and the port number where the

selected signal is located. Select the signal, and click 🔛 to add it to the signal set.

Once you have specified the logged signal in the Create Signal Set dialog box, the signal appears in the Create Requirement dialog box.

- 2 Specify the radii of the bounding ellipse as real positive finite values in **Semi-axis length**. You specify r_x and r_y that are the x-axis and y-axis radii before any rotation about the ellipse center.
- **3** Specify the location of the center of the bounding ellipse in **Center**. You specify x_0 and y_0 , the x and y coordinates of the center, as real finite values.
- 4 Specify the angle of rotation of the ellipse about its center as a real finite scalar in **Angle (rad)**.
- **5** Specify the bound **Type** as one of the following:
 - '<=' Ellipse is an upper bound. The phase plane trajectory of the two signals should lie inside or on the ellipse.
 - '>=' Ellipse is a lower bound. The phase plane trajectory of the two signals should lie outside or on the ellipse.
- 1 (Optional) To create an iteration plot that shows the evaluated requirement value for each optimization iteration, select **Create Plot**. The plot is populated when you perform optimization. During optimization, the software computes the signed minimum distance of each point in the phase plane trajectory to the bounding ellipse. The maximum of these signed distances is returned and plotted at each iteration. A positive value indicates that the requirement has been violated and at least one of the trajectory points lies outside the bounding region.
- 2 Click OK.

A new variable, with the specified requirement name, appears in the **Data** area of the **Response Optimizer**. A graphical display of the requirement also appears in the **Response Optimizer** app window.

Specify Custom Requirements

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

1 In the **Response Optimizer**, in **New** drop-down menu, select **Custom Requirement**. The Create Requirement dialog box opens where you specify the custom requirement.

- 2 Specify a requirement name in Name.
- **3** Specify the requirement type in the **Type** drop-down menu.
- 4 Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click

to review	or edit th	ne function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

5 (Optional) To prevent the solver from considering specific parameter combinations, select **Error if constraint is violated**. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next iteration.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current iteration. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

For more information, see "Skip Model Simulation Based on Parameter Constraint Violation (GUI)" on page 3-163.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

6 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			x
Signal set: Sig			
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal	
		OK Cancel Help	
			:

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**. A graphical display of the requirement also appears in the **Response Optimizer** app window.

See Also

- "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103
- "Design Optimization to Meet a Custom Objective (Code)" on page 3-118

Edit Design Requirements

The Edit Design Requirement dialog box allows you to exactly position constraint segments and to edit other properties of these constraints. The dialog box has two main components:

- An upper panel to specify the constraint you are editing
- A lower panel to edit the constraint parameters

The upper panel of the Edit Design Requirement dialog box resembles the image in the following figure.

Design requirement:	Upper time response bound from 0 to 10 sec	•
---------------------	--------------------------------------------	---

In the **Control System Designer** app in Control System Toolbox^m, you can edit design requirements from the analysis plots. The **Design requirement** drop-down list will contain all the requirements on that plot.

Edit Design Requirement Dialog Box Parameters

The particular parameters shown within the lower panel of the Edit Design Requirement dialog box depend on the type of constraint/requirement. In some cases, the lower panel contains a grid with one row for each segment and one column for each constraint parameter. The following table summarizes the various constraint parameters.

Edit Design Requirement Dialog Box Parameters

Parameter	Found in	Description
Time	Upper and lower time response bounds on step and impulse response plots	Defines the time range of a segment within a constraint/requirement.
Amplitude	Upper and lower time response bounds on step and impulse response plots	Defines the beginning and ending amplitude of a constraint segment.
Slope (1/s)	Upper and lower time response bounds	Defines the slope, in 1/s, of a constraint segment. It is an alternative method of specifying the magnitude values. Entering a new Slope value changes any previously defined magnitude values.
Final value	Step response bounds	Defines the input level after the step occurs.
Rise time	Step response bounds	Defines a constraint segment for a particular rise time.
% Rise	Step response bounds	The percentage of the step's range used to describe the rise time.
Settling time	Step response bounds	Defines a constraint segment for a particular settling time.
% Settling	Step response bounds	The percentage of the final value that defines the settling region used to describe the settling time.
% Overshoot	Step response bounds	The percentage amount by which the signal can exceed the final value before settling.
% Undershoot	Step response bounds	Defines the constraint segments for a particular percent undershoot.

See Also

Related Examples

- "Specify Design Variables" on page 3-56
- "Specify Variable Requirements in the App" on page 3-31
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43
- "Specify Optimization Options" on page 3-62
- "Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)" on page 6-59

Edit Design Requirements

The Edit Design Requirement dialog box allows you to exactly position constraint segments and to edit other properties of these constraints. The dialog box has two main components:

- An upper panel to specify the constraint you are editing
- A lower panel to edit the constraint parameters

The upper panel of the Edit Design Requirement dialog box resembles the image in the following figure.

Design requirement:	Upper time response bound from 0 to 10 sec	•	
---------------------	--------------------------------------------	---	--

In the context of the **Control System Designer** app in Control System Toolbox, **Design requirement** is associated with both the analysis plot or editor that contains the requirement and the particular requirement itself. To edit other constraints within the app, select another design requirement from the drop-down menu.

Edit Design Requirement Dialog Box Parameters

The particular parameters shown within the lower panel of the Edit Design Requirement dialog box depend on the type of constraint/requirement. In some cases, the lower panel contains a grid with one row for each segment and one column for each constraint parameter. The following table summarizes the various constraint parameters.

Parameter	Found in	Description
Time	Upper and lower time response bounds on step and impulse response plots	Defines the time range of a segment within a constraint/requirement.
Amplitude	Upper and lower time response bounds on step and impulse response plots	Defines the beginning and ending amplitude of a constraint segment.
Slope (1/s)	Upper and lower time response bounds	Defines the slope, in 1/s, of a constraint segment. It is an alternative method of specifying the magnitude values. Entering a new Slope value changes any previously defined magnitude values.
Final value	Step response bounds	Defines the input level after the step occurs.
Rise time	Step response bounds	Defines a constraint segment for a particular rise time.
% Rise	Step response bounds	The percentage of the step's range used to describe the rise time.
Settling time	Step response bounds	Defines a constraint segment for a particular settling time.
% Settling	Step response bounds	The percentage of the final value that defines the settling region used to describe the settling time.
% Overshoot	Step response bounds	The percentage amount by which the signal can exceed the final value before settling.
% Undershoot	Step response bounds	Defines the constraint segments for a particular percent undershoot.

Edit Design Requirement Dialog Box Parameters

Specify Variable Requirements in the App

In the **Response Optimizer**, you can specify the following constraints on Simulink model parameters that are specified as variables:

- Monotonic Variable "Impose Monotonic Constraint Requirement on Variable" on page 3-31
- Smoothness Constraint "Impose Upper Bound on Gradient Magnitude of Variable" on page 3-33
- Function Matching "Specify Linear or Quadratic Function Matching Constraint" on page 3-36
- Vector Property "Specify Requirement on a Vector Property" on page 3-39
- Relational Constraint "Impose Relational Constraint Between Two Variables" on page 3-41

For information about how to specify a model parameter as a variable, see "Add Model Parameters as Variables for Optimization" on page 3-56. After you specify the constraints, you can see if the requirements are satisfied by optimizing the design variables. For more information, see "Specify Optimization Options" on page 3-62.

Impose Monotonic Constraint Requirement on Variable

You can impose a monotonic constraint requirement on a design variable in your Simulink model. For example, constrain a variable to be monotonically increasing. The variable can be a vector, matrix, or multidimensional array that is a parameter in your model, such as the breakpoints of a lookup table.

To specify the requirement:

1 In the **Response Optimizer**, in **New** drop-down menu, select **Monotonic Variable**.

📣 Response Opti	A Response Optimization* - sdoShipSteering									
RESPONSE OF	TIMIZATION	VIEW							i i s	2 🗖 🕐
Open San Session ❤ Sessi	/e New Select	Evaluate	Design Variables Set: ☞ None ▼ ∠ Uncertain Variables Set: 🔀 None ▼ ∠	Sensitivity Analysis 👻	Data to Plot:	Add Plot	Plot Model Response	© Options	Optimize	
FILE		IDEMENTS	VARIABLES			PLOTS		OPTIONS	OPTIMIZE	

In the Create Requirement dialog box, specify the requirement.

Create Requirement ×					
	Monotonic Variable				
Specify a requirement that a vector, matrix or array be monotonic.					
Name: MonotonicVariable					
▼ Specify Monotonic Constraint					
Variable	Type or Select Show in Model				

- 2 Specify a requirement name in **Name**.
- **3** Specify the name of the variable in **Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single.

You can type the name of a nonscalar variable, or select the variable from the drop-down list. The list is prepopulated with all the nonscalar variables in your model. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1,:) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the **Response Optimizer**, see "Add Model Parameters as Variables for Optimization" on page 3-56.

1 Specify the monotonicity for each dimension of the variable.

After you select the variable, the dialog updates to show **Dimension 1** to **Dimension n**, corresponding to the n dimensions of the variable. For example, for a 2-dimensional variable K of size 3-by-5, the dialog updates as shown.

Smoothness Constraint Specify a smoothness requirement on a vector, matrix or array.						
Specify a smoothness requirement on a vector, matrix or array.						
Name: SmoothnessReq						
▼ Specify Smoothness Constraint						
Gradient maximum magnitude: 1						
Dependent Variable: 🔣 👻	Show in Model					
Independent Variable						
Dimension 1: Type or Select	Show in Model					
Dimension 2: Type or Select <5 element vector, or scalar> 	Show in Model					

Specify the monotonicity for the first dimension in **Dimension 1** and for the n^{th} -dimension in **Dimension n** as one of the following options:

- Strictly increasing Each element of the variable is greater than the previous element in that dimension.
- Increasing Each element of the variable is greater than or equal to the previous element in that dimension.
- **Decreasing** Each element of the variable is less than or equal to the previous element in that dimension.
- Strictly decreasing Each element of the variable is less than the previous element in that dimension.

- Not constrained No constraint exists between the elements of the variable in that dimension.
- (Optional) To create an iteration plot that shows the evaluated requirement value for each optimization iteration, select Create Plot. The plot is populated when you perform optimization. The plot shows the evaluated requirement value corresponding to each dimension of the variable. A positive value indicates that the requirement has been violated.
- 2 Click OK.

A new variable, with the specified requirement name, appears in the **Data** area of the **Response Optimizer** app.

Impose Upper Bound on Gradient Magnitude of Variable

You can impose an upper bound on the gradient magnitude of a variable in your Simulink model. The variable can be a vector, matrix, or multidimensional array that is a parameter in your model, such as the data of a lookup table. For example, consider a car engine controller whose gain changes under different operating conditions determined by the car speed. You can use a gradient bound constraint to limit the rate at which the controller gain changes per unit change in vehicle speed.

For an N-dimensional variable F that is a function of independent variables $x_1, ..., x_N$, the gradient magnitude is defined as:

$$|\nabla F| = \sqrt{\left(\frac{\partial F}{\partial x_1}\right)^2 + \left(\frac{\partial F}{\partial x_2}\right)^2 + \dots + \left(\frac{\partial F}{\partial x_N}\right)^2}$$

To compute the gradient magnitude, the software computes the partial derivative in each dimension by computing the difference between successive F data in that dimension and dividing by the spacing between the data in that dimension. You specify F and the spacing between the data. The software checks whether the gradient magnitude of the variable data is less than or equal to a specified bound. If the gradient magnitude of the data is greater than the required bound, the variable data is not smooth.

To specify the requirement:

1 In the **Response Optimizer**, in **New** drop-down list, select **Smoothness Constraint**.



In the Create Requirement dialog box, specify the requirement.

Create Requirement > Smoothness Constraint					
Specify a smoothness requirement on a vector, matrix or array.					
specify a smoothless requirement on a rector, matrix of analy					
Name: SmoothnessReg					
 Specify Smoothness Constraint 					
Gradient maximum magnitude: 1					
Dependent Variable: Type or Select 🔹	Show in Model				

- 2 Specify a requirement name in Name.
- **3** Specify the gradient magnitude bound as a nonnegative finite real scalar in **Gradient maximum magnitude**.
- 4 Specify the variable F that you want to impose the requirement on in **Dependent Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single. The variable must be a parameter in your model or a constant that you enter.

You can type the name of a nonscalar variable or constant, or select the variable from the dropdown list. The list is prepopulated with all the nonscalar variables in your model. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1,:) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the **Response Optimizer**, see "Add Model Parameters as Variables for Optimization" on page 3-56.

1 Specify the spacing between points of **Dependent Variable** data in each dimension in **Independent Variable**.

After you select the **Dependent Variable**, the dialog updates to show **Dimension 1** to **Dimension n**, corresponding to the *n* dimensions of the dependent variable. For example, for a 1-dimensional variable K, the dialog updates as shown.

Create Requirement	×						
Smoothness Constraint							
Specify a smoothness requirement on a vector, matrix or array.							
Name: SmoothnessReq							
▼ Specify Smoothness Constraint							
Gradient maximum magnitude: 1							
Dependent Variable: 🔣 👻	Show in Model						
Independent Variable							
Dimension 1: Type or Select <4 element vector, or scalar> 	Show in Model						

The first dimension specifies the spacing going down the dependent variable data rows, and the second specifies spacing across the columns. The *N*th dimension specifies the spacing along the *N*th dimension of dependent variable data. You can specify the independent variables in each dimension as scalars or vectors.

- Scalars Specify the spacing between dependent variable data F in the corresponding dimension as a nonzero scalar. For example, suppose that **Dependent Variable** is two-dimensional, and the spacing between data in the first dimension is 5 and in the second dimension is 2. In the **Independent Variable** section, specify **Dimension 1** as 5 and **Dimension 2** as 2.
- Vectors Specify the coordinates of F data in the corresponding dimension as real, numeric, monotonic vectors. The software uses the coordinates to compute the spacing between the dependent variable data points in the corresponding dimension. The length of the vector must match the length of F in the corresponding dimension. You do not have to specify coordinates with uniform spacing. For example, suppose that F is two-dimensional, and the length of the data in the first and second dimension is 3 and 5, respectively. The coordinates of the data in the first dimension are [1 2 3]. In the second dimension, the spacing is not uniform and the coordinates of the data are [1 2 10 20 30]. In the Independent Variable section, specify Dimension 1 as [1 2 3] and Dimension 2 as [1 2 10 20 30].

You can also specify the independent variables by typing the name of a variable, or selecting a variable from the drop-down list. The list is prepopulated with all the variables in your model that have the appropriate size. To choose a subset of an array or matrix variable V, type an expression. For example, specify as V(1, :) to use the first row of the variable. To use a numeric field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

- (Optional) To create an iteration plot that shows the evaluated requirement value for each optimization iteration, select Create Plot. The plot is populated when you perform optimization. A positive value indicates that the requirement has been violated.
- 2 Click OK.

A new variable, with the specified requirement name, appears in the **Data** area of the **Response Optimizer**.

Specify Linear or Quadratic Function Matching Constraint

In the app, you can constrain a variable's values to match a linear or quadratic function. The variable can be a vector, matrix, or a multidimensional array that is a parameter in your model, such as the data of a lookup table in your model. To specify the requirement:

1 In **Response Optimizer**, from the **New** drop-down list, select **Function Matching**.

RESPONSE OF	TIMIZATION	VIEW			AH I I I I			i i s	¢ 🗗
Open Sav	re M Select		Design Variables Set: 😰 None 👻 🖉	Sensitivity Analysis 👻		Id Plot Model	© Options	Optimize	
ession - Sessi FILE	,		VARIABLES	Analysis •	PLOT	 Response 	OPTIONS	OPTIMIZE	

In the Create Requirement dialog box, specify the requirement. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Requirement	×
	Function Matching
S	pecify that values must match a designated function.
Name: FunctionMatc	hing
 Specify Function M 	_
Functional Relation:	Linear ~
Dependent Variable:	K ~ Show in Model
Independent Variable	
Dimension 1:	bpK ~ <4 element vector> Show in Model
_	
Center and scale i	independent variables
 Center and Scale S 	Settings
Use automatic ce	nters and scales
O Use custom cente	ers and scales
Independent Variable	Center Scale
	Dimension 1: 12 2.5820
Create Plot	OK Cancel Help

- 2 Specify the function to be matched. To do so, set **Functional Relation** to one of the following values:
 - Linear Data from variable V are fit to a linear function. For example, for a two-dimensional variable with independent variables, X_1 and X_2 , the linear function has the form:

 $V = a_0 + a_1 X_1 + a_2 X_2$

The software calculates the fit coefficients a_0 , a_1 , and a_2 and then calculates the sum of squares of the error between the data and the linear function.

• Quadratic with no cross-terms — Data are fit to a quadratic function with no cross-terms. For a two-dimensional variable, the pure quadratic function has the form:

 $V = a_0 + a_1 X_1 + a_2 X_1^2 + a_3 X_2 + a_4 X_2^2$

• Quadratic with all cross-terms — Variable data are fit to a quadratic function that includes cross-terms. For a two-dimensional variable, the quadratic function has the form:

 $V = a_0 + a_1 X_1 + a_2 X_1^2 + a_3 X_2 + a_4 X_2^2 + a_5 X_1 X_2$

If the variable is one-dimensional, there are no cross-terms and so the computation is the same as when **Functional relation** is Quadratic with no cross-terms.

3 Specify the variable V to which you want to apply the requirement in **Dependent Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single that is a parameter in your model.

Type the name of a nonscalar variable, or select a variable from the drop-down list. The list is prepopulated with all the nonscalar variables in your model. To see where the selected variable is used on your model, click **Show in Model**. To choose a subset of an array or matrix variable A, type an expression. For example, specify A(1, :) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as design variables in the app, see "Add Model Parameters as Variables for Optimization" on page 3-56.

4 Specify the independent variable vectors used for computing the function in **Independent Variable**. The independent variables are specified as real, numeric, monotonic vectors.

The number of independent variables must equal the number of dimensions of the dependent variable V. For example, you specify two independent variables when V is a matrix, and use three independent variables when V is three-dimensional. The first independent variable vector specifies coordinates going down the rows of V, and the second independent variable vector specifies coordinates going across the columns of V. The n^{th} independent variable vector specifies coordinates along the n^{th} dimension of V. The number of elements in each independent variable vector such a specifies of V in the corresponding dimension. The independent variable vectors must be monotonically increasing or decreasing.

You can also specify the independent variables by typing the name of a variable, or selecting a variable from the drop-down list. The list is prepopulated with all the variables in your model that have the appropriate size. To choose a subset of an array or matrix variable A, type an expression. For example, specify A(1,:) to use the first row of the variable. To use a numeric field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b. To use an equally spaced vector, select $[1 \ 2 \ \dots N]$ from the drop-down menu.

5 Specify whether you want to center and scale the independent variables. When you select the Center and scale independent variables option, the independent variable vectors you specify are divided by a scale value after subtracting a center value. Centering can improve numerical conditioning when one or more independent variable vectors have a mean that differs from 0 by several orders of magnitude. Scaling can improve numerical conditioning when independent variable vectors of magnitude.

To specify the center and scale values for each independent variable, expand the **Center and Scale Settings** section, and select one of the following:

• Use automatic centers and scales - The center and scale values are the mean and standard deviation for each independent variable. Using the mean and standard deviation values to center and scale the independent variables is the default option.

- Use custom centers and scales Specify the Center and Scale values for each independent variable. The independent variable vectors are divided by the corresponding Scale value after subtracting the value you specify in Center.
- 6 (Optional) Select the Create Plot option to create an iteration plot that shows the evaluated requirement value for each optimization iteration. The software computes an error signal that is the difference between the dependent variable data and the specified function of the independent variables. The sum of squares of this error is plotted when you perform optimization. A positive value indicates that the requirement has been violated, and 0 value indicates that the requirement is satisfied. The closer the value is to 0, the better the match between the function and dependent variable data.
- 7 Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

Specify Requirement on a Vector Property

You can specify a requirement on a vector property, such as the mean value of the vector. The vector must be a parameter in your model. To specify the requirement:

1 In the **Response Optimizer**, in **New** drop-down list, select **Vector Property**.

A Response Optimization* - sdoShipSteering										
RESPONSE OPTIMIZ	ATION	VIEW			S#15	\times		84	i i s	e 🗗 ?
Open Save Session Session	New V		Design Variables Set: ☞ None ▼ ∠ Uncertain Variables Set: 🔀 None ▼ ∠	Sensitivity Analysis ▼	Data to Plot:	Add Plot	Plot Model Response	© Options	Optimize	
FILE	REOU	IDEMENITS	VARIABLES			PLOTS		OPTIONS	OPTIMIZE	

Create Requi	irement ×	١					
	Vector Property						
	Specify a requirement on a vector property such as the mean value.						
Name: VectorProperty							
	Vector roberty						
▼ Specify Vector Property							
Property:	Vector mean 👻						
Туре:	Constrain property to be <= the bound 💌						
Bound:	0						
Variable:	Type or Select Show in Model						
-							

In the Create Requirement dialog box, specify the requirement.

- 2 Specify a requirement name in **Name**.
- **1** Specify the vector property in **Property**. For a vector *V* with *N* elements, you can specify one of the following properties:
 - Vector mean mean(V)
 - Vector median *median(V)*
 - Vector variance variance(V)
 - Vector inter-quartile range Difference between the 75th and 25th percentiles of the vector values.

Vector sum
$$-\sum_{i=1}^{N} V(i)$$

Vector sum of squares $-\sum_{i=1}^{N} V(i)^2$

Vector sum of absolute values
$$-\sum_{i=1}^{N} |V(i)|$$

- Vector minimum min(V)
- Vector maximum max(V)
- 2 Specify the type of requirement you want to impose on the vector property in Type. You can set an upper or lower bound on the vector property, or require the property to equal a particular value. You can also choose to maximize or minimize the vector property. For example, to maximize the mean value of your vector, specify **Property** as Vector mean and Type as Maximize the property.
- 3 Specify the value of the bound imposed on the vector property in **Bound**. Specify the bound as a finite real scalar value. For example, if for a vector variable V you require mean(V) = 5, specify **Property** as Vector mean, **Type** as Constrain property to be == the bound, and **Bound** as 5.
- 1 Specify the name of the variable in **Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single.

You can type the name of a nonscalar variable, or select the variable from the drop-down list. The list is prepopulated with all the nonscalar variables in your model. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1, :) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the **Response Optimizer**, see "Add Model Parameters as Variables for Optimization" on page 3-56.

- 2 (Optional) To create an iteration plot that shows the evaluated requirement value for each optimization iteration, select Create Plot. The plot is populated when you perform optimization. A positive value indicates that the requirement has been violated.
- 3 Click OK.

A new variable, with the specified requirement name, appears in the **Data** area of the **Response Optimizer** app.

Impose Relational Constraint Between Two Variables

You can impose a relational constraint requirement on a pair of variables in your Simulink model. For example, require that variable *a* is always greater than variable *b*. To specify the requirement:

1 In the **Response Optimizer**, in **New** drop-down list, select **Relational Constraint**.

RESPONSE OPTIMIZ	ATION VIEV					84.	h É S	₫ 🗗 (
Open Save Gession - Session -	New Version Evalu	te	Sensitivity Analysis 👻	Data to Plot:	ot Plot Model Response	Options	Optimize	
FILE	RECHIDEMENTS	VARIABLES		PLOTS		OPTIONS	OPTIMIZE	

In the Create Requirement dialog box, specify the requirement.

Create Requirement	Create Requirement X					
Relational Constraint						
Specify a requirement that two variables obey a relational constraint.						
Name: RelationalReg						
▼ Specify Relational Constraint						
Variable	Relationship	Variable				
Type or Select 🛛 👻	< 🗸	Type or Select 🛛 👻				
Show in Model		Show in Model				

- 2 Specify a requirement name in Name.
- **3** Specify the name of the two variables in **Variable**. The variables can be vectors or arrays but must be the same size.

Type the names of two variables, or select the variables from the drop-down lists. The lists are prepopulated with all the variables in your model. To see where a selected variable is used on your model, click **Show in Model**. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1, :) to use the first row of the variable. To use a numeric field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the **Response Optimizer**, see "Add Model Parameters as Variables for Optimization" on page 3-56.

1 Specify the relation between the elements of the two variables as one of the following in **Relationship**:

- '<' Each data element in the first variable is less than the corresponding element in the second variable.
- '<=' Each data element in the first variable is less than or equal to the corresponding element in the second variable.
- '>' Each data element in the first variable is greater than the corresponding element in the second variable.
- '>=' Each data element in the first variable is greater than or equal to the corresponding element in the second variable.
- '==' Each data element in the first variable is equal to the corresponding element in the second variable.
- '~=' Each data element in the first variable is not equal to the corresponding element in the second variable.
- (Optional) To create an iteration plot that shows the evaluated requirement value for each optimization iteration, select **Create Plot**. The plot is populated when you perform optimization. The plot shows the evaluated requirement value corresponding to each element of the variables. The interpretation of the evaluated requirement value depends on the requirement **Type**.

Туре	Evaluated Requirement Value					
	Requirement is Satisfied	Requirement is Violated				
'>' or '<'	Negative number	Positive number, or 0 if the elements are equal				
'>=' or '<='	Negative number, or 0 if the elements are equal	Positive number				
'=='	0	Non-zero number				
'~='	0	1				

2 Click OK.

A new variable, with the specified requirement name, appears in the **Data** area of the **Response Optimizer** app.

See Also

Related Examples

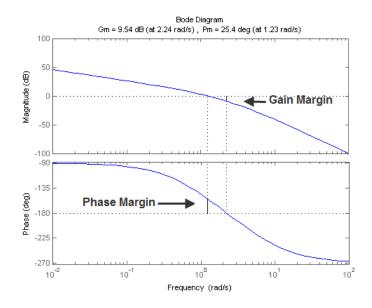
- "Specify Design Variables" on page 3-56
- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43
- "Specify Optimization Options" on page 3-62
- "Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)" on page 6-59

Specify Frequency-Domain Design Requirements in the App

Specify Lower Bounds on Gain and Phase Margin

To specify lower bounds on the gain and phase margin of a linear system:

- 1 In the **Response Optimizer**, select **Gain and Phase Margin** in the **New** list. A window opens where you specify lower bounds on the gain and phase margin of your linear system.
- 2 Specify a requirement name in Name.
- **3** Specify bounds on the gain margin or phase margin, or both.



- **Gain margin** Amount of gain increase or decrease required to make the loop gain unity at the frequency where the phase angle is -180°.
- **Phase margin** Amount of phase increase or decrease required to make the phase angle 180° when the loop gain is 1.0

To specify a lower bound on the gain margin or phase margin, or both, select the corresponding check box and enter the lower bound value.

4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- a Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A variable with the specified requirement name appears in the **Data** area of the app. A graphical display of the requirement also appears in the **Response Optimizer** app window.

- **6** (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Gain and Phase Margins block to specify bounds on the gain and phase margin. (Requires Simulink Control Design.)

Specify Piecewise-Linear Lower and Upper Bounds on Frequency Response

To specify upper or lower bounds on the magnitude of a system response:

- 1 In the **Response Optimizer**, select **Bode Magnitude** in the **New** list. A window opens where you specify the lower or upper bounds on the magnitude of the system response.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify the requirement type using the **Type** list.
- 4 Specify the edge start and end frequencies and corresponding magnitude in the **Frequency** and **Magnitude** columns.
- **5** Insert or delete bound edges.

Click to specify additional bound edges.

Select a row and click **to** delete a bound edge.

6 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

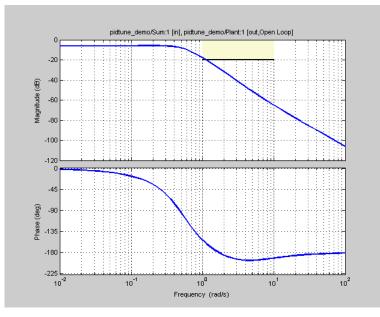
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app window. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 8 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Bode Characteristics block to specify bounds on the magnitude of the system response. (Requires Simulink Control Design.)

Specify Bound on Closed-Loop Peak Gain

To specify an upper bound on the closed-loop peak response of a system:

- 1 In the **Response Optimizer**, select **Closed-Loop Peak Gain** in the **New** list. A window opens where you specify an upper bound on the closed-loop peak gain of the system.
- 2 Specify a requirement name in the Name box.
- 3 Specify the upper bound on the closed-loop peak gain in the **Closed-Loop peak gain** box.
- 4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

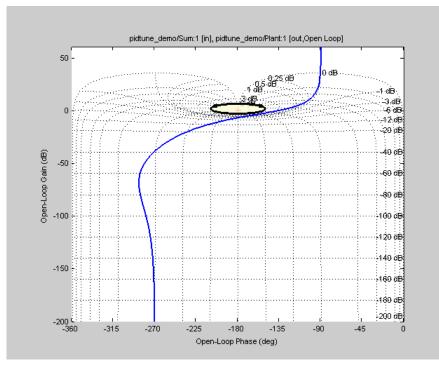
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app window. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- **6** (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Nichols Characteristics block to specify bounds on the magnitude of the system response. (Requires Simulink Control Design.)

Specify Lower Bound on Damping Ratio

To specify a lower bound on the damping ratio of the system:

- 1 In the **Response Optimizer**, select **Damping Ratio** in the **New** list. A window opens where you specify a lower bound on the damping ratio of the system.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify the lower bound on the damping ratio in the **Damping ratio** box.
- 4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

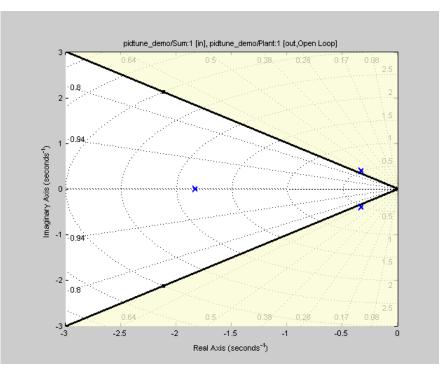
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 6 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Pole-Zero Characteristics block to specify a bound on the damping ratio. (Requires Simulink Control Design.)

Specify Upper and Lower Bounds on Natural Frequency

To specify a bound on the natural frequency of the system:

- 1 In the **Response Optimizer**, select **Natural Frequency** in the **New** list. A window opens where you specify a bound on the natural frequency of the system.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify a lower or upper bound on the natural frequency in the **Natural frequency** box.
- **4** In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

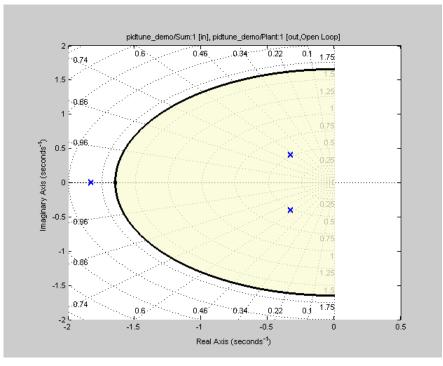
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 6 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Pole-Zero Characteristics block to specify a bound on the natural frequency. (Requires Simulink Control Design.)

Specify Upper Bound on Approximate Settling Time

To specify an upper bound on the approximate settling time of the system:

- 1 In the **Response Optimizer**, select **Settling Time** in the **New** list. A window opens where you specify an upper bound on the approximate settling time of the system.
- 2 Specify a requirement name in the Name box.

- 3 Specify the upper bound on the approximate settling time in the **Settling time** box.
- 4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- a Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

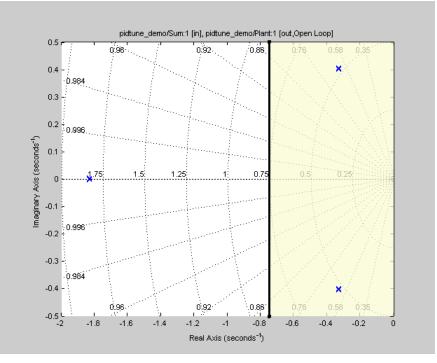
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



6 (Optional) In the graphical display, you can:

- "Move Constraints Graphically" on page 3-14
- "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Pole-Zero Characteristics block to specify the approximate settling time. (Requires Simulink Control Design.)

Specify Piecewise-Linear Upper and Lower Bounds on Singular Values

To specify piecewise-linear upper and lower bounds on the singular values of a system:

- 1 In the **Response Optimizer**, select **Singular Values** in the **New** list. A window opens where you specify the lower or upper bounds on the singular values of the system.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify the requirement type using the **Type** list.
- 4 Specify the edge start and end frequencies and corresponding magnitude in the **Frequency** and **Magnitude** columns, respectively.
- **5** Insert or delete bound edges.

Click to specify additional bound edges.

Select a row and click to delete a bound edge.

6 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

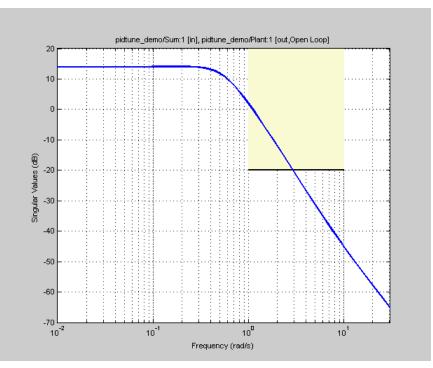
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 8 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Singular Value Characteristics block to specify bounds on the singular value. (Requires Simulink Control Design).

Specify Step Response Characteristics

To apply a step response requirement to a linearization of your model (requires Simulink Control Design), specify the step response characteristics as follows:

1 Select a step response requirement from the **Response Optimizer**.

In the **New** drop-down menu of the app, in the **New Frequency Domain Requirement** section, select **Step Response Envelope**.

A Create Requirement dialog box opens where you specify the step response requirements.

- 2 Specify a requirement name in the **Name** field of the dialog box.
- **3** Specify the step response characteristics:
 - Initial value Input level before the step occurs
 - Step time Time at which the step takes place
 - Final value Input level after the step occurs
 - **Rise time** The time taken for the response signal to reach a specified percentage of the step range. The step range is the difference between the final and initial values.

- % **Rise** The percentage of the step range used with **Rise time** to define the overall rise time characteristics.
- **Settling time** Time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
- % **Settling** The percentage of the final value that defines the settling range of settling time characteristic specified in **Settling time**.
- **% Overshoot** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **% Undershoot** The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **4** Specify the systems to be bound.

To apply this requirement to a linearization of your Simulink model:

- **a** In the **Select Systems to Bound** area, specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

Alternatively, you can use the Check Step Response Characteristics block to specify step response bounds for a signal.

Specify Custom Requirements

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

- 1 In the **Response Optimizer**, in **New** drop-down menu, select **Custom Requirement**. The Create Requirement dialog box opens where you specify the custom requirement.
- 2 Specify a requirement name in Name.
- **3** Specify the requirement type in the **Type** drop-down menu.
- 4 Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click

to review or edit the function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

5 (Optional) To prevent the solver from considering specific parameter combinations, select **Error if constraint is violated**. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next iteration.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current iteration. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

For more information, see "Skip Model Simulation Based on Parameter Constraint Violation (GUI)" on page 3-163.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

6 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			х
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		Ĩ
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**. A graphical display of the requirement also appears in the **Response Optimizer** app window.

See Also

- "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103
- "Design Optimization to Meet a Custom Objective (Code)" on page 3-118

See Also

Related Examples

- "Specify Design Variables" on page 3-56
- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Variable Requirements in the App" on page 3-31
- "Specify Optimization Options" on page 3-62

Specify Design Variables

This topic shows how to specify design variables for optimization.

Before running the optimization, you must specify the model parameters to optimize. These parameters form the design variables set for optimization. By tuning these parameters, Simulink Design Optimization software attempts to make the signals meet the requirements. Simulink Design Optimization software optimizes the response signals of the model by varying the tuned parameters so that the response signals lie within the constraint bound segments or closely match a specified reference signal. The design variables can be scalar, vector, matrix, or an expression that evaluates to one of these values.

You can also use sensitivity analysis for finding the parameters that most influence the optimization problem and use these as design variables. To open the **Sensitivity Analyzer**, in the **Response**

Optimization tab, click **EulSensitivity Analysis**. In the **Sensitivity Analyzer** app, you can explore the response optimization design space by altering the design variables, identify the parameters that most influence the optimization problem, and compute initial values.

Add Model Parameters as Variables for Optimization

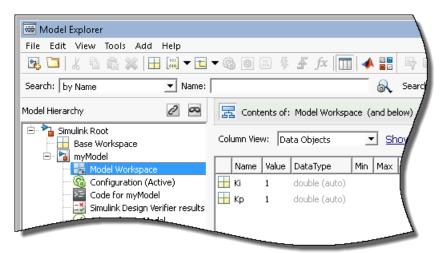
The software can only optimize variables that are in use by the Simulink model. Create variables for optimization in the MATLAB or model workspace, and specify your model or block parameters using these variables.

In this figure, the **Proportional (P)** and **Integral (I)** gain parameters of a PID Controller block are specified as numerical values.

The Function Block Parameters: PID Controller
PID Controller
This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune' button (requires Simulink Control Design).
Controller: PI
Time domain:
Continuous-time
O Discrete-time
Main PID Advanced Data Types State Attributes
Controller parameters
Source: internal
Proportional (P): 1.599340
Integral (I): 0.079967 $P + I \frac{1}{s}$
Tune

To optimize the gain parameters, specify them as variables Kp and Ki:

- **1** Create the variables Kp and Ki in one of the following ways:
 - Add the variables to the model workspace, and specify initial values.



- Write initialization code in the **PreloadFcn** callback of the model. For more information, see "Model Callbacks".
 - Kp = 1; Ki = 1;

erete time

2 Specify the gain parameters as the variables Kp and Ki in the PID Controller block dialog box.

Source:	internal 🔹	Compensator formula
Proportional (P):	Кр	
Integral (I):	Ki	$P+I\frac{1}{s}$
	Tune	5

You can now select Kp and Ki for optimization. See, "Specify Design Variables" on page 3-58.

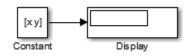
Specify Independent Parameters for Optimization

You can also specify independent parameters that do not appear explicitly in the model as variables for optimization. However, you cannot use this workflow with Simulink fast restart.

Suppose that a model parameter Kint is related to independent parameters x and y such that Kint = x+y. To optimize x and y instead of Kint:

• Create the independent variables x and y by adding them to the model workspace and specifying initial values.

• The software only allows tuning of variables that are used by model blocks. To ensure that the software detects x and y for tuning, add a Constant block to your model, and specify the **Constant value** of the block as [x y]. Connect the block to a Display block.



• Write code in the **InitFcn** callback of the model that defines the relationship between Kint, x, and y. You must first use the get_param function to get the variables x and y from the model workspace before you can use them to define Kint.

```
wks = get_param(gcs, 'ModelWorkspace')
x = evalin(wks, 'x')
y = evalin(wks, 'y')
Kint = x+y;
```

You can now select x and y for optimization. Do not optimize the independent and dependent parameters simultaneously. Doing so can lead to incorrect results. For example, do not optimize Kint, x and y together.

Specify Design Variables

To specify the parameters to be tuned using the **Response Optimizer**:

1 In the **Design Variables Set** list, select **New**.

A window opens where you specify design variables. All parameters in use by the model are displayed in this window.

2 Select one or more parameter names and click



to add the selected parameters to a design variables set.

Note You can add the same parameter to multiple design variable sets.

3 (Optional) Specify design variable settings.

Setting	Description	Default
Variable	The name of the parameter.	Not an editable field
Value	modified during optimization.	Current value of the parameter in the model. If you edit this column, click Update model variable values to update the values in the model.
Minimum	The minimum value or lower bound for the parameter. You can edit this field to provide an alternate minimum value.	-Inf

Setting	Description	Default
Maximum	The maximum value or upper bound for the parameter. You can edit this field to provide an alternate maximum value.	Inf
		Next power of 2 greater than the current value of the parameter

The check-box indicates whether the parameter is selected as a design variable in the set. Select it if you want this parameter to be tuned during the optimization. Deselect if you do not want this parameter to be tuned during the optimization but you would like to keep it on the list of tuned parameters (for a subsequent optimization).

Expand Variable Detail to see the block in the model that contains this parameter.

4 Click **OK** to create a design variable set.

If your model contains referenced models, you can select the referenced variables from the **Create Design Variables Set** dialog box. For example, the first variable in the dialog box, Slew, is listed as sdoRateLimitedController:Slew.sdoRateLimitedController is the name of the referenced model with the variable Slew. The Slew variable has the same value for all instances of the sdoRateLimitedController model. In contrast, the variable Kd can have a different value for each instance of the referenced model containing it. For example, the second variable in the dialog box is listed as sdoMultipleMotors/Control_1:Kd. The upper-level model sdoMultipleMotors has block Control_1, which is a referenced model that has variable Kd. The value of this variable can be different than Kd in block Control_2, which is the third variable in the dialog box. To enable instance-specific values, Kd is specified as a model argument in the referenced model workspace.

pritalables sets Designifies, 1			
Value Value 55	nimum Malmum Scale		Tanialda
		_	stoket initiation of a few
		100	statute Manufacture (184
Add result satisfies to	radicula.	1.961	skilling@Miter/Control 244
		-	winMultipleMeter/Control_142
			stoMatphMeters/Cortesl.240
			stolking&Mean/Cornel,14p

See Also

Related Examples

- "Optimize Parameters for Robustness (GUI)" on page 3-172
- "Update Model with Design Variables Set" on page 3-60
- "Save Design Variable Values for Specific Iteration" on page 3-74

Update Model with Design Variables Set

This example shows how to update a model with a set of design variables.

Open the Simulink model and load the pre-configured **Response Optimizer** session.

load('pidtune_demo_sdosession_update_dv.mat')
sdotool(SDOSessionData)

The **Response Optimizer** opens and loads the preconfigured session. In the **Data** area, **DesignVars1** is a set of tuned design variables.

In the **Design Variables Set** list, select the design variable set, DesignVars1.

A Response Optimization - pidtune_demo - Time plot 2								
RESPONSE OPTIMIZ	ATION	TIME PLO	т	VIEW			Al.	
	🔤 New 👻	Me	D	esign Variables Se	🗑 DesignV	ars 🔻 🖊	E Da	ata to Plot:
Open Save	Select	Evaluate	- Unc	ertain Variables Se		ESIGN VARIA	BLES SETS	lo data selected
Session - Session -	_	Requirement	sono		😰 Desigi	nVars		
FILE Data Browser	REQUI		Tim	e plot 1 🛛 🗐 Tir	🗑 Design	nVars1	J.	
Search workspace varial	bles	<u>ہ</u> - م		is no data for pidtur	CREATE NE	W SET OF DE	SIGN VARIABLE	ES ponse" or ru
✓ MATLAB Workspace					New	o specify varia	ables used	utput Const
Name 🔺	Value			1.2	by the	Simulink mo	del to optimize	2
Kd	0	A						
Kinana								

Open the Edit dialog box.



Select the variables you want to update in the model.

For this example, select Kd, Ki, and Kp.

Edit: Desi	gnVars1					×
	Variable	Value	Minimum	Maximum	Scale	
	Kd	0.3624295	-Inf	Inf		+
	Ki	0.1170876	-Inf	Inf		
	Кр	0.2855104	-Inf	Inf	0.1	
	Variable Detail					
						?

Click Update model variables.

Plot the model response.

In the **Response Optimization** tab, click **Plot Current Response**.

See Also

Related Examples

• "Save Design Variable Values for Specific Iteration" on page 3-74

Specify Optimization Options

This topic shows how to specify optimization options in the **Response Optimizer**, after you have configured the design variables and design requirements.

After you have configured the design variables and design requirements, specify the following optimization options:

1 Optimization progress and result options for optimization task

To specify these options, in the **Response Optimization** tab, click **Options** to open the Response Optimization Options dialog box. In the **General Options** tab, specify the optimization progress and result options. For details about the options, click the **Help** button.

Response Optimization Options	×
General Options Optimization Options Parallel Options Linearization Options	
Progress Options	
Show optimization progress window during optimization	
✓ Update plots during optimization	
Result Options	
Update model at end of optimization	
Save optimization termination information	
Save requirement values at the end of optimization.	
Design variable sets:	
Overwrite optimized variable values in design variable set	
Save optimized variable values as new design variable set	
OK Cancel Help	

2 Optimization method and termination options

Specify these options in the **Optimization Options** tab of the Response Optimization Options dialog box.

3 Parallel computing options

Specify these options in the **Parallel Options** tab of the Response Optimization Options dialog box. For details about the options, see "Optimize Design Using Parallel Computing (GUI)" on page 3-184.

4 Linearization options

Specify these options in the **Linearization Options** tab of the Response Optimization Options dialog box.

See Also

Related Examples

- "Specify Design Variables" on page 3-56
- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43
- "Specify Custom Requirements in the App" on page 3-11

More About

• "How the Optimization Algorithm Formulates Minimization Problems" on page 3-3

Create Linearization I/O Sets

This example shows how to create a linearization input/output set in the **Response Optimizer** or **Sensitivity Analyzer**.

To create a linearization I/O set:

- **1** Open the Create Linearization I/O Set dialog box using one of the following methods:
 - In a requirement dialog box, in the **Select Systems to Bound** section, click
 - In the **Response Optimizer**, in the **New** drop-down list, select **Linearization I/Os**.
- 1 In your Simulink model, select one or more signals that you want to define as analysis points.

The selected signals appear in the Create linearization I/O set dialog box under **Currently** selected signals.

Create linearization I/O set	
Variable name: 10s1	1
Filter by name 👂 🔻	1
Currently selected signals	
└── ⁺ = magball/Controller : 1	

2 Under **Currently selected signals**, click the signal you want to add. To select multiple signals, hold **Ctrl** and click each signal you want to add.

To add a signal from within a bus signal, expand the bus and select the signal. For example, select the data signal within the COUNTERBUS signal.

Create linearization I/O set	
Variable name: IOs1	
Filter by name 🔎 🗸	
Currently selected signals Currently selected signals /COUNTERBUSCreator : 1(COUNTER data limits	

3 To add the signal to list of **Analysis I/Os**, click **Add**.

Create linearization I/O set					×
Variable name: IOs1					
Filter by name	+ م		Analysis I/	Os	
Currently selected signals			Active	Signal Description	Configuration
magball/Controller : 1				Block: magball/Controller Port: 1 Signal Name:	±ঠ- Input Perturbation 💽
		Add			

- 4 In the **Configuration** drop-down list for the signal, select the type of analysis point you want to define:
 - **<u>+</u>** Input Perturbation Specifies an additive input to a signal.
 - **Utput Measurement** Takes a measurement at a signal.
 - Loop Break Specifies a loop opening.

 - J * Open-Loop Output Specifies an output measurement followed by a loop break.
 - Loop Transfer Specifies an output measurement before a loop break followed by an input perturbation.
 - **Sensitivity** Specifies an input perturbation followed by an output measurement.
 - Complementary Sensitivity Specifies an output measurement followed by an input perturbation.

For more information on the different types of analysis points, see "Specify Portion of Model to Linearize" (Simulink Control Design).

5 Repeat steps 1–4 for any other signals you want to define as analysis points.

Tip To highlight the source block of an analysis point in the Simulink model, in the **Analysis I/Os** list, select the analysis point, and click **Highlight**.

- 6 In the Variable name box, enter a name for the I/O set.
- 7 Click OK.

See Also

Related Examples

• "Design Optimization to Meet Frequency-Domain Requirements (GUI)" on page 3-129

More About

• "What Is Linearization?" (Simulink Control Design)

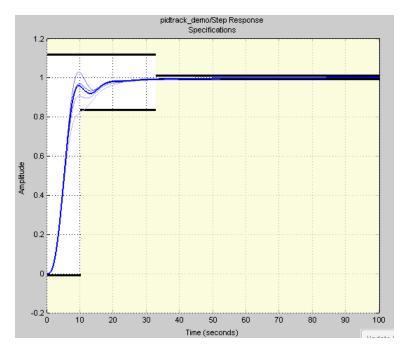
Interact with Plots

This topic shows how to interact with plots in the **Response Optimizer**.

Response Plots

View model signals and the requirements applied to the signal using a response plot. You can also plot the frequency response of a system (requires Simulink Control Design software).

The response plot shows the system response as it varies during optimization. You can also view the uncertain system responses in the plot.



To plot a response plot:

1 In the **Response Optimization** tab of the app, in the **Data to Plot** drop-down list, choose a signal, linear system, or requirement.



2 Select a response plot in the **Add Plot** drop-down list.

You can add to an existing plot, or create a plot. The drop-down list has entries for the supported plot types for the given plot variable.

3 Display the current model response.

Click **Plot Model Response**. The current response appears as a thick line on the response plot.

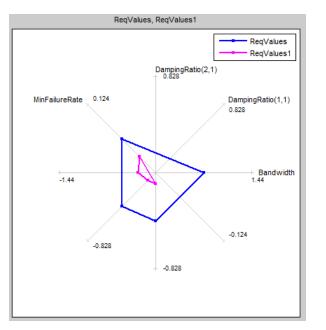
- 4 Configure the plot display.
 - To select the signal or systems to display, right-click within the white space in the plot, and choose from **Signals** and **Systems** in the menu.
 - To control whether the response signal displays at intermediate steps during optimization, right-click within the white space in the plot and select **Responses** > **Show Iteration Responses**. The response at an intermediate step is based on parameter values at that intermediate step in the optimization.
 - Modify plot properties.

You can change plot properties such as plot title, axis labels, axes limits, and units. Right-click the white space in a plot and select **Axes Properties** to open the Property Editor dialog box.

Spider Plots

Compare the values of design variable sets or evaluated requirements using a spider plot.

Spider plots depict multivariate data using a separate axis for each variable. The various axes are arranged clockwise and have a common intersecting point.



To plot a spider plot for a design variable set or evaluated requirements:

- 1 In the **Response Optimization** tab of the app, in **Data to Plot**, choose design variables, or evaluated requirements.
- 2 Select a Spider plot in the Add Plot drop-down list.

You can add to an existing plot, or create a new one. The drop-down list has entries for the supported plot types for the given plot variable.

3 Configure the plot display.

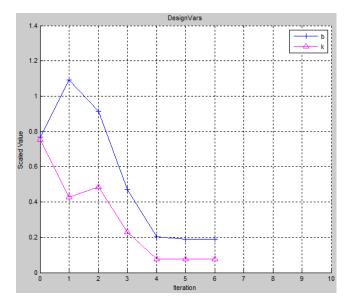
To view only some of the variables or requirement values in a given plot, right-click the plot, and select the variables or requirements in the **Show** list.

For information on using a spider plot to compare design variables sets or evaluated requirements, see "Compare Requirements and Design Variables Using Spider Plot" on page 3-71

Iteration Plots

Plot the values of design variables and requirements as they vary during optimization using an iteration plot.

Iteration plots depict the value of the plot variables for each iteration. The x-axis represents the iteration number.



To plot an iteration plot for a design variable set or requirements:

- **1** In the **Response Optimization** tab of the app, in **Data to Plot**, choose design variables or requirements.
- 2 Select an iteration plot in the **Add Plot** drop-down list.

You can add to an existing plot, or create a plot. The drop-down list has entries for the supported plot types for the given plot variable.

3 Configure the plot display.

To view scaled values of the plotted variables, right-click the plot, and select **Show scaled values**.

4 Save design variables and requirements values for a given iteration using the iteration plots.

For more information, see "Save Design Variable Values for Specific Iteration" on page 3-74.

See Also

Related Examples

• "Save Design Variable Values for Specific Iteration" on page 3-74

• "Compare Requirements and Design Variables Using Spider Plot" on page 3-71

Compare Requirements and Design Variables Using Spider Plot

This example shows how to use a spider plot to compare requirement evaluations before and after optimizing the response. You can use a similar procedure to compare the values of sets of design variables.

Open the Simulink model and load the pre-configured Response Optimizer App session.

For this example, which uses a distillation column model, the step response requirements are preconfigured and loaded in the model workspace.

1 Open the distillation model.

sys = 'distillation_demo';
open_system(sys)

2 Open the **Response Optimizer**.

In the Simulink model window, from the **Apps** tab, in the gallery, under **Control Systems** select **Response Optimizer**.

Alternatively, click the Response Optimization GUI with preloaded data block in the model and skip the next step.

3 Load the preconfigured **Response Optimizer** session.

Click the **Response Optimization** tab. In the **Open Session** drop-down list, select **Open from model workspace**. A window opens where you select the **Response Optimizer** session to load. Select distillation_optim and click **OK**.

The preconfigured step response requirements are loaded in the **Response Optimizer**.

Evaluate the requirement before optimization.

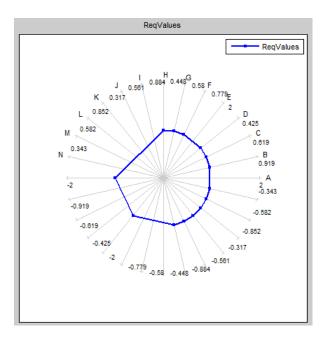
In the **Response Optimization** tab, click **Evaluate Requirements**.

A new variable, ReqValues, containing the evaluation of the requirements appears in the Data area.

When optimizing the model response, you create a set of requirements that it must satisfy. If the requirements are violated, meaning that they evaluate to non-negative values, the design variables must be optimized. After the optimization, you can compare the original requirement value with the requirement evaluated using the optimized design variable values.

Plot the requirement value before optimization.

- 1 In the **Data to Plot** list, select ReqValues.
- 2 In the Add Plot list, select Spider plot.



The plot has an axis for each edge-and-signal combination defined in the distillation_demo/ Desired Step Response check block. Points on each axis represent the violation for that signaledge combination and the plot connects these points to form a closed polygon representing the initial design. Note that some points are negative, representing satisfied constraints, and some positive, representing violated constraints.

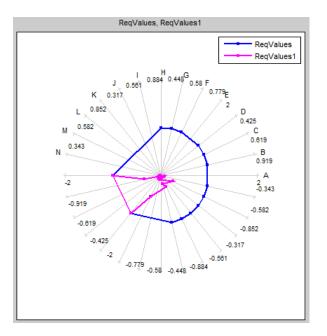
Optimize the model.

Click **Optimize**.

A new variable, ReqValues1, containing the evaluation of the requirements using the optimized design variables appears in the **Data** area.

Compare the requirement values before and after optimization.

- 1 In the **Data to Plot** list, select ReqValues1.
- 2 In the Add Plot list, select Spider plot 1.



The optimized requirement values, ReqValues1, are all negative or zero, indicating that all the constraints are satisfied.

See Also

More About

• "Save Design Variable Values for Specific Iteration" on page 3-74

Save Design Variable Values for Specific Iteration

This example shows how to save the design variable values for specific optimization iterations.

During optimization, the optimization solver simulates the model using a different set of design variables at each iteration. After the optimization completes, you can export the values for an iteration from the iteration plot of the design variable set.

For this example, load a preconfigured **Response Optimizer** session. Optimize the model, and export the design variable set values for the third iteration.

Open the Simulink model and load the preconfigured **Response Optimizer** session.

```
load('distillation_demo_sdosession_export_iter_dv.mat')
sdotool(SDOSessionData)
```

The **Response Optimizer** opens and loads the preconfigured session. **Iteration Plot 1** is configured to plot the values of **DesignVars** for each optimization iteration.

Click Optimize.

The optimization completes after four iterations.

Select the iteration plot of the design variable set.

Click Iteration plot 1.

Open the Save Iteration Data dialog box.

Right-click on the iteration plot, and select Save Iteration Data.

Save Iteration Data	×
Data to save:	DesignVars 🗸
Iteration(s) to save:	4
Save to a variable named:	DesignVars
	OK Cancel Help

You can also access the Save Iteration Data dialog box from the Optimization Progress Report. To do so, in the progress report, click **Save Iteration**.

Specify details about the design variable data to save:

- In the **Data to save** list, select DesignVars.
- In Iteration(s) to save, enter 3.

To specify multiple iterations, use a vector of integers. For example, [0 2 5].

• In **Save to a variable named**, enter DesignVars_iter.

Export the design variable values set.

Click **OK**. The exported data variable, **DesignVars_iter_3**, appears in the **Data** area of the app.

Note The iteration number is added as a suffix to the saved data variable name.

See Also

Related Examples

- "Update Model with Design Variables Set" on page 3-60
- "Compare Requirements and Design Variables Using Spider Plot" on page 3-71

More About

• "Interact with Plots" on page 3-67

Design Optimization to Meet Time-Domain and Frequency-Domain Requirements (GUI)

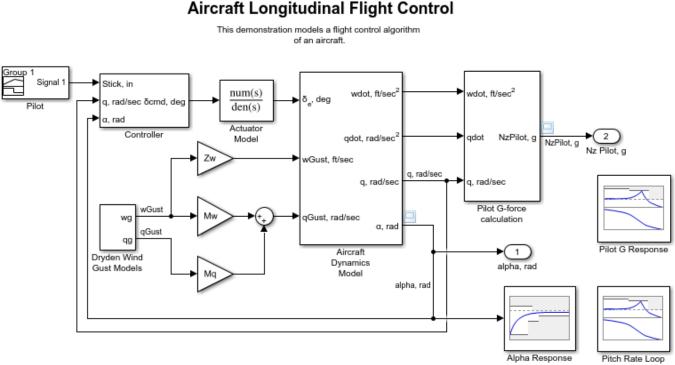
This example shows how to tune a controller to satisfy time-domain and frequency-domain design requirements using the **Response Optimizer**.

The example requires Simulink[®] Control Design[™] software.

Aircraft Longitudinal Flight Control Model

Open the Simulink model.

sys = 'sdoAircraft';
open system(sys);



Copyright 1990-2012 The MathWorks, Inc.

The aircraft model is based on the Simulink slexAircraftExample model. The model includes:

- Subsystems to model aircraft dynamics (Aircraft Dynamics Model), wind gusts (Dryden Wind Gust Models), and pilot G-forces (Pilot G-force calculation).
- A step change applied to the aircraft joystick at 1 second into the simulation that causes the aircraft to pitch upward.

Controller Design Problem

You tune the controller gains to meet the following time-domain and frequency-domain design requirements:

- Angle-of-attack alpha response to a step change in the joystick has a rise time of less than 1 second, less than 1% overshoot, and settles to within 1% of steady state within less than 5 seconds
- Pitch-rate control loop has good tracking below 1 rad/s and 20 dB noise rejection above 100 rad/s
- Closed-loop response from joystick to pilot G-Force is below 0 dB above 5 rad/s.

These requirements reduce the high frequency G-forces experienced by the pilot in response to joystick changes while still maintaining flight performance.

The model includes the following blocks (from Simulink® Design Optimization^m and Simulink Control Design Model Verification libraries):

• Alpha Response specifies the alpha step response requirement.

- Check Step Re	esponse Cha	aracteristics		
Assert that the	e input signa	al satisfies bounds s	pecified by step respo	onse characteristics.
Bounds As	sertion			
☑ Include st	ep response	e bound in assertion		
Step time (se	econds):	1		
Initial value:		0	Final value:	0.5
Rise time (se	conds):	1	% Rise:	80
Settling time	(seconds):	5	% Settling:	1
% Overshoot	:	1	% Undershoot:	1
<table-cell> Enable ze</table-cell>	ro-crossing	detection		
Show Plot	Show plo	t on block open		Response Optimization
		ОК	Cancel	Help Apply

• Pitch Rate Loop specifies the pitch-rate performance requirement.

The linearization inputs/outputs are already selected in the **Linearizations** tab. The pitch-rate loop starts from the input of the controller (the controller error signal) and ends at the output of the pitch-rate sensor. The angle-of-attack loop is opened signal so that the block only computes the pitch-rate loop response. The linear system is computed at a simulation time of 0.

Bode Plot			
	play a linear system o ear system and asser		
Linearizations	Bounds Logging	Assertion	
Linearization inp	outs/outputs:		
Block : Po	rt : Bus Element	Co	nfiguration
sdoAircraft/Co	ntroller/error : 1	Input Perturba	ation +
sdoAircraft/Co	ntroller/Kq : 1	Open-loop O	utput
sdoAircraft/Co	ntroller/Ka : 1	Loop Break	
			×
•	m		4
Linearize on:	Simulation snapsho	ts	•
Snapshot times:	[0]		
Trigger type:	Rising edge		▼
 Algorithm (Options		
▶ Labels			
Labels			
Show Plot	Show plot on block op	en Re	sponse Optimization
0	ОК	Cancel	Help Apply

The **Bounds** tab specifies the following pitch-rate loop shape requirements:

- Greater than 20 dB over the range 0.01 rad/s to 0.1 rad/s
- Greater than 0 dB over the range 0.1 rad/s to 1 rad/s
- Less than -20 dB over the range 100 rad/s to 1000 rad/s

Bode Plot					
	linear system on a Bode plot. You can also specify stem and assert that the bounds are satisfied.				
Linearizations Bour	ds Logging Assertion				
🔲 Include upper mag	nitude bound in assertion				
Frequencies (rad/s):	[100 1000]				
Magnitudes (dB):	[-20 -20]				
🗵 Include lower mag	nitude bound in assertion				
Frequencies (rad/s): [0.01 0.1; 0.1 1]					
Magnitudes (dB):	[20 20; 0 0]				
Show Plot Show	plot on block open Response Optimization				
0	OK Cancel Help Apply				

• Pilot G Response specifies the G-force requirement.

The linearization inputs/outputs are already selected in the **Linearizations** tab. The linear system is computed at a simulation time of 0.

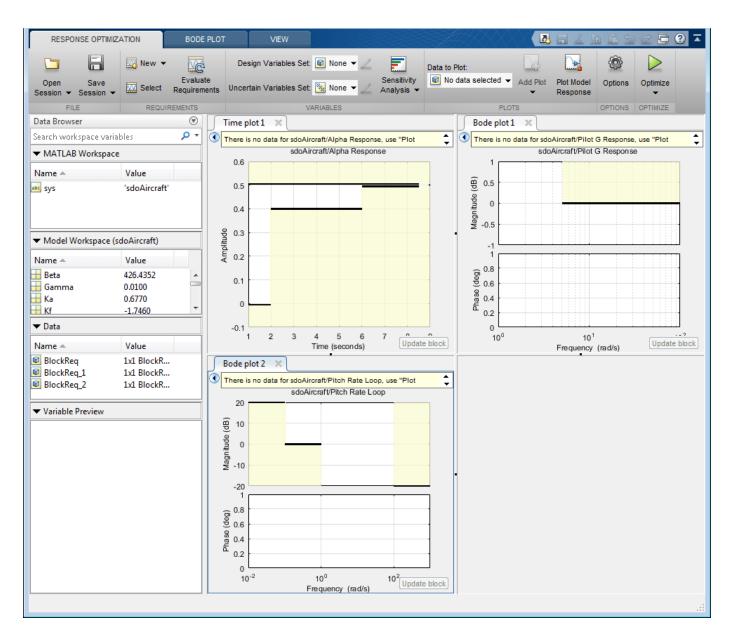
Bode Plot					
Compute and disp bounds on the lin					
	car system				
Linearizations	Bounds	Logging	Assertion		
Linearization inp	outs/output	s:			
Block : Por	rt : Bus Ele	ement	C	onfiguration	
sdoAircraft/Pilo	ot : 1		Input Perturk	ation	+
sdoAircraft/Pilo	ot G-force	calculati	Output Mea	surement	
					×
•				+	
Linearize on:	Simulatio	on snapshot	S		•
Snapshot times:	[0]				
Trigger type:	Rising ed	lge			-
Algorithm (Options				
Labels					
Show Plot	Show plot o	on block ope	en Re	esponse Optimiz	ation
0			Cancel	Halp	Apple
	0	K	Cancel	Help	Apply

The ${\bf Bounds}$ tab specifies the G-force requirements of less than 0 dB over the range 5 rad/s 100 rad/s.

Bode Plot					
Compute and display a bounds on the linear sy					
Linearizations Bour	nds L	ogging	Assertio	on	
📝 Include upper mag	initude l	bound in	assertion		
Frequencies (rad/s):	[5 100]]			
Magnitudes (dB):	[0 0]				
🔲 Include lower mag	nitude t	ound in	assertion		
Frequencies (rad/s): []					
Magnitudes (dB):	[]				
Show Plot Show	plot on	block ope	n	Response	Optimization
0	ОК		Cancel	Help	Apply

Open the Response Optimizer

Open the **Response Optimizer** to configure and run design optimization problems interactively. Click **Response Optimization** on the Block Parameters dialog of Alpha Response, Pitch Rate Loop or Pilot G Response block. Alternatively, type sdotool('sdoAircraft'). To show multiple requirement plots at the same time, use the **View** tab in the app.



The app detects the requirements specified in the Model Verification blocks and automatically includes them as requirements to satisfy.

Specify Design Variables

Specify the following model parameters as design variables for optimization:

- Controller gains Ki and Kf
- Pitch-rate sensor gain Kq
- Alpha sensor gain Ka

In the **Design Variables Set** drop-down list, select **New**. A dialog to select model parameters for optimization opens.

Select Ki, Kf, Kq and Ka. Click << to add the selected parameters to the design variables set.

Specify minimum and maximum gain values, the Ki and Kf values must remain negative while Ka and Kq must remain positive.

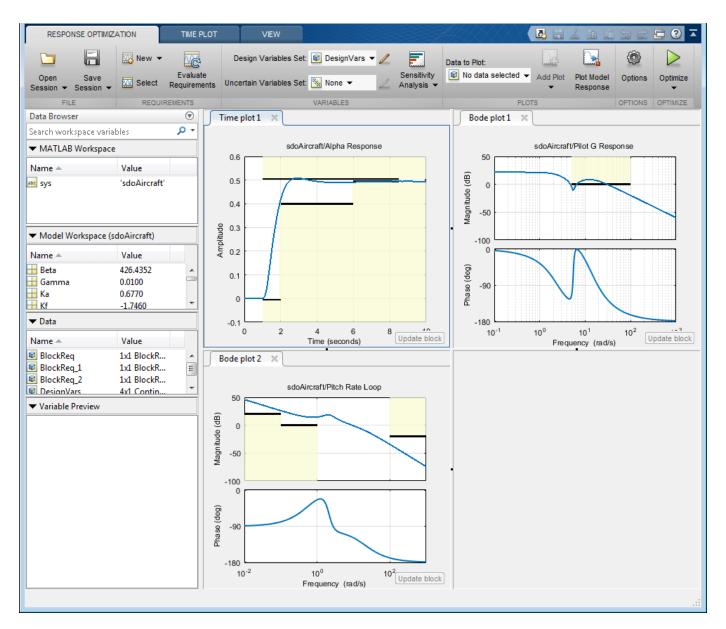
Edit: Desi	gnVars					×
	Variable	Value	Minimum	Maximum	Scale	
V	Ka	0.677000	0.001	Inf	1	+
V	Kf	-1.746	-Inf	-0.001	2	
	Ki	-3.864	-Inf	-0.001	4	
V	Kq	0.8156	0.001	Inf	1	(A)
	H- D1		Up	date model	variables]
Varia	ible Detail					?

Press **Enter** after you enter the values.

Click **OK**. A new variable **DesignVars** appears in the **Response Optimizer** browser.

Evaluate the Initial Design

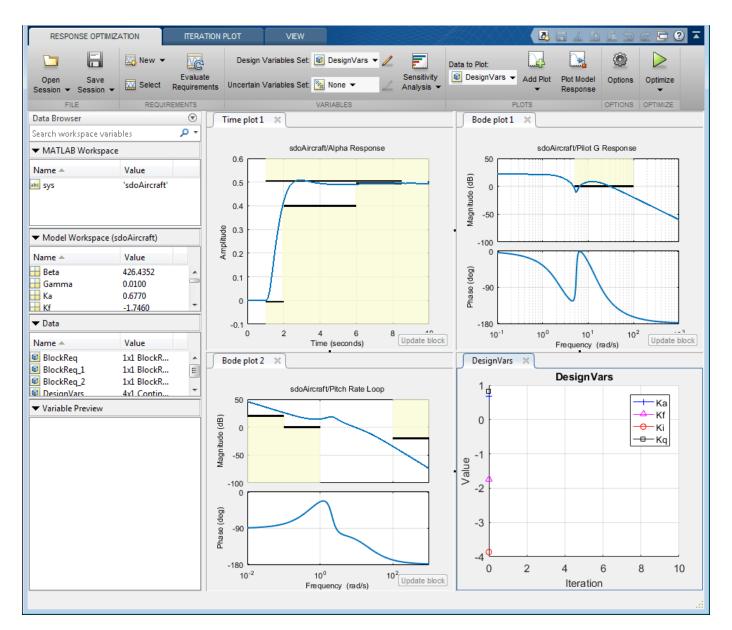
Click **Plot Model Response** to simulate the model and check how well the initial design satisfies the design requirements.



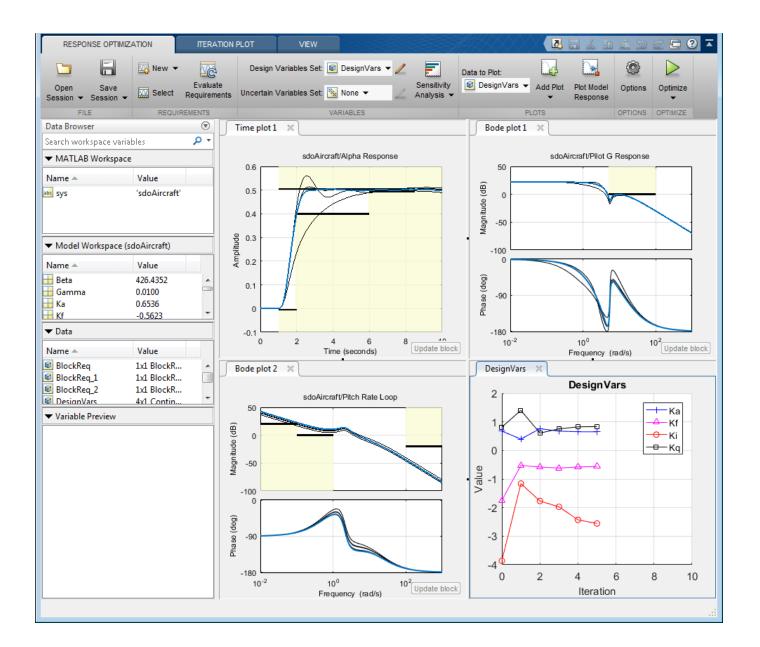
The plots indicate that the current design does not satisfy the pilot G-force requirement and the alpha step response overshoot requirement is violated.

Optimize the Design

Create a plot to display how the controller variables are modified during the optimization. In the **Data To Plot** drop-down list, select **DesignVars**, which contains the optimization design variables Ki, Kf, Kq and Ka. In the **Add Plot** drop-down list, select **Iteration plot**.



Click **Optimize**.



Iteration	F-count	Alpha Response ((<=0)	Pilot G Response (<=0)	Pitch Rate Loop ((<=0)	Pitch Rate Loop ((>=0)
0	9	0.1450	8.5841	-0.7160	0.3122
1	20	0.4060	-1.7578	-1.0067	0.0215
2	31	0.1081	-1.0047	-1.3244	-0.1505
3	41	0.2407	-0.0478	-1.2014	-0.0129
4	51	0.0067	0.0584	-1.1881	0.1180
5	61	2.2579e-04	0.0034	-1.1965	0.1432
		11-Oct-2012 11:15:57 ed, 01-Oct-2012 11:16	:39		
Optimized	variable va	lues written to 'Design\	/ars' in the Design Opti	mization workspace	-
				Display Options	

To load a pre-configured file and run the optimization, click **Open** in the **Response Optimization** tab and select sdoAircraft_sdosession.mat. Alternatively load the project by typing:

>> load sdoAircraft_sdosession

>> sdotool(SDOSessionData)

The optimization progress window updates at each iteration and shows that the optimization converged after 5 iterations.

The Alpha Response and Pilot G Response plots indicate that the design requirements are satisfied. The DesignVars plot shows that the controller gains converged to new values.

To view the optimized design variable values, click DesignVars in the **Response Optimizer** browser. The optimized values of the design variables are automatically updated in the Simulink model.

% Close the model
bdclose('sdoAircraft')

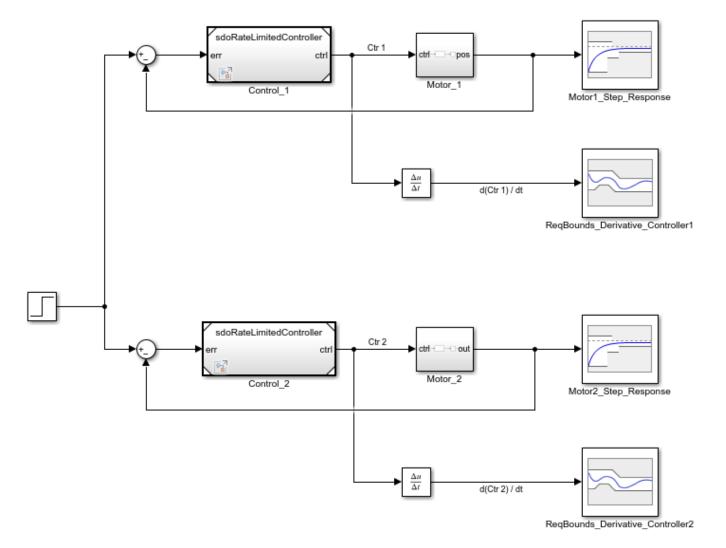
Design Optimization Tuning Parameters in Referenced Models (GUI)

This example shows how to tune parameters in referenced models using the **Response Optimizer** app.

Motor Control Model

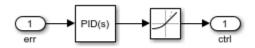
The model shows the control of angular position for two motors. Open the Simulink® model.

```
open_system('sdoMultipleMotors')
```



Model references are used for the two controllers, which are instances of the same model. Open the controller.

open_system('sdoRateLimitedController')



Each controller has PID gains Kp, Ki, and Kd, and a slew rate, Slew. The Slew value limits the rate at which the control signal changes, so that currents drawn from the power supply stay within the power supply's limits. The Slew value is common to both instances of the controller. In contrast, the PID gains need to be different for each instance of the controller, since the motors being controlled have different characteristics. Therefore, the PID gains are specified as model arguments in the controller's model workspace. The PID gain values are set at the level of the sdoMultipleMotors model.

Design Problem

The control reference signal is a step change in position, which occurs at 1 second. Each motor's angular position should follow the reference signal, but the first motor has a smaller moment of inertia and can respond more quickly to changes in the reference signal. The angular position of the first motor must satisfy the following requirements:

- Rise time: 2 seconds
- Settling time: 7 seconds

The second motor has a larger moment of inertia so it can't respond as quickly. The angular position of the second motor must satisfy the following requirements:

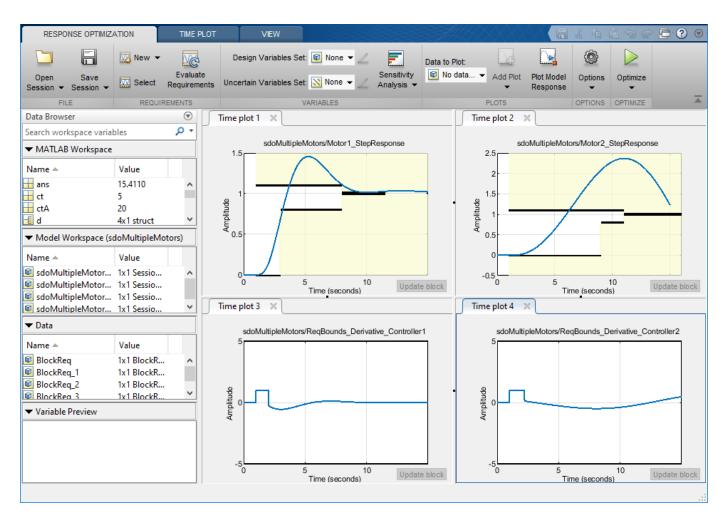
- Rise time: 8 seconds
- Settling time: 10 seconds

Also, the derivative of each controller signal is required to be in the range from -5 to 5, so that currents drawn from the power supply stay within the power supply's limits.

All these requirements are specified using blocks in the Simulink model.

Open the Response Optimizer

In the Simulink model from the **Apps** tab, click **Response Optimizer** under **Control Systems** to launch the **Response Optimizer** app. With the current settings of the controller variables, observe that neither step response requirement is satisfied.



Specify Design Variables

In the **Design Variables Set** list, select **New**. The dialog shows tunable variables. The Slew variable is listed as sdoRateLimitedController:Slew, indicating that the variable is set at the level of the sdoRateLimitedController model. The colon is a delimiter between the model and variable. The Slew variable has the same value for all instances of the controller model. In contrast, the derivative gain for the first controller is listed as sdoMultipleMotors/Control_1:Kd, indicating that the variable is set at the level of the sdoMultipleMotors model, where it appears in the Control_1 block. The forward slash is the delimiter for Simulink blockpaths.

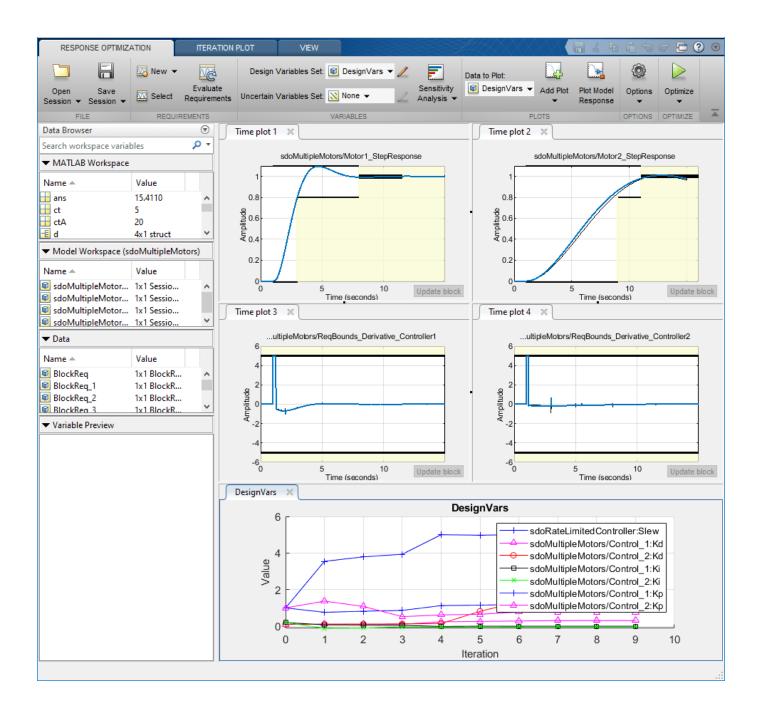
Select the common Slew variable, as well as the Kd, Ki, and Kp parameters for each model. Add these variables to the design variable set. Specify the minimum value for the Slew variable as 0.

	1	1			1		-	
Variable	Value	Minimum	Maximum	Scale		Variable	Current value	Used By
sdoRateLimitedController:Slew	1	0	Inf	1				
sdoMultipleMotors/Control_1:Kd	0.1	-Inf	Inf	0.125				
sdoMultipleMotors/Control_2:Kd	0.1	-Inf	Inf	0.125				
sdoMultipleMotors/Control_1:Ki	0.2	-Inf	Inf	0.25	-			
sdoMultipleMotors/Control_2:Ki	0.2	-Inf	Inf	0.25				
sdoMultipleMotors/Control_1:Kp	1	-Inf	Inf	1				
sdoMultipleMotors/Control_2:Kp	1	-Inf	Inf	1	Ū			
		U	odate model	variables	1			
riable Detail					_	Specify exp	ression indexing	if necessary (e.g., a(3) or s.x)

Click **OK**. A new variable, **DesignVars**, appears in the **Response Optimizer** browser. Create a plot of the design variables to see how they evolve during optimization. Under **Data to Plot**, select **DesignVars**. Click **Add Plot** and add an iteration plot. You can use the **View** tab to arrange the plots so they are all visible.

Optimize the Design

To optimize the design, click **Optimize**. The plots are updated during optimization. When optimization is complete, observe that all design requirements are satisfied. The step responses are within the prescribed bounds, as are the controller output signal derivatives.



Iteration	F-count	Motor1_St	Motor2_St	ReqBound	ReqBound	ReqBound	ReqBound			
		(<=0)	(<=0)	(<=0)	(>=0)	(<=0)	(>=0)			
0	15	0.9773	211.8094	-0.8000	0.8805	-0.8000	0.9000			
1	1 30 5.5294 8.4928 -0.2908 0.9383 -0.2908 0.9200									
2	50	3.4705	6.6123	-0.2419	0.9270	-0.2419	0.9398			
3	70	1.9300	0.9395	-0.2133	0.9099	-0.2133	0.9750			
4	85	0.2025	0.0523	1.0658e-15	0.8747	1.0658e-15	0.9764			
5	100	0.0019	0.0453	-0.0048	0.8704	-0.0048	0.8258			
6	115	1.3695e-04	0.0266	3.1926e-04	0.8092	3.1926e-04	0.9426			
7	131	8.6228e-05	0.0125	9.5779e-05	0.8616	9.5779e-05	0.8219			
8	151	-0.0016	0.0118	7.9682e-05	0.7940	7.9682e-05	0.9677			
9	172	-0.0016	0.0118	7.9682e-05	0.7940	7.9682e-05	0.9677			
Optimizati Optimized sdoMultip	on converg variable v	pdated with op	18 10:58:21 D'DesignVars' in Dtimized values							
Optimizati Optimized SdoMultip Optimized	on converg variable v	ged, 08-Jul-201 alues written to pdated with op nt values writte	18 10:58:21 o 'DesignVars' in							
Optimizati Optimized 'sdoMultip Optimized Optimizati	ion converg I variable v IeMotors' u I requireme	ged, 08-Jul-201 alues written to pdated with op int values writte utput:	18 10:58:21 D'DesignVars' in Dtimized values	s es_1' in the De						
Optimizati Optimized 'sdoMultip Optimized Optimizati Local min Optimizati feasible di	on converg l variable v leMotors' u l requireme ion solver o imum foun on comple rections, to	ged, 08-Jul-201 alues written to pdated with op ent values writte utput: d that satisfies ted because th within the sel	18 10:58:21 b 'DesignVars' in otimized values en to 'ReqValue	s es_1' in the De nction is non-d the optimality	sign Optimizat lecreasing in tolerance,	ion workspace				
Optimizati Optimized 'sdoMultip Optimized Optimizati Local min Optimizati feasible di and constr	on converg l variable v leMotors' u l requireme ion solver o imum foun on comple rections, to	ged, 08-Jul-201 alues written to pdated with op int values writte utput: d that satisfies ited because th o within the sel- atisfied to withi	18 10:58:21 b 'DesignVars' in otimized values en to 'ReqValue the constraints ne objective fu ected value of	s es_1' in the De nction is non-d the optimality	sign Optimizat lecreasing in tolerance,	ion workspace				
Optimizati Optimized 'sdoMultip Optimized Optimizati Local min Optimizati feasible di and constr Stopping o Optimizati is less thar	on converg l variable v: leMotors' u l requireme ion solver o imum foun ion comple rections, to aints are se criteria deta on comple n options.O	ged, 08-Jul-201 alues written to pdated with op int values writte utput: d that satisfies ited because th o within the sel- atisfied to withi ails: tted: The relati ptimalityTolera	18 10:58:21 b 'DesignVars' in otimized values en to 'ReqValue the constraints ne objective fu ected value of	s es_1' in the De a. nction is non-o the optimality value of the co ptimality mease 00e-03, and th	sign Optimizat lecreasing in tolerance, onstraint tolera ure, 0.0000006 e relative max	ion workspace nce. 2+00, imum constrain				

Close the models.

```
bdclose('sdoMultipleMotors')
bdclose('sdoRateLimitedController')
```

See Also

More About

• "Design Optimization Tuning Parameters in Referenced Models (Code)" on page 3-95

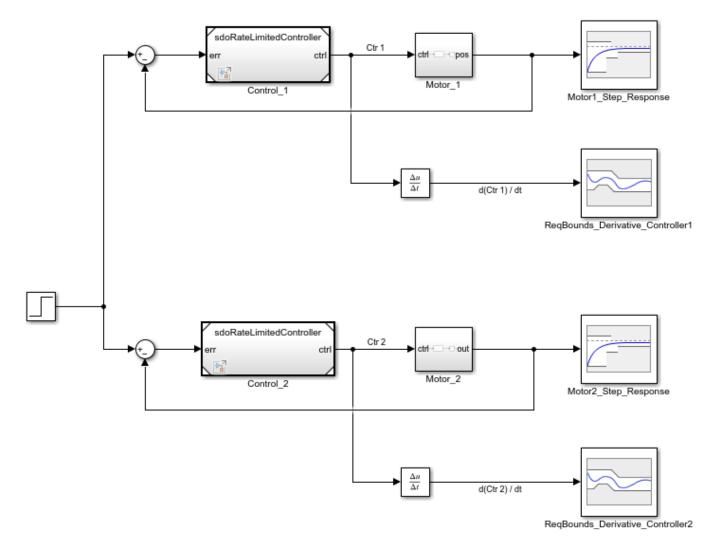
Design Optimization Tuning Parameters in Referenced Models (Code)

This example shows how to tune parameters in referenced models, using the ${\tt sdo.optimize}$ command.

Motor Control Model

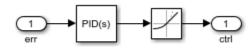
The model shows the control of angular position for two motors. Open the Simulink model.

```
open_system('sdoMultipleMotors')
```



Two motors are controlled. Model references are used for the controllers, which are instances of the same model. Open the controller.

open_system('sdoRateLimitedController')



Design Problem

The two motors in the main model have different characteristics, so each controller needs to be tailored to its motor. Each controller has PID gains Kp, Ki, and Kd, and a slew rate, Slew. The Slew value limits the rate at which the control signal changes. The Slew value is common to both instances of the controller. In contrast, the PID gains need to be different for each instance of the controller, since the motors being controlled have different characteristics. Therefore, the PID gains are specified as model arguments in the controller's model workspace. The PID gain values are set at the level of the sdoMultipleMotors model.

The control reference signal is a step change in position, which occurs at 1 second. Each motor's angular position should follow the reference signal, but the first motor has a smaller moment of inertia and can respond more quickly to changes in the reference signal. Also, the derivative of each controller signal must be limited, so that currents drawn from the power supply stay within the power supply's limits.

Specify Design Variables

Specify the design variables to be tuned by the optimization routine in order to satisfy the requirements. The Slew variable is specified as sdoRateLimitedController:Slew, indicating that the variable is set at the level of the sdoRateLimitedController model. The colon is a delimiter between the model and variable. The Slew variable has the same value for all instances of the controller model. In contrast, the proportional gain for the first controller is specified as sdoMultipleMotors/Control_1:Kp, indicating that the variable is set at the level of the sdoMultipleMotors model, where it appears in the Control_1 block. The forward slash is the delimiter for Simulink blockpaths.

Specify that design variables include the gains Kp, Ki, and Kd, for both PID controllers. Also include the slew rate, Slew, which is common to both controllers. Finally, specify that the slew rate cannot be negative.

```
DesignVars = sdo.getParameterFromModel('sdoMultipleMotors', ...
{'sdoRateLimitedController:Slew', ...
'sdoMultipleMotors/Control_1:Kp', ...
'sdoMultipleMotors/Control_1:Kd', ...
'sdoMultipleMotors/Control_2:Kp', ...
'sdoMultipleMotors/Control_2:Ki', ...
'sdoMultipleMotors/Control_2:Kd' });
DesignVars(1).Minimum = 0;
```

Specify Design Requirements

Each motor's angular position angle should follow the reference signal, but the first motor has a smaller moment of inertia and can respond more quickly to changes in the reference signal. We want the angular position of the first motor to satisfy the following requirements:

• Rise time: 2 seconds

• Settling time: 7 seconds

This requirement is specified in a step response check block in the Simulink model. We can refer to the block and include the requirement in a variable, to be passed to the optimization objective function.

```
Requirements = struct;
bnds = getbounds('sdoMultipleMotors/Motor1_Step_Response');
Requirements.Motor1_StepResponse = bnds{1};
```

The second motor has a larger moment of inertia, so it can't respond as quickly. We want the angular position of the second motor to satisfy the following requirements:

- Rise time: 8 seconds
- Settling time: 10 seconds

This requirement is also specified in a step response check block in the Simulink model, and we refer to the block to include the requirement in a variable, to be passed to the optimization objective function.

```
bnds = getbounds('sdoMultipleMotors/Motor2_Step_Response');
Requirements.Motor2_StepResponse = bnds{1};
```

Also, the derivative of each controller signal is required to be in the range from -5 to 5, so that currents drawn from the power supply stay within the power supply's limits. These requirements are specified in bound check blocks in the Simulink model, and these requirements should also be collected among requirements to be passed to the optimization objective function.

```
bnds = getbounds('sdoMultipleMotors/ReqBounds_Derivative_Controller1');
Requirements.Controller1_DerivBound1 = bnds{1};
Requirements.Controller1_DerivBound2 = bnds{2};
bnds = getbounds('sdoMultipleMotors/ReqBounds_Derivative_Controller2');
Requirements.Controller2_DerivBound1 = bnds{1};
Requirements.Controller2_DerivBound2 = bnds{2};
```

Prevent check block assertions during optimization.

```
CheckBlockStatus = sdo.setCheckBlockEnabled('sdoMultipleMotors','off');
```

Simulation Definition

The cost function requires a simulation scenario to run the model. Create a simulation scenario and add model signals to log, so their values are available to the cost function.

Simulator = sdo.SimulationTest('sdoMultipleMotors');

The motor angular positions need to be logged during optimization, to evaluate the requirements on their step responses.

```
Motor1_Position = Simulink.SimulationData.SignalLoggingInfo;
Motor1_Position.BlockPath = 'sdoMultipleMotors/Motor1_Step_Response/u';
Motor1_Position.LoggingInfo.LoggingName = 'Motor1_Position';
Motor1_Position.LoggingInfo.NameMode = 1;
```

```
Motor2_Position = Simulink.SimulationData.SignalLoggingInfo;
Motor2_Position.BlockPath = 'sdoMultipleMotors/Motor2_Step_Response/u';
```

```
Motor2_Position.LoggingInfo.LoggingName = 'Motor2_Position';
Motor2 Position.LoggingInfo.NameMode = 1;
```

The controller signal derivatives also need to be logged, to evaluate the bound requirements on them.

```
Controller1_Derivative = Simulink.SimulationData.SignalLoggingInfo;
Controller1_Derivative.BlockPath = 'sdoMultipleMotors/ReqBounds_Derivative_Controller1/u';
Controller1_Derivative.LoggingInfo.LoggingName = 'Controller1_Derivative';
Controller1_Derivative.LoggingInfo.NameMode = 1;
```

```
Controller2_Derivative = Simulink.SimulationData.SignalLoggingInfo;
Controller2_Derivative.BlockPath = 'sdoMultipleMotors/ReqBounds_Derivative_Controller2/u';
Controller2_Derivative.LoggingInfo.LoggingName = 'Controller2_Derivative';
Controller2_Derivative.LoggingInfo.NameMode = 1;
```

To log these signals during optimization, collect them into the simulation scenario, Simulator.

```
Simulator.LoggingInfo.Signals = [...
Motor1_Position ; ...
Motor2_Position ; ...
Controller1_Derivative ; ...
Controller2_Derivative ];
```

Create Optimization Objective Function

Create an objective function, which will be called at each optimization iteration, to evaluate the design requirements as the design variables are tuned. This cost function has input arguments for the design variables, simulation scenario, and design requirements.

type sdoMultipleMotors_Design

```
function Vals = sdoMultipleMotors Design(P,Simulator,Requirements)
%SDOMULTIPLEMOTORS DESIGN Objective function for multiple motors
%
% Function called at each iteration of the optimization problem.
%
% The function is called with the model named mdl, a set of parameter
\% values, P, a Simulator, and the design Requirements to evaluate. It
% returns the objective value and constraint violations, Vals, to the
% optimization solver.
%
% See the sdoExampleCostFunction function and sdo.optimize for a more
% detailed description of the function signature.
%
% See also sdoMultipleMotors cmddemo
% Copyright 2018 The MathWorks, Inc.
%% Model Evaluation
% Simulate the model.
Simulator.Parameters = P;
Simulator = sim(Simulator);
% Retrieve logged signal data.
SimLog = find(Simulator.LoggedData, get_param('sdoMultipleMotors','SignalLoggingName'));
Motorl_Position = find(SimLog, 'Motorl_Position');
```

```
Motor2 Position = find(SimLog, 'Motor2 Position');
Controller1 Derivative = find(SimLog, 'Controller1 Derivative');
Controller2_Derivative = find(SimLog, 'Controller2_Derivative');
% Evaluate the design requirements.
Cleq Motor1 StepResponse
                               = evalRequirement(Requirements.Motor1 StepResponse,
                                                                                              Motorl Posi
Cleq Motor2 StepResponse
                               = evalRequirement(Requirements.Motor2 StepResponse,
                                                                                              Motor2 Posi
Cleg Controller1 DerivBound1 = evalRequirement(Requirements.Controller1 DerivBound1, Controller1
Cleq Controller1 DerivBound2 = evalRequirement(Requirements.Controller1 DerivBound2, Controller1
Cleq_Controller2_DerivBound1 = evalRequirement(Requirements.Controller2_DerivBound1, Controller2_
Cleq_Controller2_DerivBound2 = evalRequirement(Requirements.Controller2_DerivBound2, Controller2
%% Return Values.
%
\% Collect the evaluated design requirement values in a structure to
% return to the optimization solver.
Vals.Cleq = [...
    Cleq Motor1 StepResponse(:); ...
    Cleq Motor2 StepResponse(:); ...
    Cleg Controller1 DerivBound1(:); ...
    Cleq_Controller1_DerivBound2(:); ...
    Cleq Controller2 DerivBound1(:); ...
    Cleq_Controller2_DerivBound2(:)];
```

end

To optimize, define a handle to the cost function that uses the Simulator and Requirements defined above. Use an anonymous function that takes one argument (the design variables) and calls the objective function. Finally, call sdo.optimize to optimize the design variables to try to meet the requirements.

```
optimfcn = @(P) sdoMultipleMotors_Design(P, Simulator, Requirements);
[Optimized_DesignVars, Info] = sdo.optimize(optimfcn, DesignVars);
```

Optimization started 02-Sep-2021 00:21:48

				max		First-order
Iter	F-count	f (x)	constraint	Step-size	optimality
0	15		0	211.8		
1	30		0	8.464	2.92	26.5
2	50		0	6.531	0.396	15.1
3	70		0	1.931	0.675	15.7
4	85		0	0.2419	1.67	41.2
5	100		0	0.04524	0.271	1.02e+03
6	110		0	0.04524	0.00399	Θ
local	minimum	found that	t sa	tisfies the	constraints	

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the value of the optimality tolerance, and constraints are satisfied to within the value of the constraint tolerance.

Restore check block assertions.

sdo.setCheckBlockEnabled('sdoMultipleMotors', CheckBlockStatus);

Update Model

Update the model with the optimized parameter values.

sdo.setValueInModel('sdoMultipleMotors',Optimized_DesignVars);

Close the models.

```
bdclose('sdoMultipleMotors')
bdclose('sdoRateLimitedController')
```

See Also

More About

• "Design Optimization Tuning Parameters in Referenced Models (GUI)" on page 3-88

Specify Steady-State Operating Point for Response Optimization

What is a Steady-State Operating Point?

An *operating point* of a dynamic system defines the states and root-level input signals of the model at a specific time. For example, in a car engine model, variables such as engine speed, throttle angle, engine temperature, and surrounding atmospheric conditions typically describe the operating point.

A *steady-state operating point* of a model, also called an equilibrium or *trim* condition, includes state variables that do not change with time.

A model can have several steady-state operating points. For example, a hanging damped pendulum has two steady-state operating points at which the pendulum position does not change with time. A *stable steady-state operating point* occurs when a pendulum hangs straight down. When the pendulum position deviates slightly, the pendulum always returns to equilibrium. In other words, small changes in the operating point do not cause the system to leave the region of good approximation around the equilibrium value.

When using optimization search to compute operating points for nonlinear systems, your initial guesses for the states and input levels must be near the desired operating point to ensure convergence.

When linearizing a model with multiple steady-state operating points, it is important to have the right operating point. For example, linearizing a pendulum model around the stable steady-state operating point produces a stable linear model, whereas linearizing around the unstable steady-state operating point produces an unstable linear model.

For more information on operating points, see "What Is an Operating Point?" (Simulink Control Design) and "What Is a Steady-State Operating Point?" (Simulink Control Design).

Setting up a Steady-State Operating Point

This topic shows how to setup a steady-state operating point in **Response Optimizer**. To improve the fit between the model and measured data, the model must be set to steady-state before optimization.

- **1** Open the **Response Optimizer** and specify your requirements using the steps outlined in "Design Optimization to Meet Frequency-Domain Requirements (GUI)" on page 3-129.
- 2 In the toolstrip, click **Options** and select **Operating Point Options** from the drop down menu.

ALIPONE OPINZA	00x			6 mil						
	Career a	Custom		an fai 😸 han e	data to Pat.	- 40000	Des Noos	2000	Comice	
								-	-	
Outs Brancer			019						Ex sheains	
	40		p -					plats at	disaraping results	
· Millia Nodapere								Entrol	when Carlinson	
Name o	Value			24	 		Popul		For sales and on and br coloniador	
Finites In	bi dealer.								or and of sheeping	a a part of
Properties a	Mideable.			-				People	Cydore	
	Veralities.								for using paratet	
	556.00.		~					104gul	ing for model evaluation	•
· Model Nodopers Pr	<i>cubiceble</i>	90						Lawrence	attus cations	
Name o	Value			14				Cotion	For the model	
ill and										
and b	1592			10				fipes6	ng Paint Options	1 1
	1,1258							Cotion	The setting model	

3 The following **Operating Point** dialog box opens.



The **Optimize at steady-state** option is checked by default when you open the operating point dialog. Use the **States**, **Inputs** and **Outputs** tabs to specify the known parameters, bounds and deviations. For instance, there is one state in the above figure. Use the operating point dialog to specify that this state should be treated as an unknown, and it should be set to steady state. During response optimization, the operating point computation will vary this state to set it at steady-state.



You can also sync operating point specifications from your Simulink model using the button.

4 The Simulink Design Optimization software uses optimization methods to search for operating points in a model. Use the **Options** tab of the dialog to specify these optimization methods. These options specify the optimization algorithm, tolerances, and stopping conditions. For instance, the option Gradient descent with projection is often used to find the operating point for systems that use physical modeling. For more information, click on the methods.



5 Having specified the operating point parameters, continue with the optimization workflow as described in "Design Optimization to Meet Frequency-Domain Requirements (GUI)" on page 3-129.

See Also

More About

- "What Is an Operating Point?" (Simulink Control Design)
- "What Is a Steady-State Operating Point?" (Simulink Control Design)
- "Set Model to Steady-State When Estimating Parameters (GUI)" on page 2-117
- "Set Model to Steady-State When Estimating Parameters (Code)" on page 2-97

Design Optimization to Meet a Custom Objective (GUI)

This example shows how to optimize a design to meet a custom objective using the **Response Optimizer** app. You optimize the cylinder parameters to minimize the cylinder geometry and satisfy design requirements.

Hydraulic Cylinder Model

The hydraulic cylinder model is based on the Simulink® model sldemo_hydcyl. The model includes:

- Pump and Cylinder Assembly subsystems. For more information on the subsystems, see "Single Hydraulic Cylinder Simulation".
- A step change applied to the cylinder control valve orifice area that causes the cylinder piston position to change.

Hydraulic Cylinder Design Problem

You tune the cylinder cross-sectional area and piston spring constant to meet the following design requirements:

- Ensure that the piston position has a step response rise time of less than 0.04 seconds and setting time of less than 0.05 seconds.
- Limit the maximum cylinder pressures to 1.75e6 N/m.
- Minimize the cylinder cross-sectional area.

Open the Response Optimizer

Open the **Response Optimizer** to configure and run design optimization problems interactively using the following command.

```
sdotool('sdoHydraulicCylinder')
```

RESPONSE OPTIMIZ	ATION	VIEW					d i s	๔ 🗗 🕐 革
	New 👻	Evaluate	Design Variables Set: 😰 None 👻 🖉	Sensitivity	Data to Plot:		٢	
Open Save Session ▼ Session ▼	Select	Requirements	Uncertain Variables Set: 📉 None 💌 🚄	Analysis 👻	No data selected Add Plot	Plot Model Response	Options	Optimize
FILE	REQUIP	REMENTS	VARIABLES		PLOTS		OPTIONS	OPTIMIZE
Data Browser		$\overline{\mathbf{v}}$						
Search workspace varia	bles	- ۹						
 MATLAB Workspace 	1							
Name 🔺	Value							
 ✓ Model Workspace (s Name ▲ A Ac Beta C1 ✓ Data 	doHydraulicC Value 0.0100 1.0000e-03 70000000 2.0000e-08	ylinder)						
Name 🔺	Value							
▼ Variable Preview								

Specify Design Variables

Specify the following model parameters as design variables for optimization:

- Cylinder cross-sectional area Ac
- Piston spring constant K

In the **Design Variables Set** drop-down list, select **New**. A dialog to select model parameters for optimization opens.

reate D	esign Variables set: DesignVars								
	Variable	Value	Minimum	Maximum	Scale]	Variable	Current value	Used By
							Ac	0.001	sdoHydraulio
							Beta	70000000	sdoHydraulio
						-	C1	2e-008	sdoHydraulio
							C2	3e-009	sdoHydraulio
							Cd	0.61	
							К	50000	sdoHydraulio
							Qmax	0.005	sdoHydraulio
			Upda	te model varia	able values		V30	2.5e-005	
							rho	800	sdoHydraulio
Varia	able Detail						Specify expression	ion (e.g., s.x or a(3))	
									ОКС

Select Ac and K. Click \Leftrightarrow to add the selected parameters to the design variables set.

Limit the cylinder cross-sectional area to circular area with radius between 1 and 2 centimeters and the piston spring constant to a range of 1e4 to 10e4 N/m. To do so, specify the maximum and minimum for the corresponding variable in the **Maximum** and **Minimum** columns.

Because the variable values are different orders of magnitude, scale Ac by 1e-3 and K by 1e5.

Press Enter after you specify the values.

									X
Create I	Design Variables set: DesignVars								
	Variable	Value	Minimum	Maximum	Scale	I	Variable	Current value	Used By
V	Ac	0.001	0.0003	0.0013	0.001		Ac	0.001	sdoHydraulicCylinder/Cylinde
	κ	50000	10000	100000	100000		Beta	70000000	sdoHydraulicCylinder/Cylinde
						-	C1	2e-008	sdoHydraulicCylinder/Cylinde
							C2	3e-009	sdoHydraulicCylinder/Pump
							Cd	0.61	
							К	50000	sdoHydraulicCylinder/Cylinde
							Qmax	0.005	sdoHydraulicCylinder/Pump
			Upda	te model vari	able values	J	V30	2.5e-005	
							rho	800	sdoHvdraulicCvlinder/Cvlinde
▶ Vari	able Detail						Specify expression (e.g., s	.x or a(3))	
									OK Cancel Help

Click **OK**. A new variable **DesignVars** appears in the **Response Optimizer** browser.

Specify Design Requirements

The design requirements require logged model signals. During optimization, the model is simulated using the current value of the design variables and the logged signal is used to evaluate the design requirements.

Log the cylinder pressures, which is the first output port of the Cylinder Assembly block.

In the **New** drop-down list, select **Signal**. A dialog to select model signals to log opens.

RESPONSE OPTIMIZ	ATION	VIEW					HANN (4	1 B B	9	⊑ ? ≖
	No.	ew 🕶 📈	Design Variables Set: 👔	DesignVa	rs 🕶 🖊	E	Data to Plot:			٢	\triangleright
Open Save Session ← Session ← FILE	×××	Damping Ratio Specify a bound on t ratio of the poles of			4	Sensitivity Analysis ▼	No data selected PLC	Add Plot	Plot Model Response	Options OPTIONS	Optimize OPTIMIZE
Data Browser Search workspace varia ▼ MATLAB Workspace	Gunid	Natural Frequency Specify a bound on frequency of the po	the natural les of a linear system.								
Name 📥	×××	Settling Time Specify an upper bo approximate settling	und on the 1 time of a linear system.								
▼ Model Workspace (s		Singular Value Specify a piecewise I the singular values o									
Name A A Ac		Step Response Enve Specify bounds on t response of a linear	he step	Γ							
Beta C1	fx	Custom Requiremen Create a custom requirement									
	NEW	V REQUIREMENT SOUR	CE								
Name 🔺 😰 DesignVars	-6		nals to log so design requirements.	E	=						
▼ Variable Preview	Ĵå	Linearization I/Os Specify Linearization can be used in desig									
	NEW	V REQUIREMENT USING	MODEL BLOCKS								
	•		Blocks Design Optimization library n check blocks to a Simulin		-						.:

		X
Signal set: Sig		
		Signal
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	
		Remove Signal
		OK Cancel Help

Enter Pressures as the signal name in the **Signal set** field. Then, in the Simulink model, click the first output port of the Cylinder Assembly block named Pressure. The dialog updates to display the selected signal.

Select the signal in the dialog and click \Rightarrow to add it to the signal set.

			×
Signal set: Pressures			
Lurrently selected signals]	Signal	
/Cylinder Assembly : 1(Pressures)		sdoHydraulicCylinder/Cylinder Assembly:1 (Pressures)	
	⇒		
			Remove Signal
			OK Cancel Help

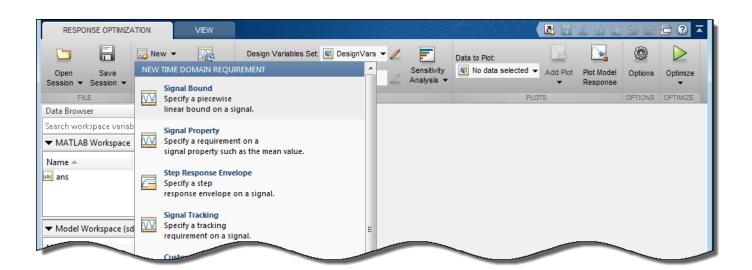
Click **OK**. A new variable **Pressures** appears in the **Response Optimizer** browser.

Similarly, log the piston position, which is the second output of the Cylinder Assembly block, in a variable named PistonPosition.

	×
Signal set: PistonPosition	
Currently selected signals	Signal
/Cylinder Assembly : 2(PistonPosition)	sdoHydraulicCylinder/Cylinder Assembly:2 (PistonPosition)
	Remove Signal
	OK Cancel Help

Specify the maximum cylinder pressure requirement of less than 1.75e6 N/m.

In the **New** drop-down list, select **Signal Bound**. A dialog to create a signal bound requirement opens.



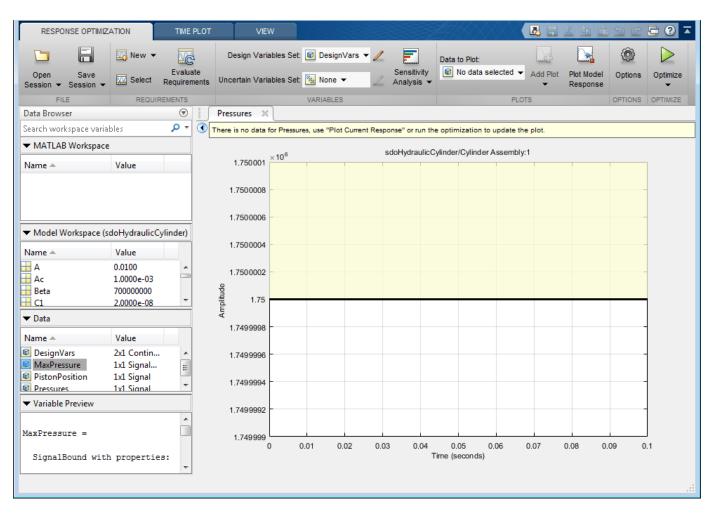
					×								
	Signal Bound												
Use to specify a piecewise linear bound on a signal.													
Name: SignalBo	ound												
 Specify Sign 	al Bound												
Type: Constrai	Type: Constrain signal to be <= the bound 👻												
Edge	Start	Edge	End										
Time (s)	Amplitude	Time (s)	Amplitude	Slope (1/s)	+								
0	1	10	1	0									
					Ū								
]								
Select Signa	ls to Bound												
Create Plot			ОК	Cancel	Help								

Designate the **Requirement Name** as MaxPressure. In both the start and end **Amplitude** columns, enter the maximum pressure requirement of 1.75e6 N/m, and set the **Edge End Time** to 0.1 s. In the **Select Signals to Bound** area, select **Pressures**, the signal on which this requirement applies.

					X	
Signal Bound						
	Use to specif	y a piecewise lin	ear bound on a si	ignal.		
Name: MaxPres	sure					
▼ Specify Signa						
Type: Constrain	n signal to be <= 1	the bound 👻				
Edg	e Start	Edge	e End			
Time (s)	Amplitude	Time (s)	Amplitude	Slope (1/s)	+	
0	1750000	0.1000	1750000	0		
					1	
▼ Select Signal	s to Bound					
		Cinnal			1	
Piston	Signal PistonPosition (sdoHydraulicCylinder/Cylinder Assembly:2)					
	res (sdoHydraulic			2)	+	
		-,	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,			
]	
Create Plot OK Cancel Help						

Click OK.

- A new MaxPressure variable appears in the **Response Optimizer** browser.
- A graphical view of the maximum pressure requirement is automatically created.



Specify the piston position step response requirement of rise time of less than 0.04 seconds and a settling time of less than 0.05 seconds.

In the **New** drop-down list of the **Response Optimization** tab, select **Step Response Envelope**. A dialog to create a step response requirement opens.

Specify a requirement named PistonResponse, and the required rise and settling time bounds. Select PistonPosition as the signal to apply the step response requirement to.

					×	
Step Response Envelope Use to specify a step response envelope on a signal.						
Name: PistonR	esponse					
 Specify Ste 	p Response Characteris	tics				
Initial value:	0		Final value:		0.0400	
Step time :	0	seconds				
Rise time :	0.0400	seconds	% Rise:		80	
Settling time :	0.0500	seconds	% Settling:		1.0000	
% Overshoot:	10.0000		% Undershoot:		1	
▼ Select Signals to Bound						
Signal						
PistonPosition (sdoHydraulicCylinder/Cylinder Assembly:2)						
Pressures (sdoHydraulicCylinder/Cylinder Assembly:1)						
					1	
Create Plot			ОК	Cancel	Help	

Click OK.

Specify Custom Objective

The custom objective is to minimize the cylinder cross-sectional area.

In the **New** drop-down list, select **Custom Requirement**. A dialog to create a custom requirement opens.

Create Requirement ×			
Custom Requirement			
Create a custom requirement. The optimizer evaluates the specified function during optimization passing a structure with fields containing the optimized design variable values and logged simulation results.			
Name: Cu	stomReg		
Specify	Function		
Туре:	Constrain function output to be <= 0		
Function:	@myCustomRequirement		
Error if constraint is violated			
Select Signals and Systems to Bound (Optional)			
Create	Plot OK Cancel Help		
	.::		

Specify a function to call during optimization in the **Requirement Function** field. At each optimization iteration, the software calls the function and passes the current design variable values. You can also optionally pass logged signals to the custom requirement. Here, you use sdoHydraulicCylinder_customObjective as the custom requirement function, which returns the value of the cylinder cross-sectional area.

In the **Requirement Type** drop-down list, specify whether the requirement is an objective to minimize (min), an inequality constraint (<=), or an equality constraint (==).

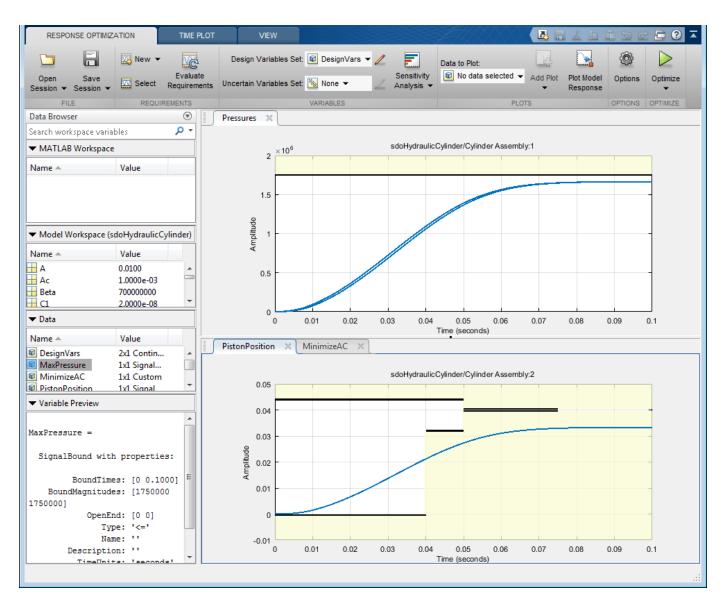
Create Requirement ×			
Custom Requirement			
Create a custom requirement. The optimizer evaluates the specified function during optimization passing a structure with fields containing the optimized design variable values and logged simulation results.			
Name: Mi	nimizeAC		
 Specify 	Function		
Туре:	Minimize the function output 🔹		
Function:	@sdoHydraulicCylinder_customObjective		
Error if constraint is violated			
Select Signals and Systems to Bound (Optional)			
V Create	Plot OK Cancel Help		

type sdoHydraulicCylinder_customObjective

```
function objective = sdoHydraulicCylinder customObjective(data)
%SDOHYDRAULICCYLINDER CUSTOMOBJECTIVE
%
% The sdoHydraulicCylinder_customObjective function is used to define a
% custom requirement that can be used in the graphical SDTOOL environment.
%
% The |data| input argument is a structure with fields containing the
% design variable values chosen by the optimizer.
%
% The |objective| return argument is the objective value to be minimized by
% the SDOTOOL optimization solver.
%
% Copyright 2011 The MathWorks, Inc.
%For the cylinder design problem we want to minimize the cylinder
%cross-sectional area so return the cylinder cross-sectional area as an
%objective value.
Ac = data.DesignVars(1);
objective = Ac.Value;
end
```

Evaluate the Initial Design

Click **Plot Model Response** to simulate the model and check how well the initial design satisfies the design requirements. To show both requirement plots at the same time, use the plot layout widgets in the **View** tab.

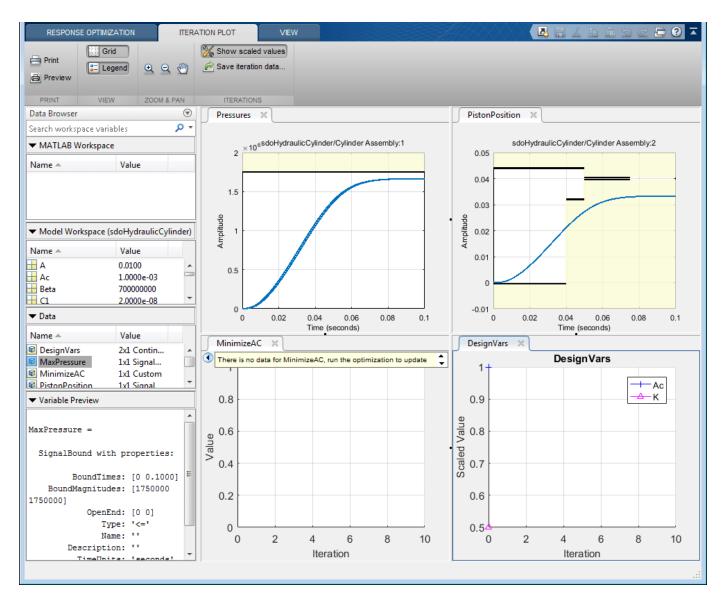


From the plots, see that the maximum pressure requirement is satisfied but the piston position step response requirement is not satisfied.

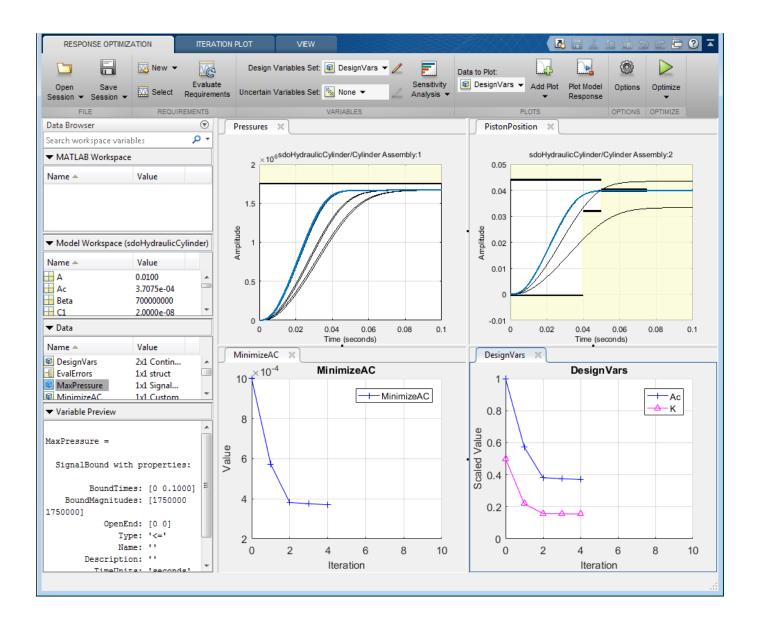
Optimize the Design

Create a plot to display how the cylinder cross-sectional area and piston spring constant are modified during optimization.

In the **Data to Plot** drop-down list, select **DesignVars**, which contains the optimization design variables Ac and K. In the **Add Plot** drop-down, create a new iteration plot to show the design variable trajectories. For this new plot, click **Show scaled values** in the **Iteration Plot** tab, to facilitate viewing the two trajectories on the same axes.



Click **Optimize** in the **Response Optimization** tab.



Iteration	F-count	MinimizeAC	MaxPressure	PistonResponse
		(Minimize)	(<=0)	(<=0)
0	5	1.0000e-03	-0.0480	0.3033
1	11	5.7281e-04	-0.0476	0.0729
2	17	3.8184e-04	-0.0476	-4.2157e-04
3	22	3.7527e-04	-0.0476	-4.7753e-04
4	27	3.7075e-04	-0.0476	-4.0574e-04
Optimization started 19-Oct-2014 19:58:06 Optimization converged, 19-Oct-2014 19:58:13 The optimizer encountered 1 errors during the optimization. Details of the errors have been written to				
'EvalErrors' in the Design Optimization workspace.				

The optimization progress window updates at each iteration and shows that the optimization converged after 4 iterations.

The Pressures and PistonPosition plots indicate that the design requirements are satisfied. The MinimizeAC plot shows that the cylinder cross-sectional area Ac is minimized.

To view the optimized design variable values, click the variable name in the **Response Optimizer** browser. The optimized values of the design variables are automatically updated in the Simulink model.

Related Examples

To learn how to optimize the cylinder design using the sdo.optimize command, see "Design Optimization to Meet a Custom Objective (Code)" on page 3-118.

Design Optimization to Meet a Custom Objective (Code)

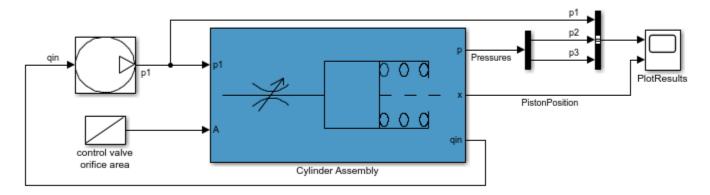
This example shows how to optimize a design to meet a custom objective using sdo.optimize. You optimize the cylinder parameters to minimize the cylinder geometry and satisfy design requirements.

Hydraulic Cylinder Model

Open the Simulink® model.

sys = 'sdoHydraulicCylinder';
open_system(sys);

Single Hydraulic Cylinder Simulation



Copyright 1990-2011 The MathWorks, Inc.

The hydraulic cylinder model is based on the Simulink model sldemo hydcyl. The model includes:

- Pump and Cylinder Assembly subsystems. For more information on the subsystems, see "Single Hydraulic Cylinder Simulation".
- A step change applied to the cylinder control valve orifice area that causes the cylinder piston position to change.

Hydraulic Cylinder Design Problem

You tune the cylinder cross-sectional area and piston spring constant to meet the following design requirements:

- Ensure that the piston position has a step response rise time of less than 0.04 seconds and setting time of less than 0.05 seconds.
- Limit the maximum cylinder pressures to 1.75e6 N/m.
- Minimize the cylinder cross-sectional area.

Specify Design Variables

Select the following model parameters as design variables for optimization:

- Cylinder cross-sectional area Ac
- Piston spring constant K

```
Ac = sdo.getParameterFromModel('sdoHydraulicCylinder','Ac');
K = sdo.getParameterFromModel('sdoHydraulicCylinder','K');
```

Limit the cylinder cross-sectional area to a circular area with radius between 1 and 2 centimeters.

Ac.Minimum = pi*1e-2^2; % m^2 Ac.Maximum = pi*2e-2^2; % m^2

Limit the piston spring constant to a range of 1e4 to 10e4 N/m.

K.Minimum = 1e4; % N/m
K.Maximum = 10e4; % N/m

Specify Design Requirements

The design requirements require logged model signals. During optimization, the model is simulated using the current value of the design variables and the logged signal is used to evaluate the design requirements.

Log the following signals:

• Cylinder pressures, available at the first output port of the Cylinder Assembly block

```
Pressures = Simulink.SimulationData.SignalLoggingInfo;
Pressures.BlockPath = 'sdoHydraulicCylinder/Cylinder Assembly';
Pressures.OutputPortIndex = 1;
```

Piston position, available at the second output port of the Cylinder Assembly block

```
PistonPosition = Simulink.SimulationData.SignalLoggingInfo;
PistonPosition.BlockPath = 'sdoHydraulicCylinder/Cylinder Assembly';
PistonPosition.OutputPortIndex = 2;
```

Create an object to store the logging information and use later to simulate the model

```
simulator = sdo.SimulationTest('sdoHydraulicCylinder');
simulator.LoggingInfo.Signals = [PistonPosition,Pressures];
```

Specify the piston position step response requirement of rise time of less than 0.04 seconds and settling time less than of 0.05 seconds.

```
PistonResponse = sdo.requirements.StepResponseEnvelope;
set(PistonResponse, ...
'RiseTime', 0.04, ...
'FinalValue', 0.04, ...
'SettlingTime', 0.05, ...
'PercentSettling', 1);
```

Specify the maximum cylinder pressure requirement of less than 1.75e6 N/m.

```
MaxPressure = sdo.requirements.SignalBound;
set(MaxPressure, ...
    'BoundTimes', [0 0.1], ...
    'BoundMagnitudes', [1.75e6 1.75e6], ...
    'Type', '<=');</pre>
```

For convenience, collect the performance requirements into a single structure to use later.

```
requirements = struct(...
'PistonResponse', PistonResponse, ...
'MaxPressure', MaxPressure);
```

Create Objective/Constraint Function

To optimize the cylinder cross-sectional area and piston spring constant, create a function to evaluate the cylinder design. This function is called at each optimization iteration.

Here, use an anonymous function with one argument that calls the sdoHydraulicCylinder design function.

```
evalDesign = @(p) sdoHydraulicCylinder_design(p,simulator,requirements);
```

The function:

- Has one input argument that specifies the cylinder cross-sectional area and piston spring constant values.
- Returns the optimization objective value and optimization constraint violation values.

The optimization solver minimizes the objective value and attempts to keep the optimization constraint violation values negative. Type help sdoExampleCostFunction for more details on how to write the objective/constraint function.

The sdoHydraulicCylinder_design function uses the simulator and requirements objects to evaluate the design. Type edit sdoHydraulicCylinder_design to examine the function in more detail.

type sdoHydraulicCylinder_design

```
function design = sdoHydraulicCylinder design(p,simulator,requirements)
%SDOHYDRAULICCYLINDER DESIGN
% The sdoHydraulicCylinder design function is used to evaluate a cylinder
% design.
%
% The |p| input argument is the vector of cylinder design parameters.
%
% The [simulator] input argument is a sdo.SimulinkTest object used to
% simulate the |sdoHydraulicCylinder| model and log simulation signals
%
% The |requirements| input argument contains the design requirements used
% to evaluate the cylinder design
%
% The |design| return argument contains information about the design
% evaluation that can be used by the |sdo.optimize| function to optimize
% the design.
%
% see also sdo.optimize, sdoExampleCostFunction
% Copyright 2011 The MathWorks, Inc.
%% Simulate the model
%
```

```
\% Use the simulator input argument to simulate the model and log model
% signals.
%
% First ensure that we simulate the model with the parameter values chosen
% by the optimizer.
%
simulator.Parameters = p;
% Simulate the model and log signals.
simulator = sim(simulator);
% Get the simulation signal log, the simulation log name is defined by the
% model |SignalLoggingName| property
logName = get param('sdoHydraulicCylinder','SignalLoggingName');
simLog = get(simulator.LoggedData,logName);
%% Evaluate the design requirements
%
% Use the requirements input argument to evaluate the design requirements
%
% Check the PistonPosition signal against the stepresponse requirement
%
PistonPosition = get(simLog, 'PistonPosition');
cPiston = evalRequirement(requirements.PistonResponse,PistonPosition.Values);
% Check the Pressure signals against the maximum requirement
%
Pressures = find(simLog, 'Pressures');
cPressure = evalRequirement(requirements.MaxPressure,Pressures.Values);
% Use the PistonResponse and MaxPressure requirements as non-linear
% constraints for optimization.
design.Cleg = [cPiston(:);cPressure(:)];
% Add design objective to minimize the Cylinder cross-sectional area
Ac = p(1);
                    %Since we called sdo.optimize(evalDesign,[Ac;K])
design.F = Ac.Value;
end
```

Evaluate the Initial Design

Call the objective function with the initial cylinder cross-sectional area and initial piston spring constant.

```
initDesign = evalDesign([Ac;K]);
```

The function simulates the model and evaluates the design requirements. The scope shows that the maximum pressure requirement is satisfied but the piston position step response requirement is not satisfied.

initDesign is a structure with the following fields:

• Cleq shows that some of the inequality constraints are positive indicating they are not satisfied by the initial design.

initDesign.Cleq

ans =

-0.3839

- -0.1861 -0.1836 -1.0000 0.3033 0.2909 0.1671 0.2326 -0.0480 -0.0480
- F shows the optimization objective value (in this case the cylinder cross-sectional area). The initial design cross-sectional area, as expected, has the same value as the initial cross-sectional area parameter Ac.

initDesign.F

ans =

1.0000e-03

Optimize the Design

Pass the objective function, initial cross-sectional area and piston spring constant values to sdo.optimize.

```
[pOpt,optInfo] = sdo.optimize(evalDesign,[Ac;K]);
```

Optimization started 02-Sep-2021 00:18:14

			max		First-order
Iter	F-count	f(x)	constraint	Step-size	optimality
Θ	5	0.001	0.3033		
1	11	0.00057281	0.07293	0.48	85.4
2	17	0.000391755	0	0.128	28
3	22	0.000387625	0	0.00291	0.00459
4	27	0.000382932	0	0.00331	0.00187
5	32	0.000378204	Θ	0.00331	0.000219
Local	minimum	found that	satisfies the	constraints.	

Optimization completed because the objective function is non-decreasing in feasible directions, to within the value of the optimality tolerance, and constraints are satisfied to within the value of the constraint tolerance.

The optimization repeatedly evaluates the cylinder design by adjusting the cross-sectional area and piston spring constant to meet the design requirements. From the scope, see that the maximum pressure and piston response requirements are met.

The sdo.optimize function returns:

• pOpt shows the optimized cross-sectional area and piston spring constant values.

p0pt

pOpt(1,1) =

```
Name: 'Ac'
Value: 3.7820e-04
Minimum: 3.1416e-04
Maximum: 0.0013
Free: 1
Scale: 0.0020
Info: [1x1 struct]
pOpt(2,1) =
Name: 'K'
Value: 1.5809e+04
Minimum: 10000
Maximum: 100000
Free: 1
Scale: 65536
Info: [1x1 struct]
```

2x1 param.Continuous

• optInfo is a structure that contains optimization termination information such as number of optimization iterations and the optimized design.

optInfo

```
optInfo =
struct with fields:
        Cleq: [10x1 double]
        F: 3.7820e-04
        Gradients: [1x1 struct]
        exitflag: 1
        iterations: 5
        SolverOutput: [1x1 struct]
            Stats: [1x1 struct]
```

For example, the Cleq field shows the optimized non-linear inequality constraints are all non-positive to within optimization tolerances, indicating that the maximum pressure and piston response requirements are satisfied.

optInfo.Cleq

ans =

-0.0972 -0.0131 -0.0131 -1.0000 -0.2064 -0.0047 -0.0069 -0.0004 -0.0476 -0.0476

The F field contains the optimized cross-sectional area. The optimized cross-sectional area value is nearly 50% less that the initial value.

optInfo.F

ans = 3.7820e-04

Update the Model Variable Values

By default, the model variables Ac and K are not updated at the end of optimization. Use the setValueInModel command to update the model variable values.

sdo.setValueInModel('sdoHydraulicCylinder',p0pt)

Related Examples

To learn how to optimize the cylinder design using the **Response Optimizer**, see "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103.

% Close the model
bdclose('sdoHydraulicCylinder')

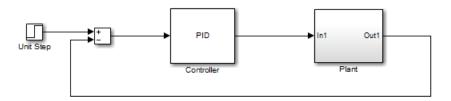
Design Optimization to Meet Custom Signal Requirements (GUI)

This example shows how to optimize a design to meet a custom signal requirement. You optimize the controller parameters to minimize the plant actuation signal energy while satisfying step response requirements.

1 Load a saved **Response Optimizer** session.

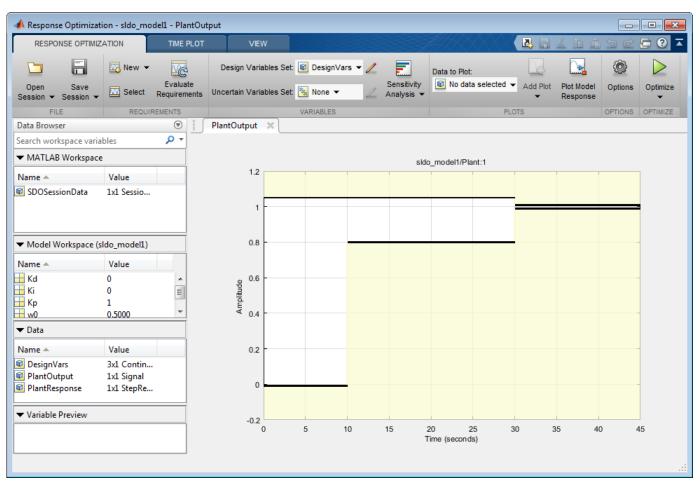
load sldo_model1_custom_signal_session
sdotool(SD0SessionData);

The following Simulink model opens.



The **Response Optimizer**, configured with the following settings, also opens:

- Step response characteristics, specified on the output of the Plant block, that the model output must satisfy:
 - Maximum overshoot of 5%
 - Maximum rise time of 10 seconds
 - Maximum settling time of 30 seconds
- Design variable set with the controller parameters $Kp,\,Ki$ and Kd. These parameters have a minimum value of 0.
- The variables for step requirements (PlantResponse), logged signal (PlantOutput) and design variables (DesignVars) which appear in the **Data** area.



- 2 Specify a signal to log. You apply the custom requirement on this logged signal.
 - a Select New > Signal.

A window opens where you select a signal to log.

b In the Simulink model window, click the output of the Controller block.

The window updates to display the selected signal.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set, enter PlantActuator.

Click **OK**. A new variable **PlantActuator** appears in the **Data** area.

3 Specify the custom requirement to apply to the signal.

The custom requirement calls the objective function sldo_model1_minimize_energy which returns the energy in the PlantActuator signal. The signal energy is minimized. This function accepts:

- An input argument data which is a structure with fields for the design variables in the **Data** area. Signals are logged for the nominal and uncertain parameter values if there are any.
- Returns the objective value to be minimized.

Tip To see the contents of this function, type edit sldo_model1_minimize_energy.

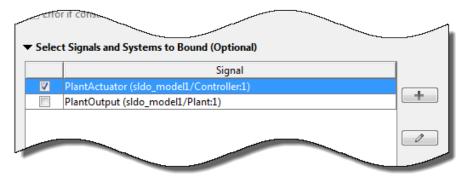
a Select New > Custom Requirement.

A window opens where you specify the custom requirement.

- **b** Specify MinimizeEnergy as the Name.
- c Specify @sldo_model1_minimize_energy as the Function.
- d Select Minimize the function output as the Type.

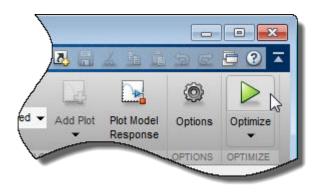
Create Requirement	×
Custom Requirement	
Create a custom requirement. The optimizer evaluates the optimization passing a structure with fields containing the cand logged simulation result	ptimized design variable values
Name: MinimizeEnergy	
Type: Minimize the function output	
Function: @sldo_model1_minimize_energy	
int is violated	

4 In the **Select Signals and Systems to Bound** area, select the **PlantActuator** check box to associate the custom requirement with that signal.



Click **OK**. A new variable appears in the **Data** area of the app. The window also updates to graphically display the custom signal requirement.

5 Click **Optimize**.



After a few iterations, the optimization converges to meet both the custom signal and step response requirements.

Iteration	F-count	MinimizeEnergy (Minimize)	PlantResponse (<=0)
0	5	328.9064	201.4682
1	12	3.5137e+04	8.5712
2	19	2.4809e+04	1.9011
3	26	3.4691e+04	0.3936
4	33	5.1240e+04	0.4952
5	40	5.0944e+04	0.0613
6	49	2.9535e+04	3.9387e-04
7	58	3.0751e+04	-0.0046
8	71	3.0748e+04	-0.0045
9	79	3.0585e+04	-0.0019
10	100	3.0585e+04	-0.0019
Optimizati	on converged, 2	w-2014 10:48:49 18-Nov-2014 10:47:23 written to 'DesignVars' in the Design Optin	nization workspace

6 Close the model.

```
setOption(sdotool('sldo_model1'),'NoPromptClose',true)
bdclose('sldo_model1')
```

See Also

Related Examples

• "Design Optimization Using Frequency-Domain Check Blocks (GUI)"

Design Optimization to Meet Frequency-Domain Requirements (GUI)

This example shows how to tune model parameters to meet frequency-domain requirements using the **Response Optimizer** app.

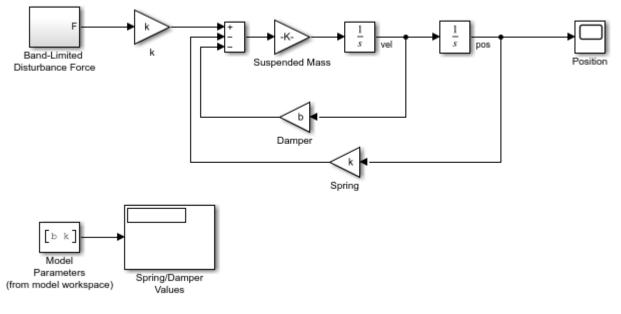
This example requires Simulink[®] Control Design[™].

Suspension Model

Open the Simulink model.

open_system('sdoSimpleSuspension')

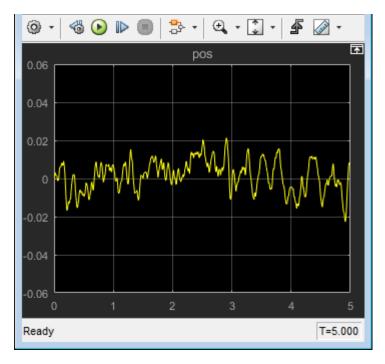
Simple Suspension System



Copyright 2002-2015 The MathWorks, Inc.

Mass-spring-damper models represent simple suspension systems and for this example you tune the system to meet typical suspension requirements. The model implements the second order system representing a mass-spring-damper using Simulink blocks and includes:

- a Mass gain block parameterized by the total suspended mass, m0+mLoad. The total mass is the sum of a nominal mass, |m0|, and a variable load mass, mLoad.
- a Damper gain block parameterized by the damping coefficient, b.
- a Spring gain block parameterized by the spring constant, k.
- two integrator blocks to compute the mass velocity and position.
- a Band-Limited Disturbance Force block applying a disturbance force to the mass. The disturbance force is assumed to be band-limited white noise.



Simulate the model to view the system response to the applied disturbance force.

Design Problem

The initial system has a bandwidth that is too high. This can be seen from the spiky position signal. You tune the spring and damper values to meet the following requirements:

- The -3dB system bandwidth must not exceed 10 rad/s.
- The damping ratio of the system must be less than 1/sqrt(2). This ensures that no frequencies in pass band are amplified by the system.
- Minimize the expected failure rate of the system. The expected failure rate is described by a Weibull distribution dependent on the mass, spring, and damper values.
- These requirements must all be satisfied as the load mass ranges from 0 to 20.

Open the Response Optimizer

In the Apps tab, click Response Optimizer under Control Systems.

Specify Design Variables

In the **Design Variables Set** list, select **New**. Add the b and k model variables to the design variable set.

- Specify the Minimum and Maximum values for the b variable as 100 and 10000 respectively.
- Specify the Minimum and Maximum values for the k variable as 10000 and 100000 respectively.

Create Desig	gn Variables Set								×
Create Desi	gn Variables set:	DesignVars							
	Variable	Value	Minimum	Maximum	Scale		Variable	Current value	Used By
V b		6283	100	10000	8192		NoiseBW	1000	/Band-Limited Disturbance Force/Band-Limited White Noise
🗹 k		98696	10000	100000	131072		NoisePower	100000	/Band-Limited Disturbance Force/Band-Limited White Noise
							m0	100	sdoSimpleSuspension/Suspended Mass
							mLoad	0	sdoSimpleSuspension/Suspended Mass
Variable	: Detail		Up	date model v	variables]	Specify expr	ression (e.g., s.x	or a(3)) OK Cancel Help

Click **OK**. A new variable, **DesignVars**, appears in the **Response Optimizer** browser.

In the **Uncertain Variables Set** list, select **New**. Add the mLoad variable to the uncertain variables set.

• Specify the Uncertain Values value for the mLoad variable as [10 15 20]

Create Uncertain Variables Set			×
Create Uncertain Variables set: UncVars			
Variable Nominal Value Uncertain Values	 Variable	Current value	Used By
MLoad 0 [101520]	NoiseBW	1000	/Band-Limited Disturbance Force/Band-Limited White Noise
	NoisePower	100000	/Band-Limited Disturbance Force/Band-Limited White Noise
	ь	6283	sdoSimpleSuspension/Damper impleSuspension/Model Parameters (from model
Sampling method	k	98696	/Band-Limited Disturbance Force/Band-Limited White Noise impleSuspension/Model Parameters (from model workspace) sdoSimpleSuspension/Spring
	m0	100	sdoSimpleSuspension/Suspended Mass
 Min and Max combinations only (2 combinations) All combinations (3 combinations) 			
			OK Cancel Help

Click OK. A new variable, UncVars, appears in the Response Optimizer browser.

Specify Linear Analysis Input/Output Points

Specify the input/output points defining the linear system used to compute the bandwidth and damping ratio.

To specify the input/output points:

- In the New list, select Linearization I/Os.
- In the Simulink model, click the signal at the output of the Band-Limited Disturbance Force block. The **Create linearization I/O set** dialog box is updated and the chosen signal appears in it.
- In the Create linearization I/O dialog box, select the signal and click Add.

• In the **Configuration** list for the selected signal, choose **Input Perturbation** to specify it as an input signal.

Create linearization I/O set					X
Variable name: IOs					
Filter by name 🔎 🔻		Analysis I/	'Os		
Currently selected signals		Active		Signal Description	Configuration
			Block:spensic Port: 1	on/Band-Limited Disturbance Force	
		✓	Signal Name:		±å- Input Perturbation 🚽
		_			
	Add >>				
				Highlight	sh Signal Names Delete
					OK Cancel Help

• Similarly, add the **pos** signal from the Simulink model. Specify this signal as an output. In the **Configuration** list, select **Output Measurement**.

Create linearization I/O set				X
Variable name: IOs				
Filter by name 🔎 🔻		Analysis I/	'Os	
Currently selected signals		Active	Signal Description	Configuration
sdoSimpleSuspension/x_dot : 1 (pos)		V	Block:spension/Band-Limited Disturbance Force Port: 1 Signal Name:	±å- Input Perturbation ▼
	Add >>		Block: sdoSimpleSuspension/x_dot Port: 1 Signal Name: pos	Output Measurement
			Highlight Refre	sh Signal Names Delete
				OK Cancel Help

• Click **OK**. A new variable, **IOs**, appears in the **Response Optimizer** browser.

Add Bandwidth and Damping-Ratio Requirements

Tune the spring and damper values to satisfy bandwidth and damping ratio requirements.

To specify the bandwidth requirement:

- Open a dialog to specify bounds on the Bode magnitude. In the **New** list, select **Bode Magnitude**.
- Specify the requirement name as Bandwidth.
- Specify the edge start frequency and magnitude as 10 rad/s and -3db, respectively.
- Specify the edge end frequency and magnitude as 100 rad/s and -3db, respectively.
- Specify the input/output set to which the requirement applies by clicking **Select Systems to Bound**. Select the **IOs** check box.

Create Requirement	t				×
		Bode Magn	itude		
Specify a	piecewise linear l	bound on the free	quency response	of a linear system	
Name: Bandwidth					
Bandwidti					
 Specify Magnit 	tude Bound				
Type: Constrain	system to be <= f	the bound 🔻			
Edge	Start	Edge	End		
Freq. (rad/s)	Mag. (dB)	Freq. (rad/s)	Mag. (dB)	Slope (dB/dec	+
10.0000	-3	100	-3	0	
L					
▼ Select System	s to Bound				
Snapshot Times:	[0]				
	[*]	Linearization	1/0		
IOs (sdo	SimpleSuspensio			e:1 [in]. sdoSim	
					+
					0
L					
Create Plot			OK	Cancel	Help

• Click **OK**. A new requirement, **Bandwidth**, appears in the **Response Optimizer** browser and a graphical view of the bandwidth requirement is automatically created.

To specify the damping ratio requirement:

- Open a dialog to specify bounds on the damping ratio. In the New list, select Damping Ratio.
- Specify the damping ratio bound value as 0.7071.
- Specify the input/output set to which the requirement applies by clicking **Select Systems to Bound**. Select the **IOs** check box.

Create Requirement	X
Damping Ratio	
Specify a bound on the damping ratio of the poles of a linear system.	
Name: DampingRatio	
▼ Specify Bound	
Damping ratio > 0.7071	
▼ Select Systems to Bound	
Snapshot Times: [0]	
Linearization I/O	
IOs (sdoSimpleSuspension/Band-Limited Disturbance Force:1 [in], sdoSim	
Create Plot OK Cancel Hel	р

• Click **OK**. A new requirement, **DampingRatio**, appears in the **Response Optimizer** browser and a graphical view of the damping ratio requirement is automatically created.

Add a Reliability Requirement

Tune the spring and damper values to minimize the expected failure rate over a lifetime of 100e3 miles. The failure rate is computed using a Weibull distribution on the damping ratio of the system. As the damping ratio increases the failure rate is expected to increase.

Specify the reliability requirement as a custom requirement:

- Open a dialog box to specify the custom requirements. In the **New** list, select **Custom Requirement**.
- Specify the custom requirement name as MinFailureRate.

- In the Specify Function area, select Minimize the function output from the Type list.
- Specify the function as @sdoSuspension FailureRate.

Create Requi	irement ×
	Custom Requirement
specified	a custom requirement. The optimizer evaluates the function during optimization passing a structure with aining the optimized design variable values and logged simulation results.
Name: Mir	nFailureRate
 Specify 	Function
Туре:	Minimize the function output 🔹
Function:	@sdoSuspension_FailureRate
Error if	f constraint is violated
Select S	Signals and Systems to Bound (Optional)
Create	Plot OK Cancel Help

 Click OK. A new requirement, MinFailureRate, appears in the Response Optimizer browser and a graphical view of the custom requirement is automatically created.

The @sdoSuspension_FailureRate function returns expected failure rate for a lifetime of 100e3 miles.

```
type sdoSuspension_FailureRate
```

```
function pFailure = sdoSuspension FailureRate(data)
%SDOSUSPENDION FAILURERATE
%
% The sdoSuspension FailureRate function is used to define a custom
\% requirement that can be used in the graphical SDTOOL environment.
%
% The |data| input argument is a structure with fields containing the
% design variable values chosen by the optimizer.
%
\% The <code>|pFailure|</code> return argument is the failure rate to be minimized by the
% SDOTOOL optimization solver. The failure rate is given by a Weibull
% distribution that is a function of the mass, spring and damper values.
% The design minimizes the failure rate for a 100e3 mile lifetime.
%
% Copyright 2012 The MathWorks, Inc.
%Get the spring and damper design values
allVarNames = {data.DesignVars.Name};
idx
           = strcmp(allVarNames, 'k');
```

```
k
          = data.DesignVars(idx).Value;
idx
          = strcmp(allVarNames, 'b');
h
           = data.DesignVars(idx).Value;
%Get the nominal mass from the model workspace
wksp = get param('sdoSimpleSuspension', 'ModelWorkspace');
  = evalin(wksp,'m0');
m
%The expected failure rate is defined by the Weibull cumulative
%distribution function, 1-exp(-(x/l)^k), where k=3, l is a function of the
%mass, spring and damper values, and x the lifetime.
d
        = b/2/sqrt(m*k);
pFailure = 1-exp(-(100e3*d/250e3)^3);
end
```

Optimize the Design

Before running the optimization be sure to have completed the earlier steps. Alternatively, you can load the sdoSimpleSuspension_sdosession from the model workspace into the **Response Optimizer**.

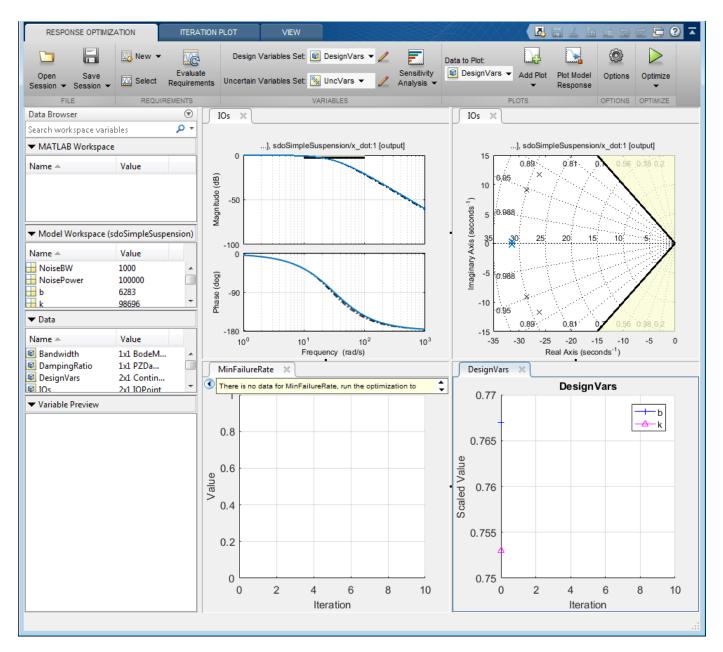
To save the initial design variable values and later compare them with the optimized values configure the optimization:

- Click **Options**.
- Select the Save optimized variable values as new design variable set option.

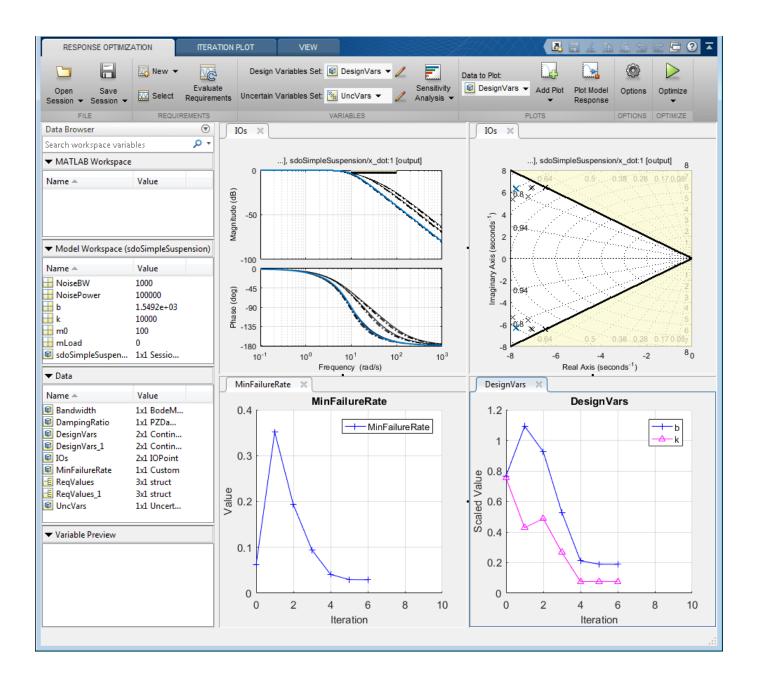
To study how the design variable values change during optimization:

- In the **Data to Plot** list, select **DesignVars**.
- In the Add Plot list, and select Iteration Plot.
- View the design variables in an appropriately scaled manner. Right-click on the DesignVars plot and select Show scaled values.

To evaluate the requirements at the initial design point, click **Evaluate Requirements**. The requirement plots are updated and a **ReqValues** variable is added to the **Response Optimizer** browser.



To optimize the design, click **Optimize**. The plots are updated during optimization. At the end of optimization, the optimal design values are written to the DesignVars1 variable. The requirement values for the optimized design are written to the ReqValues1 variable.

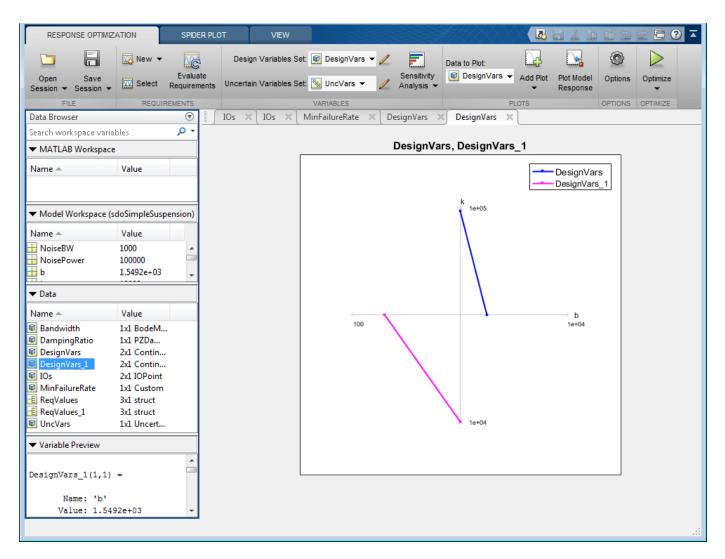


Iteration	F-count	MinFailureRate (Minimize)	Bandwidth (<=0)	DampingRatio (>=0)	
0	5	0.0620	0.7642	0.2910	-
1	10	0.3513	-0.6699	0.4142	
2	15	0.1933	-0.0462	0.4142	
3	20	0.0942	0.0253	0.4142	
4	24	0.0408	-0.5935	0.1192	
5	28	0.0297	-0.2810	0.0047	Ξ
6	32	0.0293	-0.2674	-1.5701e-16	
Optimizat	ion converge	9-Dec-2015 18:37:37 ed, 29-Dec-2015 18:39:53 lues written to 'DesignVars 1	' in the Design Optimization v	vorkspace	•

Analyze the Design

To compare design variables before and after optimization:

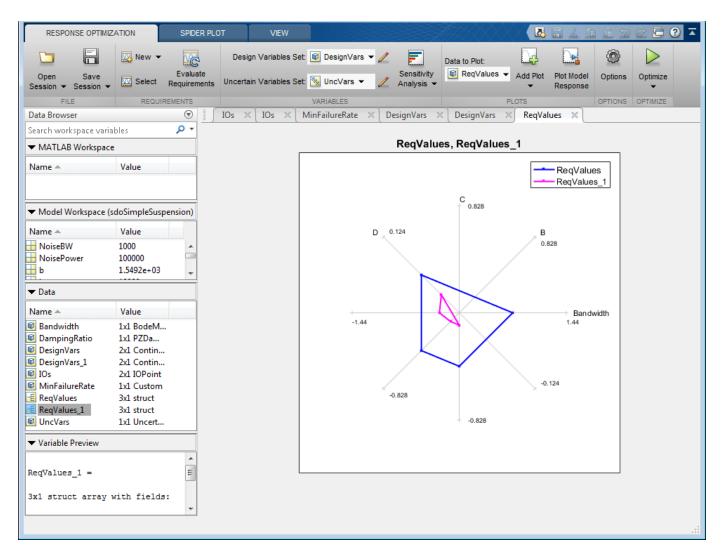
- In the **Data to Plot** list, select **DesignVars**.
- In the Add Plot list, select Spider Plot.
- To add the optimized design variables to the same plot, select DesignVars1 in the Response Optimizer browser and drag it onto the Spider plot. Alternatively, in the Data to Plot list, select DesignVars1. Then, in the Add Plot list, select Spider plot 1 from the Add to Existing Plot section.



The plot shows that the optimizer reduced both the k and b values for the optimal design.

To compare requirements before and after optimization:

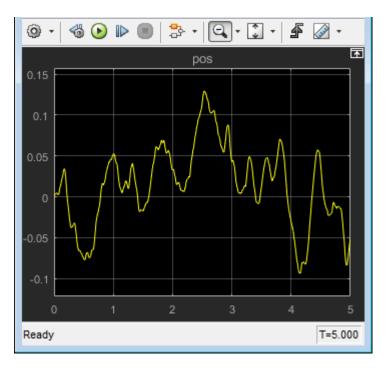
- In the Data to Plot list, select ReqValues.
- In the Add Plot list, select Spider Plot.
- To add the optimized requirement values to the same plot, select ReqValues1 in the Response **Optimizer** browser and drag it onto the Spider plot. Alternatively, in the **Data to Plot** list, select ReqValues1. Then, in the **Add Plot** list, select **Spider plot 2** from the **Add to Existing Plot** section.



The plot shows that the optimal design has a lower failure rate (the MinFailureRate axis) and better satisfies the bandwidth requirement. The value plotted on the bandwidth axis is the difference between the bandwidth bound and the bandwidth value. The optimization satisfies the bound by keeping this value negative; a more negative value indicates better satisfaction of the bound.

The improved reliability and bandwidth are achieved by pushing the damping ratio closer to the damping ratio bound. The plot has two axes for the damping ratio requirement, one for each system pole, and the plotted values are the difference between the damping ratio bound and the damping ratio value. The optimization satisfies the bound by keeping this value negative.

Finally the simulated mass position is smoother than the initial position response (indication of a lower bandwidth as required) at the expense of larger position deflection.



Related Examples

To learn how to optimize the suspension design using the sdo.optimize command, see "Design Optimization to Meet Frequency-Domain Requirements (Code)" on page 3-216.

```
% Close the model
bdclose('sdoSimpleSuspension')
```

Specify Custom Signal Objective with Uncertain Variable (GUI)

This example shows how to specify a custom objective function for a model signal. You calculate the objective function value using a variable that models parameter uncertainty.

Competitive Population Dynamics Model

The Simulink® model sdoPopulation models a simple two-organism ecology using the competitive Lotka-Volterra equations:

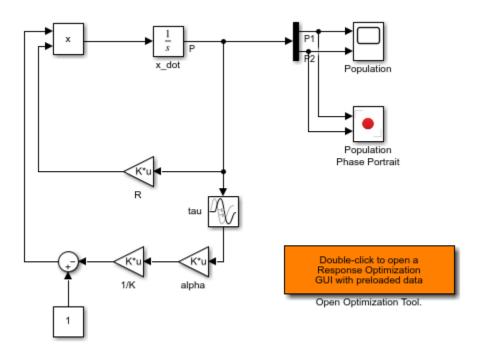
$$\frac{dP_1}{dt} = R_1 P_1 (1 - \frac{P_1(t - \tau_1) + \alpha P_2(t - \tau_2)}{K})$$
$$\frac{dP_2}{dt} = R_2 P_2 (1 - \frac{P_2(t - \tau_2) + \alpha P_1(t - \tau_1)}{K})$$

- P_n is the population size of the n-th organism.
- R_n is the inherent per capita growth rate of each organism.
- τ_n is the competitive delay for each organism.
- *K* is the carrying capacity of the organism environment.
- α is the proximity of the two populations and how strongly they affect each other.

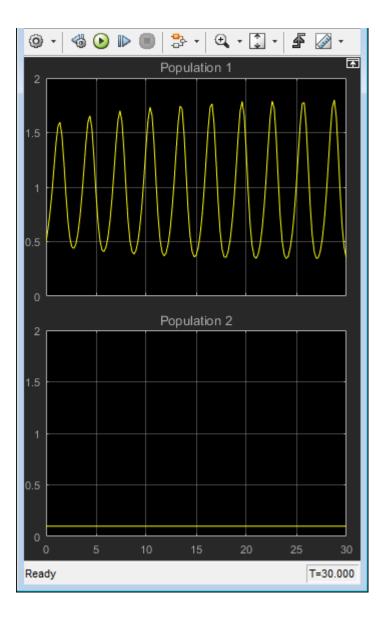
The model uses normalized units.

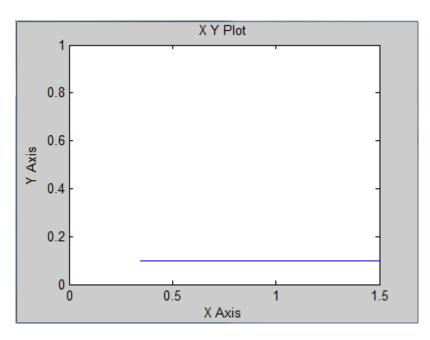
Open the model.

```
open_system('sdoPopulation')
```



Copyright 2012-2013 The MathWorks, Inc.





The two-dimensional signal, P, models the population sizes for P1 (first element) and P2 (second element). The model is initially configured with one organism, P1, dominating the ecology. The Population scope shows the P1 population oscillating between high and low values, while P2 is constant at 0.1. The Population Phase Portrait block shows the population sizes of the two organisms in relation to each other.

Population Stabilization Design Problem

Tune the R_2 , au_2 , and lpha values to meet the following design requirements.

- Minimize the population range, that is, the maximum difference between P1 and P2.
- Stabilize P1 and P2, that is, ensure that neither organism population dies off or grows extremely large.

You must tune the parameters for different values of the carrying capacity, K. This ensures robustness to environment carrying-capacity uncertainty.

Open Response Optimizer

Double-click the Open Optimization Tool block in the model to open a pre-configured **Response Optimizer** session. The session specifies the following variables:

- DesignVars Design variables set for the R_2 , τ_2 , and α model parameters.
- K_unc Uncertain parameter modeling the carrying capacity of the organism environment (*K*). K_unc specifies the nominal value and two sample values.
- P1 and P2 Logged signals representing the populations of the two organisms.

Specify Custom Signal Objective Function

Specify a custom requirement to minimize the maximum difference between the two population sizes. Apply this requirement to the P1 and P2 model signals.

- **1** Open the Create Requirement dialog box. In the **New** list, select **Custom Requirement**.
- 2 Specify the following in the Create Requirement dialog box:
- Name Enter PopulationRange.
- Type Select Minimize the function output from the list.
- Function Enter @sdoPopulation_PopRange. For more information about this function, see Custom Signal Objective Function Details.
- Select Signals and Systems to Bound (Optional) Select the P1 and P2 check boxes.

Create Requi	irement	×
	Custom Requirement	
optimizati	e a custom requirement. The optimizer evaluates the specified function du ion passing a structure with fields containing the optimized design variab and logged simulation results. pulationRange	-
▼ Specify	Function	
Туре:	Minimize the function output 🔹	
Function:	@sdoPopulation_PopRange	
Error in	f constraint is violated	
▼ Select S	Signals and Systems to Bound (Optional)	
	Signal	
V P	1 (sdoPopulation/Demux:1)	
P	2 (sdoPopulation/Demux:2)	+
		0
Snapshot [*]	Times: [0]	
	Linearization I/O	
Create Lin	nearization I/Os defining a system so that it can be used in requirements.	+
✓ Create	Plot OK Cancel	Help .:

3. Click **OK**.

A new variable, PopulationRange, appears in the **Response Optimizer** browser.

Custom Signal Objective Function Details

PopulationRange uses the sdoPopulation_PopRange function. This function computes the maximum difference between the populations, across different environment carrying capacity values. By minimizing this value, you can achieve both design goals. The function is called by the optimizer at each iteration step.

To view the function, type edit sdoPopulation_PopRange. The following discusses details of this function.

Input/Output

The function accepts data, a structure with the following fields:

- DesignVars Current iteration values of R_2 , τ_2 , and α .
- Nominal Logged signal data, obtained by simulating the model using parameter values specified by data.DesignVars and nominal values for all other parameters. The Nominal field is itself a structure with fields for each logged signal. The field names are the logged signal names. The custom requirement uses the logged signals, P1 and P2. Therefore, data.Nominal.P1 and data.Nominal.P2 are timeseries objects corresponding to P1 and P2.
- Uncertain Logged signal data, obtained by simulating the model using the sample values of the uncertain variable K_unc. The Uncertain field is a vector of N structures, where N is the number of sample values specified for K_unc. Each element of this vector is similar to data.Nominal and contains simulation results obtained from a corresponding sample value specified for K_unc.

The function returns the maximum difference between the population sizes across different carrying capacities. The following code snippet in the function performs this action:

val = max(maxP(1) - minP(2), maxP(2) - minP(1));

Data Time Range

When computing the design goals, discard the initial population growth data to eliminate biases from the initial-condition. The following code snippet in the function performs this action:

```
%Get the population data
tMin = 5; %Ignore signal values prior to this time
iTime = data.Nominal.P1.Time > tMin;
sigData = [data.Nominal.P1.Data(iTime), data.Nominal.P2.Data(iTime)];
```

iTime represents the time interval of interest, and the columns of sigData contain P1 and P2 data for this interval.

Optimization for Different Values of Carrying Capacity

The function includes the effects of varying the carrying capacity by iterating through the elements of data.Uncertain. The following code snippet in the function performs this action:

```
for ct=1:numel(data.Uncertain)
```

```
iTime = data.Uncertain(ct).P1.Time > tMin;
sigData = [data.Uncertain(ct).P1.Data(iTime), data.Uncertain(ct).P2.Data(iTime)];
maxP = max([maxP; max(sigData)]); %Update maximum if new signals are bigger
minP = min([minP; min(sigData)]); %Update minimum if new signals are smaller
end
...
```

The maximum and minimal populations are obtained across all the simulations contained in data.Uncertain.

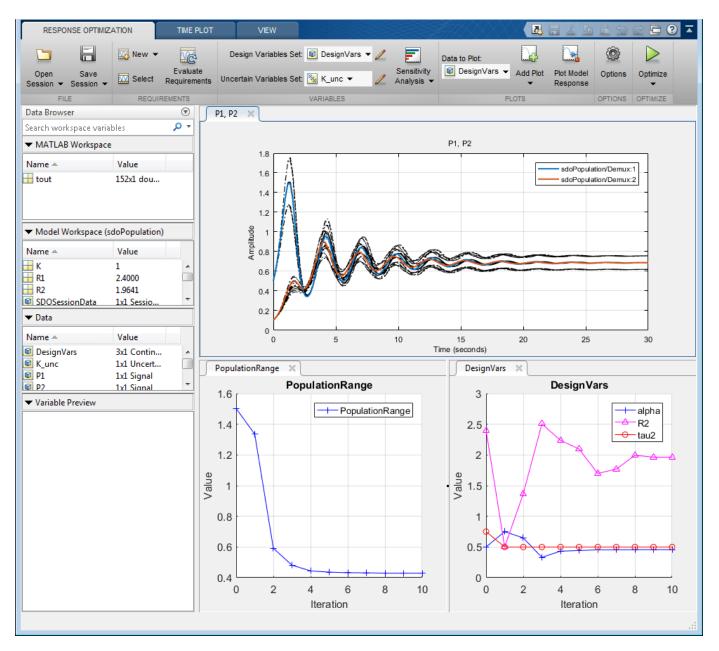
Optimize Design

Click Optimize.

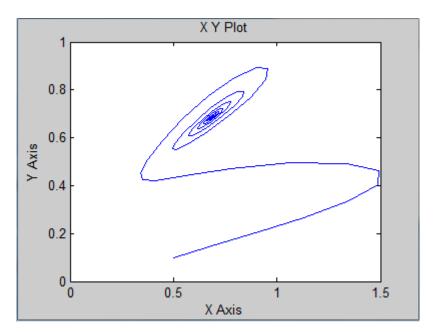
The optimization converges after a number of iterations.

teration	F-count	PopulationRange	
		(min)	
0	8		1.5024
1	13		1.3350
2	20		0.5918
3	29		0.4814
4	36		0.4443
5			0.4361
6	51		0.4322
7			0.4316
8	65		0.4294
9	77		0.4294
10	88		0.4294
Optimizati	on started 0	9-Oct-2012 12:01:35	
otimizati		ed, 09-Oct-2012 12:03:32	
-particulario	en oonreig		
ptimized	variable va	lues written to 'DesignVars' in the Design Optimization workspace	-
		Save Iteration Display Options	Optimize

The P1, P2 plot shows the population dynamics, with the first organism population in blue and the second organism population in red. The dotted lines indicate the population dynamics for different environment capacity values. The PopulationRange plot shows that the maximum difference between the two organism populations reduces over time.



The **Population Phase Portrait** block shows the populations initially varying, but they eventually converge to stable population sizes.



% Close the model.
bdclose('sdoPopulation')

Design Optimization with Uncertain Variables (Code)

This example shows how to optimize a design when there are uncertain variables. You optimize the dimensions of a Continuously Stirred Tank Reactor (CSTR) to minimize product concentration variation and production cost in case of varying, or uncertain, feed stock.

Continuously Stirred Tank Reactor (CSTR) Model

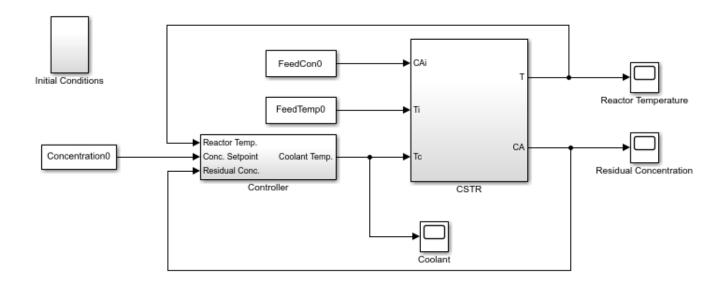
Continuously Stirred Tank Reactors (CSTRs) are common in the process industry. The Simulink® model, sdoCSTR, models a jacketed diabatic (i.e., non-adiabatic) tank reactor described in [1]. The CSTR is assumed to be perfectly mixed, with a single first-order exothermic and irreversible reaction, $A \rightarrow B$. A, the reactant, is converted to B, the product.

In this example, you use the following two-state CSTR model, which uses basic accounting and energy conservation principles:

$$\begin{aligned} \frac{dC_A}{dt} &= \frac{F}{A*h}(C_{feed} - C_A) - r*C_A\\ \frac{dT}{dt} &= \frac{F}{A*h}(T_{feed} - T) - \frac{H}{c_p\rho}r - \frac{U}{c_p*\rho*h}(T - T_{cool})\\ r &= k_0*e^{\frac{-E}{R*T}} \end{aligned}$$

- *C_A*, and *C_{feed}* Concentrations of A in the CSTR and in the feed [kgmol/m³]
- T, T_{feed} , and T_{cool} CSTR, feed, and coolant temperatures [K]
- F and ρ Volumetric flow rate [m^3/h] and the density of the material in the CSTR [1/m^3]
- h and A Height [m] and heated cross-sectional area [m^2] of the CSTR.
- k_0 Pre-exponential non-thermal factor for reaction $A \rightarrow B$ [1/h]
- E and H Activation energy and heat of reaction for $A \rightarrow B$ [kcal/kgmol]
- *R* Boltzmann's gas constant [kcal/(kgmol * K)]
- \mathcal{G}_{P} and U Heat capacity [kcal/K] and heat transfer coefficients [kcal/(m^2 * K * h)] Open the Simulink model.

open system('sdoCSTR');



Copyright 2012-2015 The MathWorks, Inc.

The model includes a cascaded PID controller in the Controller subsystem. The controller regulates the reactor temperature, T, and reactor residual concentration, C_A .

CSTR Design Problem

Assume that the CSTR is cylindrical, with the coolant applied to the base of the cylinder. Tune the CSTR cross-sectional area, A, and CSTR height, h, to meet the following design goals:

- Minimize the variation in residual concentration, C_A . Variations in the residual concentration negatively affect the quality of the CSTR product. Minimizing the variations also improves CSTR profit.
- Minimize the mean coolant temperature T_{cool} . Heating or cooling the jacket coolant temperature is expensive. Minimizing the mean coolant temperature improves CSTR profit.

The design must allow for variations in the quality of supply feed concentration, C_{feed} , and feed temperature, T_{feed} . The CSTR is fed with feed from different suppliers. The quality of the feed differs from supplier to supplier and also varies within each supply batch.

Specify Design Variables

Select the following model parameters as design variables for optimization:

- Cylinder cross-sectional area A
- Cylinder height h
- p = sdo.getParameterFromModel('sdoCSTR', {'A', 'h'});

Limit the cross-sectional area to a range of [1 2] m².

p(1).Minimum = 1; p(1).Maximum = 2; Limit the height to a range of [1 3] m.

p(2).Minimum = 1; p(2).Maximum = 3;

Specify Uncertain Variables

Select the feed concentration and feed temperature as uncertain variables. You evaluate the design using different values of feed temperature and concentration.

```
pUnc = sdo.getParameterFromModel('sdoCSTR',{'FeedCon0','FeedTemp0'});
```

Create a parameter space for the uncertain variables. Use normal distributions for both variables. Specify the mean as the current parameter value. Specify a variance of 5% of the mean for the feed concentration and 1% of the mean for the temperature.

```
uSpace = sdo.ParameterSpace(pUnc);
uSpace = setDistribution(uSpace, 'FeedCon0', makedist('normal', pUnc(1).Value, 0.05*pUnc(1).Value));
uSpace = setDistribution(uSpace, 'FeedTemp0', makedist('normal', pUnc(2).Value, 0.01*pUnc(2).Value))
```

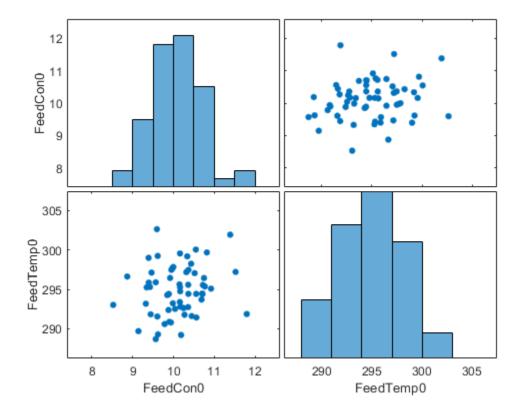
The feed concentration is inversely correlated with the feed temperature. Add this information to the parameter space.

```
%uSpace.RankCorrelation = [1 -0.6; -0.6 1];
```

The rank correlation matrix has a row and column for each parameter with the (i,j) entry specifying the correlation between the i and j parameters.

Sample the parameter space. The scatter plot shows the correlation between concentration and temperature.

```
rng('default'); %For reproducibility
uSmpl = sdo.sample(uSpace,60);
sdo.scatterPlot(uSmpl)
```



Ideally you want to evaluate the design for every combination of points in the design and uncertain spaces, which implies 30*60 = 1800 simulations. Each simulation takes around 0.5 sec. You can use parallel computing to speed up the evaluation. For this example you instead only use the samples that have maximum & minimum concentration and temperature values, reducing the evaluation time to around 1 min.

```
[~,iminC] = min(uSmpl.FeedCon0);
[~,imaxC] = max(uSmpl.FeedCon0);
[~,iminT] = min(uSmpl.FeedTemp0);
[~,imaxT] = max(uSmpl.FeedTemp0);
uSmpl = uSmpl(unique([iminC,imaxC,iminT,imaxT]) ,:);
```

Specify Design Requirements

The design requirements require logging model signals. During optimization, the model is simulated using the current value of the design variables. Logged signals are used to evaluate the design requirements.

Log the following signals:

CSTR concentration, available at the second output port of the sdoCSTR/CSTR block

```
Conc = Simulink.SimulationData.SignalLoggingInfo;
Conc.BlockPath = 'sdoCSTR/CSTR';
Conc.OutputPortIndex = 2;
Conc.LoggingInfo.NameMode = 1;
Conc.LoggingInfo.LoggingName = 'Concentration';
```

• Coolant temperature, available at the first output of the sdoCSTR/Controller block

```
Coolant = Simulink.SimulationData.SignalLoggingInfo;
Coolant.BlockPath = 'sdoCSTR/Controller';
Coolant.OutputPortIndex = 1;
Coolant.LoggingInfo.NameMode = 1;
Coolant.LoggingInfo.LoggingName = 'Coolant';
```

Create and configure a simulation test object to log the required signals.

```
simulator = sdo.SimulationTest('sdoCSTR');
simulator.LoggingInfo.Signals = [Conc,Coolant];
```

Create Objective/Constraint Function

Create a function to evaluate the CSTR design. This function is called at each optimization iteration.

Use an anonymous function with one argument that calls the sdoCSTR design function.

```
evalDesign = @(p) sdoCSTR_design(p,simulator,pUnc,uSmpl);
```

The evalDesign function:

- Has one input argument that specifies the CSTR dimensions
- Returns the optimization objective value

The sdoCSTR_design function uses a for loop that iterates through the sample values specified for the feed concentration. Within the loop, the function:

- Simulates the model using the current iterate, feed concentration, and feed temperature values
- · Calculates the residual concentration variation and coolant temperature costs

To view the objective function, type edit sdoCSTR_design.

Evaluate Initial Design

Call the evalDesign function with the initial CSTR dimensions.

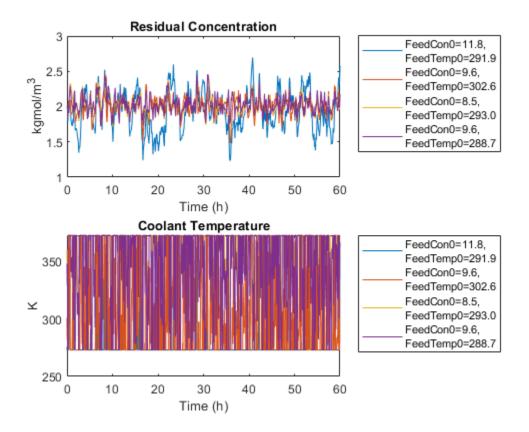
```
dInit = evalDesign(p)
```

```
dInit =
```

```
struct with fields:
F: 11.3356
costConc: 6.4390
costCoolant: 4.8965
```

Plot the model response for the initial design. Simulate the model using the sample feed concentration values. The plot shows the variation in the residual concentration and coolant temperature.

sdoCSTR_plotModelResponse(p,simulator,pUnc,uSmpl);



The sdoCSTR_plotModelResponse function plots the model response. To view this function, type edit sdoCSTR_plotModelResponse.

Optimize Design

Pass the objective function and initial CSTR dimensions to sdo.optimize.

```
pOpt = sdo.optimize(evalDesign,p)
```

```
Optimization started 02-Sep-2021 00:25:33
```

			max			First-order
Iter F-count		f(x)	constrai	int	Step-size	optimality
0	4	5.17535		0		
1	8	3.81522		0	2.01	7.83
2	12	2.63963		0	0.57	3.03
3	16	2.53426		0	0.159	0.292
4	20	2.50995		0	0.017	0.432
5	24	2.50266		0	0.125	1.44
6	28	2.50097		0	0.0966	0.234
7	32	2.47473		0	0.0135	0.0719
8	37	2.47473		0	0.000955	0.0694
Local	minimum	possible. Con	straints	sat	isfied.	

fmincon stopped because the size of the current step is less than the value of the step size tolerance and constraints are satisfied to within the value of the constraint tolerance.

```
pOpt(1,1) =
      Name: 'A'
      Value: 2
   Minimum: 1
   Maximum: 2
      Free: 1
      Scale: 0.5000
      Info: [1x1 struct]
pOpt(2,1) =
      Name: 'h'
      Value: 2.2850
   Minimum: 1
   Maximum: 3
      Free: 1
      Scale: 2
      Info: [1x1 struct]
```

2x1 param.Continuous

Evaluate Optimized Design

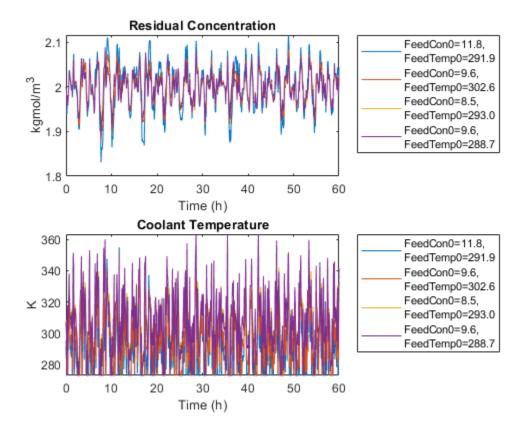
dFinal = evalDesign(pOpt)

Call the evalDesign function with the optimized CSTR dimensions.

```
dFinal =
struct with fields:
F: 2.4747
costConc: 1.4505
costCoolant: 1.0242
```

Plot the model response for the optimized design. Simulate the model using the sample feed concentration values. The optimized design reduces the residual concentration variation and average coolant temperature for different feed stocks.

sdoCSTR_plotModelResponse(p0pt,simulator,pUnc,uSmpl);



Related Examples

To learn how to use sensitivity analysis to explore the CSTR design space and select an initial design for optimization, see "Design Exploration Using Parameter Sampling (Code)" on page 4-157.

References

[1] Bequette, B.W. *Process Dynamics: Modeling, Analysis and Simulation*. 1st ed. Upper Saddle River, NJ: Prentice Hall, 1998.

```
% Close the model
bdclose('sdoCSTR')
```

Generate MATLAB Code for Design Optimization Problems (GUI)

This example shows how to automatically generate a MATLAB® function to solve a Design Optimization problem. You use the **Response Optimizer** to define an optimization problem for a hydraulic cylinder design and generate MATLAB code to solve this optimization problem.

Hydraulic Cylinder Design Problem

The "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103 example shows how to use the **Response Optimizer** to optimize a cylinder design. In this example we load a pre-configured **Response Optimizer** session based on that example.

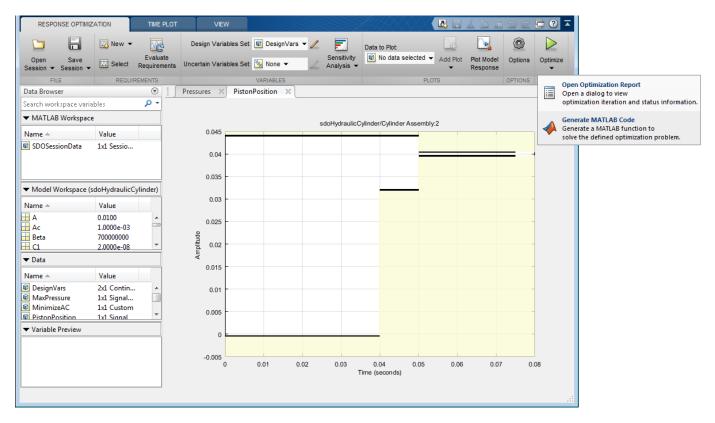
load sdoHydraulicCylinder_sdosession

Use the following command to open the Response Optimizer

sdotool(SD0SessionData)

Generate MATLAB Code

From the **Optimize** list, select **Generate MATLAB Code**.



The generated code is added to the MATLAB editor as an unsaved MATLAB function.

```
Editor - Untitled29*
                                                                                                          Ξ×
1
      function [Optimized DesignVars, Info] = sdo sdoHydraulicCylinder(DesignVars)
                                                                                                           .
2
     SDO SDOHYDRAULICCYLINDER
3
        읗
4
       % Solve a design optimization problem for the sdoHydraulicCylinder model.
                                                                                                          Ξ
5
       $
 6
       % The function returns optimized parameter values, Optimized DesignVars,
7
        % and optimization termination information, Info.
8
9
        % The, DesignVars, input argument defines the model parameters to optimize,
10
        % if omitted the parameters specified in the function body are optimized.
11
12
        % Modify the function to include or exclude new design requirements or
13
        % change the optimization options.
14
15
       % Auto-generated by SDOTOOL on 15-Mar-2012 13:14:45.
16
       읗
17
18
        %% Open the model.
19
       open_system('sdoHydraulicCylinder')
20
21
       %% Specify Design Variables
22
       몿
23
        % Specify model parameters as design variables to optimize.
        if nargin < 1 || isempty(DesignVars)</pre>
24
25
           DesignVars = sdo.getParameterFromModel('sdoHydraulicCylinder', {'Ac', 'K'});
26
           DesignVars(1).Minimum = 0.0003;
27
           DesignVars(1).Maximum = 0.0013;
28
           DesignVars(1).Scale = 0.001;
                                                   111
4
Untitled29*
         ×
```

Examine the generated code. Significant code portions are:

- Specify Design Variables Definition of the model parameters being optimized.
- Specify Design Requirements Definition of the design requirements.
- Create Optimization Objective Function Creation of an anonymous function that calls the subfunction sdoHydraulicCylinder_optFcn, which evaluates the cylinder design. sdo.optimize calls the anonymous function at each iteration.
- **Evaluate custom parameter requirement functions** Evaluates the custom requirement, MinimizeAC, that uses the sdoHydraulicCylinder_customObjective function.
- **Optimize the Design** Optimization using the sdo.optimize command.

Select Save from the MATLAB editor to save the generated function.

Run Generated Code

Run the generated function.

```
Command Window
New to MATLAB? Watch this <u>Video</u>, see <u>Demos</u>, or read <u>Getting Started</u>.
  >> [pOpt,optInfo] = sdo_sdoHydraulicCylinder;
  Optimization started 14-Mar-2012 12:04:23
                             max Step-size First-order
  Iter F-count
                   f(x) constraint
                                                  optimality
          5
                  0.001 0.3033
     0
         10 0.000414827
                              3.401
                                          4e+04
                                                       0.001
     1
         15 0.000506357
                             0.4482 6.74e+03
                                                    0.00148
     2
     3
        20 0.000537775
                            0.05098 4.36e+03
                                                      0.257
                                          10.1
     4
        25 0.000511641 0.003752
                                                     0.0977
         30 0.000503989 0.000875
     5
                                            335
                                                       0.001
 Local minimum found that satisfies the constraints.
 Optimization completed because the objective function is non-decreasing in
 feasible directions, to within the selected value of the function tolerance,
 and constraints are satisfied to within the selected value of the constraint tolerance.
f_{x} >>
```

 \odot

x

The first output argument, pOpt, contains the optimized parameter values and the second output argument, optInfo, contains optimization information.

Modify the Generated Code

You can:

- Modify the generated sdo_sdoHydraulicCylinder function to include or exclude new design requirements or change the optimization options.
- Call the generated sdo_sdoHydraulicCylinder function with a different set of parameters to optimize.

For details on how to write an objective/constraint function to use with the sdo.optimize command, type help sdoExampleCostFunction at the MATLAB command prompt.

Close the model.

Skip Model Simulation Based on Parameter Constraint Violation (GUI)

This example shows how to optimize a design and specify parameter-only constraints that prevent the model from being evaluated in an invalid solution space.

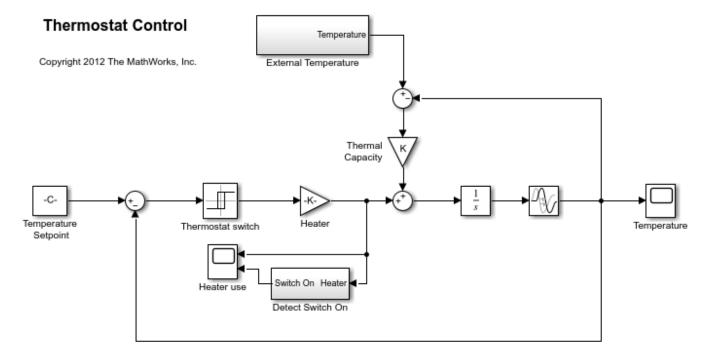
During optimization, the solver may try a design variable set that results in a model simulation error, which can be computationally expensive. If you can define a parameter-only constraint that identifies such a design variable set, then the solver can use the constraint to skip such sets. In other words, you can configure the optimization to be more efficient by disallowing design variable sets that lead to simulation errors.

In this example, you optimize thermostat settings to minimize temperature set-point deviations while satisfying some constraints. One of the constraints applies to the model parameters that define the thermostat switch on/switch off points. If the switch-off point is greater than the switch-on point, evaluating the model leads to a simulation error.

Thermostat Model

Open the model.

open_system('sdoThermostat');



The model describes a simple heater & thermostat that regulate the temperature of a room. The room is subject to external temperature fluctuations. The room temperature is computed using a first-order heat-flow equation:

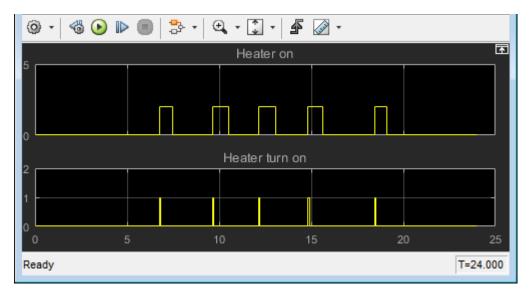
$$\frac{dT}{dt} = K(T_e - T) + Q$$

Where:

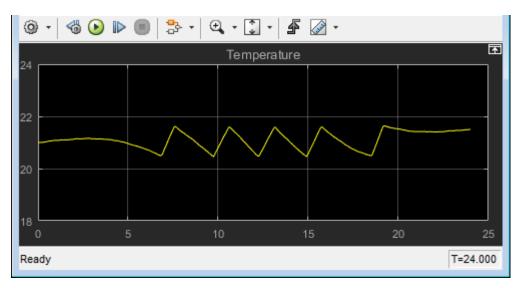
- *T* is the room temperature (C).
- T_e the external temperature (C).
- Q the heat supplied by the heater (W).
- *K* the room thermal capacity (J/C).

The heater is controlled by a thermostat that turns on when the difference between the room temperature and temperature set-point exceeds a threshold. The heater turns off when the error drops below a threshold.

The heater operation is displayed in the Heater use scope. The upper axis is the delivered heat and the lower axis shows the times when the heater is switched on.



The room temperature is displayed in the Temperature scope.



Thermostat Design Problem

You tune the thermostat turn-on and turn-off temperature thresholds, and also the heater power. The Thermostat switch block specifies the turn-on and turn-off thresholds using the variables H_on and H_off. The Heater block specifies the heater power using the variable Hgain.

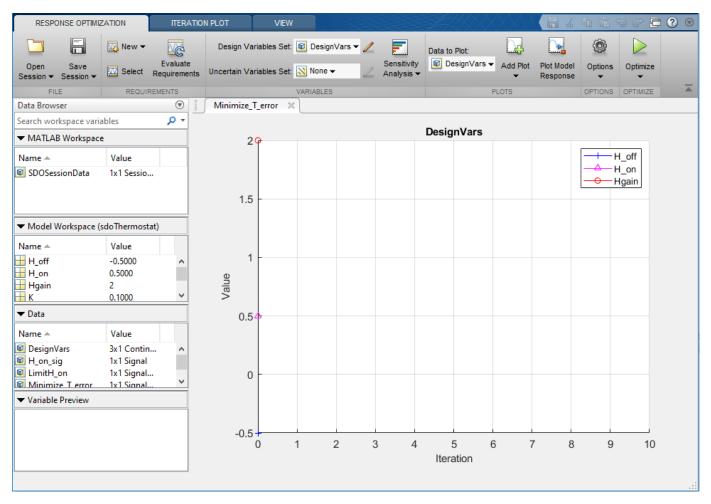
The design requirements are:

- Minimize the difference between the room temperature and temperature set-point over a 24 hour period.
- The heater must not turn on more than 12 times during the 24 hour period.
- The thermostat turn-on temperature must be greater than the thermostat turn-off temperature. If this constraint is violated, the model is invalid and cannot be simulated or evaluated.

Open the Response Optimizer

Open a pre-configured **Response Optimizer** session using the following commands.

load sdoThermostat_sdosession
sdotool(SDOSessionData)



The pre-configured session specifies the following variables:

- DesignVars Design variables set for the H_on, H_off, and Hgain model parameters.
- Minimize_T_error Requirement to minimize the temperature deviation from the set-point.
- LimitH_on Requirement to limit the number of times the thermostat is turned on.
- H_on_sig and T_error Logged signals. H_on_sig represents when the heater is on. T_error is the difference between the room temperature and the set-point.

Specify Parameter Constraint

The $H_on > H_off$ requirement is not yet defined. Use a custom requirement to specify this constraint and configure the requirement to error if it is not satisfied.

In the **New** drop-down list, select **Custom Requirement**. The Create Requirement dialog opens.

In this dialog, specify the following:

- Name SwitchConstraint.
- Type Select Constrain the function output to be >= 0 from the Type list.
- Function @sdoThermostat_SwitchingConstraint.
- Error if constraint is violated Select this check box.

Create Requi	irement ×					
	Custom Requirement					
Create a custom requirement. The optimizer evaluates the specified function during optimization passing a structure with fields containing the optimized design variable values and logged simulation results.						
Name: Swi	itchConstraint					
 Specify 	Function					
Туре:	Constrain function output to be $\geq = 0$ \bullet					
Function:	@sdoThermostat_SwitchingConstraint					
Error if	f constraint is violated					
Select Signals and Systems to Bound (Optional)						
Create	Plot OK Cancel Help					

The software calls the sdoThermostat_SwitchingConstraint function at each optimization iteration with a structure containing all the design variables. The output of the sdoThermostat_SwitchingConstraint function is the difference between the H_on and H_off values. This difference must be positive for the requirement to be satisfied.

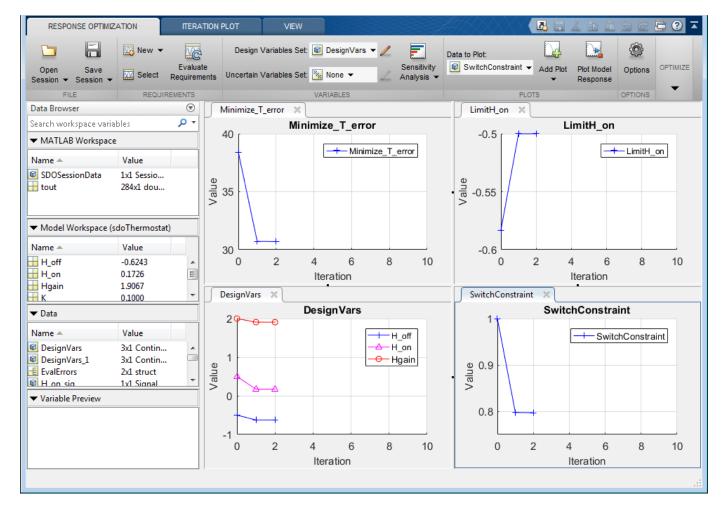
The software evaluates custom requirements that test parameter-only constraints, such as SwitchConstraint, before simulating the model and evaluating the remaining requirements.

- If the constraint is violated while the **Error if constraint is violated** check box is *selected*, the software does not simulate the model to evaluate the remaining requirements. Instead, the solver assigns the cost function a NaN value for this iteration, evaluates the terminating conditions, and continues.
- If the constraint is violated while the **Error if constraint is violated** check box is *cleared*, the solver will attempt to simulate the model to evaluate the remaining requirements. Simulating the model may lead to a hard error; for example, simulating the thermostat model when SwitchConstraint is violated will lead to an error. In this case, the solver assigns the cost function a NaN value for this iteration, evaluates the terminating conditions, and continues.

To examine the constraint function, type edit sdoThermostat_SwitchingConstraint. The requirement that $H_{on} > H_{off}$ is implemented as $H_{on} - H_{off} > 0$

Optimize the Design

Click Optimize.



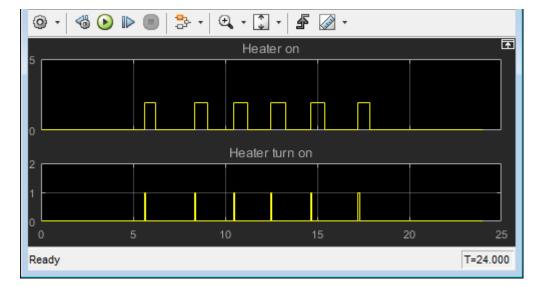
Iteration	F-count	SwitchConstraint (>=0)	Minimize_T_error (Minimize)	LimitH_on (<=0)
0	7	1	38.4084	-0.5833
1	28	0.7973	30.7171	-0.5000
2	133	0.7969	30.7003	-0.5000
Optimizati	on started 24-	Oct-2014 15:57:03		
Optimizati	on converged	. 24-Oct-2014 15:57:27		
	_		ation. Details of the errors have l	been written to
The optim	izer encounte	, 24-Oct-2014 15:57:27 red 2 errors during the optimiz o Optimization workspace.	ation. Details of the errors have l	been written to

The Optimization Progress window appears and updates at each iteration. The optimization successfully minimizes the temperature error while satisfying the switching constraints.

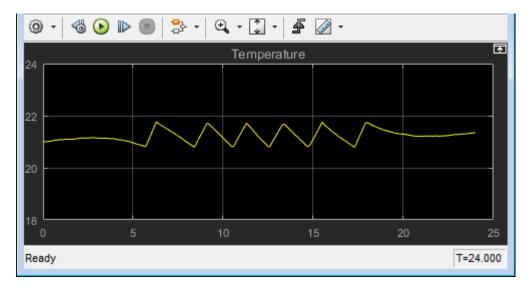
During this optimization, the H_on and H_off values never approach the H_on > H_off constraint boundary. So, there is never a danger of violating the constraint. However, changing the optimization algorithm may produce different behavior. For example, changing the optimization algorithm from the one used here, 'Interior-Point', to 'Active-Set' results in H_on and H_off values that are at the constraint boundary. This violation triggers the SwitchConstraint requirement and prevents model simulation for the relevant iterations.

View Optimized Model Response

Simulate the model with the optimized thermostat settings. The optimized heater operation is displayed in the Heater use scope where the upper axis is the delivered heat and the lower axis the heater switch on times.



The optimized room temperature is displayed in the Temperature scope.



Close the model.

bdclose('sdoThermostat')

Optimizing Parameters for Robustness

What Is Robustness?

A design is robust when its response does not violate design requirements under model parameter variations. Your model may contain parameters whose values are not precisely known. Such parameters vary over a given range of values and are defined as uncertain parameters. You may know the nominal value and the range of values in which these uncertain parameters vary.

You can use Simulink Design Optimization software to incorporate the parameter uncertainty to test the robustness of your design. When you optimize parameters for robustness, the optimization solver uses the responses computed using all the uncertain parameter values to adjust the design variable values.

You can specify the same parameter both as a design *and* uncertain variable. However, you cannot use a parameter both as a design and uncertain variable in the same optimization run. Also, you cannot add uncertainty to controller or plant parameters during optimization-based control design in the **Control System Designer**.

The uncertain variables can be scalar, vector, matrix or an expression.

You can test and optimize parameters for model robustness in the following ways:

- **Before Optimization**. Specify the parameter uncertainty *before* you optimize the parameters to meet the design requirements. In this case, the optimization method optimizes the signals based on both nominal parameter values as well as the uncertain values. This mode requires more computational time.
- After Optimization. Specify the parameter uncertainty *after* you have optimized the model parameters to meet design requirements. You can then test the effect of the uncertain parameters by plotting the model's response. If the response violates the design requirements, you can optimize the parameters again by including the parameter uncertainty during the optimization.

Related Examples

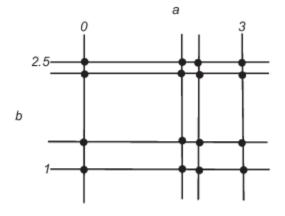
"Optimize Parameters for Robustness (GUI)" on page 3-172

More About

"Sampling Methods for Uncertain Parameters" on page 3-170

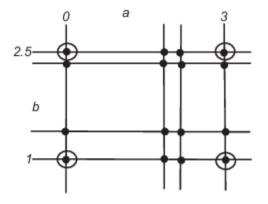
Sampling Methods for Uncertain Parameters

Sample values for uncertain parameters are a vector of numerical values. You can specify the vector yourself or generate a vector of random numbers using the software. The sample values you specify can be uniformly distributed or random. For example, four sample values for two uncertain parameters a and b in the range [0 3] and [1 2.5] may look like the following figure.



There are two methods to determine the number of sample values to use during optimization:

• Only the combination of minimum and maximum values (circled)



• Combination of the entire set of values (all solid dots in the previous figure)

Tip Using only the minimum and maximum values during optimization increases the computation speed when compared to using the entire set of values.

For the previous example, there are 4 combinations using the minimum and maximum values and 16 combinations if you use all sample values.

In the **Response Optimizer**, you specify the sampling method using the options as shown in the following figure.

Create U	ncertain Variables set: Unc	/ars		
V	Variable	Nominal Value	Uncertain Values	
	w0	0.5	[0.45 0.47 0.49 0.51 0.53 0.55]	
				H
			Set Uncertain Values	
Sampli	ing method			
C	n and Max combinations or combinations (6 combinati)		(

Related Examples

"Optimize Parameters for Robustness (GUI)" on page 3-172

More About

• "What Is Robustness?" on page 3-170

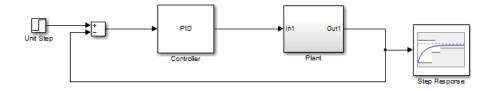
Optimize Parameters for Robustness (GUI)

This example shows how to optimize parameters for model robustness.

1 Load a saved **Response Optimizer** session.

```
load sldo_model1_desreq_optim_sdosession;
sdotool(SD0SessionData);
```

The sdotool command opens the following Simulink model and a saved **Response Optimizer** session.



The parameters of this model, Kp, Ki and Kd, have already been optimized to meet the following step response requirements:

- Maximum overshoot of 5%
- Maximum rise time of 10 seconds
- Maximum settling time of 30 seconds
- 2 Specify parameter uncertainty.
 - a In the Uncertain Variables Set drop-down list, select New.

ſ	RESPO	NSE OPTIMIZ	ATION	VIEW				X			$\sim \sim $
			🔤 New 👻	Ne	Design Variables Set:	😰 D	esignVars 🥆	1	E	Data to Plot:	
	Open Session 👻	Save Session 💌	w Select	Evaluate Requirements	Uncertain Variables Set:	<u></u> N	ione 💌	2	Sensitivity Analysis 💌	😰 No data	aselected
						EXIS	TING UNCER	RTAIN	VARIABLES		_
						111	None				
						CRE	ATE NEW UN	ICERT	AIN VARIABLE	s	
						÷			iables that ca ues during of		

A window opens where you specify uncertain variables.

eate Uncertain Variables set: UncVars Variable Nominal Value	Uncertain Values	Variable	Current value	Used By
		- Kd	0	sldo model1 desreg og
			0	sldo model1 desreg g
		Ki Kp	1	sido model1 desreg
		w0	0.5	
		zeta	0.5	sido model1 desreq
Sampling method	Set Uncertain Values]		
Min and Max combinations only (0 com	binations)			
All combinations (0 combinations)				
			C	OK Cancel He

- **b** Click w0 and zeta to select them.
- С

Click to add the selected parameters to an uncertain variables set.

Create U	Incertain Variable	es set: UncVars						
V	Variable	Nominal Value	U	ncertain Values	Variabl			
V	w0	0.5		[0.45 0.55]	Kd			
1	zeta	0.5		[0.45 0.55]	Ki			
				Set Uncertain Values				
Sampl Mi All								

The software displays the following parameter settings:

- Variable Parameter name
- Nominal Value Nominal value of the parameters as specified in the Simulink model
- **Uncertain Values** Values that the uncertain parameter can take. By default, the maximum and minimum values vary by 10% of the nominal value.

The total number of sample values to use during optimization is a combination of the maximum and minimum values of the uncertain parameters.

The check-box indicates that the parameter is included in the uncertain variable set. The default uncertain variable set name is UncVars.

Click **OK**. A new variable UncVars appears in **Data** area of the **Response Optimizer**.

Specify Random Values

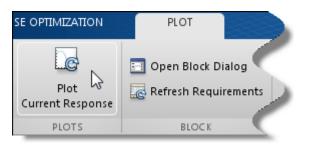
Instead of specifying sample values, you can auto-generate random values in a specific range. Select a parameter and click **Set Uncertain Values**.

Create U	Incertain V	ariables s	et: UncVars					
V	Variable	Nomi	Uncertain Values					
V	w0	0.5	[0.45 0.55]					
Set Uncertain Values								

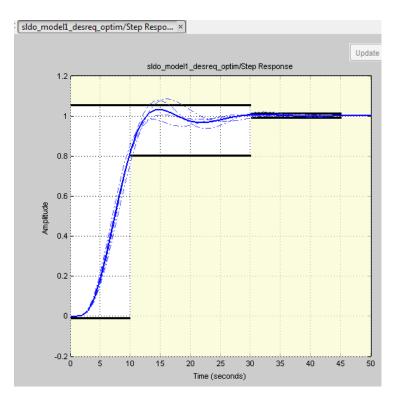
A window opens where you specify the range and the number of samples.

Generate uncertain values for 'w0' Nominal value: 0.5							
Distribution: Uniform							
Minimum:	0.45						
Maximum:	0.55						
Number of samples: 2							

- **3** Test the model robustness to the uncertain parameters.
 - a Click Plot Model Response.



The step response plot, displaying the requirements, updates.



- The solid curve corresponds to the model response computed using the optimized parameters and nominal values of the uncertain parameter.
- The four dashed curves correspond to the model response with the minimum and maximum values of the uncertain parameters.

The dashed plot lines show that the response during the period of 10 to 20 seconds violates the design requirements.

4 Optimize the parameters for model robustness. Click **Optimize**.



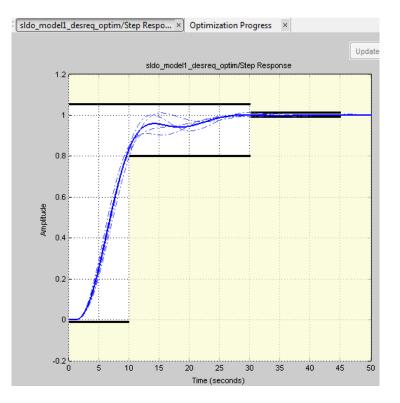
The Optimization Progress window opens which displays the optimization iterations.

After the optimization completes, the message **Optimization** converged indicates that the final model response computed by varying the uncertain parameters meets the specified design requirements.

teration	F-count	Step Response (Upper)
		(<=0)
0	7	0.0317
1	15	0.0031
2	23	4.8922e-05
3	31	5.0549e-09
Dotimizati	on started 0	4-Feb-2013 19:56:27
-		
Optimizati	on converge	d, 04-Feb-2013 19:56:58
		ues written to 'DesignVars' in the Design Optimization workspace

5 Examine the responses.

Tip To view only the final responses of the model, right-click the white area in the plot and uncheck **Responses > Show Iteration Responses**.



The final responses appear as the thick solid and dashed curves. The nominal and uncertain responses with parameter variations now meet the design requirements.

If your model contains referenced models, you can create an uncertain variable set using variables in the referenced models, using the **Create Uncertain Variables Set** dialog. For example, the first variable in the dialog box, Slew, is listed as sdoRateLimitedController:Slew. sdoRateLimitedController is the name of the referenced model with the variable Slew. The Slew variable has the same value for all instances of the sdoRateLimitedController model. In contrast, the variable Kd can have a different value for each instance of the referenced model containing it. For example, the second variable in the dialog box is listed as sdoMultipleMotors/Control_1:Kd. The upper-level model sdoMultipleMotors has block Control_1, which is a referenced model that has variable Kd. The value of this variable can be different than Kd in block Control_2, which is the third variable in the dialog box. To enable instance-specific values, Kd is specified as a model argument in the referenced model workspace.



More About

- "What Is Robustness?" on page 3-170
- "Sampling Methods for Uncertain Parameters" on page 3-170

•

See Also

Related Examples

"Design Optimization with Uncertain Variables (Code)" on page 3-152

Use Accelerator Mode During Simulations

About Accelerating Optimization

Simulink Design Optimization software supports Normal and Accelerator simulation modes. You can accelerate the design optimization computations by changing the simulation mode of your Simulink model to Accelerator. For information about these modes, see "How Acceleration Modes Work".

The default simulation mode is Normal. In this mode, Simulink uses interpreted code, rather than compiled C code during simulations.

In the Accelerator mode, Simulink Design Optimization software runs simulations during optimization with compiled C code. Using compiled C code speeds up the simulations and reduces the time to optimize the model response signals.

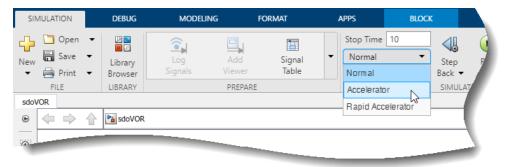
Limitations

If the model structure changes during optimization, the model is compiled to regenerate the C code for each iteration. In this case, using the Accelerator mode increases the computation time. To learn more about code regeneration, see "Code Regeneration in Accelerated Models".

Setting Accelerator Mode

To set the simulation mode to Accelerator, open the Simulink model window and perform one of the following actions:

• Under **Simulation**, choose Accelerator from the drop-down list as shown in the next figure.



Tip To obtain the maximum performance from the Accelerator mode, close all Scope blocks in your model.

See Also

More About

• "Ways to Speed Up Design Optimization Tasks"

Speed Up Response Optimization Using Parallel Computing

When to Use Parallel Computing for Response Optimization

You can use Simulink Design Optimization software with Parallel Computing Toolbox software to speed up the response optimization of a Simulink model. Using parallel computing may reduce model optimization time in the following cases:

- The model contains a large number of tuned parameters, and the Gradient descent method is selected for optimization.
- The Pattern search method is selected for optimization.
- The model contains a large number of uncertain parameters and uncertain parameter values.
- The model is complex and takes a long time to simulate.

When you use parallel computing, the software distributes independent simulations to run them in parallel on multiple MATLAB sessions, also known as *workers*. Distributing the simulations significantly reduces the optimization time because the time required to simulate the model dominates the total optimization time.

For information on how the software distributes the simulations and the expected speedup, see "How Parallel Computing Speeds Up Optimization" on page 3-180.

For information on configuring your system and using parallel computing, see "Use Parallel Computing for Response Optimization" on page 3-183.

How Parallel Computing Speeds Up Optimization

You can enable parallel computing with the Gradient descent and Pattern search optimization methods. When you enable parallel computing, the software distributes independent simulations during optimization on multiple MATLAB sessions. The following sections describe which simulations are distributed and the potential speedup using parallel computing.

Parallel Computing with the Gradient Descent Method

When you select Gradient descent as the optimization method, the model is simulated during the following computations:

- Constraint and objective value computation One simulation per iteration
- Constraint and objective gradient computations Two simulations for every tuned parameter per iteration
- Line search computations Multiple simulations per iteration

The total time, *Ttotal*, taken per iteration to perform these simulations is given by the following equation:

 $Ttotal = T + (Np \times 2 \times T) + (Nls \times T) = T \times (1 + (2 \times Np) + Nls)$

where T is the time taken to simulate the model and is assumed to be equal for all simulations, Np is the number of tuned parameters, and Nls is the number of line searches. Nls is difficult to estimate and you generally assume it to be equal to one, two, or three.

When you use parallel computing, the software distributes the simulations required for constraint and objective gradient computations. The simulation time taken per iteration when the gradient computations are performed in parallel, *TtotalP*, is approximately given by the following equation:

$$TtotalP = T + (ceil\left(\frac{Np}{Nw}\right) \times 2 \times T) + (Nls \times T) = T \times (1 + 2 \times ceil\left(\frac{Np}{Nw}\right) + Nls)$$

where *Nw* is the number of MATLAB workers.

Note The equation does not include the time overheads associated with configuring the system for parallel computing and loading Simulink software on the remote MATLAB workers.

The expected speedup for the total optimization time is given by the following equation:

$$\frac{TtotalP}{Ttotal} = \frac{1 + 2 \times ceil\left(\frac{Np}{Nw}\right) + Nls}{1 + (2 \times Np) + Nls}$$

For example, for a model with $N_{p}{=}3,\,N_{w}{=}4,$ and $N_{ls}{=}3,$ the expected speedup equals

$$\frac{1+2 \times ceil\left(\frac{3}{4}\right)+3}{1+(2\times3)+3} = 0.6$$

For an example of the performance improvement achieved with the **Gradient descent** method, see "Improving Optimization Performance Using Parallel Computing" on page 3-200.

Parallel Computing with the Pattern Search Method

The Pattern search optimization method uses search and poll sets to create and compute a set of candidate solutions at each optimization iteration.

The total time, *Ttotal*, taken per iteration to perform these simulations, is given by the following equation:

$$Ttotal = (T \times Np \times Nss) + (T \times Np \times Nps) = T \times Np \times (Nss + Nps)$$

where *T* is the time taken to simulate the model and is assumed to be equal for all simulations, *Np* is the number of tuned parameters, *Nss* is a factor for the search set size, and *Nps* is a factor for the poll set size. *Nss* and *Nps* are typically proportional to *Np*.

When you use parallel computing, Simulink Design Optimization software distributes the simulations required for the search and poll set computations, which are evaluated in separate parfor loops. The simulation time taken per iteration when the search and poll sets are computed in parallel, *TtotalP*, is given by the following equation:

$$\begin{split} TtotalP &= (T \times ceil(Np \times \frac{Nss}{Nw})) + (T \times ceil(Np \times \frac{Nps}{Nw})) \\ &= T \times (ceil(Np \times \frac{Nss}{Nw}) + ceil(Np \times \frac{Nps}{Nw})) \end{split}$$

where *Nw* is the number of MATLAB workers.

Note The equation does not include the time overheads associated with configuring the system for parallel computing and loading Simulink software on the remote MATLAB workers.

The expected speed up for the total optimization time is given by the following equation:

$$\frac{TtotalP}{Ttotal} = \frac{ceil(Np \times \frac{Nss}{Nw}) + ceil(Np \times \frac{Nps}{Nw})}{Np \times (Nss + Nps)}$$

For example, for a model with N_p=3, N_w=4, N_{ss}=15, and N_{ps}=2, the expected speedup equals $\frac{ceil(3 \times \frac{15}{4}) + ceil(3 \times \frac{2}{4})}{3 \times (15 + 2)} = 0.27.$

Note Using the Pattern search method with parallel computing may not speed up the optimization time. To learn more, see "Why do I not see the optimization speedup I expected using parallel computing?" on page 3-194

For an example of the performance improvement achieved with the Pattern search method, see "Improving Optimization Performance Using Parallel Computing" on page 3-200.

See Also

Related Examples

• "Use Parallel Computing for Response Optimization" on page 3-183

Use Parallel Computing for Response Optimization

Configure Your System for Parallel Computing

You can speed up model optimization using parallel computing on multicore processors or multiprocessor networks. Use parallel computing with the **Response Optimizer** and sdo.optimize to optimize using the fmincon, lsqnonlin, and patternsearch methods. Parallel computing is not supported for the fminsearch (**Simplex search**) method.

When you optimize model parameters using parallel computing, the software uses the available parallel pool. If none is available, and you select **Automatically create a parallel pool** in your Parallel Computing Toolbox preferences, the software starts a parallel pool using the settings in those preferences. To open a parallel pool that uses a specific cluster profile, use:

parpool(MyProfile);

MyProfile is the name of a cluster profile.

For information regarding creating a cluster profile, see "Add and Modify Cluster Profiles" (Parallel Computing Toolbox).

Model Dependencies

Model dependencies are any referenced models, data such as model variables, S-functions, and additional files necessary to run the model. Before starting the optimization, verify that the model dependencies are complete. Otherwise, you may get unexpected results.

Making Model Dependencies Accessible to Remote Workers

When you use parallel computing, the Simulink Design Optimization software helps you identify model dependencies. To do so, the software uses the Dependency Analyzer. The dependency analysis may not find all the files required by your model. To learn more, see "Dependency Analyzer Scope and Limitations". If your model has dependencies that are undetected or inaccessible by the parallel pool workers, then add them to the list of model dependencies.

The dependencies are made accessible to the parallel pool workers by specifying one of the following:

- File dependencies: the model dependency files are copied to the parallel pool workers.
- Path dependencies: the paths to the model dependencies are added to the paths of the parallel pool workers. If you are working in a multi-platform scenario, ensure that the paths are compatible across platforms.

Using file dependencies is recommended, however, in some cases it can be better to choose path dependencies. For example, if parallel computing is set up on a local multi-core computer, using path dependencies is preferred as using file dependencies creates multiple copies of the dependent files on the local computer.

For more information, see:

- "Optimize Design Using Parallel Computing (GUI)" on page 3-184 (Not supported in Simulink Online.)
- "Optimize Design Using Parallel Computing (Code)" on page 3-186

Optimize Design Using Parallel Computing (GUI)

To optimize a model response using parallel computing in the **Response Optimizer**:

1 Ensure that the software can access parallel pool workers that use the appropriate cluster profile.

For more information, see "Configure Your System for Parallel Computing" on page 3-183.

- 2 Open the **Response Optimizer** for the Simulink model.
- **3** Configure the design variables, design requirements, and, optionally, optimization settings.

For more information, see "Specify Design Variables" on page 3-56, "Specify Time-Domain Design Requirements in the App" on page 3-16, "Specify Frequency-Domain Design Requirements in the App" on page 3-43, and "Specify Optimization Options" on page 3-62.

4

On the **Response Optimization** tab, click Options to open the **Response Optimization Options** dialog box.

5 Select the **Parallel Options** tab.

Response Optimization Options		x
General Options Optimization Options	Parallel Options	Linearization Options
🔲 Use the parallel pool during optimiza	tion	
Model file dependencies Model path of	lependencies	
No model file dependencies specified.		
	4	Add file dependency Sync file dependencies from model
	ОК	Cancel Help

6 Select the **Use the parallel pool during optimization** check box.

This option checks for dependencies in your Simulink model. The file dependencies are displayed in the **Model file dependencies** list box, and corresponding path to the files in **Model path dependencies**. The files listed in **Model file dependencies** are copied to the remote workers.

Note The automatic dependencies check may not detect all the dependencies in your model.

For more information, see "Model Dependencies" on page 3-183. In this case, add the undetected dependencies manually.

Response Optimization Options	¢
General Options Optimization Options Parallel Options Linearization Options	
✓ Use the parallel pool during optimization	
Model file dependencies Model path dependencies	
C:\matlab\toolbox\parallelModelDependencies\file1.mat	
Add file dependency Sync file dependencies from model	
OK Cancel Help	-
	2

7 Add any file dependencies that the automatic check does not detect.

Specify the files in the **Model file dependencies** list box separated by semicolons or on separate lines.

Response Optimization Options	×
General Options Optimization Options Parallel Options Linearization Options	
✓ Use the parallel pool during optimization	
Model file dependencies Model path dependencies	
C:\matlab\toolbox\parallelModelDependencies\file1.mat; C:\myproject\file2.m C:\matlab\work\file3.mat	
Add file dependency Sync file dependencies from model	
OK Cancel Help	-

Alternatively, click Add file dependency to open a dialog box, and select the file to add.

Note If you do not want to copy the files to the remote workers, delete all entries in the **Model** file dependencies list box. Populate the **Model path dependencies** list box by clicking the **Sync path dependencies from model**, and add any undetected path dependencies. In addition,

in the list box, update the paths on local drives to make them accessible to remote workers. For example, change C:\ to \\\\hostname\\C\$\\.

8 If you modify the Simulink model, resync the dependencies to ensure that any new dependencies are detected. Click **Sync file dependencies from model** in the **Parallel Options** tab to rerun the automatic dependency check for your model.

This action updates the **Model file dependencies** list box with any new file dependency found in the model.

- 9 Click OK.
- **10** In the **Response Optimizer**, click **Optimize** to optimize the model response using parallel computing.

For information on troubleshooting problems related to optimization using parallel computing, see "Troubleshooting" on page 3-187.

Optimize Design Using Parallel Computing (Code)

To optimize a model response using parallel computing at the command line:

1 Ensure that the software can access parallel pool workers that use the appropriate cluster profile.

For more information, see "Configure Your System for Parallel Computing" on page 3-183.

- 2 Open the model.
- **3** Specify design requirements and design variables. For example see, "Design Optimization to Meet Step Response Requirements (Code)".
- 4 Enable parallel computing using an optimization option set, opt.

```
opt = sdo.OptimizeOptions;
opt.UseParallel = true;
```

5 Find the model dependencies.

```
[dirs,files] = sdo.getModelDependencies(modelname)
```

Note sdo.getModelDependencies may not detect all the dependencies in your model. For more information, see "Model Dependencies" on page 3-183. In this case, add the undetected dependencies manually.

6 Modify files to include any file dependencies that sdo.getModelDependencies does not detect.

```
files = vertcat(files, 'C:\matlab\work\filename.m')
```

Note If you do not want to copy the files to the remote workers, use the path dependencies. Add any undetected path dependencies to dirs and update the paths on local drives to make them accessible to remote workers. See sdo.getModelDependencies for more details.

7 Add the file dependencies for optimization.

```
opt.ParallelFileDependencies = files;
```

8 Run the optimization.

[pOpt,opt_info] = sdo.optimize(opt_fcn,param,opt);

For information on troubleshooting problems related to optimization using parallel computing, see "Troubleshooting" on page 3-187.

Troubleshooting

Why Are the Optimization Results With and Without Using Parallel Computing Different?

- Different numerical precision on the client and worker machines can produce marginally different simulation results. Thus, the optimization method can take a different solution path and produce a different result.
- When you use parallel computing with the Pattern search method, the search is more comprehensive and can result in a different solution.

To learn more, see "Parallel Computing with the Pattern Search Method" on page 3-181.

Why Don't I See the Optimization Speed up I Expected Using Parallel Computing?

- When you optimize a model that does not have a large number of parameters or does not take long to simulate, you might not see a speedup in the optimization time. In such cases, the overhead associated with creating and distributing the parallel tasks outweighs the benefits of running the optimization in parallel.
- Using the Pattern search method with parallel computing might not speed up the optimization time. Without parallel computing, the method stops the search at each iteration when it finds a solution better than the current solution. The candidate solution search is more comprehensive when you use parallel computing. Although the number of iterations might be larger, the optimization without using parallel computing might be faster.

To learn more about the expected speedup, see "Parallel Computing with the Pattern Search Method" on page 3-181.

Why Doesn't the Optimization Using Parallel Computing Make Any Progress?

To troubleshoot the problem:

- **1** Run the optimization for a few iterations without parallel computing to see if the optimization progresses.
- 2 Check whether the remote workers have access to all model dependencies. Model dependencies include data variables and files required by the model to run.

To learn more, see "Model Dependencies" on page 3-183.

Why Doesn't the Optimization Using Parallel Computing Stop When I Click the Stop Optimization Button?

When you use parallel computing with the Pattern search method, the software must wait until the current optimization iteration completes before it notifies the workers to stop. The optimization does not terminate immediately when you click **Stop**, and, instead, appears to continue running.

See Also

sdo.optimize | sdo.OptimizeOptions | sdo.getModelDependencies | parpool

Related Examples

 "Optimizing Time-Domain Response of Simulink Models Using Parallel Computing" on page 3-209

More About

- "Speed Up Response Optimization Using Parallel Computing" on page 3-180
- "Ways to Speed Up Design Optimization Tasks"

Use Fast Restart Mode During Response Optimization

This topic shows how to speed up response optimization using Simulink fast restart. You can use the fast restart feature to speed up response optimization of tunable parameters of a model.

Fast restart enables you to perform iterative simulations without compiling a model or terminating the simulation each time. Using fast restart, you compile a model only once. You can then tune parameters and simulate the model again without spending time on compiling. Fast restart associates multiple simulation phases with a single compile phase to make iterative simulations more efficient. You see a speedup of design optimization tasks using fast restart in models that have a long compilation phase. See "How Fast Restart Improves Iterative Simulations".

When you enable fast restart, you can only change tunable properties of the model during simulation. For more information about the limitations, see "Limitations".

You can optimize using fast restart in the **Response Optimizer** (Not supported in Simulink Online) or at the command line on page 3-189.

Response Optimizer App Workflow for Fast Restart

To optimize a model response using fast restart in the **Response Optimizer**:

- **1** Open the Simulink model.
- 2 Enable fast restart in the model.

Click **Fast Restart** in the model window.

- **3** Open the **Response Optimizer** for the model.
- **4** Configure the design variables, design requirements, and, optionally, optimization settings.

For more information, see "Specify Design Variables" on page 3-56, "Specify Time-Domain Design Requirements in the App" on page 3-16, "Specify Frequency-Domain Design Requirements in the App" on page 3-43, and "Specify Optimization Options" on page 3-62.

- 5 Click **Optimize** to optimize the model response in fast restart mode.
- **6** Disable fast restart.

In the model window, click Fast Restart

Command-Line Workflow for Fast Restart

To optimize a model response using fast restart at the command line:

- **1** Open the Simulink model.
- 2 Create a model simulation scenario. You must create a simulation scenario with logging information before configuring the model for fast restart. You cannot modify logging information once the model has been compiled for fast restart.

Simulator = sdo.SimulationTest('model');

Specify model signals to log during model simulation.

For response optimization problems that include frequency-domain requirements, the model is linearized using Simulink Control Design. Use the SystemLoggingInfo property of the sdo.SimulationTest object, Simulator, to specify linear systems to log when simulating the model. For an example, see "Design Optimization to Meet Frequency-Domain Requirements (Code)" on page 3-216.

Note In fast restart mode, you cannot use the linearize command from Simulink Control Design to specify and compute linear systems. Using linearize generates an error.

- 3 Specify design requirements, Requirements, and design variables, param. For and example, see "Design Optimization to Meet Step Response Requirements (Code)".
- 4 Configure the model and simulation scenario for fast restart.

```
Simulator = fastRestart(Simulator, 'on');
```

5 Create an optimization cost function, myCostfcn, and pass Simulator to the cost function as an input. For more information, see "Write a Cost Function" on page 2-49. In the cost function, the simulator configured for fast restart is used to update the model parameters, simulate the model, and log signals.

Use an anonymous function with one argument that calls myCostfcn.

optimfcn = @(param) myCostfcn(param,Simulator,Requirements);

Here, myCostfcn is a cost function that takes design variables, param, simulation scenario, Simulator, and design requirements, Requirements, as inputs.

6 Run the optimization.

[param_opt,opt_info] = sdo.optimize(optimfcn,param);

7 Restore the simulator fast restart settings.

```
Simulator = fastRestart(Simulator, 'off');
```

Troubleshooting

Why Don't I See the Optimization Speedup I Expected Using Fast Restart?

You see a speedup of design optimization tasks using fast restart in models that have a long compilation phase. If the compilation phase of your model is not long, you do not see a significant change in optimization speed.

See Also

sdo.SimulationTest|sdo.optimize|fastRestart

More About

• "Ways to Speed Up Design Optimization Tasks"

Optimization Does Not Make Progress

Should I worry about the scale of my responses and how constraints and design requirements are discretized?

No, Simulink Design Optimization software automatically normalizes constraints, design requirement and response data.

Why don't the responses and parameter values change at all?

The optimization problem you formulated might be nonsmooth. This means that small parameter changes have no effect on the amount by which response signals satisfy or violate the constraints and only large changes will make a difference. Try switching to a search-based method such as simplex search or pattern search. Alternatively, look for initial guesses outside of the dead zone where parameter changes have no effect. If you are optimizing the response of a Simulink model, you could also try removing nonlinear blocks such as Quantizer or Dead Zone.

Why does the optimization stall?

When optimizing a Simulink model, certain parameter combinations can make the simulation stall for models with strong nonlinearities or frequent mode switching. In these cases, the ODE solvers take smaller and smaller step sizes. Stalling can also occur when the model's ODEs become too stiff for some parameter combinations. A symptom of this behavior is when the Simulink model status is Running and clicking the **Stop** button fails to interrupt the optimization. When this happens, you can try one of the following solutions:

- Switch to a different ODE solver, especially one of the stiff solvers.
- Specify a minimum step size.
- Disable zero crossing detection if chattering is occurring.
- Tighten the lower and upper bounds on parameters that cause simulation difficulties. In particular, eliminate regions of the parameter space where some model assumptions are invalid and the model behavior can become erratic.

Optimization Convergence

What to do if the optimization does not get close to an acceptable solution?

- If you are using pattern search, check that you have specified appropriate maximum and minimum values for all your tuned parameters or compensator elements. The pattern search method looks inside these bounds for a solution. When they are set to their default values of Inf and -Inf, the method searches within ±100% of the initial values of the parameters. In some cases this region is not large enough and changing the maximum and minimum values can expand the search region.
- Your optimization problem might have local minima. Consider running one of the search-based methods first to get closer to an acceptable solution.
- Reduce the number of tuned parameters and compensator elements by removing from the design variables or from the **Compensators** pane those parameters that you know only mildly influence the optimized responses. After you identify reasonable values for the key parameters, add the fixed parameters back to the tunable list and restart the optimization using these reasonable values as initial guesses.
- The software may have encountered errors during the optimization. Review the errors to determine if you can make changes to improve the optimization results. Changes may require modifications to the model, requirements, or optimization settings.
 - In the **Response Optimizer**, the software creates a structure named EvalErrors in the **Data** area when the optimization completes with errors. Export this structure to the MATLAB workspace and examine its contents at the command line. EvalErrors has two fields, Errors and DesignVars, containing the errors encountered during optimization and the corresponding design variable values. To reproduce a specific error, use sdo.setValueInModel to run the model using the design variables that correspond to the error.
 - At the command line, the second output of sdo.optimize, opt_info, is a structure that provides information regarding the optimization. opt_info.exitflag identifies the reason the optimization terminated. For more information regarding exit flags, see "Exit Flags and Exit Messages".

Why does the optimization terminate before exceeding the maximum number of iterations, with a solution that does not satisfy all the constraints or design requirements?

- It might not be possible to achieve your specifications. Try relaxing the constraints or design requirements that the response signals violate the most. After you find an acceptable solution to the relaxed problem, tighten some constraints again and restart the optimization.
- The optimization might have converged to a local minimum that is not a feasible solution. Restart the optimization from a different initial guess and/or use one of the search-based methods to identify another local minimum that satisfies the constraints.

What to do if the optimization takes a long time to converge even though it is close to a solution?

• In the **Response Optimizer**, click **Stop** to interrupt the optimization when you think the current optimized response signals are acceptable.

When you use **Optimization Based Tuning**, click **Stop Optimization** in the **Optimization** tab of the **Response Optimization** dialog in the **Control System Designer**, when you think the current optimized response signals are acceptable.

- If you use the gradient descent method, try restarting the optimization. Restarting resets the Hessian estimate and might speed up convergence.
- Increase the convergence tolerances in the Optimization Options dialog to force earlier termination.
- Relax some of the constraints or design requirements to increase the size of the feasibility region.

What to do if the response becomes unstable and does not recover?

While the optimization formulation has explicit safeguards against unstable or divergent response signals, the optimization can sometimes venture into an unstable region where simulation results become erratic and gradient methods fail to find a way back to the stable region. In these cases, you can try one of the following solutions:

- Add or tighten the lower and upper bounds on compensator element and parameter values. Instability often occurs when you allow some parameter values to become too large.
- Use a search-based method to find parameter values that stabilize the response signals and then start the gradient-based method using these initial values.
- When optimizing responses in **Control System Designer**, you can try adding additional design requirements that achieve the same or similar goal. For example, in addition to a settling time design requirement on a step response plot, you could add a settling time design requirement on a root-locus plot that restricts the location of the real parts of the poles. By adding overlapping design requirements in this way, you can force the optimization to meet the requirements.

Optimization Speed and Parallel Computing

How can I speed up the optimization?

• The optimization time is dominated by the time it takes to simulate the model. When optimizing a Simulink model, you can enable the Accelerator mode by choosing Accelerator from the dropdown list under **Simulation** in the Simulink Editor, to dramatically reduce the optimization time.

Note The Rapid Accelerator mode in Simulink software is not supported for speeding up the optimization. For more information, see "Use Accelerator Mode During Simulations" on page 3-179.

- The choice of ODE solver can also significantly affect the overall optimization time. Use a stiff solver when the simulation takes many small steps, and use a fixed-step solver when such solvers yield accurate enough simulations for your model. (These solvers must be accurate in the entire parameter search space.)
- Reduce the number of tuned compensator elements or parameters and constrain their range to narrow the search space.
- When specifying parameter uncertainty (not available when optimizing responses in **Control System Designer**), keep the number of sample values small since the number of simulations grows exponentially with the number of samples. For example, a grid of 3 parameters with 10 sample values for each parameter requires $10^3=1000$ simulations per iteration.

Why are the optimization results with and without using parallel computing different?

- Different numerical precision on the client and worker machines can produce marginally different simulation results. Thus, the optimization method can take a different solution path and produce a different result.
- When you use parallel computing with the Pattern search method, the search is more comprehensive and can result in a different solution.

To learn more, see "Parallel Computing with the Pattern Search Method" on page 3-181.

Why do I not see the optimization speedup I expected using parallel computing?

- When you optimize a model that does not have a large number of parameters or does not take long to simulate, you might not see a speedup in the optimization time. In such cases, the overhead associated with creating and distributing the parallel tasks outweighs the benefits of running the optimization in parallel.
- Using the Pattern search method with parallel computing might not speed up the optimization time. Without parallel computing, the method stops the search at each iteration when it finds a solution better than the current solution. The candidate solution search is more comprehensive when you use parallel computing. Although the number of iterations might be larger, the optimization without using parallel computing might be faster.

To learn more about the expected speedup, see "Parallel Computing with the Pattern Search Method" on page 3-181.

Why does the optimization using parallel computing not make any progress?

To troubleshoot the problem:

- **1** Run the optimization for a few iterations without parallel computing to see if the optimization progresses.
- 2 Check whether the remote workers have access to all model dependencies. Model dependencies include data variables and files required by the model to run.

To learn more, see "Model Dependencies" on page 3-183.

Why does the optimization using parallel computing not stop when I click the Stop optimization button?

When you use parallel computing with the Pattern search method, the software must wait until the current optimization iteration completes before it notifies the workers to stop. The optimization does not terminate immediately when you click **Stop**, and, instead, appears to continue running.

Undesirable Parameter Values

What to do if the optimization drives the tuned compensator elements and parameters to undesirable values?

- When a tuned compensator element or parameter is positive, or when its value is physically constrained to a given range, enter the lower and upper bounds (**Minimum** and **Maximum**) in one of the following:
 - Dialog box to select design variables (in **Response Optimizer**)
 - Compensators pane (in Control System Designer)

This information helps guide the optimization method towards a reasonable solution.

- Specify initial guesses that are within the range of desirable values.
- In the **Compensators** pane in **Control System Designer**, verify that no integrators/ differentiators are selected for optimization. Optimizing the pole/zero location of integrators/ differentiators can result in drastic changes in the system gain and lead to undesirable values.

What to do if the optimization violates bounds on parameter values?

The Gradient descent optimization method fmincon violates the parameter bounds when it cannot simultaneously satisfy the signal constraints and the bounds. When this happens, try one of the following:

• Specify a different value for the parameter bound and restart the optimization. A guideline is to adjust the bound by 1% of the typical value.

For example, for a parameter with a typical value of 1 and lower bound of 0, change the lower bound to 0.01.

- Relax the signal constraints and restart the optimization. This approach results in a different solution path for the Gradient descent method.
- Restart the optimization immediately after it terminates by clicking **Optimize** in the **Response Optimizer**. This approach uses the previous optimization results as the starting point for the next optimization cycle to refine the results.
- Use the following two-step approach to perform the optimization:
 - **1** Run an initial optimization to satisfy the signal constraints.

For example, run the optimization using the Simplex search method. This method satisfies the signal constraints but does not support the bounds on parameter values. The results obtained using this method provide the starting point for the optimization performed in the next step. To learn more about this method, see the fminsearch function reference page in the Optimization Toolbox documentation.

2 Reconfigure the optimization by selecting a different optimization method to satisfy both the signal constraints and the parameter bounds.

For example, change the optimization method to Gradient descent and run the optimization again.

Tip If Global Optimization Toolbox software is installed, you can select the Pattern search optimization method to optimize the model response.

Reverting to Initial Parameter Values

How do I quit an optimization and revert to my initial parameter values?

- Before running an optimization, do one of the following:
 - In the **Response Optimizer**, click **Options**. Uncheck **Update model at end of optimization** in the **General Options** tab.
 - In the **Response Optimizer**, click **Options**. Select **Save optimized variable values as new design variable set** in the **General Options** tab.
 - Make a copy of the design variable set in the **Data** area.

If you want to revert to the initial parameter values after the optimization terminates or you stop the optimization by clicking **Stop**, select the design variable that contains the initial values in the

Design Variable Set drop-down list and click *adjacent to Design variables Set.* Select the design variables in the dialog box and click **Update model variable values** to revert the model parameters to their original values.

 When using the Control System Designer, the Start Optimization button becomes a Stop Optimization button after the optimization has begun. To quit the optimization, click the Stop Optimization button. To revert to the initial parameter values, select Edit > Undo Optimize compensators from the menu in the Control System Designer.

Save and Load Optimization Sessions

Structure of an Optimization Session

The **Response Optimizer** stores and organizes data from a given Simulink model inside a *session*. An optimization session includes the following information:

- Design variables and uncertain variables
- Design requirements
- Optimization results
- Optimization settings
- Plots Changes to plots layout and plot characteristics, such as axis limits, line colors, are not included.

The default session name is the same as the Simulink model name. The session name displays on the title pane of **Response Optimizer**.

Save a Session

Saving a session lets you reuse your settings and optimization results later. Each **Response Optimizer** session is associated with a Simulink model.

You can save the session as either a MAT-file or workspace variable.

- To save the session as a MAT-file, in the **Response Optimization** tab, in the **Save Session** dropdown list, click **Save to file**. A window opens where you specify the MAT-file name.
- To save the session as a model or MATLAB workspace variable, in the **Save Session** drop-down list, select **Save to model workspace** or **Save to MATLAB workspace**.

Load a Session

To load a previously saved MAT-file or workspace session:

- **1** Open the **Response Optimizer** for the model.
- 2 To load a MAT-file, in the **Response Optimization** tab, click the **Open Session** drop-down list, and select **Open from file**. A window opens where you select the MAT-file to load.

To load a workspace variable, select **Open from model workspace** or **Open from MATLAB workspace** in the **Open Session** drop-down list.

See Also

Related Examples

- "Specify Design Variables" on page 3-56
- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43
- "Specify Optimization Options" on page 3-62

Improving Optimization Performance Using Parallel Computing

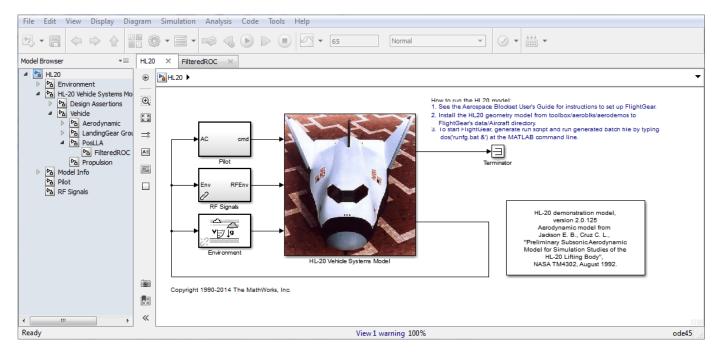
This example shows how to improve optimization performance using the Parallel Computing Toolbox[™]. The example discusses the speedup seen when using parallel computing to optimize a complex Simulink® model. The example also shows the effect of the number of parameters and the model simulation time when using parallel computing.

Requires Parallel Computing Toolbox[™]

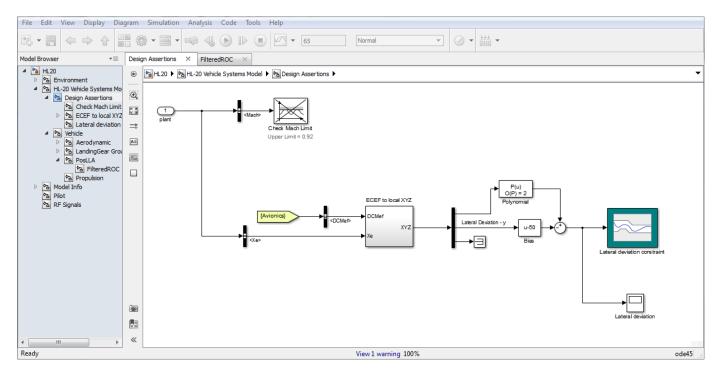
An Example Illustrating Optimization Speedup

The main computational load when using Simulink Design Optimization[™] is the simulation of a model. The optimization methods typically require numerous simulations per optimization iteration. As many of the simulations are independent, it is beneficial to use parallel computing to distribute these independent simulations across different processors.

Consider the Simulink model of an HL20 aircraft, which ships with the Aerospace Blockset[™]. The HL20 model is a complex model and includes mechanical, electrical, avionics, and environmental components. A typical simulation of the HL20 model takes around 60 seconds.



During landing, the aircraft is subjected to two wind gusts from different directions which cause the aircraft to deviate from the nominal trajectory on the runway. Simulink Design Optimization is used to tune the 3 parameters of the controller so that the lateral deviation of the aircraft from a nominal trajectory in the presence of wind gusts is kept within five meters.



The optimization is performed both with and without using parallel computing on dual-core 64bit AMD®, 2.4GHz, 3.4GB Linux® and quad-core 64bit AMD, 2.5GHz, 7.4GB Linux machines. The speed up observed for this problem is shown below.

Total optimization time (s)	Dual-core processor (two workers)		-	d-core proce four workers		
	serial	parallel	ratio	serial	parallel	ratio
Gradient descent based	2140	1360	1.57	2050	960	2.14
optimization speed-up						
Pattemsearch based optimization speed-up	3690	2140	1.72	3480	1240	2.81

The HL20 example illustrates the potential speedup when using parallel computing. The next sections discuss the speedup expected for other optimization problems.

When Will an Optimization Benefit from Parallel Computing?

The previous section shows that distributing the simulations during an optimization can reduce the total optimization time. This section quantifies the expected speedup.

In general, the following factors could indicate that parallel computing would result in faster optimization:

- There are a large number of parameters being optimized
- A pattern search method is being used
- A complex Simulink model that takes a long time to simulate
- There are a number of uncertain parameters in the model

Each of these is examined in the following sections.

Number of Parameters and Their Effect on Parallel Computing

The number of simulations performed by an optimization method is closely coupled with the number of parameters.

Gradient Descent Method and Parallel Computing

Consider the simulations required by a gradient-based optimization method at each iteration:

- A simulation for the current solution point
- Simulations to compute the gradient of the objective with respect to the optimized parameters
- Once a gradient is computed, simulations to evaluate the objective along the direction of the gradient (the so called line search evaluations)

Of these simulations, the simulations required to compute gradients are independent and are distributed. Let us look at this more closely:

```
Np = 1:16; %Number of parameters (16 = 4 filtered PID controllers)
Nls = 1; %Number of line search simulations, assume 1 for now
Nss = 1; %Total number of serial simulations, start with nominal only
```

The gradients are computed using central differences. There are 2 simulations per parameter, and include the line search simulations to give the total number of simulations per iteration:

Nss = 1+Np*2+Nls;

As mentioned above, the computation of gradients with respect to each parameter can be distributed or run in parallel. This reduces the total number of simulations that run in series when using parallel computing as follows:

```
Nw = 4; %Number of parallel processors
Nps = 1 + ceil(Np/Nw)*2+Nls;
```

The ratio Nss/Nps gives us the speedup which we plot below

```
%Vary the number of line search simulations
Nls = 0:5:
figure;
hAx = gca;
xlim([min(Np) max(Np)]);
for ct = 1:numel(Nls)
  Rf = (1+Nls(ct)+Np*2)./(1+Nls(ct)+ceil(Np/Nw)*2);
  hLine = line('parent', hAx, 'xdata', Np, 'ydata', Rf);
  if ct == 1
     hLine.LineStyle = '-';
     hLine.Color = [0 \ 0 \ 1];
     hLine.LineWidth = 1;
  else
     hLine.LineStyle = '-.';
     hLine.Color = [0.6 0.6 1];
     hLine.LineWidth = 1;
  end
end
grid on
title('Gradient descent based relative speedup')
xlabel('Number of parameters')
ylabel('Serial time/parallel time')
```

```
Gradient descent based relative speedup
     4
   3.5
Serial time/parallel time
     3
   2.5
     2
                                                Increasing number of line searches
   1.5
     1
            2
                       4
                                  6
                                             8
                                                       10
                                                                  12
                                                                             14
                                                                                       16
                                   Number of parameters
```

```
annotation('arrow',[0.55 .55],[5/6 2/6])
text(8.5,1.75,'Increasing number of line searches')
```

The plot shows that the relative speedup gets better as more parameters are added. The upper solid line is the best possible speedup with no line search simulations while the lighter dotted curves increase the number of line search simulations.

The plot also shows local maxima at 4,8,12,16 parameters which corresponds to cases where the parameter gradient calculations can be distributed evenly between the parallel processors. Recall that for the HL20 aircraft problem, which has 3 parameters, the quad core processor speedup observed was 2.14 which matches well with this plot.

Pattern Search Method and Parallel Computing

Pattern search optimization method is inherently suited to a parallel implementation. This is because at each iteration one or two sets of candidate solutions are available. The algorithm evaluates all these candidate solutions and then generates a new candidate solution for the next iteration. Evaluating the candidate solutions can be done in parallel as they are independent. Let us look at this more closely:

Pattern search uses two candidate solution sets, the search set and the poll set. The number of elements in these sets is proportional to the number of optimized parameters

<pre>Nsearch = 15*Np;</pre>	%Default number of elements in the solution set
Npoll = $2*Np$;	%Number of elements in the poll set with a 2N poll method

The total number of simulations per iteration is the sum of the number of candidate solutions in the search and poll sets. When distributing the candidate solution sets the simulations are distributed evenly between the parallel processors. The number of simulations that run in series after distribution thus reduces to:

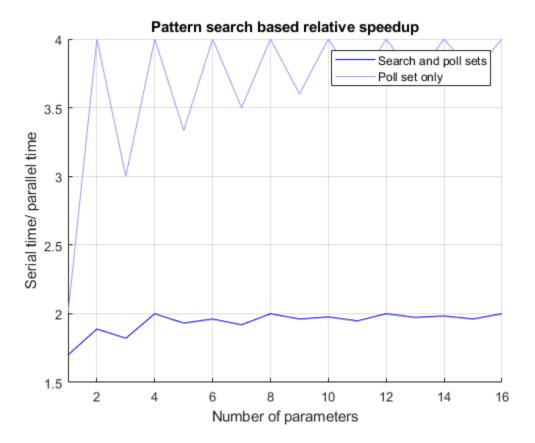
Nps = ceil(Nsearch/Nw)+ceil(Npoll/Nw);

When evaluating the candidate solutions without using parallel computing, the iteration is terminated as soon as a candidate solution that is better than the current solution is found. Without additional information, the best bet is that about half the candidate solutions will be evaluated. The number of serial simulations is thus:

```
Nss = 0.5*(Nsearch+Npoll);
```

Also note that the search set is only used in the first couple of optimization iterations, after which only the poll set is used. In both cases the ratio Nss/Nps gives us the speedup which we plot below.

```
figure;
hAx = gca;
xlim([min(Np) max(Np)]);
Rp1 = (Nss)./( Nps );
Rp2 = (Npoll)./( ceil(Npoll/Nw) );
line('parent',hAx,'xdata',Np,'ydata',Rp1,'color',[0 0 1]);
line('parent',hAx,'xdata',Np,'ydata',Rp2,'color',[0.6 0.6 1]);
grid on
title('Pattern search based relative speedup')
xlabel('Number of parameters')
ylabel('Serial time/ parallel time')
legend('Search and poll sets','Poll set only')
```



The dark curve is the speedup when the solution and poll sets are evaluated and the lighter lower curve the speedup when only the poll set is evaluated. The expected speedup over an optimization should lie between the two curves. Notice that even with only one parameter, a pattern search method benefits from distribution. Also recall that for the HL20 aircraft problem, which has 3 parameters, the quad core speedup observed was 2.81 which matches well with this plot.

Overhead of Distributing an Optimization

In the previous sections, the overhead associated with performing the optimization in parallel was ignored. Including this overhead gives an indication of the complexity of a simulation that will benefit from distributed optimization.

The overhead associated with running an optimization in parallel primarily results from two sources:

- opening the parallel pool
- loading the Simulink model and performing an update diagram on the model

There is also some overhead associated with transferring data to and from the remote processors. Simulink Design Optimization relies on shared paths to provide remote processors access to models and the returned data is limited to objective and constraint violation values. Therefore, this overhead is typically much smaller than opening the MATLAB® pool and loading the model. This was true for the HL20 aircraft optimization but may not be true in all cases. However, the analysis shown below can be extended to cover additional overhead.

```
findOverhead = false; % Set true to compute overhead for your system.
if findOverhead
```

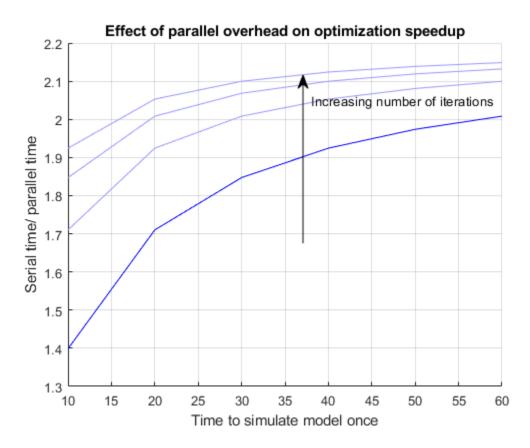
```
% Compute overhead.
 wState = warning('off', 'MATLAB:dispatcher:pathWarning');
 t0 = datetime('now');
                                  %Current time
 parpool
                                  %Open the parallel pool
 load system('airframe demo')
                                  %Open a model
 set_param('airframe_demo', 'SimulationCommand', 'update') %Run an update diagram for the model
 Toverhead = datetime('now')-t0; %Time elapsed
 Toverhead = seconds(Toverhead) %Convert to seconds
 close system('airframe demo')
 delete(acp)
                                  %Close the parallel pool
 warning(wState);
else
 % Use overhead observed from experiments.
 Toverhead = 28.6418;
end
```

Let us consider a gradient-based algorithm with the following number of serial simulations per iteration:

```
Nw = 4;%Four parallel processorsNp = 4;%Four parameters optimizedNls = 2;%Assume 2 line search simulationsNss = 1+Np*2+Nls;%Serial simulations without parallel computingNps = 1+ceil(Np/Nw)*2+Nls;%Serial simulations with parallel computing
```

The speedup is now computed as the ratio of the total parallel time including overhead to the total serial time. For worst case analysis, assume the optimization terminates after one iteration which gives:

```
Niter = 1;
     = 10:10:60;
                                 %Time to simulate model once
Ts
Tst
     = Niter*Nss*Ts;
                                 %Total serial optimization time
Tpt
     = Niter*Nps*Ts+Toverhead; %Total parallel optimization time
figure;
hAx = gca;
xlim([min(Ts) max(Ts)]);
Rp = (Tst)./(Tpt);
line('parent', hAx, 'xdata', Ts, 'ydata', Rp, 'color', [0 0 1]);
Niter = 2^{1:4};
for ct = 1:numel(Niter)
   Rp = ( Niter(ct)*Nss*Ts )./(Niter(ct)*Nps*Ts+Toverhead);
   line('parent', hAx, 'xdata', Ts, 'ydata', Rp, 'color', [0.6 0.6 1]);
end
grid on
title('Effect of parallel overhead on optimization speedup')
xlabel('Time to simulate model once')
ylabel('Serial time/ parallel time')
annotation('arrow',[0.55 0.55],[0.45 0.85]);
text(38,2.05,'Increasing number of iterations')
```



The dark lower curve shows the speedup assuming one iteration while the lighter upper curves indicate speedup with up to 16 iterations.

A similar analysis could be performed for pattern search based optimization. Experience has shown that optimization of complex simulations that take more than 40 seconds to run typically benefit from parallel optimization.

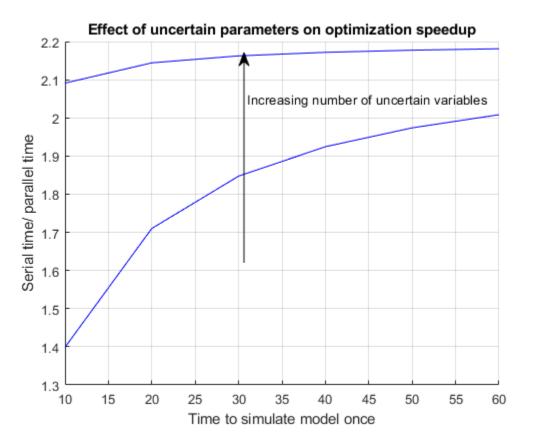
Uncertain Parameters and Their Effect on Parallel Optimization

When using Simulink Design Optimization, you can vary some parameters (say a spring stiffness) and optimize parameters in the face of these variations. The uncertain parameters define additional simulations that need to be evaluated at each iteration. However, conceptually you can think of these additional simulations as extending the simulation without uncertain parameters. This implies that wherever one simulation was carried out, now multiple simulations are carried out. As a result, uncertain parameters do not affect the overhead free optimization speedup and the calculations in earlier sections are valid.

In the case where we considered overhead, uncertain parameters have the effect of increasing the simulation time and reduce the effect of the overhead associated with creating a parallel optimization. To see this consider the following

```
Nu = [0 10]; %Number of uncertain scenarios
Nss = (1+Nu)*(1+Np*2+Nls); %Serial simulations without parallel computing
Nps = (1+Nu)*(1+ceil(Np/Nw)*2+Nls); %Serial simulations with parallel computing
figure;
hAx = gca;
```

```
xlim([min(Ts) max(Ts)]);
for ct = 1:numel(Nu)
    Rp = (Ts*Nss(ct))./(Ts*Nps(ct)+Toverhead);
    line('parent',hAx,'xdata',Ts,'ydata',Rp,'color',[0 0 1]);
end
grid on
title('Effect of uncertain parameters on optimization speedup')
xlabel('Time to simulate model once')
ylabel('Serial time/ parallel time')
annotation('arrow',[0.45 0.45],[0.4 0.9])
text(31,2.05,'Increasing number of uncertain variables')
```



The bottom curve is the case with no uncertain parameters while the top curve is the case with an uncertain parameter that can take on 10 distinct values.

Optimizing Time-Domain Response of Simulink Models Using Parallel Computing

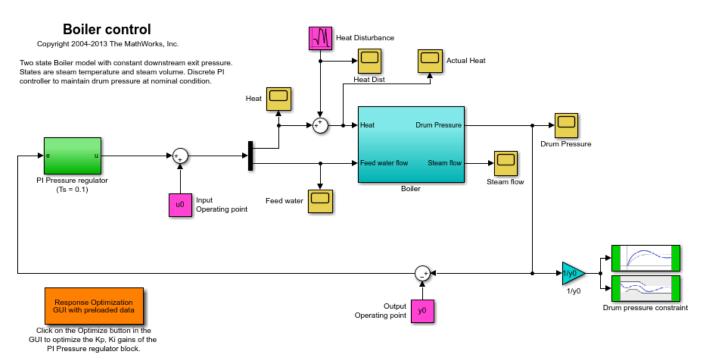
This example shows how to use parallel computing to optimize the time-domain response of a Simulink® model. You use Simulink® Design Optimization^m and Parallel Computing Toolbox^m to tune the gains of a discrete PI controller of a boiler to meet the design requirements. The example also shows how the software automatically handles model file dependencies.

This example requires Parallel Computing Toolbox[™].

Opening the Model

The Simulink model consists of a boiler model and a discrete PI controller. When using parallel computing, Simulink Design Optimization performs a model dependency check, which recognizes the boiler model library as an installed library.

open_system('boilerpressure_demo')



Design Requirements

The boiler pressure is regulated by a discrete PI controller. The design requirement for the controller is to limit the pressure variation of the boiler within +-%5 of the nominal pressure.

The initial controller has fairly good regulation characteristics but in the presence of additional heat disturbances, modeled by the Heat Disturbance block, we want to tune the controller performance to provide tighter pressure regulation.

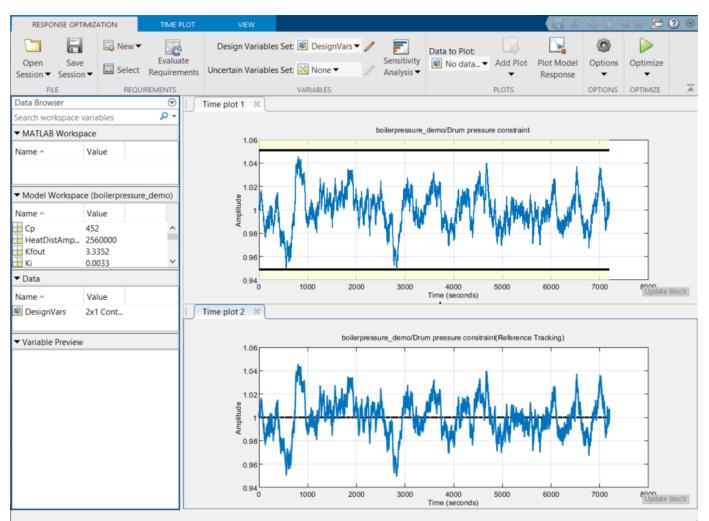
Double-click the 'Response Optimization GUI with preloaded data' block in the Simulink model to open a pre-configured **Response Optimizer** session. The **Response Optimizer** is configured with:

1. Upper and lower bounds representing a +-5% allowable range on the drum pressure

2. A reference tracking objective to minimize the deviation of the drum pressure from nominal

3. The PI controller gains, Kp and Ki, are selected for tuning

Click Plot Model Response to display the drum pressure variations with the initial controller.



Configuring and Running the Optimization in the GUI Using Parallel Computing

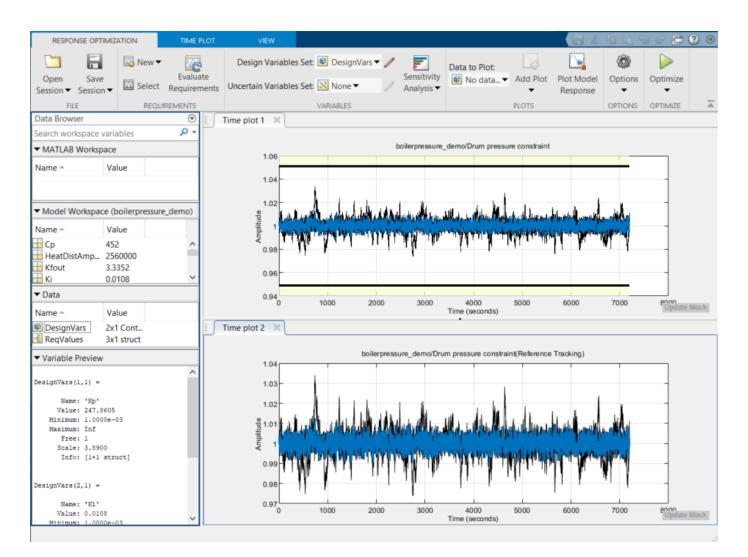
When computing the model response with the initial controller, this complex model took a long time to simulate. Using parallel computing can reduce the optimization time by simulating the model in parallel. For more information on parallel computing and optimization performance see the tutorial "Improving Optimization Performance Using Parallel Computing" on page 3-200.

To configure the optimization problem to use parallel computing click **Options** in the **Response Optimizer** and select the **Parallel Options** tab. Select the "Use the parallel pool during optimization" option. This triggers an automated search for any model dependencies. In this example, the Steam table library (libsteam.slx) is found as a model dependency (in addition to the boilerpressure_demo model itself), and is displayed in the Model file dependencies list box.

Response Optimization Options 🛛 🕹 🗙
General Options Optimization Options Parallel Options Linearization Options
Use the parallel pool during optimization
Model file dependencies Model path dependencies
H:/MATLAB/Examples/sldo/OptimizingTimeResponseOfModelUsingParallelComputingExample/boilerpressure_demo.slx H:/MATLAB/Examples/sldo/OptimizingTimeResponseOfModelUsingParallelComputingExample/libsteam.slx
Add file dependency Sync file dependencies from model
OK Cancel Help

Clicking OK configures the optimization to use parallel computing.

To run the optimization click the **Optimize** button. A progress window opens displaying optimization progress and the plots update to show the optimized response.



The final response shows that the optimized regulator tracks the reference pressure much more closely and the drum pressure constraints are satisfied.

Configuring and Running the Optimization at the Command Line

You can also use the command line functions to configure the optimization to use parallel computing and run the optimization.

Select the model variables for optimization and set lower limits

```
p = sdo.getParameterFromModel('boilerpressure_demo',{'Kp','Ki'});
p(1).Minimum = 0.001;
p(2).Minimum = 0.001;
```

Select the model signal to bound and create a simulator to simulate the model.

```
nPressure = Simulink.SimulationData.SignalLoggingInfo;
nPressure.BlockPath = 'boilerpressure_demo/1//y0';
nPressure.OutputPortIndex = 1;
nPressure.LoggingInfo.NameMode = 1;
nPressure.LoggingInfo.LoggingName = 'nPressure';
```

```
simulator = sdo.SimulationTest('boilerpressure_demo');
simulator.LoggingInfo.Signals = nPressure;
```

Get the optimization requirements defined by the check blocks in the model so that we can use them in the optimization problem.

```
bnds = getbounds('boilerpressure_demo/Drum pressure constraint');
PressureLimits = [bnds{:}];
bnds = getbounds('boilerpressure_demo/Drum pressure constraint(Reference Tracking)');
PressureRegulation = [bnds{:}];
requirements = struct(...
    'PressureLimits', PressureLimits, ...
    'PressureRegulation', PressureRegulation);
```

Define the function called during optimization. Notice that the function uses the simulator and requirements defined earlier to evaluate the design.

```
evalDesign = @(p) boilerpressure_design(p,simulator,requirements);
type boilerpressure_design
```

```
function design = boilerpressure design(p,simulator,requirements)
%BOILERPRESSURE DESIGN
%
% The boilerpressure_design function is used to evaluate a boiler
% controller design design.
% The |p| input argument is the vector of controller parameters.
%
% The [simulator] input argument is a sdo.SimulinkTest object used to
% simulate the |boilerpressure demo| model and log simulation signals.
% The |requirements| input argument contains the design requirements used
% to evaluate the boiler controller design.
%
% The |design| return argument contains information about the design
% evaluation that can be used by the [sdo.optimize] function to optimize
% the design.
%
% see also sdo.optimize, sdoExampleCostFunction
% Copyright 2011 The MathWorks, Inc.
%% Simulate the model
%
% Use the simulator input argument to simulate the model and log model
% signals.
%
% First ensure that we simulate the model with the parameter values chosen
% by the optimizer.
%
simulator.Parameters = p;
200
% Simulate the model and log signals.
%
simulator = sim(simulator);
<u>%</u>%
```

```
\% Get the simulation signal log, the simulation log name is defined by the
% model |SignalLoggingName| property
%
logName = get_param('boilerpressure_demo','SignalLoggingName');
simLog = get(simulator.LoggedData,logName);
%% Evaluate the design requirements
%
% Use the requirements input argument to evaluate the design requirements
%
% Check the Pressure signal against the [PressureLimits] requirements.
%
nPressure = get(simLog, 'nPressure');
c = [...
   evalRequirement(requirements.PressureLimits(1),nPressure.Values); ...
   evalRequirement(requirements.PressureLimits(2),nPressure.Values)];
%%
% Use the PressureLimits requirements as non-linear constraints for
% optimization.
design.Cleq = c(:);
%%
% Check the pressure signal against the |PressureRegulation| requirement.
%
f = evalRequirement(requirements.PressureRegulation,nPressure.Values);
%%
% Use the PressureRegulation requirement as an objective for optimization.
design.F = f;
end
```

Setup optimization options to use the parallel pool and specify the model and model files dependencies.

```
opt = sdo.OptimizeOptions;
opt.UseParallel = true;
opt.OptimizedModel = 'boilerpressure_demo';
[dirs,files] = sdo.getModelDependencies('boilerpressure_demo');
opt.ParallelFileDependencies = files;
```

To run the optimization using the parallel pool pass the optimization options, **opt**, to the **sdo.optimize** command.

>> [pOpt,info] = sdo.optimize(evalDesign,p,opt)
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
Configuring parallel workers for optimization...
Parallel workers configured for optimization.

Optimization started 11-May-2021 17:56:00

			max		First-order
Iter	F-count	f(x)	constraint	Step-size	optimality
0	5	17.5068	0		
1	10	11.6563	0	1.25	32.2
2	15	8.29415	0	1.27	5.64
3	20	5.67333	0	2.04	19.7
4	25	0.0951198	0	477	0.00726
5	30	0.0937752	0	1.22	0.00723
6	39	0.0937752	0	0.392	0.00723
T 1		manaihla Gar	eterinte esti	a fill a d	

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than the value of the step size tolerance and constraints are satisfied to within the value of the constraint tolerance. Removing data from parallel workers... Data removed from parallel workers.

Close the model.

bdclose('boilerpressure_demo')

Design Optimization to Meet Frequency-Domain Requirements (Code)

This example shows how to tune model parameters to meet frequency-domain requirements, using the sdo.optimize command.

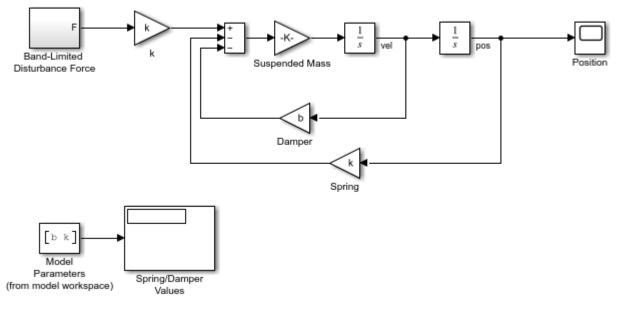
This example requires Simulink[®] Control Design[™].

Suspension Model

Open the Simulink model.

open_system('sdoSimpleSuspension')

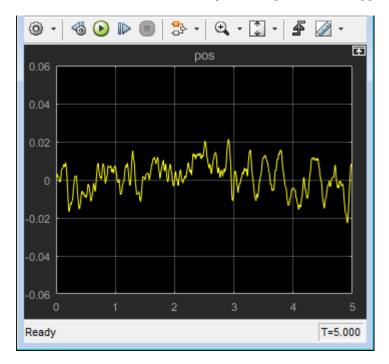
Simple Suspension System



Copyright 2002-2015 The MathWorks, Inc.

Mass-spring-damper models represent simple suspension systems and for this example we tune the system to meet typical suspension requirements. The model implements the second order system representing a mass-spring-damper using Simulink blocks and includes:

- a Mass gain block parameterized by the total suspended mass, m0+mLoad. The total mass is the sum of a nominal mass, m0, and a variable load mass, mLoad.
- a Damper gain block parameterized by the damping coefficient, b.
- a Spring gain block parameterized by the spring constant, k.
- two integrator blocks to compute the mass velocity and position.
- a Band-Limited Disturbance Force block applying a disturbance force to the mass. The disturbance force is assumed to be band-limited white noise.



Simulate the model to view the system response to the applied disturbance force.

Design Problem

The initial system has a bandwidth that is too high. This can be seen from the spiky position signal. You tune the spring and damper values to meet the following requirements:

- The -3dB system bandwidth must not exceed 10 rad/s.
- The damping ratio of the system must be less than 1/sqrt(2). This ensures that no frequencies in pass band are amplified by the system.
- Minimize the expected failure rate of the system. The expected failure rate is described by a Weibull distribution dependent on the mass, spring, and damper values.
- These requirements must all be satisfied as the load mass ranges from 0 to 20 kg.

Specify Design Variables

```
DesignVars = sdo.getParameterFromModel('sdoSimpleSuspension',{'b','k'});
DesignVars(1).Minimum = 100;
DesignVars(1).Maximum = 10000;
DesignVars(2).Minimum = 10000;
DesignVars(2).Maximum = 100000;
```

Specify Uncertain Variables

Specify mLoad, the load mass, as an uncertain variable. This will ensure the optimal solution is robust to variations in load mass.

```
UncVars = sdo.getParameterFromModel('sdoSimpleSuspension','mLoad');
UncVars_Values = {...
10; ...
20};
```

Specify Design Requirements

Specify design requirements to satisfy during optimization.

```
Requirements = struct;
Requirements.Bandwidth = sdo.requirements.BodeMagnitude(...
    'BoundFrequencies', [10 100], ...
    'BoundMagnitudes', [-3 -3]);
Requirements.DampingRatio = sdo.requirements.PZDampingRatio;
```

The reliability requirement will be specified in the optimization objective function described below. The reliability requirement is used to tune the spring and damper values to minimize the expected failure rate over a lifetime of 100e3 miles. The failure rate is computed using a Weibull distribution on the damping ratio of the system. As the damping ratio increases, the failure rate is expected to increase.

Linearization Definition

The design requirements use the bandwidth and damping ratio of the system, these frequency domain characteristics require linearizing the model. Create a simulator for the model and use the simulator to compute the linear systems used by the requirements.

```
Simulator = sdo.SimulationTest('sdoSimpleSuspension');
```

Specify linear systems to compute by specifying the linearization inputs and outputs of the system. The linear system input is the output of the Band-Limited Disturbance Force block and the linear system output is the output of the x_dot block (the position signal).

```
IOs(1) = linio('sdoSimpleSuspension/Band-Limited Disturbance Force',1,'input');
IOs(2) = linio('sdoSimpleSuspension/x_dot',1,'output');
```

Add the linearization IOs to a sdo.SystemLoggingInfo object with a logging name, and linearization snapshot time. In this case, the snapshot time is set to 0, the model initial condition.

```
sys1 = sdo.SystemLoggingInfo;
sys1.Source = 'IOs';
sys1.LoggingName = 'IOs';
sys1.LinearizationIOs = IOs;
sys1.SnapshotTimes = 0;
```

Add the system logging info to the simulator. When the simulator is used to run the model, the linear system defined by the specified linearization IOs is computed and added to the simulation log with the specified logging name.

Simulator.SystemLoggingInfo = sys1;

Create Optimization Objective Function

Create a function that is called at each optimization iteration to evaluate the design requirements.

Use an anonymous function with one argument that calls sdoSimpleSuspension_Design. The sdoSimpleSuspension_Design function has arguments for the design parameters, the simulator, the design requirements, the uncertain variables, and uncertain variable values.

optimfcn = @(P) sdoSimpleSuspension_Design(P,Simulator,Requirements,UncVars,UncVars_Values);
type sdoSimpleSuspension_Design

```
function Vals = sdoSimpleSuspension Design(P,Simulator,Requirements,UncVars,UncVars Values)
%SDOSIMPLESUSPENSION DESIGN
%
%
     The sdoSimpleSuspension Design function is used to evaluate a simple
%
     suspension system design.
%
%
     The |P| input argument is the vector of suspension design parameters.
%
%
     The |Simulator| input argument is a sdo.SimulinkTest object used to
%
     simulate the IsdoSimpleSuspension model and log simulation signals.
%
%
     The [Requirements] input argument contains the design requirements
%
     used to evaluate the suspension design.
%
%
     The |UncVars| and |UncVars Values| arguments specify the uncertain
%
     variable and uncertain variable values.
%
     The |Vals| return argument contains information about the design
%
%
     evaluation that can be used by the [sdo.optimize] function to optimize
%
     the design.
%
% See also sdoSimpleSuspension cmddemo
% Copyright 2015 The MathWorks, Inc.
%% Evaluate the suspension reliability
%Get the spring and damper design values
allVarNames = {P.Name};
idx
           = strcmp(allVarNames, 'k');
           = P(idx).Value;
k
           = strcmp(allVarNames, 'b');
idx
           = P(idx).Value:
b
%Get the nominal mass from the model workspace
wksp = get param('sdoSimpleSuspension', 'ModelWorkspace');
m = evalin(wksp,'m0');
%The expected failure rate is defined by the Weibull cumulative
%distribution function, 1 - \exp(-(x/l)^k), where k=3, l is a function of the
%mass, spring and damper values, and x the lifetime.
        = b/2/sart(m*k);
d
pFailure = 1 - exp(-(100e3*d/250e3)^3);
%% Nominal Evaluation
%
% Evaluate the model and requirements with uncertain parameters set to their
% nominal values.
% Simulate the model.
Simulator.Parameters = [P(:);UncVars(:)];
Simulator = sim(Simulator);
% Retrieve logged linearizations.
Sys = find(Simulator.LoggedData, 'IOs');
% Evaluate the design requirements.
```

```
Cleg Bandwidth = evalRequirement(Requirements.Bandwidth,Sys.values);
Cleq DampingRatio = evalRequirement(Requirements.DampingRatio,Sys.values);
%% Uncertain Evaluation
%
% Evaluate the model and requirements for all combinations of uncertain
% parameter values.
for ct=1:size(UncVars Values,1)
    UncVars(1).Value = UncVars Values{ct,1};
    % Simulate the model.
    Simulator.Parameters = [P(:);UncVars(:)];
    Simulator = sim(Simulator);
    % Retrieve logged linearizations.
    Sys = find(Simulator.LoggedData, 'IOs');
    % Evaluate the design requirements.
    Cleq Bandwidth UncVars = evalRequirement(Requirements.Bandwidth,Sys.values);
    Cleq DampingRatio UncVars = evalRequirement(Requirements.DampingRatio,Sys.values);
    Cleq Bandwidth = [Cleq Bandwidth; Cleq Bandwidth UncVars]; %#ok<AGROW>
    Cleq_DampingRatio = [Cleq_DampingRatio; Cleq_DampingRatio_UncVars]; %#ok<AGROW>
end
%% Return Values.
%
% Collect the evaluated design requirement values in a structure to
% return to the optimization solver.
Vals.F = pFailure;
Vals.Cleq = [...
    Cleq Bandwidth(:); ...
    Cleg DampingRatio(:)];
end
```

Optimization Options

Specify optimization options.

```
Options = sdo.OptimizeOptions;
Options.MethodOptions.Algorithm = 'active-set';
Options.MethodOptions.TolGradCon = le-06;
```

Optimize the Design

Call sdo.optimize with the objective function handle, parameters to optimize, and options.

[Optimized_DesignVars,Info] = sdo.optimize(optimfcn,DesignVars,Options);

Optimization started 02-Sep-2021 00:18:47

			max		First-order
Iter H	-count	f(x)	constraint	Step-size	optimality
0	5	0.0619897	0.7642		
1	10	0.351266	-0.1277	0.461	1.22
2	15	0.189345	-0.03881	0.188	0.312
3	20	0.0829063	0.01312	0.51	0.425
4	24	0.0365398	Θ	0.308	1.55

5	28	0.02	294977	7	0	0.0143	0.733
6	32	0.02	293065	5	0	0.000417	0.00142
Local	minimum	found	that	satisfies	the	constraints.	

Optimization completed because the objective function is non-decreasing in feasible directions, to within the value of the optimality tolerance, and constraints are satisfied to within the value of the constraint tolerance.

Related Examples

To learn how to optimize the suspension system using the **Response Optimizer**, see "Design Optimization to Meet Frequency-Domain Requirements (GUI)" on page 3-129.

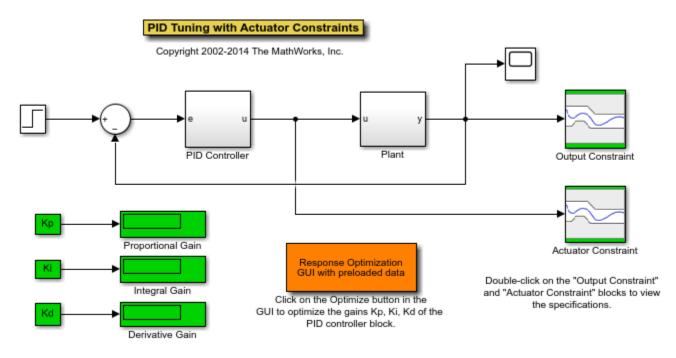
Close the model.

bdclose('sdoSimpleSuspension')

PID Tuning with Actuator Constraints

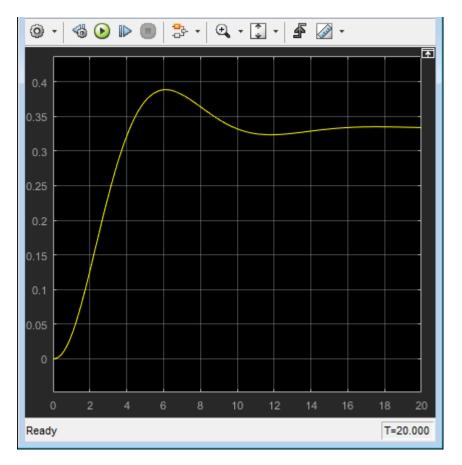
This example shows how to use Simulink® Design Optimization^m to tune the gains of the PID controller (Kp, Ki, and Kd) and optimize the step response of the plant. To view the results, use the following steps.

Open the pidtune_demo model using the command below and run the simulation. The simulation produces an unoptimized step response and the initial data for optimization.



open_system('pidtune_demo')

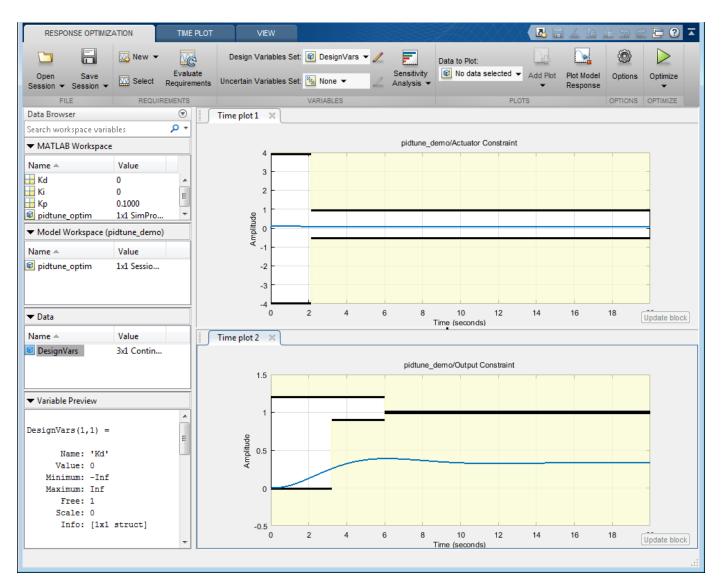
Double-click the **Scope** block to view the unoptimized step response.



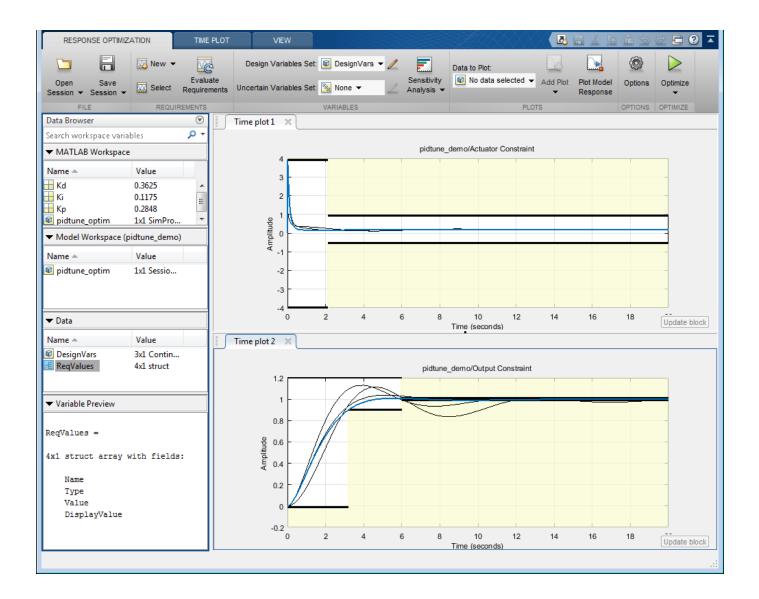
Double-click the **Output Constraint** block to view constraints on the plant response, including rise time, settling time and maximum overshoot.

Double-click the Actuator Constraint block to view constraints on the actuator signal of the controller.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



We start the optimization by pressing the **Optimize** button from the **Response Optimizer**. The plots are updated to indicate that the design requirements are now satisfied.



Iteration	F-count	Actuator Constrai	Actuator Constrai	Output Constrain	Output Constrain
		(<=0)	(>=0)	(<=0)	(>=0)
0	7	-0.9085	1	-0.6152	-0.7165
1	15	0.5718	0.9630	-0.0223	-0.3879
2	23	-0.0375	1	-0.0261	-0.1314
3	33	-0.6847	1	0.0017	-0.1559
4	41	-1.0000e-05	1	-7.3493e-04	-0.0596
5	49	-4.5433e-16	1	0.0367	-0.0052
6	57	-0.0486	1	-0.0066	-0.0078
7	65	0	1	-0.0030	-5.1137e-04
8	73	-2.2717e-16	1	-0.0019	-1.4798e-05
9	81	-1.1358e-16	1	-0.0018	1.0091e-09

Optimization started 27-Mar-2013 07:21:33				•
Optimization converged, 27-Mar-2013 07:22:22				
Optimized variable values written to 'DesignVars'	in the Design Optim	zation workspace		-
	Save Iteration	Display Options	Optimize	

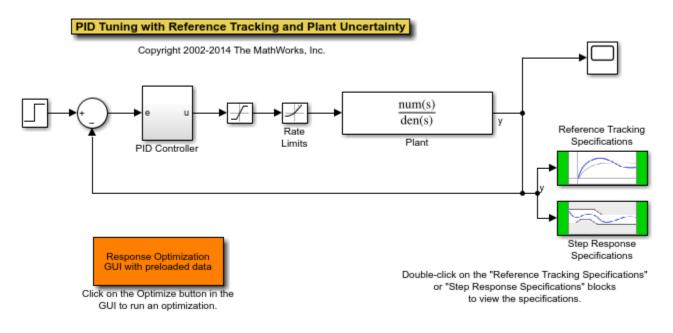
Close the model.

bdclose('pidtune_demo')

PID Tuning with Reference Tracking and Plant Uncertainty

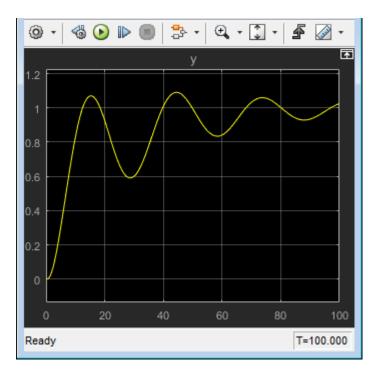
This example shows how to use Simulink® Design Optimization^m to track a reference signal and optimize the response with uncertainties in the plant model. The plant model consists of the plant transfer function and includes a Saturation block and a Rate Limits block. To view the results, use the following steps.

Open the pidtrack_demo model using the command below and run the simulation. The simulation produces an unoptimized step response and the initial data for optimization.



open_system('pidtrack_demo')

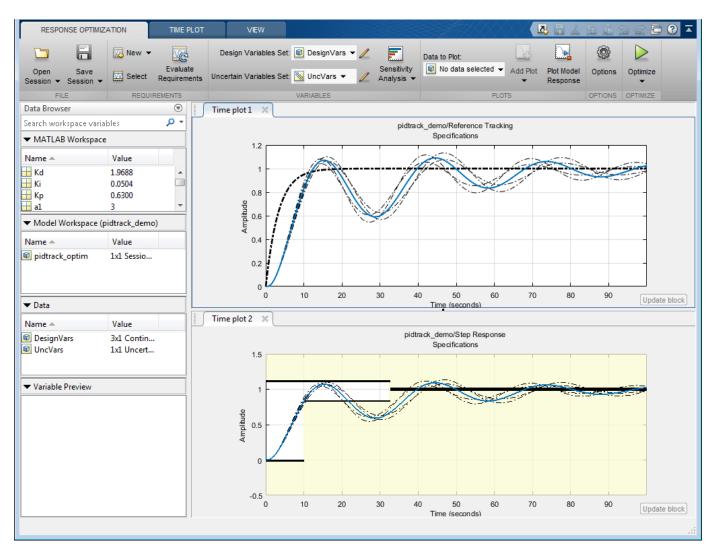
Double-click the Scope block to view the unoptimized step response.



Double-click the Step Response Specifications block to view constraints on the plant response, including rise time, settling time and maximum overshoot.

Double-click the Reference Tracking Specifications block to view the reference signal that the controller has to track.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



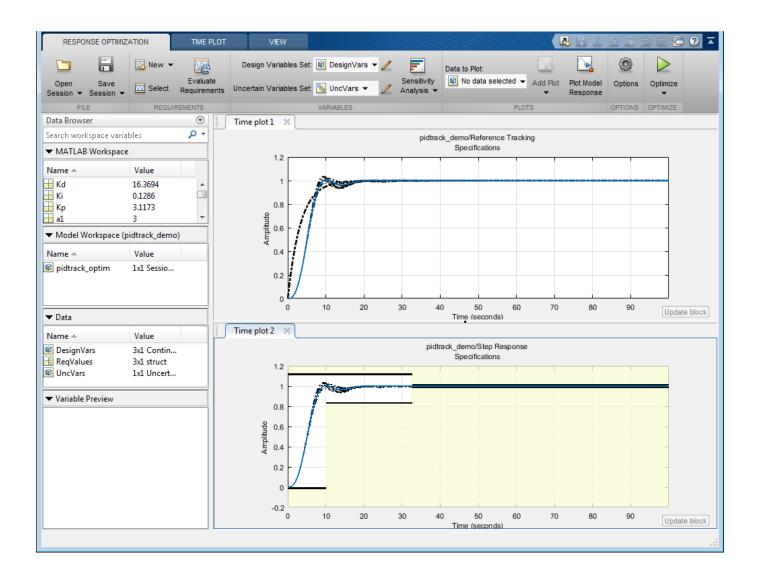
The solid lines in the plots indicate the plant response without considering the uncertainties and the dashed lines indicate the uncertain responses.

Start the optimization by pressing the **Optimize** button from the **Response Optimizer**.

The tuned parameters are the PID Controller gains Kp, Ki, and Kd. The plant parameters a1 and a2 are only known within 10% (uncertainty).

The optimization seeks to minimize the gap between the actual and ideal responses for the nominal values (solid lines) and min/max values (dotted lines) of al and a2.

The plots are updated to indicate that the design requirements are now satisfied.



teration	F-count	Reference Tracking Spec (min)	Step Response Specificat (<=0)	Step Response Specifica (>=0)
0	8	124.4264	0.1215	-0.39
1	28	107.9560	0.0851	-0.37
2	42	39.8559	0.0965	-0.09
3	55	45.9156	0.0918	0.00
4	63	37.3423	0.0347	0.00
5	74	36.2234	0.0606	0.01
6	82	32.6911	0.0038	-0.02
7	91	29.4846	0.0172	0.01
8	100	28.4823	-0.0114	-0.01
9	114	28.0177	-0.0120	-0.01
10	122	28.7452	-0.0052	0.00
11	132	27.7101	-0.0101	-6.6269e-

Optimization started 27-Mar-2013 07:24:39			c	*
Optimization converged, 27-Mar-2013 07:41:42				
Optimized variable values written to 'DesignVars'	in the Design Optim	ization workspace		Ŧ
	Save Iteration	Display Options	Optimize	

% Close the model.
bdclose('pidtrack_demo')

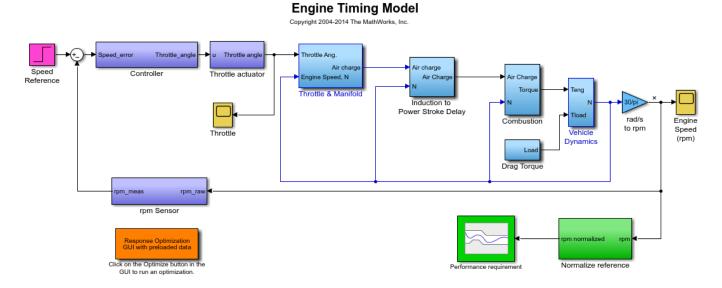
Engine Design and Cost Tradeoffs

This example shows how to use Simulink® Design Optimization[™] to optimize a design for performance and cost. In this example, you tune an automotive engine speed controller while reducing controller costs by tuning sensor accuracy and actuator response time.

Opening the Model

Open the automotive engine model using the command below. The following subsystems model the engine response: Throttle & Manifold, Induction to Power Stroke Delay, Combustion, Drag Torque and Vehicle Dynamics. The main signal of interest in the model is the engine speed in rpm. The control system consists of the lead-lag Controller, rpm Sensor, and Throttle actuator blocks. The model is driven by a step change in speed reference.

open_system('enginetradeoff_demo')



Design Overview

The design has the following objectives:

Engine performance objective:

The engine must respond to step changes in the speed reference with the following characteristics:

- Maximum overshoot of 2%
- Rise time of 4 seconds to reach 90% of the reference speed
- Settling time of 7.5 seconds to reach within 2% of the reference speed

This objective is included as a constraint in the **Performance requirement** block. The input to this block is the engine speed, which is normalized by the speed reference value. This means that although the speed reference changes, the performance requirement remains the same.

Cost minimization objective:

The design cost of the controller is to be minimized. This objective uses sensor and actuator parameterization to compute a design cost. The design cost is computed so that it is always greater than 1 and the optimization attempts to drive the cost to 1.

We use a custom requirement to minimize this cost. The enginetradeoff_cost function used by the custom requirement simply returns the cost value to be minimized.

```
type enginetradeoff_cost
```

```
function Cost = enginetradeoff cost(u)
%Compute controller cost based on sensor accuracy, actuator response, and
%controller sampling time.
% Copyright 2013 The MathWorks, Inc.
%Cost constants
min_cost = 1;
                           %Minimum cost > 0
sensor_var_min = 1e-3; %Minimum sensor variance (most expensive)
sampling_min = 1e-2; %Minimum controller sampling time (most expensive)
throttle max = 2*pi*10; %Maximum actuator response frequency (most expensive)
%Get variable names
varnames = {u.DesignVars.Name};
%Form cost
Cost = min cost;
%Add sensor cost
index = strcmp(varnames, 'sensor std');
if any(index)
 Cost = Cost + sensor var min./max(u.DesignVars(index).Value,sensor var min);
end
%Add sampling cost
index = strcmp(varnames, 'Ts');
if anv(index)
 Cost = Cost+ sampling min./max(u.DesignVars(index).Value,sampling min);
end
%Add throttle cost
index = strcmp(varnames, 'Tthrottle');
if any(index)
 Cost = Cost + u.DesignVars(index).Value/throttle max;
end
```

Model parameterization:

To achieve the performance and cost objectives, we parameterize the following in the model:

- Final step value of the speed reference: This ensures that the design works over a range of operating points, from low speed to high speed values.
- Gain, pole and zero values of the controller: This allows us to change the controller performance. We tune these values using optimization.
- Response time of the throttle actuator: The response time is optimized to minimize controller cost. The actuator cost is inversely proportional to the response time, i.e., a faster response time implies a more expensive actuator.

• Accuracy of the rpm sensor: The accuracy is specified by a standard deviation value and optimized to minimize controller cost. The sensor cost is inversely proportional to standard deviation, i.e., a smaller standard deviation implies a more accurate sensor, which is more expensive.

Running the Optimization

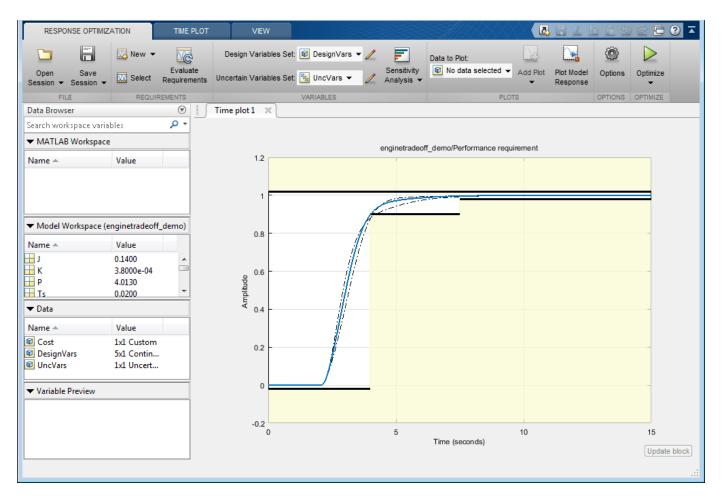
Optimizing the engine controller for performance and cost involves:

- Optimizing the controller, sensor, and actuator parameters.
- Optimizing the response over different operating conditions.

With problems of this type, it is a good practice to build the design iteratively rather than optimizing for all objectives together. Here, we use the divide and conquer strategy as shown in the table below. The idea is to use the optimized parameter values from one stage as the initial guess for the next stage.

	Tune controller parameters	Sweep model over the operating range	Tune sensor accuracy	Tune throttle response time
Stage 1	Х			
Stage 2	Х	X		
Stage 3	Х		Х	
Stage 4	х		Х	Х
Stage 5	х	X	Х	Х

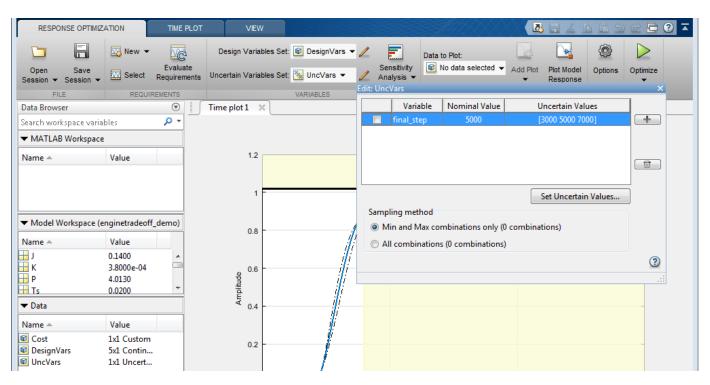
You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



The optimization project saved with the example corresponds to stage 2. In this stage, we optimize the controller parameters over the operating range. To do so, we specify K, P and Z as tuned parameters and the final step value, final_step, of the reference signal as an uncertain parameter. At this stage, the custom controller cost objective is not included in the optimization problem.

We introduce the controller cost objective in stage 3 by configuring the model as follows:

• Open the Uncertain Variables editor from the **Response Optimizer** and then unselect the check box for final_step. This removes sweeping the model over different operating points from the optimization problem and thus reduces computational load.



• Open the Design Variables editor from the **Response Optimizer** and then select the check box for sensor_std to tune this parameter.

RESPONSE OPTIMIZ	ATION	TIME	PLOT	VIEW						XX		c il s	2 🗖 ?
	🔛 New 👻	Me	4	Design Variabl	es Set: 😰 DesignVar			Data to Plot:				٢	\triangleright
Open Save Session ▼ Session ▼	Select	Evalua Requirem		Uncertain Variabl	es Set: 📉 UncVars 🔻	- EC	lit: Desi	gnVars Variable	Value	Minimum	Maximum	Scale	
FILE	REQU	IREMENTS			VARIABLES		1	К	0.00038	-1	1	0.00038	
Data Browser		$\overline{\mathbf{v}}$		Time plot 1 🛛 🛛			1	P	4.013	0	10	4.013	
Search workspace varia	bles	- م					1	Z	1.454	0	10	1.454	
▼ MATLAB Workspace								Tthrottle	63	1	100	63	
				1.2				sensor_std	0.001	0	0.1	0.001	
Name 🔺	Value			1 F			Varia	ble Detail		Up	odate model	variables	
													?
▼ Model Workspace (e	nginetradeo	ff_demo)		0.8 -		Į.							.::
Name 🔺	Value					1							
H J	0.1400					<u>II</u>							
J K P	3.8000e-04			0.6	- i	lí—							
P P	4.0130			mplitude 0.0	j j	í.							
🕂 Ts	0.0200	T		Id	il i i i i i i i i i i i i i i i i i i	í –							

• Click the **Select** button in the **Response Optimizer** to open the **Design Requirements** editor, and then select the check box for custom **Cost**. This accounts for the cost minimization objective during optimization.

RESPO	ONSE OPTIMIZA	TION	TIME PLOT	VIEW				HAN .			i i s	c 🗗	? ⊼
		🔀 New	-	Design Variables Set	: 😰 DesignVars 👻 🥖	E	Data to P				٢	\triangleright	
Open Session 👻	Save Session 👻	Select Dec		Uncertain Variables Set	t 📉 UncVars 👻 🥖	Sensitivity Analysis 👻		data selected 🤜	Add Plot	Plot Model Response	Options	Optimize	
FI	LE	Select Ke	quirements				~	PLC	DTS		OPTIONS	OPTIMIZE	
Data Brows	ser		Select I	Design Requireme	nts To Optimize		_						
Search wor	kspace variab			Requiren	nents								
▼ MATLA	B Workspace			emo/Performance requ									
			Cost (Custom 'm	in' requirement using t	he function 'enginetra	leoff_cost')		rformance req	uirement				
Name 🔺													
		Detai	I				_						
▼ Model V	Vorkspace (er												
Name 🔺		Value		0.0	1							_	
		0.1400			<u></u>								
F I		0.1400			(f)								

RESPO	NSE OPTIMIZ	ATION	TIME PLOT	VIEW							à É S	20	? -
		New 👻	Me	Design Variables S	et: 😰 DesignVars 🔻	/	.	Data to Plot:			٢	\triangleright	
Open Session 👻	Save Session 💌	Select	Evaluate Requirements	Uncertain Variables S	et: 📉 UncVars 👻	1	Sensitivity Analysis 💌	😰 No data selected 👻	Add Plot	Plot Model Response	Options	Optimize	
FI		REQUI	REMENTS		VARIABLES			PLC	TS		OPTIONS	OPTIMIZE	
Data Brows	ser			Time plot 1 🛛 🗶									
Search wor	kspace varial	bles	ب م										
▼ MATLA	B Workspace						enginetradeol	ff_demo/Performance req	uirement				
Name 🔺		Value		1.2									
				1 -							_	_	
							-						
▼ Model V	Vorkspace (e	nginetradeof	f_demo)	0.8									
Name 🔺		Value		0.0									
H J		0.1400			/								
н		4.0482e-04		o 0.6	/								
P Ts		3.7634 0.0200	-	litud									
▼ Data		0.0200		opnjildwe 0.4									
Name 🔺		Value											
Cost		1x1 Custom			- 1								
🖉 DesignV	ars	5x1 Contin.		0.2	- 1							_	
🗄 ReqValu		3x1 struct											
UncVars	:	1x1 Uncert.		0									
▼ Variable	Preview			0		_							
				-0.2									
				0			5	Time (seconds)	10			15	
								- ,,				Update	block

teration	F-count	Cost (min)	Performance requireme (<=0)	Performance requireme (>=0)
0	10	2	-0.0194	-0.00
1	20	1.6796	-0.0194	-1.4044e-
2	30	1.2639	-0.0189	0.00
3	40	1.2381	-0.0189	0.00
4	50	1.1702	-0.0186	0.01
5	60	1.1311	-0.0182	0.01
6	70	1.0982	-0.0178	0.01
7	80	1.0744	-0.0172	0.01
8	90	1.0561	-0.0163	0.01
9	100	1.0423	-0.0097	0.01
10	110	1.0322	-5.3178e-04	0.01
11	120	1.0244	-3.5049e-04	0.01
•				4
Optimizati	on converge	'-Mar-2013 08:17:01 d, 27-Mar-2013 08:18:11 ues written to 'DesignVars' in t	he Design Optimization worksp	ace .

To configure the optimization for Stage 4:

• Open the Design Variables editor from the **Response Optimizer** and then select the check box for Tthrottle to tune this parameter.

RESPO	NSE OPTIMIZ/	ATION	TIME PLO	T VIEW			All!	XXX				e 🗗 ใ
		New 👻	Me	Design Variables	Set: 😰 DesignVars 💌	-	Data to Plot				٢	\triangleright
Open Session 💌	Save Session 💌	Select	Evaluate Requirements	S Uncertain Variables	Set: 📐 UncVars 💌	Edit: Desi	gnVars Variable	Value	Minimum	Maximum	Scale	×
FIL	E	REQUI	REMENTS		VARIABLES	V	ĸ	0.000404	-1	1	0.00038	+
Data Brows	er			Time plot 1 🛛 🗶		V	P	3.763376	0	10	4.013	
Search work	space varial	oles	۰ م				Z	1.530861	0	10	1.454	
▼ MATLAB	Workspace					V	Tthrottle	63		100	63	
				1.2		V	sensor_std	0.040978	0	0.1	0.001	
Name 🔺		Value		1.2		Noris	ble Detail		Up	date model	variables]
				1		► Varia	Die Detail					?
▼ Model W Name ▲	orkspace (e	nginetradeof Value	f_demo)	0.8	/							.::

You can start the optimization by pressing the **Optimize** button from **Response Optimizer**.

To configure the optimization for stage 5:

• Open the Uncertain Variables editor from the **Response Optimizer** and then select the check box for final_step to sweep the model over the operating range.

RESPONSE O	OPTIMIZATION	TIME	PLOT	VIEW				ŻŻ	Ì	HAN (i li 2	e E	? 🔺
	Nev	v • 🕅		Design Variabl	es Set: 😰 DesignVars	• /	E		Data t				٢	\triangleright	
	sion 👻 🔤 Sel	Evalu ect Require		certain Variabl	es Set: 📉 UncVars 🔻	1	Sensit Analys	sis 👻		lo data selected 👻	Add Plot	Plot Model Response	Options	Optimize	
FILE	R	EQUIREMENTS			VARIABLES	Ean	t: UncVar	5						X	
Data Browser			Tim	ne plot 1 🛛 🛛				Varia	ble	Nominal Value	ι	Jncertain Val	ues		
Search workspac	e variables	+ م					🗾 fir	nal_ste	p	5000	[3000 5000 70	00]	+	
▼ MATLAB Wor	kspace														
Name 🔺	Value			1.2											
				1 -		-						Set Uncertai	n Values		
	·····	last dans - \					Sampling	g meth	hod						
 Model Works 	pace (enginetra	ieott_demo)		0.8		/	Min a	and M	ax cor	mbinations only (2 combina	tions)			
Name 🔺	Value					r i	◎ All co	ombin	ations	(3 combinations)				
J K	0.1400				/									?	
н	4.0482e	-04	J	_ى 0.6	/	-									
P	3.7634	-		itud	/										1
H Ts	0.0200	•		Amplitude											
▼ Data				₹ 0.4 -										-	

You can start the optimization by pressing the **Optimize** button from the **Response Optimizer**.

% Close the model
bdclose('enginetradeoff_demo')

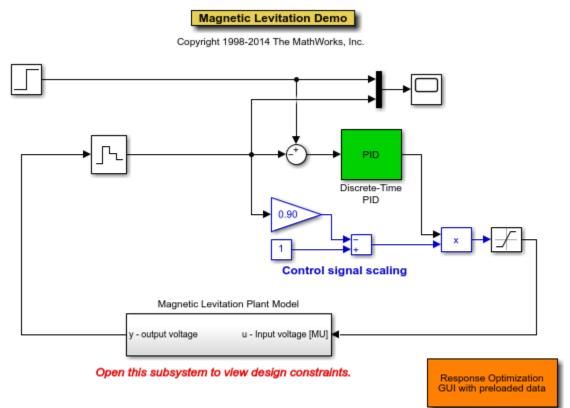
Magnetic Levitation Controller Tuning

This example shows how to use numerical optimization to tuning the controller parameters of a nonlinear system. In this example, we model a CE 152 Magnetic Levitation system where the controller is used to position a freely levitating ball in a magnetic field. The control structure for this model is fixed and the required controller performance can be specified in terms of an idealized time response.

Earnshaw's Theorem

Earnshaw's theorem proved that it is not possible to achieve stable levitation using static, macroscopic, classical electromagnetic fields. However the CE 152 system works around this by creating a potential well around the point at which the ball is to be suspended, thereby creating a non-inverse square law force. This is achieved by an inductive coil that generates a time varying electromagnetic field. The electromagnetic field is controlled through the use of feedback to keep the ball at the required location.

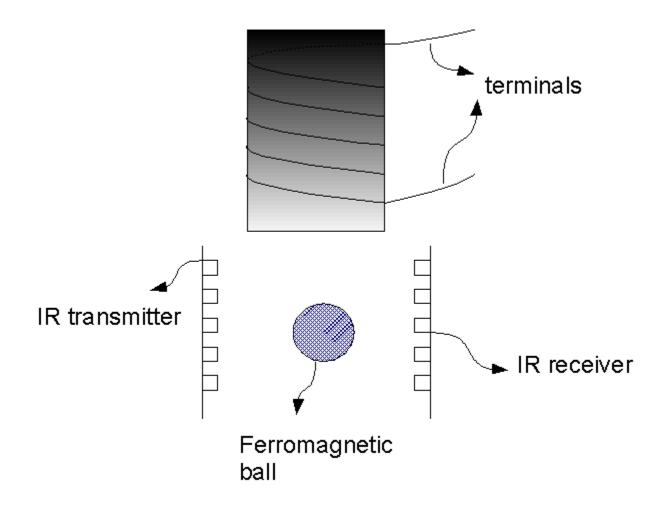
open_system('maglev_demo')



Click on the Optimize button in the GUI to run an optimization.

Model Description

The magnetic levitation system is a nonlinear dynamic system with one input and one output. Doubleclick the Magnetic Levitation Plant Model to open this subsystem. The input voltage is applied to a coil that creates the electromagnetic field. The output voltage is measured by an IR receiver and represents the position of the ball in the magnetic field. The diagram below outlines this system.

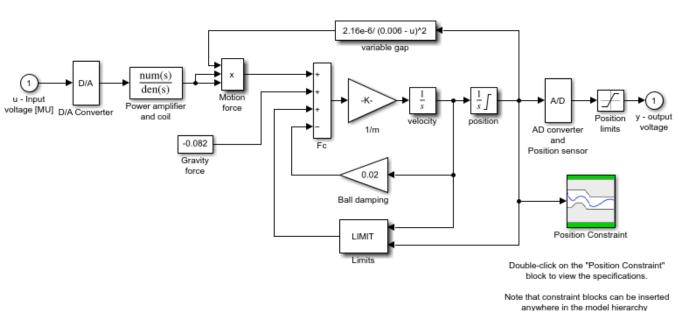


The physical system consists of a ball (with mass 0.00837 kg) which is under the influence of three forces:

- The magnetic field produced by an inductive coil. This is modeled by the Power amplifier and coil block in the Simulink® model. The input to the inductor is a voltage signal and the output a current. The force from the coil depends on the square of the current, the air-gap between the coil and the ball, and the physical properties of the ball. This produces an upward acting force on the ball.
- The gravitational force acting downwards
- A damping force which acts in a direction opposite to the velocity at any instant of time

These three forces cause the resulting motion of the ball and are modeled in Simulink as shown.

open_system('maglev_demo/Magnetic Levitation Plant Model')



Magnetic levitation detailed nonlinear model

Non-linearities arising from saturation of the coil and changes in dynamics outside the limits of the magnetic field are also modeled in the Simulink Model. As the force from the coil decays according to an inverse square law larger voltages are required the further the ball is from the coil. The control signal is scaled to account for this and the scaling is included in the *Control signal scaling* blocks.

Control Problem Description

The requirement for the controller is that it be able to position the ball at any arbitrary location in the magnetic field and that it move the ball from one position to another. These requirements are captured by placing step response bounds on the position measurement. Specifically we require the following constraints on the ball:

- Position constraint: within 20% of the desired position in less than 0.5 second
- Settling Time Constraint: within 2% of the desired position within 1.5 second

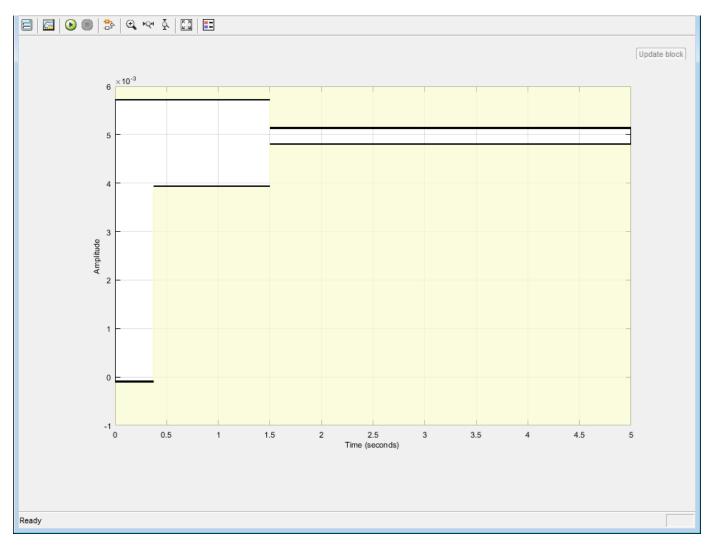
To meet the control requirements we implement a Proportional-Integral-Derivative (PID) controller. For convenience the controller uses a normalized position measurement with a range from 0 to 1, representing the bottom-most and top-most positions of the ball respectively.

Simulink \mbox{B} Design Optimization \mbox{T} and numerical optimization is ideally suited to tune the PID coefficients because:

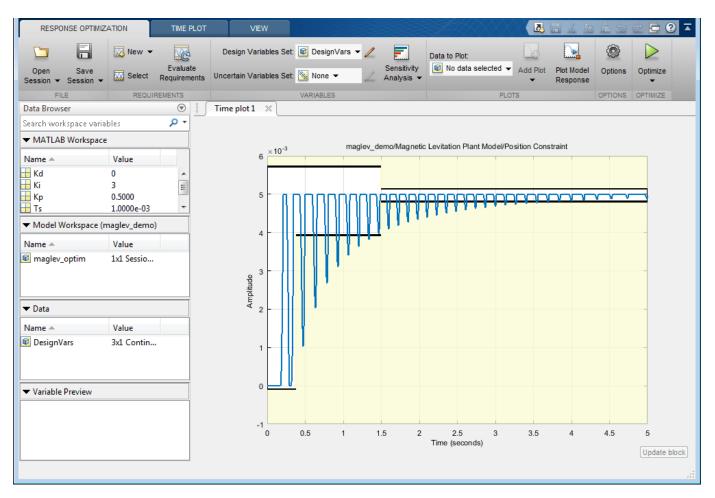
- The system dynamics are complex enough to require effort and time for analysis if we approach the problem using conventional control design techniques.
- The controller structure is fixed.
- We have knowledge of the step response we require from the system.

Setting Constraint Values

Given the step response characteristic we desire, it is simple to specify the upper and lower bounds of the response. Double-click the Position Constraint block in the Magnetic Levitation Plant Model subsystem to view constraints on the position of the ball. The constraint lines may be moved using the mouse.



You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



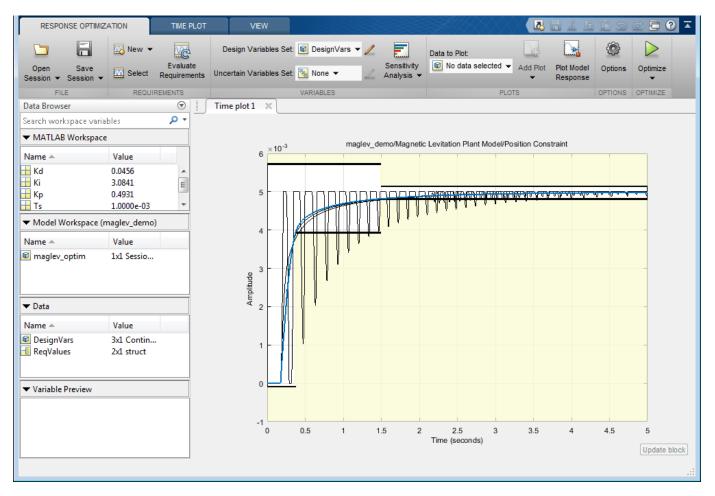
Defining Tuned Parameters

We select the PID controller parameters to tune by opening the Design Variables editor, as shown below.

RESPONSE C	OPTIMIZAT	ION	TIME PLO	T VIEW			ŚŚ	AH)	\times			Ê 9 (2 🗗 🕐
		🔀 New 👻	MC	Design Variables Set	t: 😰 DesignVars 👻		-	Data to Plot:				٢	\geq
	ave sion 👻	Select	Evaluate Requirement	s Uncertain Variables Set		dit: Des	-	Variable	Value	Minimum	Maximum	Scale	
FILE		REQUI	REMENTS		VARIABLES	1	Кр		0.5	-Inf	Inf	0.5	
Data Browser				Time plot 1 🛛 🕅		1	Ki		3	-Inf	Inf	3	
Search workspac	e variable	es	- م			1	Kd		0	-Inf	Inf	0	1 [
▼ MATLAB Work Name ▲		/alue		6 ×10 ⁻³	magl								
Η Kd	0	1	*							Up	date model	variables	
Ki Kp Ts	-	.5000 .0000e-03	Ŧ	5	ההחחחה	Varia	ible Det	ail					3
▼ Model Works	pace (ma	glev_demo)										.::
Name 🔺	١	/alue		4									

Running the Optimization

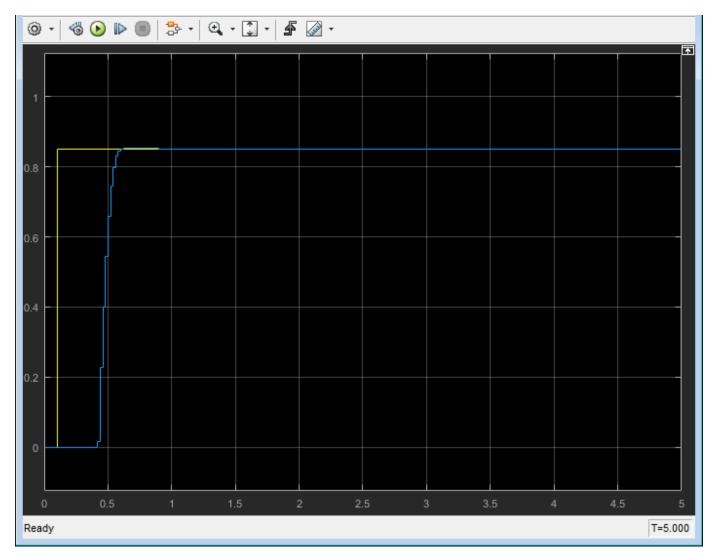
After specifying the optimization parameters and the required step response bounds we start the optimization by pressing the **Optimize** button from the **Response Optimizer**. During optimization, the plots are updated with the position of the ball for each iteration and the dark curve shows the final optimized trajectory of the ball (as shown below).



lteration	F-count	Magnetic Levitation Plant Model/Posi (<=0)	Magnetic Levitation Plant Model/Posi (>=0)
0	7	-0.0270	-0.7358
1	15	-0.0270	-0.0388
2	23	-0.0308	-0.0132
3	31	-0.0305	-0.0027
4	39	-0.0305	-5.2941e-06
Optimizati	on converg	27-Mar-2013 07:45:07 ed, 27-Mar-2013 07:45:18	mitation worknoon
optimized	variable va	lues written to 'DesignVars' in the Design Opti	mization workspace

Verifying the Results

Once we complete the optimization, it is important to validate the results against other step sizes. A successful parameter optimization should be able to provide good control for all steps sizes close to the tuned step size of 1. Step sizes from .7 to 1 should be tested to confirm the controller's performance. The following plot shows the response to a step input from 0 to 0.85 at 0.1 seconds.



Conclusion

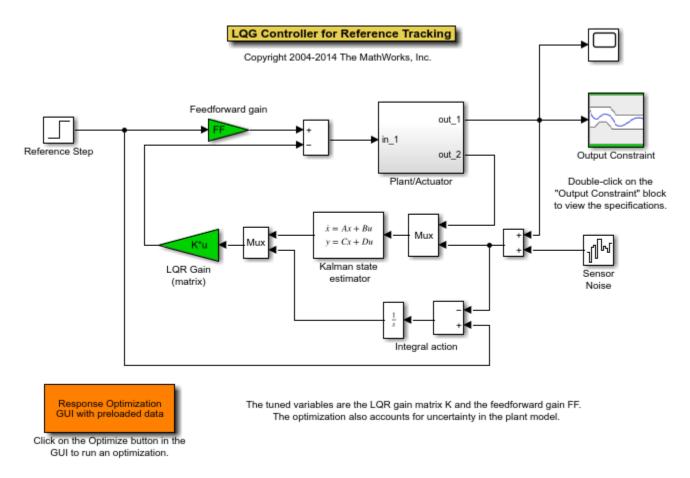
The verification step shows that controller's performance satisfies the requirements specified and the tuned parameter values are suitable for control. The tuned parameters could be used to provide baseline performance against which other control schemes can be compared, or a baseline for controllers for different operating regions.

% Close the model. bdclose('maglev_demo')

LQG Controller Tuning

This example shows how to use Simulink® Design OptimizationTM to optimize the output response of a plant by tuning the LQR gain matrix and feed-forward gain. This model includes uncertainty in the plant model and accounts for this uncertainty in the optimization process.

Open the lqg_demo model using the command below and run the simulation. The simulation produces an unoptimized response of the plant and the initial data for optimization. Double-click the Scope block to view the unoptimized response of the plant.

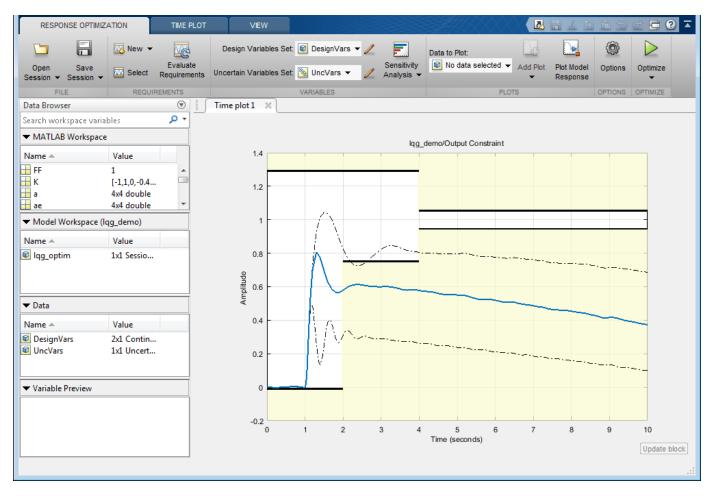


open_system('lqg_demo')

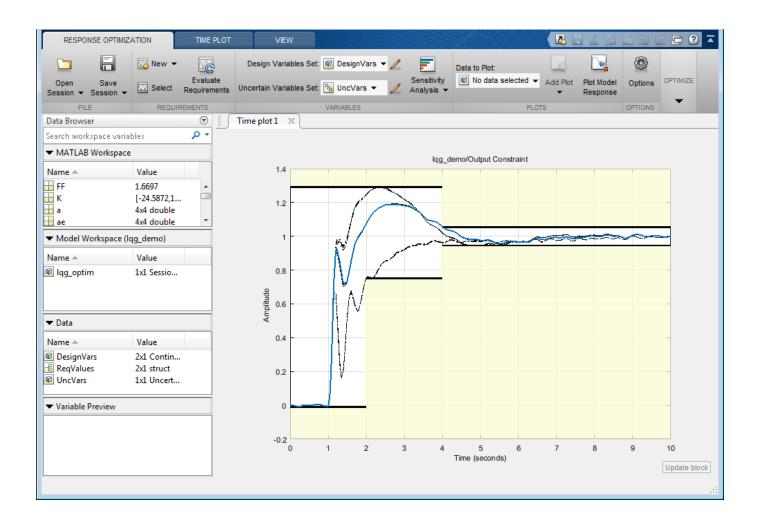
Double-click the Plant/Actuator block to view the details of the subsystem. Note that the plant is represented in State-Space form in this model and includes Rate Limiter and Saturation blocks.

Double-click the **Output** Constraint block to view constraints on the step response of the plant.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



The solid line in the plot indicates the plant response without considering the uncertainties and the dashed lines indicate the uncertain responses.



lteration	F-count	Output Constraint (Upper) (<=0)	Output Constraint (Lower) (>=0)
0	13	-0.1922	-0.8926
1	27	-0.0592	-0.4918
2	42	0.1625	-0.3928
3	71	0.1519	-0.3909
4	85	0.0188	-0.2274
5	99	0.2201	-0.073
6	113	0.1944	-0.084
7	127	0.0301	-0.032
8	141	0.0146	-0.0093
9	155	-0.0015	-3.6894e-04
10	169	-0.0015	-1.2161e-04

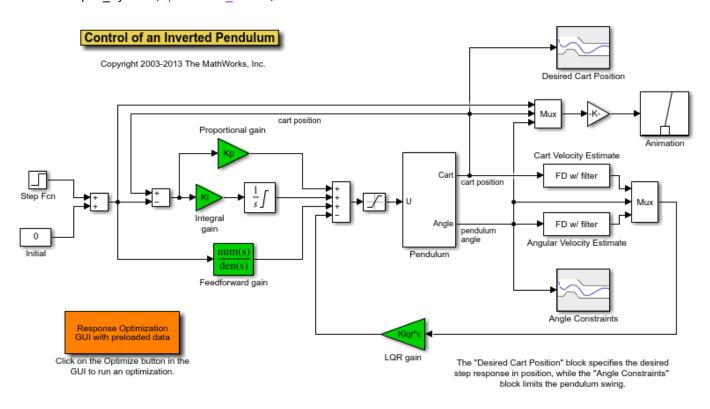
Optimization started 27-Mar-2013 07:47:22			
Optimization converged, 27-Mar-2013 07:48:33			
Optimized variable values written to 'DesignVars' in	n the Design Optimi	zation workspace	-
(Save Iteration	Display Options	Optimize

% Close the model. bdclose('lqg_demo')

Inverted Pendulum Controller Tuning

This example shows how to use Simulink® Design Optimization[™] to optimize the controller of an inverted pendulum. The inverted pendulum is on a cart and the motion of the cart is controlled. The controller's Proportional, Integral and Feed-forward gains are tuned to limit pendulum angle variations and respond to step changes in cart position optimally.

Open the pendulum_demo model using the command below and run the simulation. The simulation produces an unoptimized position and angle of the inverted pendulum and the initial data for optimization. An animation window shows the cart position and inverted pendulum angle.

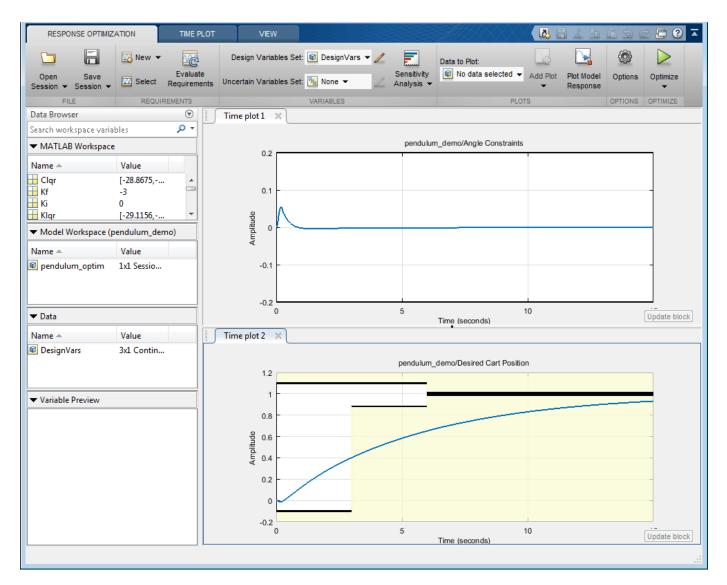


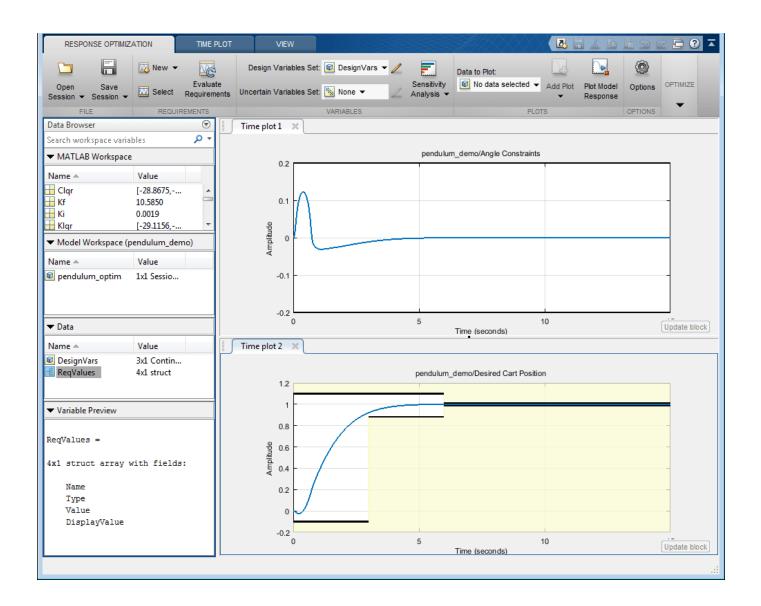
open system('pendulum demo')

Double-click the **Desired Cart Position** block to view constraints on the cart position of the inverted pendulum.

Double-click the Angle Constraints block to view constraints on the angle of the inverted pendulum.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.





Iteration	F-count	Angle Constraints			
		(<=0)	(>=0)	(<=0)	(>=0)
0	1	-0.7285	0.9794	-0.0778	-0.54
0	1	-0.7285	0.9794	-0.0778	-0.54
1	4	-0.7138	0.9781	-0.0685	-0.52
2	5	-0.7138	0.9781	-0.0685	-0.52
3	7	-0.6907	0.9753	-0.0629	-0.51
4	9	-0.6763	0.9741	-0.0513	-0.48
5	11	-0.6559	0.9715	-0.0440	-0.46
6	13	-0.5866	0.9579	-0.0311	-0.39
7	15	-0.5121	0.9227	-0.0198	-0.31
8	17	-0.4671	0.8700	-0.0183	-0.27
9	19	-0.4671	0.8700	-0.0183	-0.27
10	21	-0.4282	0.7253	-0.0165	-0.23

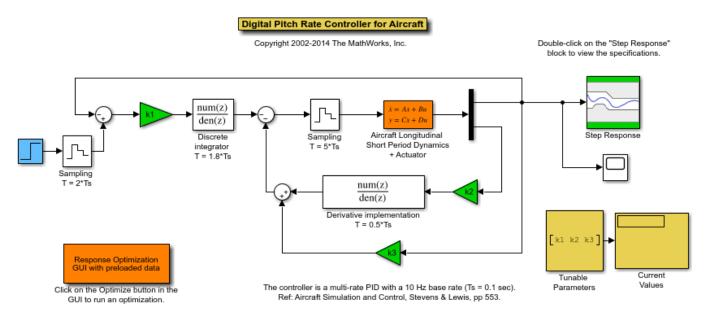
Optimization started 08-Apr-2013 14:2	4:10
Phase one: Finding a feasible solution	
Optimization converged, 08-Apr-2013	14:24:54
	Save Iteration Display Options Optimize

% Close the model.
bdclose('pendulum_demo')

Pitch Rate Controller Tuning

This example shows how to use Simulink® Design Optimization[™] to tune the gains of a Digital Pitch Rate Controller and optimize the response of an Aircraft to a step altitude change. The controller includes state derivative and integral feedback. The controller is tuned to satisfy a 10 percent overshoot and 0.9 second rise time step response characteristic.

Open the pitchrate_demo model using the command below and run the simulation. The simulation produces an unoptimized response of the aircraft and the initial data for optimization.

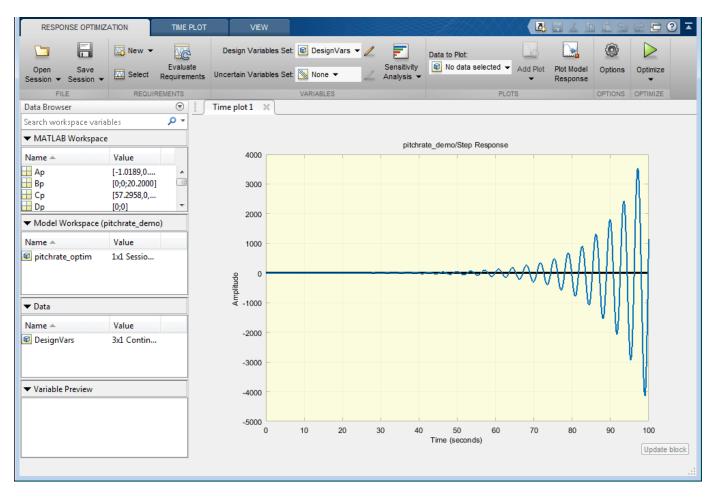


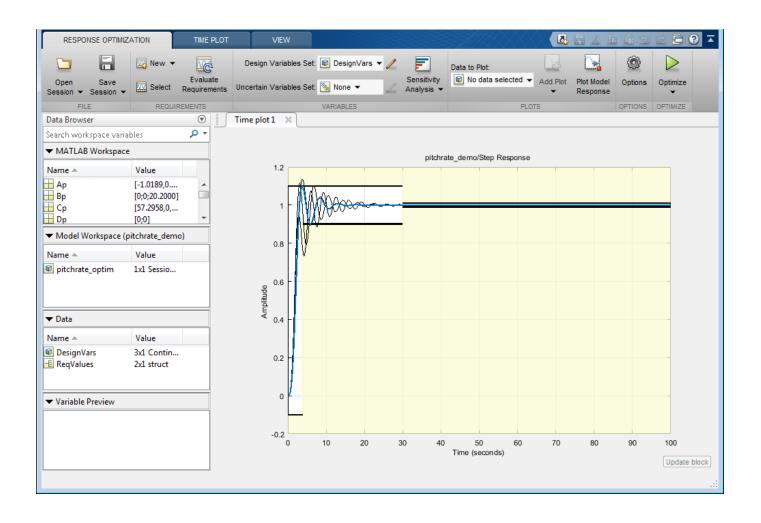
open_system('pitchrate_demo')

Double-click the Scope block to view the unoptimized response of the aircraft.

Double-click the Step Response block to view constraints on the step response of the aircraft.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.





Iteration	F-count	Step Response (Upper)	Step Response (Lower)
		(<=0)	(>=0)
0	7	3.4988e+03	-4.1820e+03
1	15	188.7803	-169.8364
2	23	8.9447	-8.5047
3	31	0.4819	-0.4885
4	39	0.2357	-0.3456
5	47	-0.0078	-0.1864
6	57	0.0131	-0.1190
7	65	0.0330	-0.0057
8	73	6.2400e-04	-0.0021
9	81	-1.8657e-04	9.1240e-05

Optimization started 27-Mar-2013 07:54:25	-	
Optimization converged, 27-Mar-2013 07:54:43		
Optimized variable values written to 'DesignVars' in the Design Optimization workspace	Ŧ	
Save Iteration Display Options Optimize		

bdclose('pitchrate_demo')

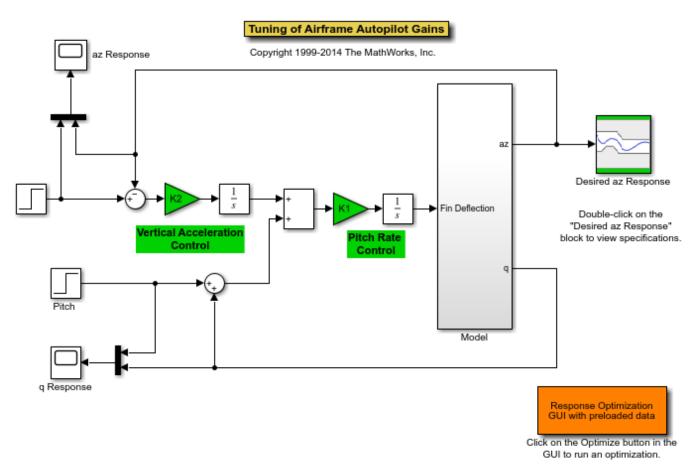
Tuning of Airframe Autopilot Gains

open_system('nlairframe demo')

This example shows how to apply Simulink® Design Optimization[™] to optimize the autopilot gains of an airframe to control its fin deflection. The model uses blocks from Aerospace Blockset[™].

The autopilot controller consists of an inner pitch loop and outer vertical acceleration loop. Both controllers are integral only controllers. The gains of both the controllers are tuned to satisfy a 2 second rise-time step response characteristic.

Open the nlairframe_demo model using the command below and run the simulation. The simulation produces an unoptimized vertical acceleration of the airframe and the initial data for optimization.



Double-click the az Response Scope block to view the unoptimized vertical acceleration az of the

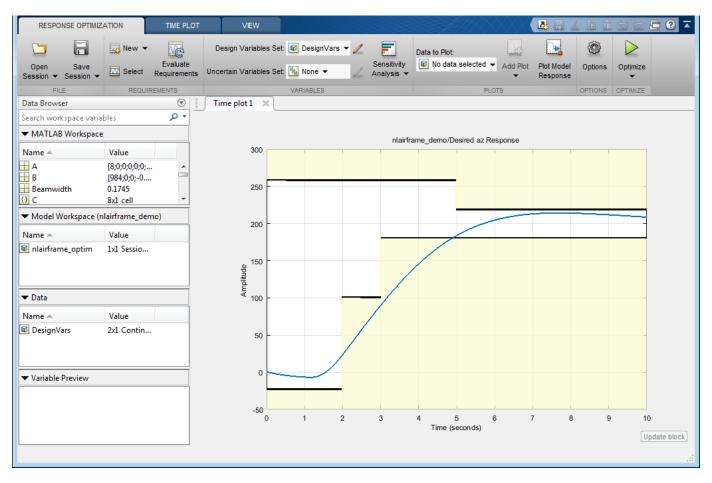
airframe.

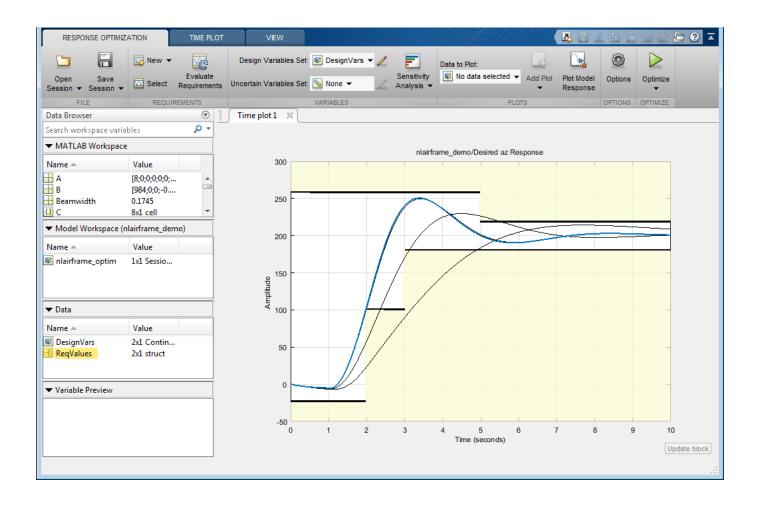
Double-click the q Response Scope block to view the unoptimized rotation rate q of the airframe.

Double-click the Model block to view the details of the subsystem. It includes an Atmosphere model and Aerodynamics and Equations of Motion model.

Double-click the **Desired az Response** block to view constraints on the vertical acceleration of the airframe. These constraints are used to simultaneously tune the gains of the two integral controllers. The first segment of the upper bound constraint represents an overshoot. Note that this is a soft constraint and can be violated.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.





Iteration	F-count	Desired az Response (Upper) (<=0)	Desired az Response (Lower) (>=0)
0	5	-0.0217	-0.7780
1	10	0.0344	-0.4451
2	15	-0.0340	-0.0250
3	20	-0.0298	-3.4917e-04
4	25	-0.0294	3.6239e-05
Optimizati	on started 2	7-Mar-2013 07:56:40	
	-	ed, 27-Mar-2013 07:56:45 lues written to 'DesignVars' in the Design Optim	ization workspace
		Save Iteration	Display Options Optimize

The darker curve shows the final optimized response of the airframe.

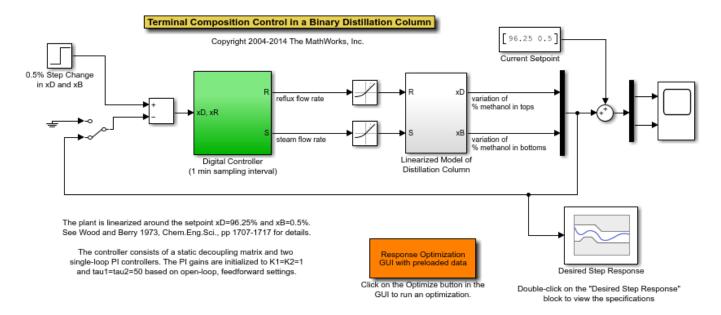
% Close the model.
bdclose('nlairframe_demo')

n

Distillation Controller Tuning

This example shows how to use Simulink® Design Optimization[™] to optimize the multi-loop controller parameters of a distillation column. The Distillation column produces methanol and is represented as a linear model with delays. The digital multi-loop controller consists of a decoupling matrix and two single-loop PI controllers. The parameters of both the single-loop controllers are tuned simultaneously to satisfy a 14 percent overshoot and 13 minute rise-time step response characteristics.

Open the distillation_demo model using the command below and run the simulation. The simulation produces the unoptimized composition of methanol in the column and the initial data for optimization.



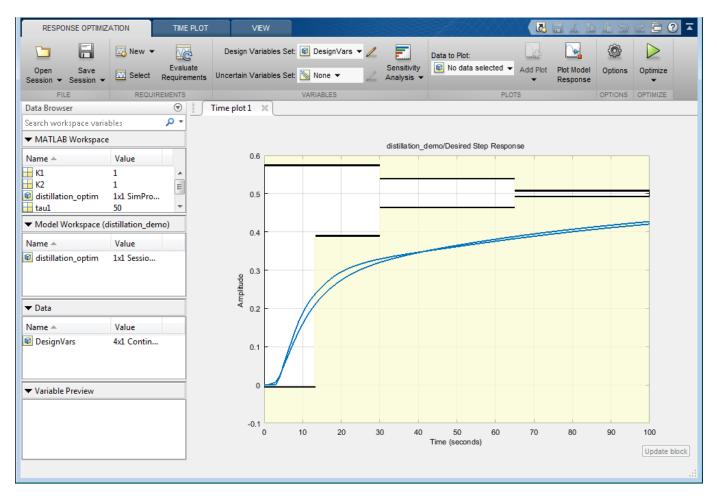
open_system('distillation_demo')

Double-click the Scope block to view the unoptimized methanol composition in the top and bottom of the column.

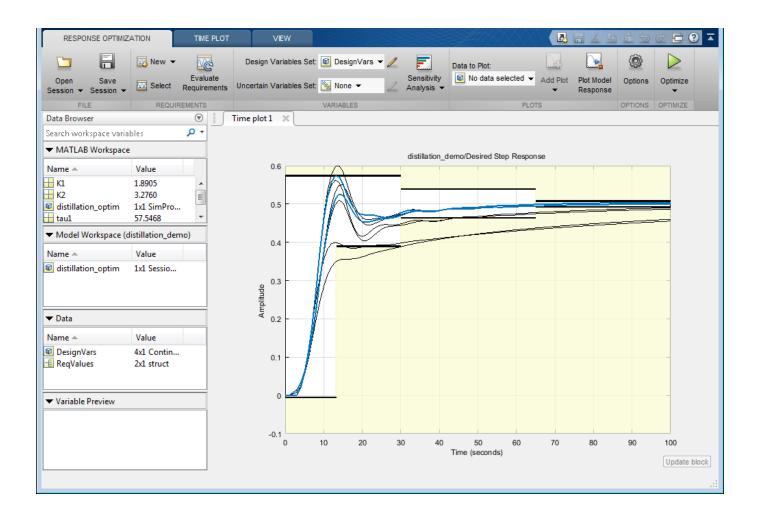
Double-click the Linearized Model of Distillation Column block. Note that this is a subsystem and shows the model for variation of methanol in the top and bottom of the distillation column.

Double-click the **Desired Step Response** block to view constraints on the step response of the distillation column. These constraints are used to simultaneously tune both of the single-loop controller parameters.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



There are two curves in the plot representing the methanol composition in the top and bottom of the column.



teration	F-count	Desired Step Response (Upper) (<=0)	Desired Step Response (Lower) (>=0)
0	9	-0.1583	-0.4593
1	19	-0.0933	-0.1625
2	29	-0.0198	-0.0421
3	39	0.0459	0.0056
4	49	5.4994e-04	0.0049
5	59	-3.6452e-05	0.0049
Optimizati	on converge	7-Mar-2013 07:58:18 ed, 27-Mar-2013 07:58:27 lues written to 'DesignVars' in the Design Optim	nization workspace

The two solid curves show the final optimized methanol composition in the top and bottom of the distillation column.

% Close the model
bdclose('distillation_demo')

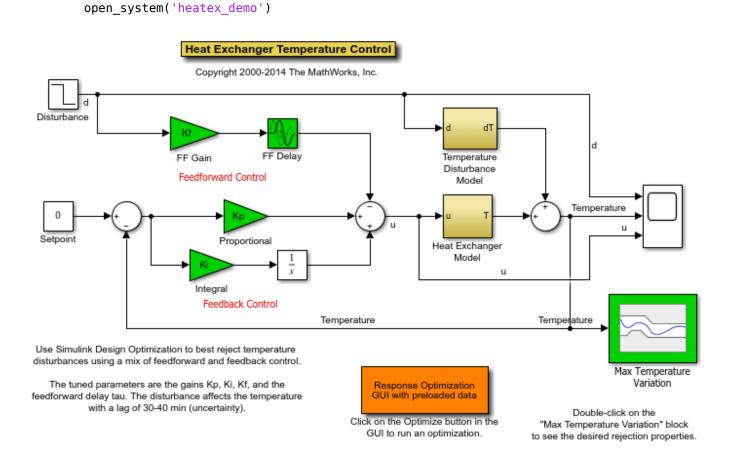
Heat Exchanger Controller Tuning

This example shows how to use Simulink[®] Design Optimization^m to optimize the temperature control of a heat exchanger around a temperature set-point.

The controller regulates the temperature around a setpoint in response to external temperature disturbances. The effect of this disturbance is modeled with an uncertain delay introduced in the Temperature Disturbance Model block.

The temperature controller includes a PI controller along with a feed-forward external temperature measurement. The controller gains are tuned to reduce the effect of external temperature variations by a factor of 50.

Open the heatex_demo model using the command below and run the simulation. The simulation produces an unoptimized temperature variation of the heat exchanger and the initial data for optimization.

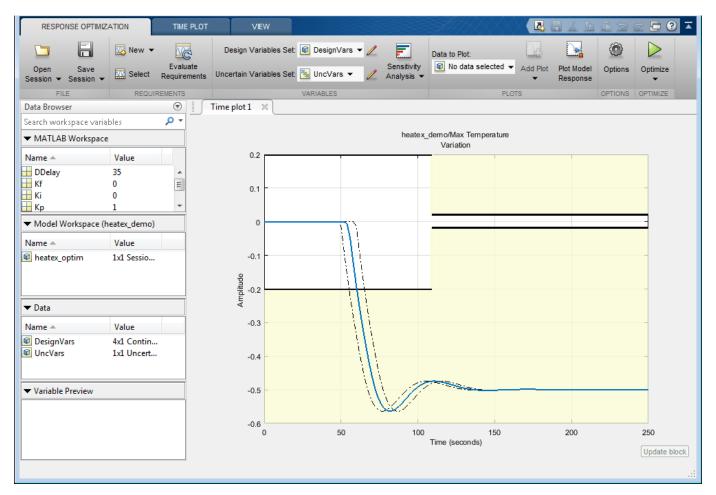


Double-click the Scope block to view the unoptimized temperature response, the disturbance signal and the control signal.

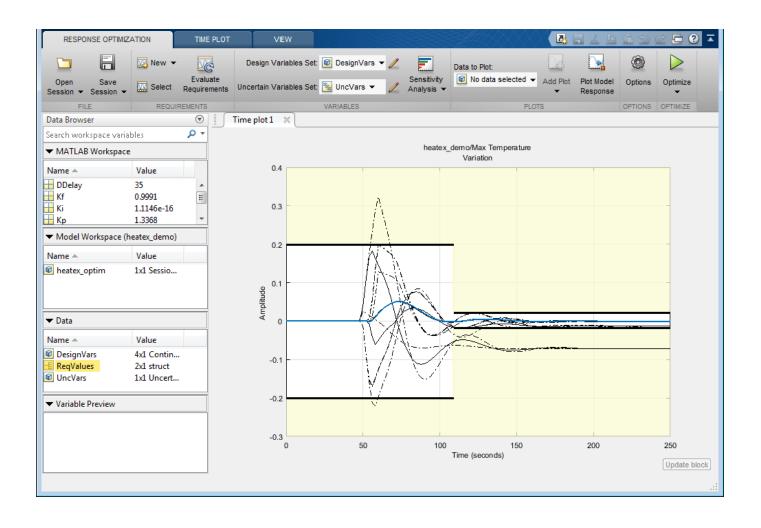
Double-click the Heat Exchanger Model block to view the model details. The exchanger is modeled as a first-order system with delay.

Double-click the Max Temperature Variation block to view constraints on the temperature variation of the heat exchanger. This constraint is used to tune the controller parameters.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



The solid line represents the current response with the mean Disturbance Delay as specified in the constraint block. The dashed lines represent the response with the maximum and minimum Disturbance Delay.



Iteration	F-count	Max Temperature Variation (Upper) (<=0)	Max Temperature Variation (Lower) (>=0)
0	7	-1	-27.3803
1	17	0.6320	-3.5122
2	27	-0.3525	-1.3447
3	37	0.0094	-0.0567
4	47	2.5871e-04	-6.8227e-04
5	57	7.5917e-06	-2.3073e-0
		7-Mar-2013 08:00:45 ed, 27-Mar-2013 08:01:09	
Optimized	variable val	lues written to 'DesignVars' in the Design Optin	nization workspace

The solid curve shows the final optimized temperature variation of the heat exchanger.

% Close the model
bdclose('heatex_demo')

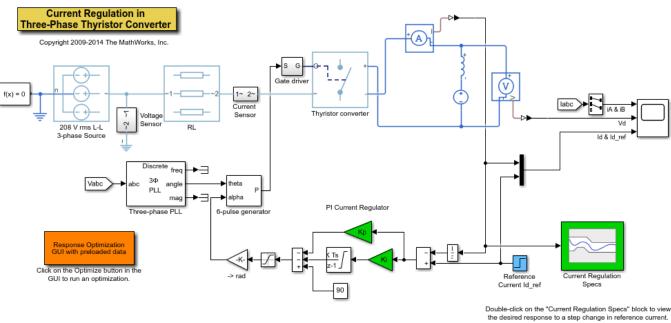
Ē

Power Converter Tuning

This example shows how to use Simulink® Design Optimization[™] to optimize the current controller parameters of a 3-phase thyristor converter. The model uses blocks from Simscape[™] and Simscape[™] Electrical[™].

The 3-phase thyristor is controlled by a pulse-width modulator with variable phase angle computed by a PI controller. The PI Current Regulator gains are tuned to track the reference DC current and limit oscillations.

Open the power demo model using the command below and run the simulation. The simulation produces an unoptimized current variation of the DC motor and the initial data for optimization.

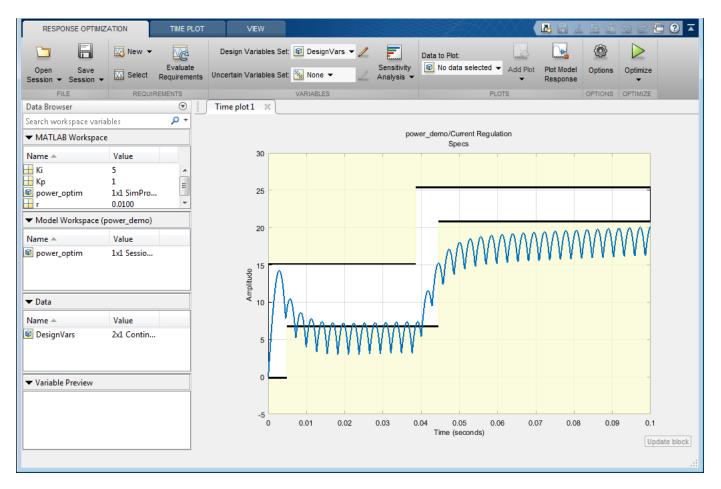


open_system('power_demo')

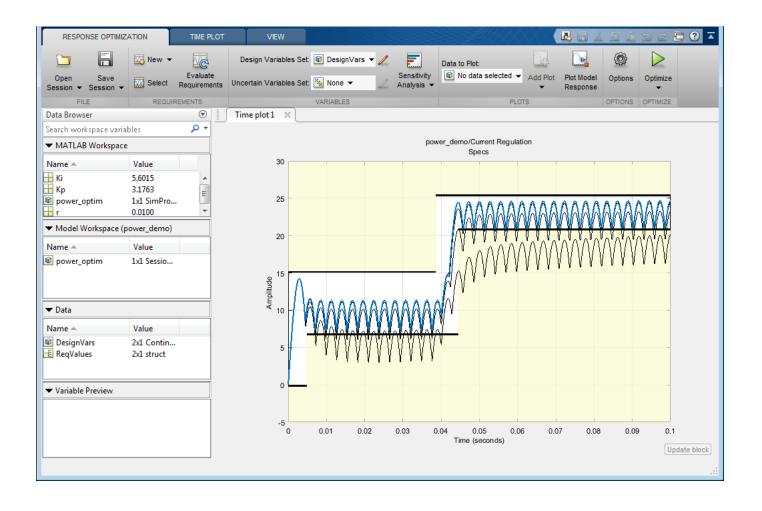
Double-click the Scope block to view the unoptimized current response. Note that two phases of the 3-phase source current and the output voltage of the DC motor are also displayed in this block.

Double-click the Current Regulation Specs block to view constraints on the output current of the DC motor.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB[®]. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



We start the optimization by pressing the **Optimize** button from the **Response Optimizer**. The plot is updated to indicate that the design requirements are now satisfied.



Iteration	F-count	Current Regulation Specs (Upper) (<=0)	Current Regulation Specs (Lower) (>=0)
0	5	-0.0583	-0.5538
1	10	-0.0583	-0.1386
2	15	-0.0387	-0.0289
3	20	-0.0279	-0.0019
4	25	-0.0271	2.1048e-04
		'-Mar-2013 08:03:18 d, 27-Mar-2013 08:04:04	
Optimized	variable val	ues written to 'DesignVars' in the Design Optim	ization workspace

The plot now shows the final optimized current response.

% Close the model. bdclose('power_demo')

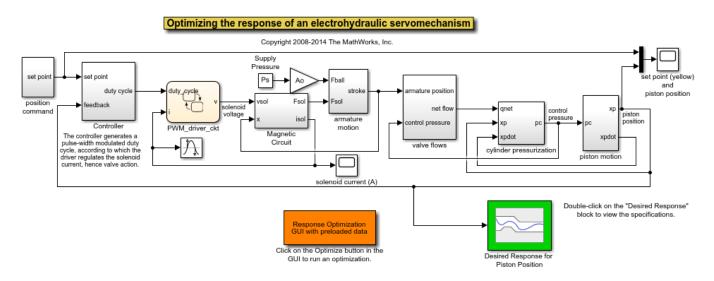
Ē

Servomechanism Tuning

This example shows how to use Simulink® Design Optimization^m to optimize the position controller parameters for a servomotor piston. This model uses blocks from Stateflow®.

The controller sets the duty cycle of a pulse-width modulation circuit for the servomotor. The antiwindup controller Proportional and Integral gains are tuned to optimize the step response characteristics of the servomotor.

Open the servo_demo model using the command below and run the simulation. The simulation produces an unoptimized position of the piston and the initial data for optimization.



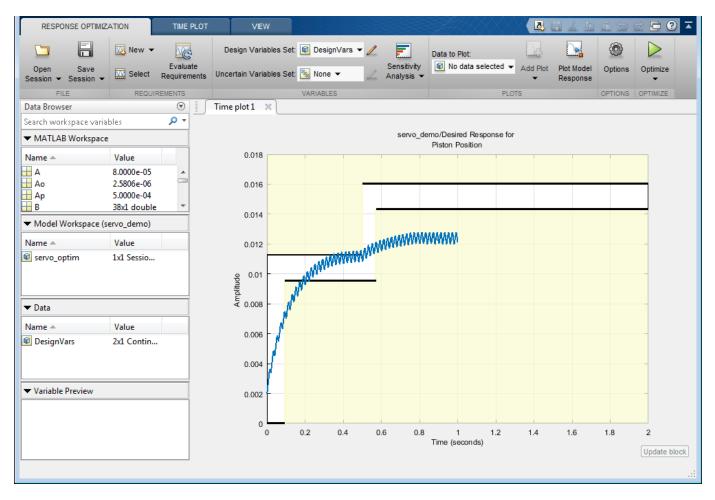
open_system('servo_demo')

There are two Scope blocks in the model. Double-click the set point(yellow) and piston position Scope block to view the set-point and the unoptimized position of the piston.

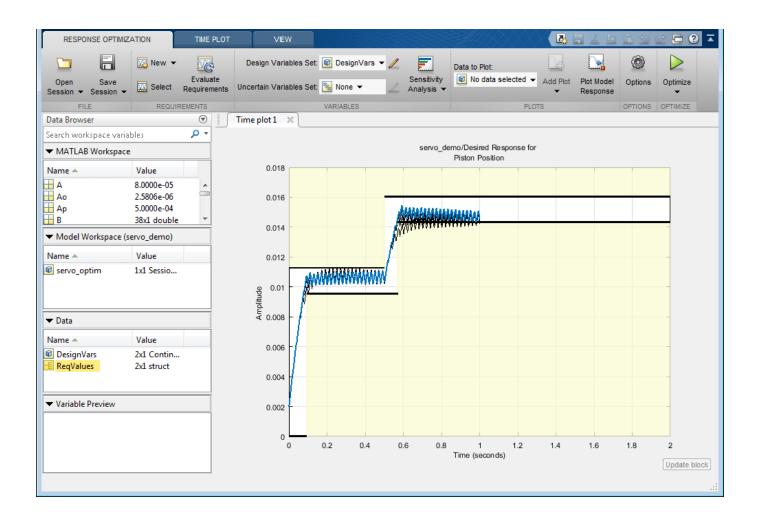
Double-click the **solenoid current(A)** Scope block to view the output solenoid current of Magnetic Circuit.

Double-click the Desired Response for Piston Position block to view constraints on the position of the piston.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



We start the optimization by pressing the **Optimize** button from the **Response Optimizer**. The plots are updated to indicate that the design requirements are now satisfied.



lteration	F-count	Desired Response for Piston Position ((<=0)	Desired Response for Piston Position ((>=0)
0	5	0.0283	-0.2660
1	12	-0.0331	-0.243
2	17	-7.2782e-04	-0.067
3	22	-0.0197	-0.020
4	27	-0.0231	-0.008
5	32	-0.0188	9.0716e-0
6	37	-0.0188	9.0716e-0
Optimizati	on started 2	7-Mar-2013 08:06:30	
	-	ed, 27-Mar-2013 08:08:13 Ilues written to 'DesignVars' in the Design Opti	mization workspace ,

The plot shows the final optimized piston position.

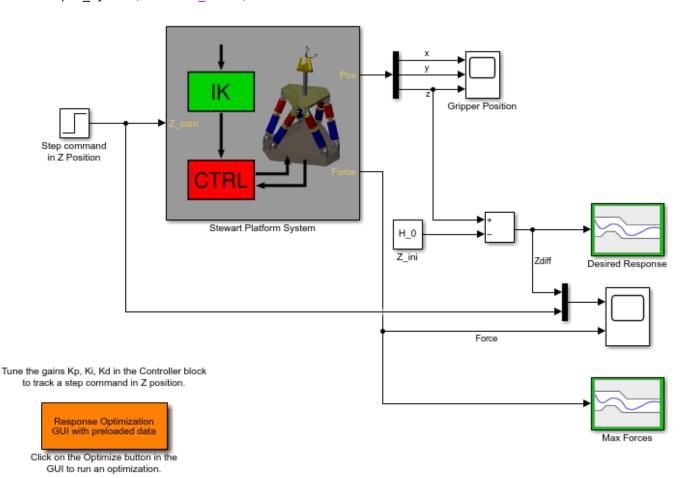
% Close the model. bdclose('servo_demo')

Stewart Platform Controller Tuning

This example shows how to use Simulink® Design Optimization^M to optimize the position controller parameters of a Stewart platform. The Stewart platform is modeled using Simscape^M Multibody^M blocks.

The model includes a vertical PID position controller and the gains are tuned to limit the maximum forces and track position with minimum overshoot and 0.05 second rise time.

Open the stewart_demo model using the command below and run the simulation. The simulation produces an unoptimized vertical position (Z direction) of the Stewart platform.



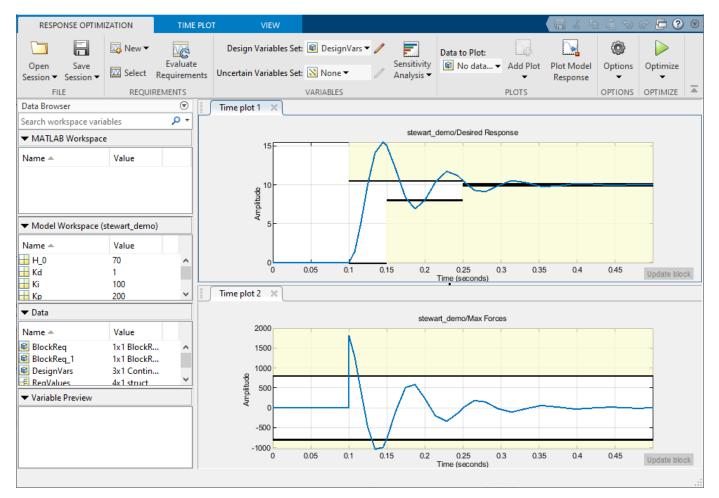
open_system('stewart_demo')

Double-click the Scope block to view the unoptimized position, the step input, and the controller's force actuation signal.

Double-click the down-arrow at the lower left corner of the Stewart Platform System block to view the details of the Stewart Platform. Note that this model uses the Simscape Multibody blockset.

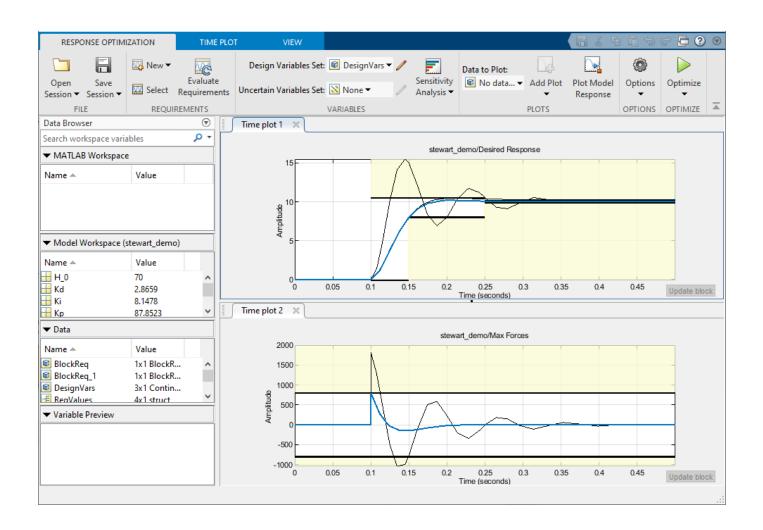
Double-click the Desired Response or the Max Forces blocks to view the constraints on the Stewart platform. The Max Forces block defines the constraints on the actuation signal of the controller.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. You can launch a pre-configured optimization task in the **Response Optimizer** by first opening the model and by double-clicking on the orange block at the bottom of the model. From the **Response Optimizer**, press the **Plot Model Response** button to simulate the model and show how well the initial design satisfies the design requirements.



There are two plots representing the Z Position and Leg Forces of the platform.

We start the optimization by pressing the **Optimize** button from the **Response Optimizer**. The plots are updated to indicate that the design requirements are now satisfied.



lteration	F-count	Desired Respons (<=0)	Desired Respons (>=0)	Max Forces (Upp (<=0)	Max Forces (Low (>=0)	
0	7			1.2765	-0.3000	~
1	14	0.0253	0.0138	-8.0520e-06	0.8172	
2	21	-0.0022	4.8856e-04	2.2846e-09	0.8166	
3	22	-0.0022	4.8856e-04	2.2846e-09	0.8166	
						4
Optimizati	ion started	30-Oct-2019 16:19:04	4			~
optimizati	ion stance	00-0012010 10.10.01				
Optimizati	ion converg	ged, 30-Oct-2019 16:2	20:21			
Optimized	l variable v	alues written to 'Desig	nVars' in the Design O	ptimization workspace	2	¥

The plots are now updated with the optimized position of the Stewart platform and the force actuation signal. $\ensuremath{\mathsf{Stewart}}$

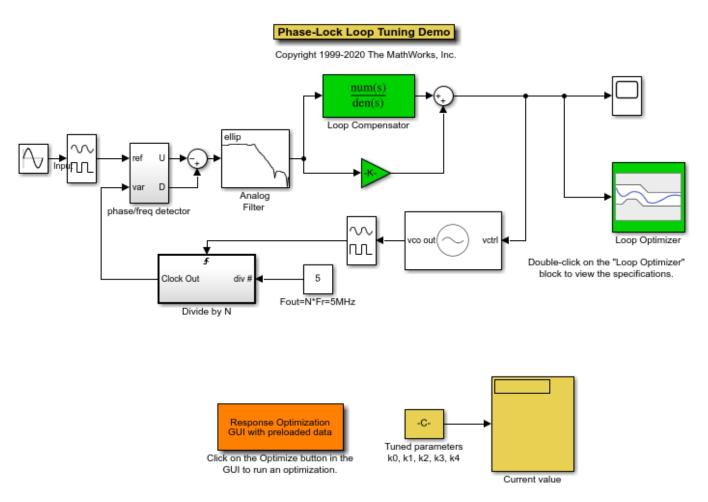
% Close the model.
bdclose('stewart_demo')

Phase Lock Loop Tuning

This example shows how to use Simulink® Design Optimization[™] to tune an all-pass filter of a Phase Lock Loop. The filter includes a second-order low pass filter and a feedthrough gain. The feedthrough gain and the second order filter coefficients are tuned to apply a steady-state input to the Voltage Controlled Oscillator (VCO).

This example requires Mixed-Signal Blockset[™] software.

Open the phaselock_demo model and run the simulation. The simulation produces an unoptimized input to the VCO and the initial data for optimization.



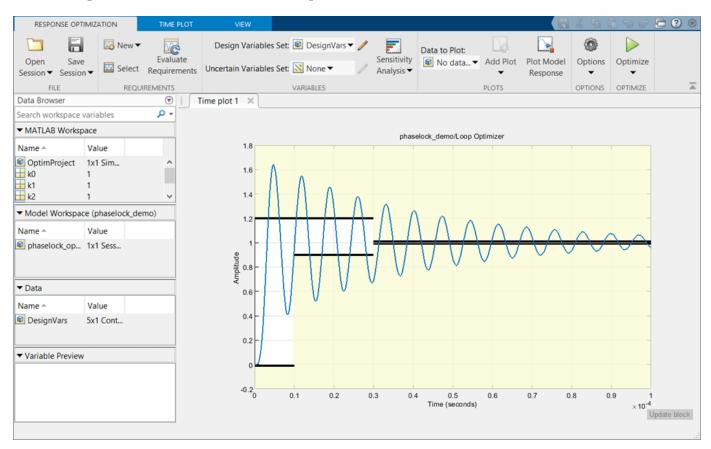
open_system('phaselock_demo')

To view the unoptimized input to the VCO, double-click the Scope block.

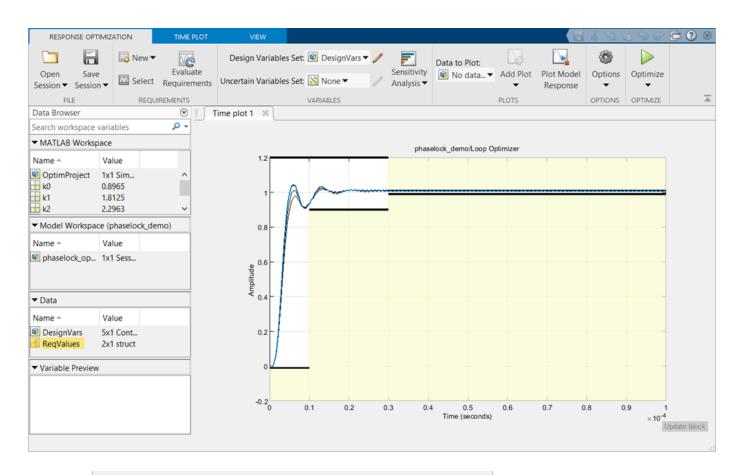
Double-click the Loop Optimizer block to view constraints on the input to the VCO. The constraints represent a step response with 0.1 second rise-time and 20 percent overshoot.

You can launch the **Response Optimizer** using the **Apps** menu in the Simulink toolstrip, or the sdotool command in MATLAB®. To launch a pre-configured optimization task in the **Response Optimizer**, open the model, and double-click the orange block at the bottom of the model. To

simulate the model and check if the initial design satisfies the design requirements, in the **Response Optimizer**, click the **Plot Model Response** button.



To start the optimization, click the **Optimize** button from the **Response Optimizer**. The plots are updated to indicate that the design requirements are now satisfied.



Iteration	F-count	Loop Optimizer (Upper) (<=0)	Loop Optimizer (Lower) (>=0)	
0	8	0.3937	-0.4956	^
1	18	0.0971	-0.0458	
2	27	0.0090	-0.0040	
3	38	0.0126	0.0230	
4	49	0.0044	0.0025	
5	60	0.0037	0.0173	
6	70	7.6888e-04	0.0150	
7	80	1.1017e-04	0.0143	
8	90	2.8262e-06	0.0142	
9	100	3.6833e-09	0.0142	ľ
				~
Optimization s	tarted 31-Dec-20	20 10:38:30		`
Optimized vari	able values writt	ec-2020 16:40:17 en to 'DesignVars' in the Design Optimization n optimized values	1 workspace	~
_			lay Options Optimize	

The solid curve shows the final optimized input to the VCO.

Close the model.

bdclose('phaselock_demo')

Surrogate Optimization in Simulink Design Optimization

This example shows how to use surrogate optimization in Simulink Design Optimization[™] to optimize the design of a hydraulic cylinder.

This example requires Parallel Computing Toolbox™ software.

Surrogate Optimization

Solving optimization problems involves using different values of the design variables and evaluating the objective function multiple times, especially if the objective function is sufficiently nonsmooth such that a derivative-based solver is not suitable. In such cases, you might need to use a derivative-free solver such as patternsearch, but these solvers tend to require running the objective function many more times. Using such a solver can be time consuming if the objective function is computationally expensive to evaluate. One way to overcome this problem is surrogate optimization. This approach creates a surrogate of the expensive objective function. The surrogate can be evaluated quickly and gives results very similar to the original objective function. Surrogate optimization also tries many starting points, which helps find a global optimum, rather than converging on a solution that, while locally optimal, might not be the global optimum.

Hydraulic Cylinder Model

This example shows how to use surrogate optimization to optimize the response of a hydraulic cylinder. Open the model.

open_system('sdoHydraulicCylinder')

The hydraulic cylinder model is based on the Simulink model sldemo hydcyl. The model includes:

- Pump and Cylinder Assembly subsystems. For more information on the subsystems, see "Single Hydraulic Cylinder Simulation".
- A step change applied to the cylinder control valve orifice area that causes the cylinder piston position to change.

Specify Design Variables

Now, specify the design variables to tune: the cylinder cross-sectional area Ac and the piston spring constant $K. \label{eq:Kappa}$

```
DesignVars = sdo.getParameterFromModel('sdoHydraulicCylinder',{'Ac','K'});
DesignVars(1).Value = 1e-3;
DesignVars(1).Minimum = 0.0003;
DesignVars(1).Maximum = 0.0013;
DesignVars(1).Scale = 0.001;
DesignVars(2).Value = 50000;
DesignVars(2).Minimum = 10000;
DesignVars(2).Maximum = 100000;
DesignVars(2).Scale = 1;
```

Specify Design Requirements

Next, specify design requirements to satisfy the following conditions during optimization:

• The pressure stays under 1,750,000 N/m.

• The piston position step response satisfies a rise time of 0.04 seconds and a settling time of 0.05 seconds.

```
Requirements = struct;
Requirements.MaxPressure = sdo.requirements.SignalBound(...
        'BoundMagnitudes', [1750000 1750000], ...
        'BoundTimes', [0 0.1]);
Requirements.PistonResponse = sdo.requirements.StepResponseEnvelope(...
        'FinalValue', 0.04, ...
        'PercentSettling', 1, ...
        'RiseTime', 0.04, ...
        'SettlingTime', 0.05);
```

Signal Logging

Next, specify model signals to log during model simulation. These signals are needed to evaluate the design requirements, that is, to determine whether they are satisfied.

```
Simulator = sdo.SimulationTest('sdoHydraulicCylinder');
```

```
PistonPosition_Info = Simulink.SimulationData.SignalLoggingInfo;
PistonPosition_Info.BlockPath = 'sdoHydraulicCylinder/Cylinder Assembly';
PistonPosition_Info.OutputPortIndex = 2;
PistonPosition_Info.LoggingInfo.LoggingName = 'PistonPosition';
PistonPosition_Info.LoggingInfo.NameMode = 1;
```

```
Pressures_Info = Simulink.SimulationData.SignalLoggingInfo;
Pressures_Info.BlockPath = 'sdoHydraulicCylinder/Cylinder Assembly';
Pressures_Info.LoggingInfo.LoggingName = 'Pressures';
Pressures_Info.LoggingInfo.NameMode = 1;
```

```
Simulator.LoggingInfo.Signals = [...
PistonPosition_Info; ...
Pressures_Info];
```

Create Optimization Objective Function

Create a function that is called at each optimization iteration to evaluate the design requirements. Use an anonymous function with one argument that calls sdoHydraulicCylinder_optFcn.

optimfcn = @(P) sdoHydraulicCylinder_optFcn(P,Simulator,Requirements);

The sdoHydraulicCylinder_optFcn function uses the simulator and requirements objects to evaluate the design. Type edit sdoHydraulicCylinder_optFcn to examine the function in more detail.

Optimization Using Derivative-Based Solver

Now, try solving this optimization problem using a derivative-based solver. Specify optimization options.

```
Options = sdo.OptimizeOptions;
Options.Method = 'fmincon';
Options.UseParallel = true;
Options.OptimizedModel = Simulator;
```

Run the optimization by calling **sdo.optimize** with the objective function handle, parameters to optimize, and options.

[Optimized_DesignVars,Info] = sdo.optimize(optimfcn,DesignVars,Options);

Starting parallel pool (parpool) using the 'local' profile ... Connected to the parallel pool (number of workers: 6). Configuring parallel workers for optimization... Analyzing and transferring files to the workers ...done. Parallel workers configured for optimization.

Optimization started 15-Jul-2021 11:36:33

			max		First-order
Iter F-c	ount	t f(x)	constraint	Step-size	optimality
Θ	5	0.001	0.3033		
1	10	0.00123343	0.3726	0.233	100
2	20	0.00117522	0.3545	0.0582	100
3	28	0.00121802	0.3678	0.0428	100
4	39	0.00117879	0.3556	0.0392	100
5	48	0.00120789	0.3646	0.0291	100
6	60	0.00119077	0.3593	0.0171	100
7	72	0.00119977	0.3621	0.00901	100
8	80	0.00119977	0.3621	0.00089	100
Convorged	+ ~	an infoncible	noint		

Converged to an infeasible point.

fmincon stopped because the size of the current step is less than the value of the step size tolerance but constraints are not satisfied to within the value of the constraint tolerance. Removing data from parallel workers... Data removed from parallel workers.

At the end of optimization iterations, the "max constraint" column is still positive, indicating that the derivative-based solver does not satisfy all of the requirements.

Optimization Options Using Surrogate Solver

Since the derivative-based solver does not satisfy all the requirements, try surrogateopt as a derivative-free solver. Specify optimization options.

```
Options = sdo.OptimizeOptions;
Options.Method = 'surrogateopt';
Options.MethodOptions.MaxFunctionEvaluations = 100;
Options.UseParallel = true;
Options.OptimizedModel = Simulator;
```

Run the optimization by calling **sdo.optimize** with the objective function handle, parameters to optimize, and options.

[Optimized_DesignVars,Info] = sdo.optimize(optimfcn,DesignVars,Options);

Configuring parallel workers for optimization... Analyzing and transferring files to the workers ...done. Parallel workers configured for optimization.

Optimization started 15-Jul-2021 11:37:23

			Current	Current	
F-count	f(x)	max constraint	f(x)	max constraint	Trial ty
1	0.001	0.303264	0.001	0.303264	initial
2	0.001	0.303264	0.0003	0.41283	random

3	0.001	0.303264	0.0008	0.982707	random
4	0.001	0.303264	0.00055	0.573642	random
5 6	0.001	0.303264	0.00105	1.25679	random
	0.001	0.303264	NaN	Inf	random
7	0.001	0.303264	0.001175	2.14476	random
8	0.001	0.303264	NaN	Inf	random
9	0.000925	0.252177	0.000925	0.252177	random
10	0.000925	0.252177	NaN	Inf	random
11	0.000925	0.252177	0.0011125	0.333109	random
12	0.000925	0.252177	NaN	Inf	random
13	0.000925	0.252177	0.0008625	0.724191	random
14	0.000925	0.252177	NaN	Inf	random
15	0.000925	0.252177	0.0009875	1.46276	random
16	0.000925	0.252177	0.0007375	4.26818	random
17	0.000925	0.252177	0.0012375	0.399354	random
18	0.000925	0.252177	0.00076875	0.985955	random
19	0.000925	0.252177	0.00126875	1.39122	random
20	0.000925	0.252177	0.00051875	0.477584	random
21	0.000925	0.252177	0.00101875	1.23392	random
22	0.000925	0.252177	NaN	Inf	random
23	0.00089375	0.226608	0.00089375	0.226608	random
24	0.00089375	0.226608	NaN	Inf	random
25	0.00089375	0.226608	0.00114375	3.56038	random
26	0.00089375	0.226608	NaN	Inf	random
27	0.00089375	0.226608	0.00095625	8.44124	random
28	0.00089375	0.226608	NaN	Inf	random
29	0.00089375	0.226608	0.00120625	0.325598	random
30	0.00058125	0.0293624	0.00058125	0.0293624	random
31	0.00058125	0.0293624	NaN	Inf	adaptiv
32	0.00058125	0.0293624	NaN	Inf	adaptiv
33	0.00058125	0.0293624	NaN	Inf	adaptiv
34	0.00058125	0.0293624	NaN	Inf	adaptiv
35	0.00058125	0.0293624	NaN	Inf	adaptiv
36	0.00058125	0.0293624	NaN	Inf	adaptiv
37	0.00058125	0.0293624	0.000373093	0.20491	adaptiv
38	0.00058125	0.0293624	NaN	Inf	adaptiv
39	0.00058125	0.0293624	NaN	Inf	adaptiv
40	0.00058125	0.0293624	NaN	Inf	adaptiv
41	0.00058125	0.0293624	NaN	Inf	adaptiv
42	0.00058125	0.0293624	0.000405619	0.394347	adaptiv
43	0.00058125	0.0293624	NaN	Inf	adaptiv
44	0.00058125	0.0293624	0.000439948	0.442546	adaptiv
45	0.00058125	0.0293624	0.000654805	0.13009	adaptiv
46	0.00058125	0.0293624	0.000545924	0.0716494	adaptiv
47	0.00058125	0.0293624	0.000507695	0.209479	adaptiv
48	0.00058125	0.0293624	0.000615123	0.368681	adaptiv
49	0.00058125	0.0293624	0.000648997	1.79744	adaptiv
50	0.00058125	0.0290879	0.00058125	0.0290879	adaptiv
			Current	Current	
F-count	f(x)	max constraint	f(x)	max constraint	Trial t
51	0.00058125	0.0290879	0.000581445	0.0291836	adaptiv
52	0.00058125	0.0290879	0.000581152	0.0291773	adaptiv
53	0.00058125	0.0290879	0.00058125	0.0292252	adaptiv
54	0.00058125	0.0290879	0.000581348	0.029273	adaptiv
55	0.00058125	0.0290879	0.00108125	0.821321	random
56	0.00058125	0.0290879	NaN	Inf	random
57	0.00058125	0.0290879	0.00083125	2.51611	random

58	0.00058125	0.0290879	NaN	Inf	random
59	0.00058125	0.0290879	0.00106562	0.768714	random
60	0.00058125	0.0290879	NaN	Inf	random
61	0.00058125	0.0290879	0.000815625	0.725618	random
62	0.00058125	0.0290879	0.000440625	0.067243	random
63	0.00058125	0.0290879	0.000940625	0.901158	random
64	0.00058125	0.0290879	0.000690625	0.475563	random
65	0.00058125	0.0290879	0.00119062	1.14814	random
66	0.00058125	0.0290879	NaN	Inf	random
67	0.00058125	0.0290879	0.000878125	1.83347	random
68	0.00058125	0.0290879	0.000628125	7.48485	random
69	0.00058125	0.0290879	0.00112812	0.359996	random
70	0.00058125	0.0290879	NaN	Inf	random
71	0.00058125	0.0290879	0.00125312	0.367383	random
72	0.00058125	0.0290879	NaN	Inf	random
73	0.00058125	0.0290879	0.00100312	2.73758	random
74	0.00058125	0.0290879	NaN	Inf	random
75	0.000398826	0.01138	0.000398826	0.01138	adaptiv
76	0.000398826	0.01138	0.000398501	0.0121406	adaptiv
77	0.000398826	0.01138	0.000342896	0.0664043	adaptiv
78	0.000398826	0.01138	0.000419212	0.0337089	adaptiv
79	0.000398826	0.01138	NaN	Inf	adaptiv
80	0.000398826	0.01138	NaN	Inf	adaptiv
81	0.000398826	0.01138	NaN	Inf	adaptiv
82	0.000398826	0.01138	NaN	Inf	adaptiv
83	0.000398826	0.01138	NaN	Inf	adaptiv
84	0.000398826	0.01138	NaN	Inf	adaptiv
85	0.000398826	0.01138	0.000379602	0.0552041	adaptiv
86	0.000398826	0.01138	NaN	Inf	adaptiv
87	0.000398826	0.01138	0.000381455	0.0919141	adaptiv
88	0.000398984	0.00300044	0.000398984	0.00300044	adaptiv
89	0.000405043	0.000165085	0.000405043	0.000165085	adaptiv
90	0.000400538	0.000170854	0.000400538	0.000170854	adaptiv
91	0.000400538	0.000170854	0.000399682	0.00578883	adaptiv
92	0.000400538	0.000170854	0.000401935	0.00574052	adaptiv
93	0.000400538	0.000170854	0.000398905	0.00720515	adaptiv
94	0.000397413	-0.00044187	0.000397413	-0.00044187	adaptiv
95	0.000397413	-0.00044187	0.000398976	-0.000446158	adaptiv
96	0.000397413	-0.00044187	0.000400538	0.000265697	adaptiv
	t solution is feasible				evaluati
easing	the function evaluati	on limit may resul	t in an improved so	olution.	

The current solution is feasible. surrogateopt stopped because it exceeded the function evolution evolution the function evaluation limit may result in an improved solution. Removing data from parallel workers... Data removed from parallel workers.

Using surrogateopt, all the design requirements are satisfied, as indicated by a negative value in the "max constraint" column.

Update the model with the optimized parameter values.

sdo.setValueInModel('sdoHydraulicCylinder',Optimized_DesignVars);

In this example, a solver using surrogates successful on an optimization problem where a derivativebased solver is unsuccessful. The surrogateopt solver is a global solver that tries many starting points. By using a surrogate of the model, <code>surrogateopt</code> needs to run the model only a moderate number of times.

See Also

sdo.OptimizeOptions|sdo.optimize

Related Examples

• "Single Hydraulic Cylinder Simulation"

Sensitivity Analysis

- "What is Sensitivity Analysis?" on page 4-2
- "Specify Parameters for Design Exploration" on page 4-4
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Specify Time-Domain Requirements" on page 4-21
- "Specify Parameters Requirements" on page 4-36
- "Specify Frequency-Domain Requirements" on page 4-48
- "Evaluate Design Requirements" on page 4-60
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67
- "Use Sensitivity Analysis to Configure Estimation and Optimization" on page 4-74
- "Interact with Plots in the Sensitivity Analyzer" on page 4-79
- "Validate Sensitivity Analysis" on page 4-96
- "Store Intermediate Data in the App" on page 4-100
- "Specify Steady-State Operating Point for Sensitivity Analysis" on page 4-102
- "Use Parallel Computing for Sensitivity Analysis" on page 4-104
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109
- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112
- "Identify Key Parameters for Estimation (GUI)" on page 4-131
- "Explore Design Reliability Using Parameter Sampling (GUI)" on page 4-145
- "Design Exploration Using Parameter Sampling (Code)" on page 4-157
- "Identify Key Parameters for Estimation (Code)" on page 4-169
- "Generate MATLAB Code for Sensitivity Analysis Statistics to Identify Key Parameters (GUI)" on page 4-179
- "Generate MATLAB Code for Sensitivity Analysis for Design Space Exploration and Evaluation (GUI)" on page 4-183

What is Sensitivity Analysis?

Sensitivity analysis is defined as the study of how uncertainty in the output of a model can be attributed to different sources of uncertainty in the model input[1]. In the context of using Simulink Design Optimization software, sensitivity analysis refers to understanding how the parameters and states (optimization design variables) of a Simulink model influence the optimization cost function. Examples of using sensitivity analysis include:

- Before optimization Determine the influence of the parameters of a Simulink model on the output. Use sensitivity analysis to rank parameters in order of influence, and obtain initial guesses for parameters for estimation or optimization.
- After optimization Test how robust the cost function is to small changes in the values of optimized parameters.

One approach to sensitivity analysis is local sensitivity analysis, which is derivative based (numerical or analytical). Mathematically, the sensitivity of the cost function with respect to certain parameters is equal to the partial derivative of the cost function with respect to those parameters. The term *local* refers to the fact that all derivatives are taken at a single point. For simple cost functions, this approach is efficient. However, this approach can be infeasible for complex models, where formulating the cost function (or the partial derivatives) is nontrivial. For example, models with discontinuities do not always have derivatives.

Local sensitivity analysis is a one-at-a-time (OAT) technique. OAT techniques analyze the effect of one parameter on the cost function at a time, keeping the other parameters fixed. They explore only a small fraction of the design space, especially when there are many parameters. Also, they do not provide insight about how the interactions between parameters influence the cost function.

Another approach to sensitivity analysis is global sensitivity analysis, often implemented using Monte Carlo techniques. This approach uses a representative (global) set of samples to explore the design space. Use Simulink Design Optimization software to perform global sensitivity analysis using the **Sensitivity Analyzer**, or at the command line. The workflow is as follows:

1 Sample the model parameters using experimental design principles. That is, for each parameter, generate multiple values that the parameter can assume. Define the parameter sample space by specifying probability distributions for each parameter. You can also specify parameter correlations.

For information about sampling parameters, see "Generate Parameter Samples for Sensitivity Analysis" on page 4-8.

- 2 Define a cost function by creating a design requirement on the model signals.
- **3** Evaluate the requirement (cost function) at each combination of parameter values using Monte Carlo simulations. You can plot the cost function output for the samples to visually analyze trends.
- 4 (Optional) Formally analyze the relation between the evaluated requirement and the samples. Analysis methods include correlation, partial correlation (requires Statistics and Machine Learning Toolbox[™] software), and standardized regression. You can configure each analysis method to use either raw or ranked data.

For information about the analysis methods, see "Analyze Relation Between Parameters and Design Requirements" on page 4-67.

References

[1] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. *Global Sensitivity Analysis. The Primer*, John Wiley and Sons, 2008.

See Also

sdo.sample|sdo.evaluate|sdo.analyze|sdo.scatterPlot

Related Examples

- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112
- "Design Exploration Using Parameter Sampling (Code)" on page 4-157
- "Identify Key Parameters for Estimation (GUI)" on page 4-131
- "Identify Key Parameters for Estimation (Code)" on page 4-169

More About

- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67
- "Validate Sensitivity Analysis" on page 4-96

Specify Parameters for Design Exploration

This topic shows how to select parameters of a Simulink model for design exploration in the **Sensitivity Analyzer**. After you select the parameters, you generate parameter samples on page 4-8 by varying the parameter values in a specific range, and evaluate your design requirements for each combination of parameter values.

Add Model Parameters as Variables

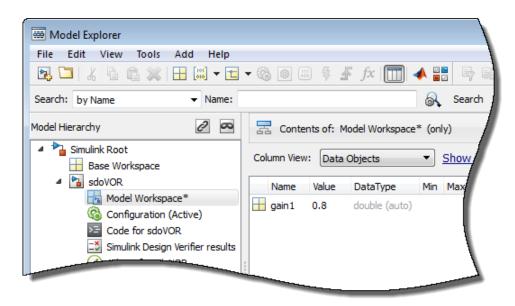
The software can only evaluate variables that are in use by the model. Create variables in the MATLAB or model workspace, and specify your Simulink model or block parameters using these variables. If you have already specified model parameters as variables, Select Parameters for Design Exploration on page 4-6.

🔁 Block	Parameters: Gain	×				
Gain						
Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).						
Main	Signal Attributes Parameter Attributes					
Gain:						
0.8						
Multiplic	ation: Element-wise(K.*u)	•				
0	OK Cancel Help Ap	ply				

In this figure, the **Gain** parameter of a Gain block is specified as a numerical value.

To evaluate design requirements using the **Gain** parameter, specify it as variable gain1:

- 1 Create the variable gain1 in one of the following ways:
 - Add the variables to the model workspace, and specify initial values.



• Write initialization code in the **PreloadFcn** callback of the model. For more information, see "Model Callbacks".

gain1 = 0.8

2 Specify the block parameter as variable gain1 in the Gain block dialog box.

-				
	Main	Signa	al Attributes	Parameter Attributes
	Gain:			
	gain1			
	Multiplic	ation:	Element-wise	se(K.*u) ▼
-				

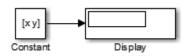
You can now select gain1 for evaluation. See, "Select Parameters for Design Exploration" on page 4-6.

Specify Independent Parameters

You can also specify independent parameters that do not appear explicitly in the model as variables. However, you cannot use this workflow with Simulink fast restart.

Suppose that a model parameter Kint is related to independent parameters x and y such that Kint = x+y. To add x and y instead of Kint:

- Create the independent variables x and y by adding them to the model workspace and specifying initial values.
- The software only allows evaluation of variables that are used by model blocks. To ensure that the software detects x and y for evaluation, add a Constant block to your model, and specify the **Constant value** of the block as [x y]. Connect the block to a Display block.



• Write code in the **InitFcn** callback of the model that defines the relationship between Kint, x, and y. You must first use the get_param function to get the variables x and y from the model workspace before you can use them to define Kint.

```
wks = get_param(gcs, 'ModelWorkspace')
x = evalin(wks, 'x')
y = evalin(wks, 'y')
Kint = x+y;
```

You can now select x and y for design exploration.

Select Parameters for Design Exploration

In the **Sensitivity Analyzer**, in the **Sensitivity Analysis** tab, click **Select Parameters** to open the Select model variables dialog box.

Use this dialog box to select parameters to vary. The table lists the variables that the model uses to set block parameter values. The variables can reside in the model workspace, the base workspace, or a data dictionary.

Select variables by clicking the check box next to each variable. If your model contains many variables, filter the list by typing in the **Filter by variable name** field. The **Used By** column lists all blocks in the model that use the variable. When a variable is used in more than one block, all blocks are listed. To highlight blocks in the model that use the variable, click the block name.

Select	model variab	les		×
Filter	by variable r	name		- م
•	Variable	Current v	Used By	
	freq1	3	engine idle speed/Transfer Fcn	
	freq2	3	engine_idle_speed/TransferFcn1	
	freq3	3	engine idle speed/Transfer Fcn2	
	gain1	100	engine idle speed/Transfer Fcn	=
	gain2	0	engine_idle_speed/TransferFcn1	
	gain3	0	engine_idle_speed/TransferFcn2	-
► Sp	ecify express	ion indexing	if necessary (e.g., a(3) or s.x)	Help

The variables that you select must have a numeric value that uses the data type double. If the value of a variable is not a double number, use these techniques:

• To select a single element or a subset of a matrix or array variable, click **Specify expression** indexing if necessary.

Specify expression indexing if necessary (e.g., a(3) or s.x.)

Enter an expression such as myArray(2), which selects the second element of an array variable myArray.

After you type the expression, press the **Enter** key to add the variable to the list of model variables.

- To use a variable of a numeric data type other than double, convert the variable to a Simulink.Parameter object, which separates a parameter value from its data type. Set the Value property to a default double number, and use the DataType property to control the data type.
- To use the value of a Simulink.Parameter object, specify the Value property. Enter the expression myParamObj.Value.
- To use a numeric field of a structure, enter myStruct.PID.P1. If you store the structure in a Simulink.Parameter object, enter myStruct.Value.PID.P1.
- To use one cell of a cell array, enter myCells{3}.

You cannot use mathematical expressions such as a + b. Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters, see "Add Model Parameters as Variables" on page 4-4.

See Also

Related Examples

- "What is Sensitivity Analysis?" on page 4-2
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8

Generate Parameter Samples for Sensitivity Analysis

This topic shows how to generate parameter samples for sensitivity analysis.

You can perform global sensitivity analysis using Simulink Design Optimization software. Using techniques such as design of experiments (DOE) (also referred to as experimental design), you can choose a parameter set for sensitivity analysis.

You generate parameter samples by varying the value of the Simulink model parameters and states of interest according to a specified probability distribution. These parameters and states are collectively referred to as parameters. Each combination of generated parameter values is referred to as a sample or sample point. A collection of samples is referred to as a design space, sample space, or parameter set.

After generating a parameter set, you define a cost function by creating design requirements on the model signals. You then evaluate the cost function for each sample in the parameter set. Then, you analyze the relation between the parameters and requirement to understand how the parameters influence the cost function.

You can generate two kinds of parameter values: random parameter values on page 4-8 or gridded parameter values on page 4-16.

Generate Random Parameter Values

When generating random parameter values, you specify the following characteristics of the parameter space:

- "Number of Samples" on page 4-8
- "Sampling Method" on page 4-8
- "Probability Distribution" on page 4-10
- "Parameter Correlations" on page 4-11

You can specify the characteristics of the parameter space either in the **Sensitivity Analyzer** or at the command line on page 4-14.

Number of Samples

Choose enough samples to yield useful results. However, each model evaluation has a computational expense and can be time intensive. As the number of parameters increases, the number of samples required to explore the design space generally increases. For correlation or regression analysis, consider using 10Np samples, where Np is the number of parameters.

Sampling Method

Specify the method used to generate the samples. You can choose from the following methods:

• **Random** — Random samples are drawn from the probability distributions specified for the parameters.

If you specify correlation between parameters, the software uses the Iman-Conover algorithm to impose the parameter correlations.

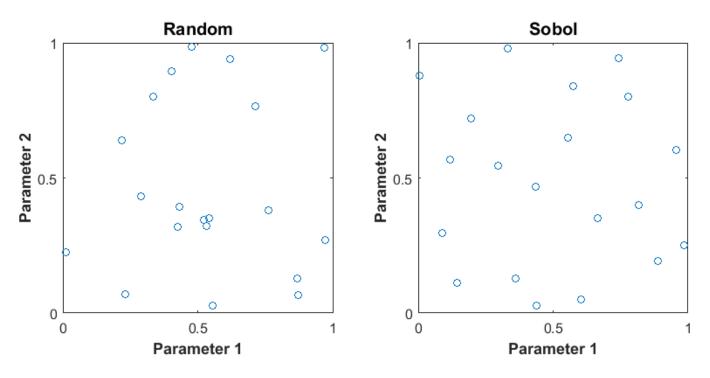
• **Latin hypercube**— Latin hypercube samples are drawn from the probability distributions specified for the parameters. Use this option for a more systematic space-filling approach than

random sampling. The Sobol and Halton quasirandom sampling methods are more space-filling than the Latin hypercube method.

If you specify correlation between parameters, the software uses the Iman-Conover algorithm to impose the parameter correlations.

• **Sobol** — Requires Statistics and Machine Learning Toolbox software. Sobol quasirandom sequences are drawn from the probability distributions specified for the parameters. Use this method for highly systematic space-filling.

The figure shows 20 samples for two parameters. The samples are generated from a uniform distribution, in the interval from 0 to 1. Random sampling can result in large gaps between some samples, and close clustering of other samples. Sobol and Halton quasirandom sampling methods avoid gaps and clustering of samples. If you have many parameters in your parameter set, Sobol sets gives more systematic space filling than Halton quasirandom sets. For more information, see "Generating Quasi-Random Numbers" (Statistics and Machine Learning Toolbox).



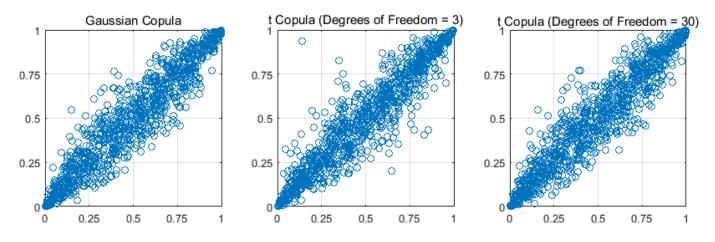
If you specify correlation between parameters, the software uses the Iman-Conover algorithm to impose the parameter correlations.

• Halton — Requires Statistics and Machine Learning Toolbox software. Halton quasirandom sequences are drawn from the probability distributions specified for the parameters. Like the Sobol method, you can use Halton method for highly systematic space-filling. However, Sobol method gives more systematic space filling if you have many parameters in your parameter set. For more information, see "Generating Quasi-Random Numbers" (Statistics and Machine Learning Toolbox).

If you specify correlation between parameters, the software uses the Iman-Conover algorithm to impose the parameter correlations.

• **Copula**— Requires Statistics and Machine Learning Toolbox software. Random samples are drawn from a copula. Use this option to impose correlations between the parameters using copulas.

You can use either a Gaussian copula (default) or a t copula. Use t copulas when the probability of extreme parameter values is not negligible (distribution is heavy-tailed), and specify the degrees of freedom. As you increase the degrees of freedom, the t copula converges to the Gaussian copula, and the probability of extreme parameter values becomes negligible. The following figure shows 1000 samples drawn for two parameters in the interval from 0 to 1 using the Gaussian and t copulas.



In comparison to the Gaussian copula, the t copula has more samples that represent the extreme values of the parameters. As the degrees of freedom are increased, the t copula converges to the Gaussian copula.

Specify the correlation type as either Spearman's rank correlation or Kendall's rank correlation.

Probability Distribution

Specify the probability distribution function and related distribution characteristics for each parameter. Use your knowledge of the system (empirical or theoretical) to choose the probability distributions.

Note Simulink Design Optimization software allows you to specify uniform (default), normal, multinomial, piecewise linear, and triangular distributions. For other distributions, you need Statistics and Machine Learning Toolbox software.

Consider the following characteristics of your parameters when choosing a distribution:

Parameter Characteristics	Applicable Distributions
Extends from -∞ to ∞	• Normal
	Extreme value
	 Generalized extreme value — Single-ended or from -∞ to ∞, depending on distributional parameter values.
	 Logistic — heavy tailed compared to normal distribution.
	• t location-scale — heavy tailed compared to normal distribution.
Bounded at both ends	• Uniform
	• Beta
	Binomial — discrete distribution
	Multinomial— discrete distribution
	Piecewise linear
	• Triangular
Extends from 0 to ∞	Birnbaum-Saunders
	• Burr
	Exponential
	• Gamma
	 Generalized extreme value — Single-ended or from -∞ to ∞, depending on parameter values.
	Inverse Gaussian
	Log-logistic
	• Log-normal
	• Nakagami
	Negative binomial— discrete distribution
	Poisson— discrete distribution
	• Rayleigh
	• Rician
	• Weibull
Custom distribution	Piecewise linear

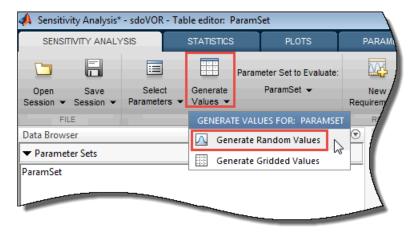
For more information about these distributions, see the "Probability Distributions" (Statistics and Machine Learning Toolbox) category.

Parameter Correlations

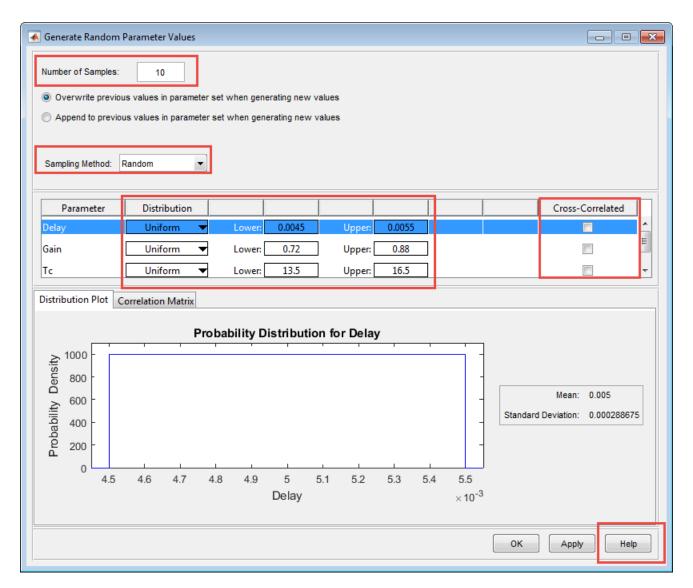
Specify the correlation between parameters. The algorithm used to impose the parameter correlations depends on the sampling method. For more information, see "Sampling Method" on page 4-8.

Generate Random Parameter Values in the App

In the **Sensitivity Analyzer**, after you have selected the parameters in the parameter set on page 4-4, click **Generate Values** and, select **Generate Random Values**.



In the Generate Random Parameter Values dialog box, specify the number of samples, probability distributions, parameter bounds and correlations, and sampling method. For information about how to specify the fields in the dialog box, click **Help**.



The generated parameter set and the corresponding parameter set table are displayed in the app. The number of rows in the parameter set table correspond to the number of samples you specified.

SENSITIVITY ANALYSIS STATISTICS PLOTS		ISTICS PLO	TS PA	ARAMETER SET VIEW 🔣 🛃			1 h h b c 🗗 🕄		
D Parameter		Set to Evaluate:		W	Store Intermedia	te Data			
Open Save Session ▼ Session ▼		Generate ParamSet Values 👻	•	New Requirement 👻	Select for Evaluation	Options		Evaluate Model	Optimize
FILE		PARAMETER SET		REQUIREN	ENTS	OPTIONS		EVALUATE	OPTIMIZE
Data Browser 💿	Paramet	er Set: ParamSet 🛛 🛛							
 Parameter Sets 									
ParamSet	Delay		Gain		Tc		Тр		
		0.0045		0.8232		14.2554			0.0108
		0.0051		0.7802		14.3713			0.0108
		0.0054		0.7505		15.3513			0.0106
		0.0052		0.7885		14.2958			0.0095
✓ Requirements		0.0047		0.7971		15.9731			0.0102
▼ Requirements		0.0049		0.7393		16.4480			0.0090
		0.0050		0.8143		15.6907			0.0099
		0.0055		0.7562		14.5316			0.0096
		0.0047		0.7815		15.2522			0.0093
		0.0054		0.8133		13.8233			0.0094

After generating the parameter values, plot them to check if generated parameter values match the intended specifications. This is relevant if you generate a small number of random samples for each parameter set. For more information, see "Inspect the Generated Parameter Set" on page 4-96.

For an example, see "Identify Key Parameters for Estimation (GUI)" on page 4-131.

This functionality is not supported in Simulink Online.

Generate Random Parameter Values at the Command Line

At the command line, use sdo.ParameterSpace to define the parameter space. This object specifies the probability distributions and correlations for the parameters. Use this object as an input to sdo.sample for generating parameter values from the specified parameter space.

To generate the random parameter values:

- Specify the number of samples as the second input argument of sdo.sample.
- Specify the method used to generate these samples using the Method property of an sdo.SampleOptions object. Use this object as an input to sdo.sample to specify the sampling options.

If the method chosen is 'sobol' or 'halton', specify the MethodOptions property of sdo.SampleOptions.

If the method chosen is 'copula', specify the choice of copula using the MethodOptions property of sdo.SampleOptions. Also specify the RankCorrelation property of the sdo.ParameterSpace object.

- Specify the probability distribution of a parameter using the ParameterDistributions property of an sdo.ParameterSpace object.
- Specify correlation between parameters, using the RankCorrelation property of the sdo.ParameterSpace.

After generating the parameter values, plot the generated values to check if they match the desired specifications. This is relevant if you generate a small number of random samples for each parameter set. For more information, see "Inspect the Generated Parameter Set" on page 4-96.

For an example, see "Identify Key Parameters for Estimation (Code)" on page 4-169.

Generate Custom Parameter Values at the Command Line

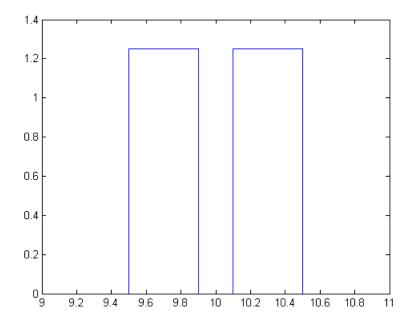
This example shows how to generate random parameter values with a custom distribution when performing sensitivity analysis at the command line. Generate a 1000 samples of a model parameter, R, in the 5% range of its nominal value, 10. R is a resistor. Resistors of 1% tolerance are removed by the manufacturer, so do not generate R values in the 1% range of its nominal value.

1 Construct a param. Continuous object.

R = param.Continuous('R',10);

2 Create a customized probability distribution, pdR, to configure the parameter space.

```
x = [0.95 0.99 1.01 1.05]*R.Value;
F = [0 0.5 0.5 1];
pdR = makedist('PiecewiseLinear','x',x,'Fx',F);
x = linspace(0.9*R.Value,1.1*R.Value,1e3);
plot(x,pdf(pdR,x));
```



makedist specifies a piecewise linear distribution for the resistor value, with a "hole" in the 1% range.

3 Specify pdR as the probability distribution for the R parameter in an sdo.ParameterSpace object.

```
ps = sdo.ParameterSpace(R,pdR);
4 Generate 1000 samples.
```

```
Ns = 1000;
```

```
x = sdo.sample(ps,Ns);
```

5 (Optional) Use sdo.scatterPlot to visualize the samples and validate the sample space.

Generate Gridded Parameter Values

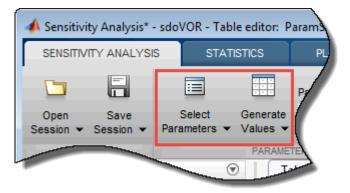
You can generate gridded parameter values in the **Sensitivity Analyzer** or at the command line.

Generate Gridded Parameter Values in the Sensitivity Analyzer

You can generate gridded parameters in the app after you have created a parameter set.

1 Create a parameter set.

In the Sensitivity Analyzer, in the Sensitivity Analysis tab, click Select Parameters.



In the Select Parameters dialog box, select all the parameters you want to include in your parameter set, and click **OK**.

	Variable Delay	Current va 0.005	Used By sdoVOR/Delay
~	Gain		sdovOly Delay sdoVOR/Gain
•	Tc	15	sdoVOR/Periphery and Prolongation
	Тр	0.01	sdoVOR/Oculomotor Plant
Spe	cify expressio	on indexing if i	necessary (e.g., a(3) or s.x)

A ParamSet variable is created in the **Parameter Sets** area of the app. The current value for each parameter in the parameter set is displayed in a table.

Data Browser	SENSITIVITY ANALYSIS	STATISTICS	PLOTS PARAMETER SET VIEW 🔀 🗐 🔏 🗎 🖄 😪 🔂 🖓					
Session × Parameters × Values × Requirement × Evaluation Model File PARAMETER SET REQUIREMENTS OPTIONS EVALUATE Data Browser Image: Comparison of the set of the			Parameter Set to E	valuate:	W	Store Intermediate Data		
Data Browser Parameter Set: ParamSet Parameter Set: ParamSet	•		then benefitien grophenia					Optimize
▼ Parameter Sets	FILE	PARA	METER SET	REQUIREM	ENTS	OPTIONS	EVALUATE	OPTIMIZE
			Parameter Set: ParamSet 🗶					
ParamSet Gain Tc 0.8000	amSet		Gain Tc 0.8000 15					

2 Specify the starting parameter values and gridding method for grid generation.

In the Sensitivity Analysis tab, click **Generate Values**, and select **Generate Gridded Values** from the drop-down menu.

Gridding Method: All Combinations (1 samples) Min-Max Combinations (1 samples) Pair-Wise Combinations (1 samples) Starting Values:	
Parameter Values	
Gain 0.8	
Tc 15	
🗮 Overwrite 🔛 Append 🕐 Help	

In the Generate Gridded Parameter Values dialog box, specify the gridding method as **All Combinations**. The app generates all possible combinations of the values specified in **Values**.

Specify the starting parameter values in **Values**. The values you enter here determine the parameter space. To see the other ways to specify starting parameter values and gridding methods, click **Help**.

Starting Values:	
Parameter	Values
Gain	[0.7,0.8,0.9]
Tc	[13:1:17]

3 Generate the parameters.

Click **Overwrite**. The parameter set table updates with the generated gridded parameter values.

Parameter Set: ParamS	et ×	
Gain		Tc
	0.7000	1
	0.8000	1
	0.9000	1
	0.7000	14
	0.8000	14
	0.9000	14
	0.7000	1
	0.8000	1
	0.9000	1
	0.7000	1
	0.8000	1
	0.9000	1
	0.7000	1
	0.8000	1
	0.9000	1

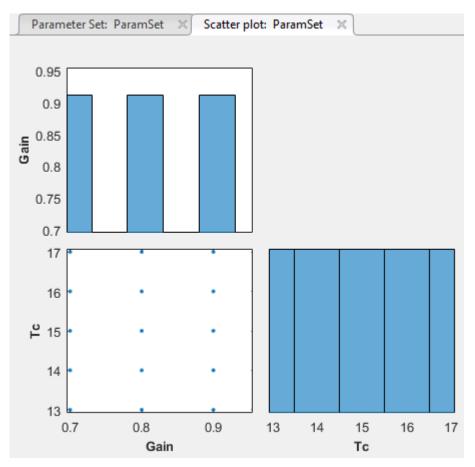
You can also append the generated values to previously generated random or gridded parameter values in the parameter set table. To do so, click **Append**.

4 (Optional) Plot the generated values.

In the **Parameter Sets** area of the app, select **ParamSet**. In the **Plots** tab of the app, select **Scatter Plot** from the plots gallery.

📣 Sensitivity Analysis* -	sdoVOR -	Parameter Set:	ParamSet
SENSITIVITY ANALYSI	S I	STATISTICS	PLOTS
ParamSet	atter plot	able editor	
Data Browser		Paramete	er Set: Param
✓ Parameter Sets			
ParamSet		Gain	

Plots are generated with histograms of the distribution of the parameter values shown on the diagonals. The off-diagonal plots display the scatter plots between pairs of parameters. To learn more about the plots, see "Interact with Plots in the Sensitivity Analyzer" on page 4-79.



This functionality is not supported in Simulink Online.

Generate Gridded Parameter Values at the Command Line

This example shows how to create a table of gridded parameter values at the command line.

Generate a grid of samples for two model parameters, A and B. Vary A between [2,3,4] and B between [20,30,40].

1 Construct a param. Continuous object.

```
A = param.Continuous('A',1);
```

- B = param.Continuous('B',10);
- **2** Specify the parameter values for grid generation.

```
Avals = [2 3 4];
Bvals = [20 30 40];
```

3 Create a table of gridded parameter values. Specify one column for each parameter, and one row for each sample. The column names must be the same as the parameter names.

```
[Agrid,Bgrid] = meshgrid(Avals,Bvals);
x = table(Agrid(:),Bgrid(:),'VariableNames',{'A','B'});
X =
    А
         В
    _
    2
         20
    2
         30
    2
         40
    3
         20
    3
         30
    3
         40
    4
         20
    4
         30
    4
         40
```

See Also

sdo.sample|sdo.SampleOptions

Related Examples

- "Inspect the Generated Parameter Set" on page 4-96
- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112
- "Design Exploration Using Parameter Sampling (Code)" on page 4-157
- "Explore Design Reliability Using Parameter Sampling (GUI)" on page 4-145

Specify Time-Domain Requirements

In the **Sensitivity Analyzer**, you can specify the following time-domain requirements:

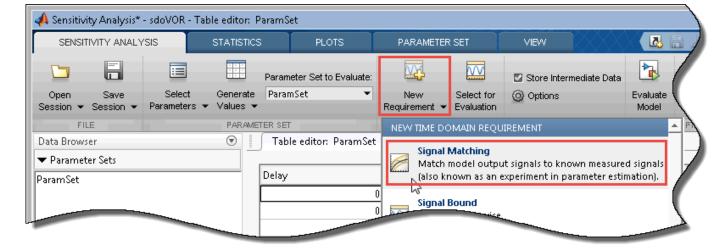
- Signal Matching "Match Model Outputs to Measured Signals" on page 4-21
- Signal Bound "Specify Piecewise-Linear Lower and Upper Bounds" on page 4-24
- Signal Property "Specify Signal Property Requirements" on page 4-26
- Step Response Envelope "Specify Step Response Characteristics" on page 4-27
- Signal Tracking "Track Reference Signals" on page 4-29
- Ellipse Region Constraint "Impose Elliptic Bound on Phase Plane Trajectory of Two Signals" on page 4-31
- Custom Requirement "Specify Custom Requirements" on page 4-33

After you specify the constraints, you can see if the requirements are satisfied by evaluating the design requirements. For more information, see "Evaluate Design Requirements" on page 4-60.

Match Model Outputs to Measured Signals

You can specify a signal matching requirement to match model outputs to measured signals. This requirement is also known as an experiment in the **Parameter Estimator**. To specify a signal matching requirement:

1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Signal Matching.



A new signal matching requirement appears in the **Requirements** area of the app. An Edit Signal Matching dialog box opens where you specify this requirement. You specify model output and input signals, and assign measured data to them. You can also specify initial state values.

📣 Sensitivity Analysis* - sdoVOR - 1	able editor: ParamS	et		
SENSITIVITY ANALYSIS	STATISTICS	PLOTS	PARAMETER SET	VIEW
		eter Set to Evaluate:		Store Intermediat
Open Save Select Session ▼ Session ▼ Parameters	Generate Paran ▼ Values ▼	nSet 💌	New Select for Requirement	Options
FILE	PARAMETER SET		REQUIREMENTS	OPTIONS
Data Browser	💿 🛛 Edit Sign	al Matching: Signal	Matching	×
ParamSet ▼ Requirements SignalMatching	sdoVC <1x1 S S S S S S S S S S S S	R/Oculomotor Plant:1 Signal, 1 points> elect Measured Outpu	✔	
▼ Results	Optiona There a		s for this requirement. states defined for this require ulate [] Plot 🖌 OK	ement.

- 2 In the **Outputs** panel of the Edit Signal Matching dialog box, select output signals, and import output data. You can select more than one output signal, but you can have only one data set for a signal. If you have multiple data sets, create multiple requirements.
 - **a** Select the model output signals that you want to add the requirement to.

By default, the root-level model output ports and logged signals are already listed in the **Outputs** panel. To remove existing outputs from the signal matching requirement, click the corresponding **X**. To add other output signals, click **Select Measured Output Signals**.

The Select Outputs dialog box opens to display the root-level model Outport blocks and logged signals.

Select Outp	uts	×
Click signals	in the simulink model to add them to the output table.	
-	Outputs	
V	sdoVOR/Oculomotor Plant:1 (EyeMotion)	
	V OK 🔀 Cancel	(?) Help
		() Help

- **b** In the Simulink model window, click the signal to add. The Select Outputs dialog box updates to show the new signals. To add these signals as outputs, select the corresponding check boxes. Click **OK**.
- **c** In the Edit Signal Matching dialog box **Outputs** panel, for each output, import measured output data in one of the following ways:

Outputs Specify measured output signals for this requirement.
sdoVOR/Oculomotor Plant:1 (EyeMotion)
<1x1 Signal, 1 points> 👻 🕁 🗴
Select Measured Output Signals

- Import signal data from spreadsheets or text files Click $\stackrel{\clubsuit}{\Longrightarrow}$, and import the data from the file.
- Import data from the MATLAB workspace Suppose that the signal data and time data are in the EyeData and time column vectors in the workspace, respectively. Specify the output data as [time, EyeData].

sdoVOR/Oculomotor Plant:1 (EyeMotion)	
[time,EyeData]	🗕 🗗 🛧

If your data is stored in a time-series object, t, specify the output data as [t.time,t.outputdata].

After importing data, to view or edit the data, click $\stackrel{1}{\Longrightarrow}$.

3 (Optional) Select input signals and import input data in the **Inputs** panel.

You can have more than one input signal, but you can have only one data set for a signal. If you have multiple data sets, create multiple requirements. By default, the root-level model input ports are already listed in the **Inputs** panel. Remove or add inputs, and import input data, in the same way as described for the output data.

4 (Optional) Specify initial states values in the **Initial States** panel.

By default, the initial conditions specified in the model are used for evaluating the requirement. To specify initial conditions other than the defaults, click **Select Initial States**. In the Select Model States dialog box, select the states to specify, and click **OK**.

Select m	odel states			x
Filter b	/ state name			+ م
•	State		Curre	ent value
	sdoVOR/Oculomotor Plant			0
	sdoVOR/Periphery and Prolongation			0
	🛫 ок	₿¢	ancel	Help

The selected states appear in the Edit Signal Matching dialog box **Initial States** panel. Specify the initial states.

Initial States
Optionally define initial states for this requirement.
sdoVOR/Oculomotor Plant
▶ 0.5]
🚟 Select Initial States

5 Close the Edit Signal Matching dialog box.

The signal matching requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **6** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

Note You can preprocess the data using the preprocess tools in the **Experiment Plot** tab. For more information, see "Preprocess Data" on page 1-13.

Specify Piecewise-Linear Lower and Upper Bounds

To specify upper and lower bounds on a signal:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Signal Bound**.

A Create Requirement dialog box opens where you specify upper or lower bounds on a signal. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

- 2 Select the requirement type from **Type**.
- **3** Specify the edge start and end times and corresponding amplitude in the **Time (s)** and **Amplitude** columns.

4 Click to specify additional bound edges.

Select a row and click to delete a bound edge.

5 In the **Select Signals to Bound** area, select a logged signal to apply the requirement to.

If you have already selected signals, as described in "Specify Signals to Log" on page 3-10, they appear in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **i** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the Create Requirement dialog box.

6 Select the check-box corresponding to the signal, and close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- 7 (Optional) Plot the requirement.
 - **a** In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60.

Alternatively, you can add a Check Custom Bounds block to your model to specify piecewise-linear bounds.

Specify Signal Property Requirements

To specify signal property requirements:

- 1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Signal Property**. A Create Requirement dialog box opens where you specify signal property requirements. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.
- 2 In the **Specify Property** area, specify a signal property requirement using the **Property** and **Type** lists and the **Bound** box.

Property	Description	Time weighting available
Signal minimum	Minimum of the signal	No
Signal maximum	Maximum of the signal	No
Signal final value	Last signal value	No
Signal mean	Average of signal value	Yes
Signal median	Middle value of signal	Yes
Signal variance	Variance of signal	Yes
Signal interquartile range	Difference between the 75th and 25th percentiles of signal values	No
Signal sum	$\sum_{i=t_0}^{t_N} S(i), \text{ where } S(t_0), \dots, S(t_N) \text{ is the signal to constrain.}$	Yes
Signal sum square	$\sum_{i=t_0}^{t_N} S(i)^2$	Yes
Signal sum absolute	$\sum_{i=t_0}^{t_N} S(i) $	Yes

Property List

For signal properties where the **Time-Weighted** option is available, you can select it to weight the property computation by the time intervals between samples.

Custom Signal Property

You can add a custom signal property to the **Property** list by editing the function sdo.requirements.signalPropertyFcns.

- a At the MATLAB command prompt, enter edit sdo.requirements.signalPropertyFcns.
- **b** Add your signal property function to the FcnData cell array.

Your signal property function must be on the path.

3 In the **Select Signals to Bound** area, select the logged signal to which you want to apply the requirement.

The signal selected must have numeric type data (either floating-point or integer). Also, if the property selected is Signal median, Signal variance, or Signal interquartile range, then the signal data must be floating-point (either double or single).

If you have already selected a signal, as described in "Specify Signals to Log" on page 3-10, the signal appears in the list. Select the corresponding check box for that signal.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig]		
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click \Rightarrow to add it to the signal set.
- **d** In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the Create Requirement dialog box.

4 Select the check-box corresponding to the signal, and close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, a positive requirement value indicates that the requirement has been violated.

Specify Step Response Characteristics

To apply a step response requirement to a signal in your model, specify the step response characteristics as follows:

1 Select a step response requirement from the **Sensitivity Analyzer**.

In the **New Requirement** drop-down list of the app, in the **New Time Domain Requirement** section, select **Step Response Envelope**.

A Create Requirement dialog box opens where you specify the step response requirements on a signal. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

- **2** Specify the step response characteristics:
 - Initial value Input level before the step occurs
 - **Step time** Time at which the step takes place
 - **Final value** Input level after the step occurs
 - **Rise time** The time taken for the response signal to reach a specified percentage of the step range. The step range is the difference between the final and initial values.
 - % **Rise** The percentage of the step range used with **Rise time** to define the overall rise time characteristics.
 - **Settling time** Time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
 - % **Settling** The percentage of the final value that defines the settling range of settling time characteristic specified in **Settling time**.
 - **% Overshoot** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
 - **% Undershoot** The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **3** Specify the signal to be bound.

To apply this requirement to a model signal, in the **Select Signals to Bound** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check-box.

If you have not selected a signal to log:

^a Click . The Create Signal Set dialog box opens where you specify the logged signal.

b In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the Create Requirement dialog box.

4 Select the check-box corresponding to the signal, and close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60.

Alternatively, you can use the Check Step Response Characteristics block to specify step response bounds for a signal.

Track Reference Signals

Use reference tracking to force a model signal to match a desired signal. To track a reference signal:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Signal Tracking**.

A Create Requirement dialog box opens where you specify the reference signal to track. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

2 Define the reference signal by entering vectors, or variables from the workspace, in the **Time vector** and **Amplitude** fields.

Click **Update reference signal data** to use the new amplitude and time vector as the reference signal.

- **3** Specify how the optimization solver minimizes the error between the reference and model signals using the **Tracking Method** list:
 - SSE Reduces the sum of squared errors
 - SAE Reduces the sum of absolute errors
- 4 In the **Specify Signal to Track Reference Signal** area, select a logged signal to apply the requirement to.

If you already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, they appear in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			X
Signal set: Sig			
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal	1
		OK Cancel	Help .::

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the Create Requirement dialog box.

5 Select the check-box corresponding to the signal, and close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **6** (Optional) Plot the requirement.
 - **a** In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

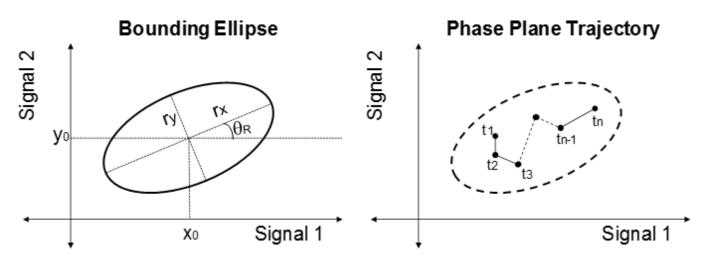
The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Against Reference block to specify a reference signal to track.

Impose Elliptic Bound on Phase Plane Trajectory of Two Signals

You can impose an elliptic bound on the phase plane trajectory of two signals in your Simulink model. The phase plane trajectory is a plot of the two signals against each other. You specify the radii, center, and rotation of the bounding ellipse. You also specify whether you require the trajectory of the two signals to lie inside or outside the ellipse.

The following image shows the bounding ellipse and an example of the phase plane trajectory of two signals.



The X-Y plane is the phase plane defined by the two signals. r_x and r_y are the radii of the bounding ellipse along the x and y axes, and θ_R is the rotation of the ellipse about the center. The ellipse center is at (x_0, y_0) . In the image, the phase plane trajectory of the signals lies within the bounding ellipse for all time points t_1 to t_n .

To specify the elliptical bound requirement:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Ellipse Region Constraint**.

📣 Sensitivi	ty Analysis* -	sdoVOR									
SENSIT	IVITY ANALY	SIS	STATISTICS	PLOTS		/IEW			🛛 🖪 🖁	1 <u>/</u> §	i d d 🗗 🕄
Open	Save	Select Parameters V	Generate	er Set to Evaluate: ameter Set Defined	d 🔻	New D	Select for	Store Intermediate Data Options	Evaluate	Optimize	
Session -	Session -	Farameters •	Values •	 	_	Requirement -	IENTS	OPTIONS	Model	OPTIMIZE	Ā

In the Create Requirement dialog box, specify the signals and elliptical bound. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Requirement		×							
Ellipse Region Constraint									
Constrain a pair of signals to an elli	ptical region in sigr	al phase plane.							
Name: PhasePlaneEllipse									
 Specify ellipse bound 									
Signal	Semi-axis length	Center							
Select Click to select signal.	1	0							
Select Click to select signal.	0.5	0							
Select Click to select signal.	0.0								
	Angle (rad): 0								
Type: Signal <= bound 👻									
and the second se									

1 Specify the two signals that you want to impose the requirement on. The signals define the X-Y plane of the bounding ellipse. To specify the signals, click the corresponding **Select** buttons.

▼ Specify ellipse bound		
Signal	Semi-axis length Center	
Select/Controller/Derivative with LPF	1 rx 0)	Ко
Select sdoShipSteering/Controller/Speed	0.5 ry 0 y	Yo
	Angle (rad): 0	

When you click **Select**, the Create Signal Set dialog box opens.

Create Signal Set			Х
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		1
		OK Cancel	Help

In the Simulink model window, click the signal to which you want to add the requirement. The Create Signal Set dialog box updates with the name of the block and the port number where the

selected signal is located. Select the signal, and click 🔛 to add it to the signal set.

Once you have specified the logged signal in the Create Signal Set dialog box, the signal appears in the Create Requirement dialog box.

- 2 Specify the radii of the bounding ellipse as real positive finite values in **Semi-axis length**. You specify r_x and r_y that are the x-axis and y-axis radii before any rotation about the ellipse center.
- **3** Specify the location of the center of the bounding ellipse in **Center**. You specify x_0 and y_0 , the x and y coordinates of the center, as real finite values.
- 4 Specify the angle of rotation of the ellipse about its center as a real finite scalar in **Angle (rad)**.
- **5** Specify the bound **Type** as one of the following:
 - '<=' Ellipse is an upper bound. The phase plane trajectory of the two signals should lie inside or on the ellipse.
 - '>=' Ellipse is a lower bound. The phase plane trajectory of the two signals should lie outside or on the ellipse.
- **1** Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

- **2** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60.

Specify Custom Requirements

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Custom Requirement**.

A Create Requirement dialog box opens where you specify the reference signal to track. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

- 2 Specify the requirement type in the **Type** drop-down menu.
- **3** Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click

to rev

to review or edit the function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

4 (Optional) To prevent the solver from considering specific parameter combinations, select **Error if constraint is violated**. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next combination of parameters in the parameter set.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current combination of parameter values. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

5 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- **d** In **Signal set** field, enter a name for the selected signal set.

 $\operatorname{Click} \mathbf{OK}.$ A new variable, with the specified name, appears in the Create Requirement dialog box.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

6 Select the check-box corresponding to the signal or system, and close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, a positive requirement value indicates that the requirement has been violated.

See Also

Related Examples

- "Specify Parameters Requirements" on page 4-36
- "Specify Frequency-Domain Requirements" on page 4-48
- "Evaluate Design Requirements" on page 4-60
- "Identify Key Parameters for Estimation (GUI)" on page 4-131
- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112

Specify Parameters Requirements

In the **Sensitivity Analyzer**, you can specify the following constraints on Simulink model parameters that are specified as variables:

- Monotonic Variable "Impose Monotonic Constraint Requirement on Variable" on page 4-36
- Smoothness Constraint "Impose Upper Bound on Gradient Magnitude of Variable" on page 4-38
- Function Matching "Specify Linear or Quadratic Function Matching Constraint" on page 4-41
- Vector Property "Specify Requirement on a Vector Property" on page 4-43
- Relational Constraint "Impose Relational Constraint Between Two Variables" on page 4-45

For information about how to specify a model parameter as a variable, see "Add Model Parameters as Variables" on page 4-4.

Impose Monotonic Constraint Requirement on Variable

You can impose a monotonic constraint requirement on a variable in your Simulink model. For example, constrain a variable to be monotonically increasing. The variable can be a vector, matrix, or multidimensional array that is a parameter in your model, such as the breakpoints of a lookup table.

To specify the requirement:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Monotonic Variable**.

📣 Sensitivi	ty Analysis* ·	- sdoVOR									
SENSI	TIVITY ANALY	'SIS	STATISTICS	PLOTS	·	VIEW		<u>III III III III III III III III III II</u>		1 % 1	i 5 c 🗗 🖓
				Parameter Set to Evaluate:		M ,	M	Store Intermediate Data			
Open Session 👻	Save Session 💌	Select Parameters	Generate Values 	No Parameter Set Define	ed ▼	New 😡 Requirement 👻	Select for Evaluation	Options	Evaluate Model	Optimize *	
_					_	REQUIREM	ENTS	OPTIONS	EVALUATE	OPTIMIZE	Ā

In the Create Requirement dialog box, specify the requirement. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Require	ment						
	Monotonic Variable						
	Specify a requirement that a vector, matrix or array be monotonic.						
Name: Mono	otonicVariable						
▼ Specify Monotonic Constraint							
Variable	Type or Select						

2 Specify the name of the variable in **Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single.

You can type the name of a nonscalar variable, or select the variable from the drop-down list. The list is prepopulated with all the nonscalar variables in your model. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1,:) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the app, see "Add Model Parameters as Variables" on page 4-4.

1 Specify the monotonicity for each dimension of the variable.

After you select the variable, the dialog updates to show **Dimension 1** to **Dimension n**, corresponding to the n dimensions of the variable. For example, for a 2-dimensional variable K of size 3-by-5, the dialog updates as shown.

Create Requirement			×							
	Smoothness Constraint									
Sp	Specify a smoothness requirement on a vector, matrix or array.									
Name: SmoothnessRe	q									
▼ Specify Smoothnes	ss Constraint									
Gradient maximum m	agnitude: 1									
Dependent Variable:	K -		Show in Model							
Independent Variable		d								
Dimension 1:	Type or Select 🔹	<3 element vector, or scalar>	Show in Model							
Dimension 2:	Type or Select 🔹	<5 element vector, or scalar>	Show in Model							

Specify the monotonicity for the first dimension in **Dimension 1** and for the n^{th} -dimension in **Dimension n** as one of the following options:

- Strictly increasing Each element of the variable is greater than the previous element in that dimension.
- Increasing Each element of the variable is greater than or equal to the previous element in that dimension.
- **Decreasing** Each element of the variable is less than or equal to the previous element in that dimension.
- Strictly decreasing Each element of the variable is less than the previous element in that dimension.
- Not constrained No constraint exists between the elements of the variable in that dimension.

1 Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, the app returns the evaluated requirement value corresponding to each dimension of the variable. A positive requirement value indicates that the requirement has been violated.

Impose Upper Bound on Gradient Magnitude of Variable

You can impose an upper bound on the gradient magnitude of a variable in your Simulink model. The variable can be a vector, matrix, or multidimensional array that is a parameter in your model, such as the data of a lookup table. For example, consider a car engine controller whose gain changes under different operating conditions determined by the car speed. You can use a gradient bound constraint to limit the rate at which the controller gain changes per unit change in vehicle speed.

For an N-dimensional variable F that is a function of independent variables $x_1, ..., x_N$, the gradient magnitude is defined as:

$$|\nabla F| = \sqrt{\left(\frac{\partial F}{\partial x_1}\right)^2 + \left(\frac{\partial F}{\partial x_2}\right)^2 + \dots + \left(\frac{\partial F}{\partial x_N}\right)^2}$$

To compute the gradient magnitude, the software computes the partial derivative in each dimension by computing the difference between successive F data in that dimension and dividing by the spacing between the data in that dimension. You specify F and the spacing between the data. The software checks whether the gradient magnitude of the variable data is less than or equal to a specified bound. If the gradient magnitude of the data is greater than the required bound, the variable data is not smooth.

To specify the requirement:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Smoothness Constraint**.



In the Create Requirement dialog box, specify the requirement. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Requirement	X
Smoothness Constraint	
Specify a smoothness requirement on a vector, matrix	or array.
Name: SmoothnessReq	
▼ Specify Smoothness Constraint	
Gradient maximum magnitude: 1	
Dependent Variable: Type or Select 🔹	Show in Model

- 2 Specify the gradient magnitude bound as a nonnegative finite real scalar in **Gradient maximum magnitude**.
- **3** Specify the variable F that you want to impose the requirement on in **Dependent Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single. The variable must be a parameter in your model or a constant that you enter.

You can type the name of a nonscalar variable or a constant, or select the variable from the dropdown list. The list is prepopulated with all the nonscalar variables in your model. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1,:) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the app, see "Add Model Parameters as Variables" on page 4-4.

1 Specify the spacing between points of **Dependent Variable** data in each dimension in **Independent Variable**.

After you select the **Dependent Variable**, the dialog updates to show **Dimension 1** to **Dimension n**, corresponding to the n dimensions of the dependent variable. For example, for a 1-dimensional variable K, the dialog updates as shown.

reate Requirement	>						
Smoothness Constraint							
Specify a smoothness requirement on a vector, matrix or arra	ay.						
Name: SmoothnessReq							
▼ Specify Smoothness Constraint							
Gradient maximum magnitude: 1							
Dependent Variable: 🔣 👻	Show in Model						
Independent Variable							
Dimension 1: Type or Select <4 element vector, or scalar>	Show in Model						

The first dimension specifies the spacing going down the dependent variable data rows, and the second specifies spacing across the columns. The *N*th dimension specifies the spacing along the *N*th dimension of dependent variable data. You can specify the independent variables in each dimension as scalars or vectors.

- Scalars Specify the spacing between dependent variable data F in the corresponding dimension as a nonzero scalar. For example, suppose that **Dependent Variable** is two-dimensional, and the spacing between data in the first dimension is 5 and in the second dimension is 2. In the **Independent Variable** section, specify **Dimension 1** as 5 and **Dimension 2** as 2.
- Vectors Specify the coordinates of F data in the corresponding dimension as real, numeric, monotonic vectors. The software uses the coordinates to compute the spacing between the dependent variable data points in the corresponding dimension. The length of the vector must match the length of F in the corresponding dimension. You do not have to specify coordinates with uniform spacing. For example, suppose that F is two-dimensional, and the length of the data in the first and second dimension is 3 and 5, respectively. The coordinates of the data in the first dimension are [1 2 3]. In the second dimension, the spacing is not uniform and the coordinates of the data are [1 2 10 20 30]. In the Independent Variable section, specify Dimension 1 as [1 2 3] and Dimension 2 as [1 2 10 20 30].

You can also specify the independent variables by typing the name of a variable, or selecting a variable from the drop-down list. The list is prepopulated with all the variables in your model that have the appropriate size. To choose a subset of an array or matrix variable V, type an expression. For example, specify as V(1, :) to use the first row of the variable. To use a numeric field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

1 Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, the app returns the evaluated requirement value. A positive requirement value indicates that the requirement has been violated.

Specify Linear or Quadratic Function Matching Constraint

In the app, you can constrain a variable's values to match a linear or quadratic function. The variable can be a vector, matrix, or a multidimensional array that is a parameter in your model, such as the data of a lookup table in your model. To specify the requirement:

1 In Sensitivity Analyzer, from the New Requirement drop-down list, select Function Matching.

SENSI	TIVITY ANALY	'SIS	STATISTICS	PLOTS	VI	EW	***	<u>III</u>	🛛 🛃 🖁	1 % 1	i 9 ¢ 🗗
	=			Parameter Set to Evaluate:			W	Store Intermediate Data			
Open ession 💌	Save Session 💌	Select Parameters 💌	Generate	No Parameter Set Define		New 😡 Requirement 🛩	Select for	Options	Evaluate Model	Optimize	

In the Create Requirement dialog box, specify the requirement. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Requirement					×			
	Function Matching							
SI	pecify that values	must ma	atch a desi	ignated functi	on.			
Name: FunctionMatch	ning							
▼ Specify Function Matching Constraint								
Functional Relation:	Linear		\sim					
Dependent Variable:	К	~			Show in Model			
Independent Variable								
Dimension 1:	ьрК	~	<4 eleme	ent vector>	Show in Model			
Center and scale i	ndependent varia	bles						
 Center and Scale S 	ettings							
Use automatic cer	nters and scales							
O Use custom cente	rs and scales							
Independent Variable	c	enter		Scale				
	Dimension 1:			12	2.5820			
	L							
_			-					
				_				

2 Specify the function to be matched. To do so, set **Functional Relation** to one of the following values:

• Linear — Data from variable V are fit to a linear function. For example, for a two-dimensional variable with independent variables, X_1 and X_2 , the linear function has the form:

 $V = a_0 + a_1 X_1 + a_2 X_2$

The software calculates the fit coefficients a_0 , a_1 , and a_2 and then calculates the sum of squares of the error between the data and the linear function.

• Quadratic with no cross-terms — Data are fit to a quadratic function with no cross-terms. For a two-dimensional variable, the pure quadratic function has the form:

 $V = a_0 + a_1 X_1 + a_2 X_1^2 + a_3 X_2 + a_4 X_2^2$

• Quadratic with all cross-terms — Variable data are fit to a quadratic function that includes cross-terms. For a two-dimensional variable, the quadratic function has the form:

$$V = a_0 + a_1 X_1 + a_2 X_1^2 + a_3 X_2 + a_4 X_2^2 + a_5 X_1 X_2$$

If the variable is one-dimensional, there are no cross-terms and so the computation is the same as when **Functional relation** is Quadratic with no cross-terms.

3 Specify the variable V to which you want to apply the requirement in **Dependent Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single that is a parameter in your model.

Type the name of a non scalar variable, or select a variable from the drop-down list. The list is prepopulated with all the nonscalar variables in your model. To see where the selected variable is used on your model, click **Show in Model**. To choose a subset of an array or matrix variable A, type an expression. For example, specify A(1, :) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as design variables in the app, see "Add Model Parameters as Variables" on page 4-4.

4 Specify the independent variable vectors used for computing the function in **Independent Variable**. The independent variables are specified as real, numeric, monotonic vectors.

The number of independent variables must equal the number of dimensions of the dependent variable V. For example, you specify two independent variables when V is a matrix, and use three independent variables when V is three-dimensional. The first independent variable vector specifies coordinates going down the rows of V, and the second independent variable vector specifies coordinates going across the columns of V. The n^{th} independent variable vector specifies coordinates along the n^{th} dimension of V. The number of elements in each independent variable vector specifies were nust match the size of V in the corresponding dimension. The independent variable vectors must be monotonically increasing or decreasing.

You can also specify the independent variables by typing the name of a variable, or selecting a variable from the drop-down list. The list is prepopulated with all the variables in your model that have the appropriate size. To choose a subset of an array or matrix variable A, type an expression. For example, specify A(1,:) to use the first row of the variable. To use a numeric field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b. To use an equally spaced vector, select $[1 \ 2 \ \dots N]$ from the drop-down menu.

5 Specify whether you want to center and scale the independent variables. When you select the Center and scale independent variables option, the independent variable vectors you specify are divided by a scale value after subtracting a center value. Centering can improve numerical conditioning when one or more independent variable vectors have a mean that differs from 0 by several orders of magnitude. Scaling can improve numerical conditioning when independent variable vectors of magnitude.

To specify the center and scale values for each independent variable, expand the **Center and Scale Settings** section, and select one of the following:

- Use automatic centers and scales The center and scale values are the mean and standard deviation for each independent variable. Using the mean and standard deviation values to center and scale the independent variables is the default option.
- Use custom centers and scales Specify the Center and Scale values for each independent variable. The independent variable vectors are divided by the corresponding Scale value after subtracting the value you specify in Center.
- **6** Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, the app returns the evaluated requirement value. The app computes an error signal that is the difference between the dependent variable data and the specified function of the independent variables. The app returns the sum of squares of this error as the evaluated requirement value. A positive value indicates that the requirement has been violated, and 0 value indicates that the requirement is satisfied. The closer the value is to 0, the better is the match between the function and dependent variable data.

Specify Requirement on a Vector Property

In the app, you can specify a requirement on a vector property, such as the mean value of the vector. The vector must be a parameter in your model. To specify the requirement:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Vector Property**.

🗚 Sensitivit	ty Analysis* ·	- sdoVOR									
SENSIT	IVITY ANALY	'SIS	STATISTICS	PLOTS		VIEW				- <u>1</u> <u>1</u> <u>1</u>	i 9 c 6 (
	-			Parameter Set to Evaluate:		M	M	Store Intermediate Data			
Open Session 👻	Save Session 💌	Select Parameters 👻	Generate Values 💌	No Parameter Set Defined	d 🔻	New b Requirement ❤	Select for Evaluation	Options	Evaluate Model	Optimize ▼	
						REQUIREN	IENTS	OPTIONS	EVALUATE	OPTIMIZE	

In the Create Requirement dialog box, specify the requirement. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Requi	irement ×							
	Vector Property							
	Specify a requirement on a vector property such as the mean value.							
Name: Ver	torProperty							
 Specify 	Vector Property							
Property:	Vector mean 👻							
Type:	Constrain property to be <= the bound 💌							
Bound:	0							
Variable:	Type or Select							

- **1** Specify the vector property in **Property**. For a vector *V* with *N* elements, you can specify one of the following properties:
 - Vector mean mean(V)
 - Vector median median(V)
 - Vector variance variance(V)
 - Vector inter-quartile range Difference between the 75th and 25th percentiles of the vector values.

Vector sum
$$-\sum_{i=1}^{N} V(i)$$

Vector sum of squares $-\sum_{i=1}^{N} V(i)^2$

• Vector sum of absolute values
$$-\sum_{i=1}^{N} |V(i)|$$

- Vector minimum min(V)
- Vector maximum max(V)
- 2 Specify the type of requirement you want to impose on the vector property in Type. You can set an upper or lower bound on the vector property, or require the property to equal a particular value. You can also choose to maximize or minimize the vector property. For example, to maximize the mean value of your vector, specify **Property** as Vector mean and **Type** as Maximize the property.
- 3 Specify the value of the bound imposed on the vector property in **Bound**. Specify the bound as a finite real scalar value. For example, if for a vector variable V you require mean(V) = 5, specify **Property** as Vector mean, **Type** as Constrain property to be == the bound, and **Bound** as 5.
- 1 Specify the name of the variable in **Variable**. The variable must be a vector, matrix, or multidimensional array of data type double or single.

You can type the name of a nonscalar variable, or select the variable from the drop-down list. The list is prepopulated with all the nonscalar variables in your model. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1,:) to use the first row of the variable. To use a numeric nonscalar field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the app, see "Add Model Parameters as Variables" on page 4-4.

2 Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, the app returns the evaluated requirement value. A positive requirement value indicates that the requirement has been violated.

Impose Relational Constraint Between Two Variables

You can impose a relational constraint requirement on a pair of variables in your Simulink model. For example, require that variable *a* is always greater than variable *b*. To specify the requirement:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Relational Constraint**.

		VIEW			< 🛃 🖁		i 9 c ē
Para	ameter Set to Evaluate:		<u>vv</u>	C Store Intermediate Data			
	Parameter Set Defined		Select for Evaluation	Options	Evaluate Model	Optimize	
e Select	Belect Generate No	Parameter Set to Evaluate: Select Generate No Parameter Set Defined	e Select Generate No Parameter Set to Evaluate: n ← Parameters ← Values ← Requirement ← Requiremen	Parameter Set to Evaluate:	Parameter Set to Evaluate: Select Generate No Parameter Set Defined Parameters Values Parameters Values New Requirement Select for Select for Sel	Parameter Set to Evaluate: Image: Construction of the set of the	Parameter Set to Evaluate: Select Generate No Parameter Set Defined New Requirement Set Defined New Requirement Values Nodel

In the Create Requirement dialog box, specify the requirement. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

Create Requirement		×					
Relational Constraint							
Specify a requirement that two variables obey a relational constraint.							
Name: RelationalReq							
▼ Specify Relational Constraint							
Variable	Relationship	Variable					
Type or Select 🔹	< •	Type or Select 🛛 👻					
Show in Model		Show in Model					

2 Specify the name of the two variables in **Variable**. The variables can be vectors or arrays but must be the same size.

Type the name of two variables, or select the variables from the drop-down lists. The lists are prepopulated with all the variables in your model. To see where a selected variable is used on your model, click **Show in Model**. To choose a subset of an array or matrix variable V, type an expression. For example, specify **Variable** as V(1, :) to use the first row of the variable. To use a numeric field x of a structure S, type S.x. You cannot use mathematical expressions such as a + b.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain k could be defined in the MATLAB workspace as k = a + b, where a and b are not defined in the model but k is used. To add these independent parameters as variables in the app, see "Add Model Parameters as Variables" on page 4-4.

- **1** Specify the relation between the elements of the two variables as one of the following in **Relationship**:
 - '<' Each data element in the first variable is less than the corresponding element in the second variable.
 - '<=' Each data element in the first variable is less than or equal to the corresponding element in the second variable.
 - '>' Each data element in the first variable is greater than the corresponding element in the second variable.
 - '>=' Each data element in the first variable is greater than or equal to the corresponding element in the second variable.
 - '==' Each data element in the first variable is equal to the corresponding element in the second variable.
 - '~=' Each data element in the first variable is not equal to the corresponding element in the second variable.
- **1** Close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, the app returns the evaluated requirement value. The interpretation of the evaluated requirement value depends on the requirement **Type**.

Туре	Evaluated Requirement Value					
	Requirement is Satisfied	Requirement is Violated				
'>' or '<'	Negative number	Positive number, or 0 if the elements are equal				
'>=' or '<='	Negative number, or 0 if the elements are equal	Positive number				
'=='	0	Non-zero number				
'~='	0	1				

See Also

Related Examples

- "Specify Parameters for Design Exploration" on page 4-4
- "Specify Time-Domain Requirements" on page 4-21
- "Specify Frequency-Domain Requirements" on page 4-48

Specify Frequency-Domain Requirements

This topic shows how to specify frequency-domain requirements in the **Sensitivity Analyzer**. To specify frequency-domain requirements, you require the Simulink Control Design toolbox.

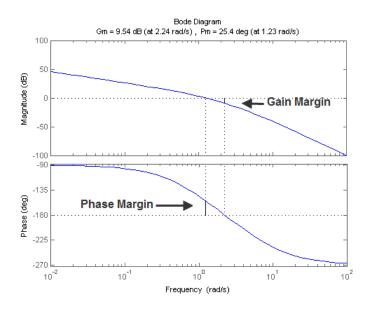
Specify Lower Bounds on Gain and Phase Margin

To specify lower bounds on the gain and phase margin of a linear system:

1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Gain and Phase Margin. A Create Requirement dialog box opens where you specify upper or lower bounds on the gain and phase margin. A new requirement with the name specified in Name appears in the Requirements area of the app.

Create Requirement	×
Gain and Phase Margin	
Specify bounds on the gain and phase margins of a linear system.	
Name: GainPhaseMarqin1	
▼ Specify Margin Bounds	
🔽 Gain margin > 10 dB	
✓ Phase margin > 60 deg	
▼ Select Systems to Bound	
Snapshot Times: [0]	
Linearization I/O	
IOs (sdoSimpleSuspension/x_dot:1 [output], sdoSimpleSuspension/Band	+ 1
	/
	~

2 Specify bounds on the gain margin or phase margin, or both.



- **Gain margin** Amount of gain increase or decrease required to make the loop gain unity at the frequency where the phase angle is -180°.
- **Phase margin** Amount of phase increase or decrease required to make the phase angle 180° when the loop gain is 1.0

To specify a lower bound on the gain margin or phase margin, or both, select the corresponding check box and enter the lower bound value.

3 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

4 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - **a** In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Specify Piecewise-Linear Lower and Upper Bounds on Frequency Response

To specify upper or lower bounds on the magnitude of a system response:

- 1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Bode Magnitude. A Create Requirement dialog box opens where you specify the lower or upper bounds on the magnitude of the system response. A new requirement with the name specified in Name appears in the Requirements area of the app.
- 2 Specify the requirement type in the **Type** drop-down list.
- **3** Specify the edge start and end frequencies and corresponding magnitude in the **Frequency** and **Magnitude** columns.
- 4 Insert or delete bound edges.

Click to specify additional bound edges.

To delete a bound edge, select a row, and click

5 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

6 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- 7 (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Bode Characteristics block to specify bounds on the magnitude of the system response. (Requires Simulink Control Design.)

Specify Bound on Closed-Loop Peak Gain

To specify an upper bound on the closed-loop peak response of a system:

- 1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Closed-Loop Peak Gain. A Create Requirement dialog box opens where you specify an upper bound on the closed-loop peak gain of the system. A new requirement with the name specified in Name appears in the Requirements area of the app.
- 2 Specify the upper bound in **Closed-Loop peak gain**.
- **3** In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- $\label{eq:b_select} b \quad \mbox{Select the linearization input/output set from the Linearization I/O area.}$

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

4 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.

b In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Nichols Characteristics block to specify bounds on the magnitude of the system response. (Requires Simulink Control Design.)

Specify Lower Bound on Damping Ratio

To specify a lower bound on the damping ratio of the poles of a system:

- 1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Damping Ratio. A Create Requirement dialog box opens where you specify a lower bound on the damping ratio of the system. A new requirement with the name specified in Name appears in the Requirements area of the app.
- 2 Specify the lower bound on the damping ratio in **Damping ratio**.
- **3** In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- $b \quad {\rm Select \ the \ linearization \ input/output \ set \ from \ the \ Linearization \ I/O \ area.}$

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

4 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Pole-Zero Characteristics block to specify a bound on the damping ratio. (Requires Simulink Control Design.)

Specify Upper and Lower Bounds on Natural Frequency

To specify a bound on the natural frequency of the poles of a system:

- 1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Natural Frequency. A Create Requirement dialog box opens where you specify a bound on the natural frequency of the system. A new requirement with the name specified in Name appears in the Requirements area of the app.
- 2 Specify a lower or upper bound on the natural frequency in **Natural frequency**.
- **3** In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

4 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Pole-Zero Characteristics block to specify a bound on the natural frequency. (Requires Simulink Control Design.)

Specify Upper Bound on Approximate Settling Time

To specify an upper bound on the approximate settling time of a system:

- 1 In the Sensitivity Analyzer, in the New Requirement drop-down list, select Settling Time. A Create Requirement dialog box opens where you specify an upper bound on the approximate settling time of the system. A new requirement with the name specified in Name appears in the Requirements area of the app.
- 2 Specify the upper bound on the approximate settling time in **Settling time**.
- **3** In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

4 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - a In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Pole-Zero Characteristics block to specify the approximate settling time. (Requires Simulink Control Design.)

Specify Piecewise-Linear Upper and Lower Bounds on Singular Values

To specify piecewise-linear upper and lower bounds on the singular values of a system:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Singular Values**. A Create Requirement dialog box opens where you specify the lower or upper bounds on the

singular values of the system. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

- 2 Specify the requirement type using the **Type** drop-down list.
- **3** Specify the edge start and end frequencies and corresponding magnitude in the **Frequency** and **Magnitude** columns, respectively.
- 4 Insert or delete bound edges.

Click to specify additional bound edges.

Select a row and click to delete a bound edge.

5 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

6 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- 7 (Optional) Plot the requirement.
 - **a** In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Singular Value Characteristics block to specify bounds on the singular value. (Requires Simulink Control Design).

Specify Step Response Characteristics

To apply a step response requirement to a linearization of your model (requires Simulink Control Design), specify the step response characteristics as follows:

1 Select a step response requirement from the **Sensitivity Analyzer**.

In the **New Requirement** drop-down list of the app, in the **New Frequency Domain Requirement** section, select **Step Response Envelope**.

A Create Requirement dialog box opens where you specify the step response requirements on a signal. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

- **2** Specify the step response characteristics:
 - Initial value Input level before the step occurs
 - Step time Time at which the step takes place
 - Final value Input level after the step occurs
 - **Rise time** The time taken for the response signal to reach a specified percentage of the step range. The step range is the difference between the final and initial values.
 - % **Rise** The percentage of the step range used with **Rise time** to define the overall rise time characteristics.
 - **Settling time** Time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
 - % **Settling** The percentage of the final value that defines the settling range of settling time characteristic specified in **Settling time**.
 - % **Overshoot** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
 - **% Undershoot** The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **3** Specify the systems to be bound.

To apply this requirement to a linearization of your Simulink model:

- **a** In the **Select Systems to Bound** area, specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

4 Close the Create Requirement dialog box.

The requirement in the **Requirements** area of the app is updated with the specified characteristics.

- **5** (Optional) Plot the requirement.
 - **a** In the **Sensitivity Analysis** tab of the app, in the **Requirements** area, select the requirement.
 - **b** In the **Plots** tab of the app, select the plot type to generate a graphical display of the requirement.

The plot is populated when you perform evaluation. A positive value indicates that the requirement has been violated.

Alternatively, you can use the Check Step Response Characteristics block to specify step response bounds for a signal.

Specify Custom Requirements

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

1 In the **Sensitivity Analyzer**, in the **New Requirement** drop-down list, select **Custom Requirement**.

A Create Requirement dialog box opens where you specify the reference signal to track. A new requirement with the name specified in **Name** appears in the **Requirements** area of the app.

- 2 Specify the requirement type in the **Type** drop-down menu.
- **3** Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click



to review or edit the function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

4 (Optional) To prevent the solver from considering specific parameter combinations, select **Error if constraint is violated**. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next combination of parameters in the parameter set.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current combination of parameter values. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

5 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
No signals have a reartly been colorted		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		Ū
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

c Select the signal and click **i** to add it to the signal set.

d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the Create Requirement dialog box.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

6 Select the check-box corresponding to the signal or system, and close the Create Requirement dialog box.

The requirement created in the **Requirements** area of the app is updated with the specified characteristics.

You can now evaluate the requirement. For more information, see "Evaluate Design Requirements" on page 4-60. When you perform evaluation, a positive requirement value indicates that the requirement has been violated.

See Also

Related Examples

- "Specify Time-Domain Requirements" on page 4-21
- "Evaluate Design Requirements" on page 4-60
- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112
- "Identify Key Parameters for Estimation (GUI)" on page 4-131

Evaluate Design Requirements

This topic shows how to evaluate your design requirements in the **Sensitivity Analyzer**. After you generate values for your parameter set and specify design requirements, you can evaluate your design requirements (cost function) at each sample of parameter values in your parameter set.

The generated parameter set, the corresponding parameter set table, and specified requirements are displayed in the app.

SENSITIVITY ANALYSIS	STATISTICS	PLOTS	PARAMETER SET	VIEW		li 🤈 🖻 🗗 🕐
		Parameter Set to Eva	luate:	M	Store Intermediate Data	
Open Save Session ▼ Session ▼ Pa	Select Generate arameters • Values •	ParamSet	 New Requirement 	Select for Evaluation	Options	Evaluate Optimize Model 👻
FILE	PARAMET	ER SET	REQUIRE	MENTS	OPTIONS	EVALUATE OPTIMIZE
Data Browser 💿	Parameter Set: Par	amSet 🛛 🗶				
 Parameter Sets 						
ParamSet	Delay	Gain	1	Tc	Тр	
		0.0049	0.8029		14.0276	0.0099
		0.0046	0.8710		15.6653	0.0103
		0.0051	0.8220		14.9205	0.0105
		0.0050	0.8732		13.9582	0.0097
		0.0052	0.7585		14.5234	0.0103
 Requirements 		0.0052	0.8282		15.3222	0.0098
SignalMatching		0.0051	0.7663		14.0752	0.0107 :
		0.0045	0.8275		15.7153	0.0107
		0.0046	0.8312		14.2285	0.0095
		0.0048	0.7309		16.2523	0.0102
		0.0050	0.7600		14.3072	0.0102
		0.0052				0.0101

To evaluate the design requirements:

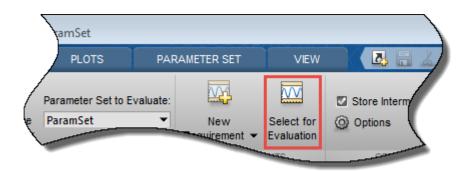
1 Select the parameter set to use for evaluation.

If you have multiple parameter sets defined, select the parameter set to use from the **Parameter Set to Evaluate** drop-down list.

SENSITIVITY ANALY	rsis	STATISTICS	PLOTS	PAF	RAMETER SET
Open Save	Selec	Generate	Parameter Set to E ParamSet	ivaluate: ▼	Ne Requirem

2 Select the requirements you want to evaluate.

If you have multiple requirements defined, they are all selected by default for evaluation. To exclude a requirement from evaluation, click **Select for Evaluation**.



3 Specify evaluation options.

Click **Options**. In the Evaluation Options dialog box, in the **General Options** tab, specify options related to evaluation progress and results. If you have Parallel Computing Toolbox, you can specify options for parallel computing in the **Parallel Options** tab. If you have Simulink Control Design software, you can specify options for linearizing your model in the **Linearization Options** tab. For details about these options, click **Help**.

Evaluation Options		×
General Options	Parallel Options	Linearization Options
Progress Option	s	
📝 Update scatt	er plots during ev	aluation
Error Options		
If evaluation erro	rs for one set of p	arameter values:
Ontinue	evaluation at oth	er parameter values
🔘 Stop eval	uation	
Result Options		
Evaluation result	S:	
Save eval	uation as new res	ult set
Overwrite	e evaluation result	with new values
	OK Cancel	Help
		.::

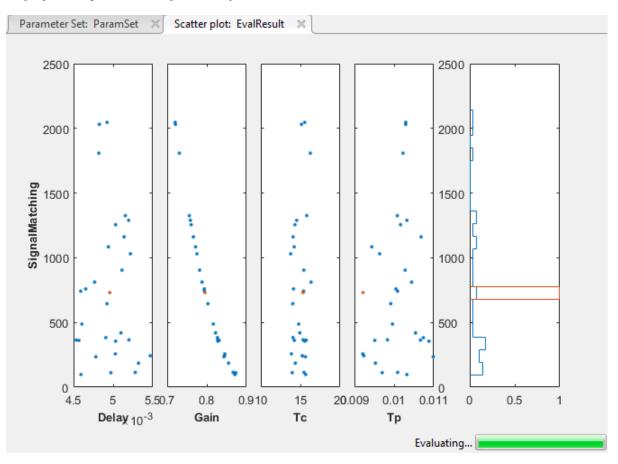
4 Speed up successive evaluations of design requirements that require the same logged signals and parameter values.

Click **Store Intermediate Data**. For more information, see "Store Intermediate Data in the App" on page 4-100.

5 Evaluate the design requirements for each sample in your parameter set.

Click **Evaluate Model**. The cost function associated with your design requirement is evaluated using each sample in your parameter set. A sample corresponds to a row of values in the parameter set table.

The app generates an evaluation result scatter plot. The plot updates to display the evaluated cost function value as a function of each parameter in the parameter set. The last column of subplots displays histograms of the probability distribution of the evaluated cost function values.



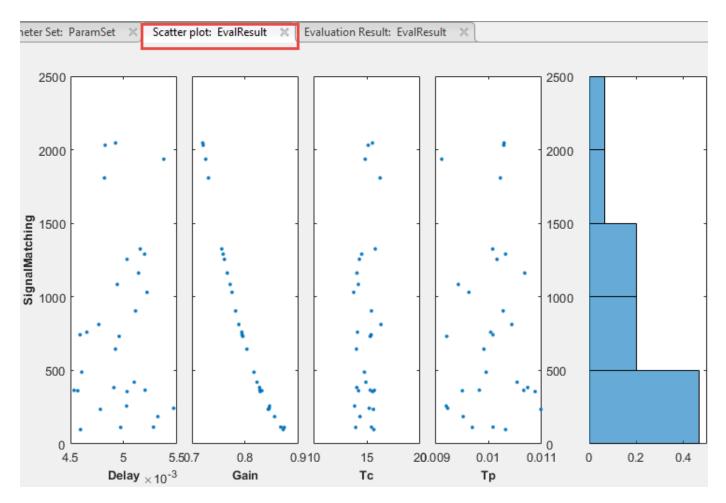
When the evaluation is complete, the app displays the evaluated requirement (normalized cost function value) and the corresponding parameter values in the **Evaluation Result** table. The app also creates a new variable with this information in the **Results** area.

SENSITIVITY ANALY	'SIS	STATISTICS	PLOTS	VIEW			🔓 🤉 e 🖻 🕐 🎙
	:		Parameter Set to Evaluate:		WV	Store Intermediate Dat	a 🔁 🖾
Open Save Session ▼ Session ▼	Sele Paramet	ct Generate ers ▼ Values ▼	ParamSet 🔹	New Requirement ▼	Select for Evaluation	Co optiono	Evaluate Optimize Model 👻
FILE		PARAMETE	R SET	REQUIREM	IENTS	OPTIONS	EVALUATE OPTIMIZE
Data Browser 💿	Pa	rameter Set: Param	nSet 🛛 🛛 Scatter plot:	EvalResult 🛛 🛛	Evaluatio	n Result: EvalResult 🛛 🛛 🖉	
▼ Parameter Sets	Sele	ct and right-click a ro	w to create an initial guess	for Response Opti	mization or	Parameter Estimation.	
ParamSet	Dela	у	Gain	Tc		Tp Sig	gnalMatching
		0.0049	0.802	9	14.0276	0.0099	644.8298 🔺
		0.0046	0.871	D	15.6653	0.0103	97.8802
		0.0051	0.822	D	14.9205	0.0105	
		0.0050	0.873	2	13.9582	0.0097	112.4360
 Requirements 		0.0052	0.758	5	14.5234	0.0103	
SignalMatching		0.0052	0.828	2	15.3222	0.0098	
		0.0051	0.766	3	14.0752	0.0107	1.1616e+03
		0.0045	0.827	5	15.7153	0.0107	364.4182
		0.0046	0.831	2	14.2285	0.0095	361.4330
		0.0048	0.730	9	16.2523	0.0102	
		0.0050	0.760	В	14.3072	0.0102	
EvalResult		0.0052	0.755	В	15.7965	0.0101	
		0.0049	0.826	9	14.0660	0.0107	383.3321
		0.0053		-	14.3625	0.0095	185.9608
		0.0052		1		0.0096	1.0309e+03
		0.0055	0				242.5879

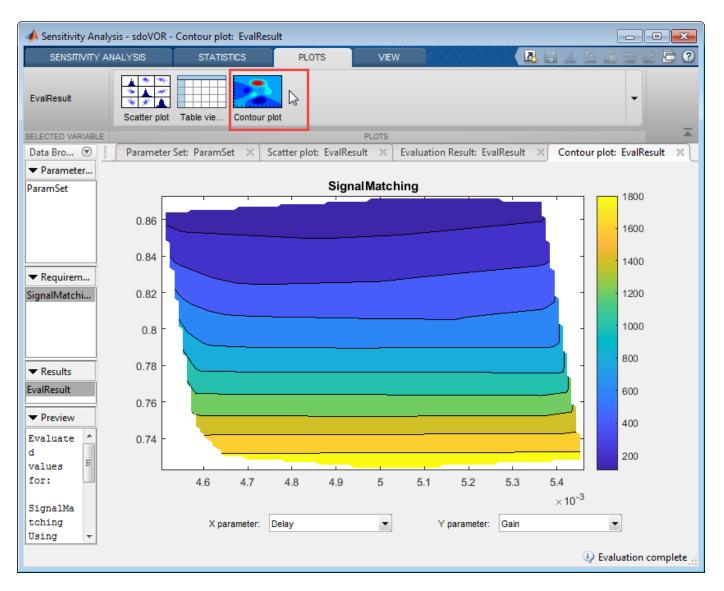
For requirements that involve a bound, a positive requirement value indicates that your requirement was violated, while a negative value indicates that the requirement was satisfied for that sample of parameter values. The parameter sample (row) corresponding to the least requirement value comes closest to satisfying your requirement.

You can employ the evaluation results to configure estimation or optimization of your parameters. For more information, see "Use Sensitivity Analysis to Configure Estimation and Optimization" on page 4-74.

The app displays the final scatter plots and probability distribution histogram. The number of data points in each scatter plot equals the number of samples (rows) in the parameter set table. Use this plot to visually analyze the relation between parameters and requirements. For example, in this case, the SignalMatching requirement looks monotonically related to the Gain parameter. To learn more about the plot, see "Interact with Plots in the Sensitivity Analyzer" on page 4-79.



You can make a contour plot of the evaluated results. To do so, select the evaluated result in the **Results** area of the app, and choose a contour plot in the **Plots** tab of the app.



The contour plot shows that the requirement does not vary systematically as a function of Delay, but it does as a function of Gain.

You can also quantify the relation between parameters and requirements. For more information, see "Analyze Relation Between Parameters and Design Requirements" on page 4-67.

See Also

Related Examples

- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Specify Time-Domain Requirements" on page 4-21
- "Specify Frequency-Domain Requirements" on page 4-48
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67

- "Validate Sensitivity Analysis" on page 4-96
- "Identify Key Parameters for Estimation (GUI)" on page 4-131

Analyze Relation Between Parameters and Design Requirements

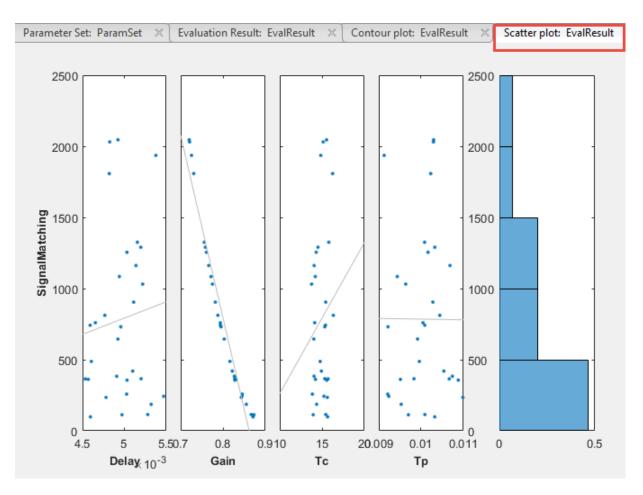
To analyze how the parameters and states (collectively referred to as parameters) of a Simulink model influence the design requirement on the model signals, you first generate samples of the parameters. You then define the cost function by creating a design requirement on the model signals, and evaluate the cost function for each sample. Finally, you analyze the relationship between the parameter variations and the cost function values. You can perform this analysis in the following ways:

- "Visual Analysis" on page 4-67
- "Statistical Analysis" on page 4-68

Visual Analysis

View a plot of the cost function evaluations against the parameter samples to identify trends. This method is informal and provides visual intuition about how the various parameters affect the cost function.

In the **Sensitivity Analyzer**, after the evaluation is complete, an evaluation result scatter plot is generated in the app. The plot displays the evaluated cost function value as a function of each parameter in the parameter set. The last column subplot displays the probability distribution of the evaluated cost function values. You can add a best-fit line to the scatter subplots by right-clicking in the plot, and selecting **Overlay linear fit** in the context menu. In this plot, the best-fit line indicates that the **Gain** parameter has a lot of influence on the requirement.



You can also plot a contour plot of the evaluated results. To learn more about these plots, see "Interact with Plots in the Sensitivity Analyzer" on page 4-79. For an example, see "Identify Key Parameters for Estimation (GUI)" on page 4-131.

At the command line, you can use tools such as:

- $\bullet \quad \mathsf{sdo.scatterPlot} \mathsf{Scatter} \ \mathsf{plot} \ \mathsf{of} \ \mathsf{the} \ \mathsf{parameter} \ \mathsf{samples} \ \mathsf{against} \ \mathsf{the} \ \mathsf{cost} \ \mathsf{function} \ \mathsf{evaluation}$
- ${\tt surf}, {\tt mesh}, {\tt contour}-{\tt 3-D}$ plot of samples of two parameters against the cost function evaluation

For an example, see "Identify Key Parameters for Estimation (Code)" on page 4-169.

Statistical Analysis

In addition to visually analyzing the effect of parameters on the cost function, you can also compute statistics to quantify the relation.

Obtain summary statistics about the relationship between cost function evaluations and parameters samples. Available analysis methods include:

Method	Description
Correlation on page 4-70	Use to analyze how a model parameter and the cost function output are correlated.
Partial Correlation on page 4-70	Use to analyze how a model parameter and the cost function are correlated, removing the effects of the remaining parameters.
Standardized Regression on page 4-70	Use when you expect that the model parameters linearly influence the cost function.

For each of these methods, you specify what data to use for the analysis by choosing from the following analysis types:

- Linear analysis, also referred to as Pearson analysis Uses raw data for analysis. Use linear analysis when you expect a linear relation between the parameters and cost function, and when the residuals about the best-fit line are expected to be normally distributed. Linear analysis is also recommended when the number of samples, and so the number of residual points is large.
- Ranked analysis, also referred to as Spearman analysis and ranked transformation Uses ranks of data for analysis. Use ranked analysis when you expect a nonlinear monotonic relation between the parameters and the cost function and when the residuals about the best-fit line are not normally distributed. Ranked analysis is also recommended when the number of samples, and so the number of residual points is small.

Linear analysis retains information about intervals between data values, whereas ranked analysis does not. Suppose that you had the following data set:

>	K ₁	X ₂	У
9)	20	340
5	5	60	106
	2.3	50.4	870.5

Here x_1 and x_2 are model parameters, and y is the cost function. Each row represents a sample and the associated cost function evaluation.

The data is ranked on a per column basis. For example, when you rank the data in column 1 (x_1), which contains the entries 9, 5, and 2.3, the ranked data is equal to 3, 2, and 1. The ranked data set for the samples of x_1 , x_2 and y are as follows:

×1	x ₂	У
3	1	2
2	3	1
1	2	3

The ranked data set can be used for correlation, partial correlation, or standardized regression analysis.

• Kendall — Kendall's tau rank correlation coefficient is calculated.

Applicable when the analysis method is Correlation. Requires Statistics and Machine Learning Toolbox software.

Correlation Method

Calculates the correlation coefficients, R. Use this method to analyze how a model parameter and the cost function outputs are correlated.

R is calculated as follows:

$$R(i, j) = \frac{C(i, j)}{\sqrt{C(i, i)C(j, j)}}$$

$$C = cov(x, y)$$

$$= E[(x - \mu_x)(y - \mu_y)]$$

$$\mu_x = E[x]$$

$$\mu_y = E[y]$$

x contains Ns samples of Np model parameters. y contains Ns rows, each row corresponds to the cost function evaluation for a sample in x.

R values are in the [-1 1] range. The (i,j) entry of R indicates the correlation between x(i) and y(j).

- R(i,j) > 0 Variables have positive correlation. The variables increase together.
- R(i,j) = 0 Variables have no correlation.
- R(i,j) < 0 Variables have negative correlation. As one variable increases, the other decreases.

Partial Correlation Method

Calculates the partial correlation coefficients, *R*. This method requires Statistics and Machine Learning Toolbox software. Use this method to analyze how a model parameter and the cost function are correlated, adjusting to remove the effect of the other parameters.

R is calculated using partialcorri from the Statistics and Machine Learning Toolbox software.

Standardized Regression Method

Calculates the standardized regression coefficients, R. Use this method when you expect that the model parameters linearly influence the cost function.

R is calculated as follows:

$$R = b_{\chi} \frac{\sigma_{\chi}}{\sigma_{\chi}}$$

Consider a single sample $(x_1,...,xNp)$ and the corresponding single output, y. b_x is the regression

coefficient vector calculated using least squares assuming a linear model $\hat{y} = b_0 + \sum_{i=1}^{Np} \hat{b}_{x_i} x_i$. R

standardizes each element of b_x by multiplying it with the ratio of the standard deviation of the corresponding x sample (σ_x) to the standard deviation of y (σ_y).

Perform Statistical Analysis

In the **Sensitivity Analyzer**, after you have evaluated the design requirements on page 4-60, specify the analysis methods and types in the **Statistics** tab of the app.

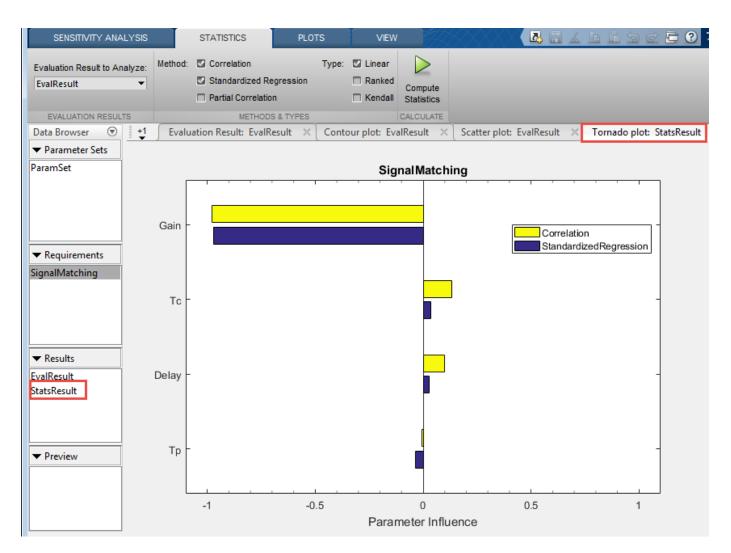
Select the evaluation results you want to analyze in the Evaluation Results to Analyze list. After

that, you specify the analysis methods and types, and click Compute Statistics. You can compute all applicable combinations of analysis methods and types.

📣 Sensitivity Analysis* - sdoVOR - Tornado plot: StatsResult							
SENSITIVITY ANALYSIS	STATISTICS	PLOTS	VIEW		\otimes		
Evaluation Result to Analyze: EvalResult	Method: Correlation Standardized Re Partial Correlation	gression	Linear Ranked Kendall	Compute Statistics			
EVALUATION RESULTS	METHODS	S & TYPES		CALCULATE	_		

The results of the analysis are returned in the StatsResult variable, in the **Results** area of the app. In this case, the StatsResult variable includes the linear (Pearson) correlation coefficients and linear standardized regression coefficients calculated between the cost function and each parameter. To see the coefficients, right-click StatsResult, and select **Open** in the context-menu.

A tornado plot is generated that displays the results of the analysis in order of influence of parameters on the cost function. The parameter that most influences the cost function is displayed on the top. As was seen in the results scatter plot, in this tornado plot the Gain parameter has the most influence on the design requirement cost function.



To learn more about tornado plots, see "Interact with Plots in the Sensitivity Analyzer" on page 4-79. For an example, see "Identify Key Parameters for Estimation (GUI)" on page 4-131.

At the command line, specify the analysis methods and types using sdo.analyze. This function performs linear correlation analysis by default. To specify other analysis methods, use sdo.AnalyzeOptions. For an example, see "Identify Key Parameters for Estimation (Code)" on page 4-169.

See Also

sdo.AnalyzeOptions|sdo.analyze|sdo.sample|sdo.evaluate

Related Examples

- "What is Sensitivity Analysis?" on page 4-2
- "Evaluate Design Requirements" on page 4-60
- "Identify Key Parameters for Estimation (GUI)" on page 4-131
- "Identify Key Parameters for Estimation (Code)" on page 4-169

• "Use Sensitivity Analysis to Configure Estimation and Optimization" on page 4-74

Use Sensitivity Analysis to Configure Estimation and Optimization

This topic shows how to use the results generated in the **Sensitivity Analyzer** to configure parameter estimation or response optimization.

You can use sensitivity analysis to evaluate how the parameters of a Simulink model influence the model output or model design requirements. You first generate samples of the parameters, and then define a cost function by creating a design requirement on the model signals. For more information see, "Generate Parameter Samples for Sensitivity Analysis" on page 4-8, "Specify Time-Domain Requirements" on page 4-21, and "Specify Frequency-Domain Requirements" on page 4-28. You then evaluate the requirement (cost function) for each sample. You can use the evaluated results to configure parameter estimation or response optimization in the following ways:

- 1 Analyze the relationship between the parameters and the evaluated requirement values, and rank parameters in order of influence. For more information, see "Analyze Relation Between Parameters and Design Requirements" on page 4-67. You can then choose to estimate or optimize the more influential parameters.
- **2** Obtain initial guesses for parameter values for estimation or optimization.

After you evaluate the requirements in the **Sensitivity Analyzer**, the evaluated requirement (cost function value) and the corresponding parameter values are displayed in the **Evaluation Result** table of the app. A new variable, EvalResults, with this information is created in the **Results** area of the app. For example, the table below lists the evaluated signal matching requirement and corresponding values for the parameters Gain, Delay, Tc, and Tp of the model, sdoVOR.

SENSITIVITY ANALYSI	IS ST.	ATISTICS	B PLOTS	VIEW	AH-1-1-1		li i d c 🗗 🕐 🤅
	=		Parameter Set to Evaluate:		Store Intermedi	ate Data	
Open Save Session ▼ Session ▼ F		enerate alues 💌	ParamSet 💌	New Select Requirement • Evaluation	Co optionio	Evaluate Optimi Model 👻	ze
FILE		PARAME	TER SET	REQUIREMENTS	OPTIONS	EVALUATE OPTIMI	ZE
Data Browser		•	Parameter Set: Params	Set 🛛 Scatter plot: E	valResult 🛛 Evaluati	on Result: EvalResult 🛛 🛛	
 Parameter Sets 		•	Select and right-click a row	to create an initial guess f	or Response Optimization of	r Parameter Estimation.	
ParamSet			Delay	Gain	Tc	Тр	SignalMatching
			0.0049	0.8029	14.027	6 0.0099	644.8298 🔺
			0.0046	0.871	15.665	3 0.0103	97.8802
			0.0051	0.822) 14.920	5 0.0105	
			0.0050	0.873	2 13.958	2 0.0097	112.4360
 Requirements 			0.0052	0.758	5 14.523	4 0.0103	1.2900e+03
ignalMatching			0.0052	0.828	2 15.322	2 0.0098	365.7245
			0.0051	0.766	3 14.075	2 0.0107	1.1616e+03
			0.0045	0.827	5 15.715	3 0.0107	364.4182
			0.0046	0.8312	2 14.228	5 0.0095	361.4330
			0.0048	0.730	9 16.252	3 0.0102	
Results			0.0050	0.760	3 14.307	2 0.0102	
valResult			0.0052	0.755	3 15.796	5 0.0101	1.3255e+03
			0.0049	0.826	9 14.066	0 0.0107	
			0.0053	0.855	L 14.362	5 0.0095	
			0.0052	0.77 <u>5</u>		0.0096	
		-	0.0055			0.0092	242,5879

To extract parameter values to use as an initial guess during estimation or optimization:

a Sort the evaluated cost function values in ascending order by clicking the evaluated requirement column.

Delay	Gain	Tc	Тр	SignalMatching
0.0046	0.8710	15.6653	0.0103	97.8802
0.0050	0.8732	13.9582	0.0097	112.4360
0.0053	0.8666	15.4429	0.0101	114.2653
0.0053	0.8551	14.3625	0.0095	185.9608
0.0048	0.8432	15.6278	0.0110	234.8702
0.0055	0.8449	15.2286	0.0092	242.5879
0.0050	0.8456	13.8582	0.0092	257.6335
0.0050	0.8281	15.5501	0.0109	355.8483
0.0046	0.8312	14.2285	0.0095	361.4330
0.0045	0.8275	15.7153	0.0107	364.4182
0.0052	0.8282	15.3222	0.0098	365.7245
0.0049	0.8269	14.0660	0.0107	383.3321
	0.8220	and the second second	9.0105	419.7438
				102-00

b To choose the parameter values that minimize the cost function, right-click corresponding row, and select **Extract Parameter Values**.

Delay	Gain	Tc	Тр	SignalMatching
0.0046	0.8710	15.6653		802 ^
0.0050	0.8732	13.9582		t Parameter Values 360
0.0053	0.8666	15.4429	Plot al	l result data 653
0.0053	0.8551	14.3625	0.0095	185.9608
0.0048	0.8432	15.6278	0.0110	234.8702
0.0055	0.8449	15.2286	0.0092	242.5879
9.0050	0.8456	13.8582	0.0092	257.6335
	0.8281			355.8482

A new variable, ParamValues, is created in the **Results** area of the app.

▼ Results	
EvalResult	
ParamValues	

c When exporting to a Parameter Estimator or Response Optimizer app session on page 4-76, choose this variable to specify the initial guess for parameters.

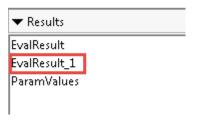
3 To test the robustness of your design during optimization in the **Response Optimizer** app, specify the values for uncertain parameters. For more information, see "Optimizing Parameters for Robustness" on page 3-170.

You can specify the values for uncertain parameters using all the parameter values in EvalResults. You can also choose a subset of the parameter values:

Delay	Gain	Tc	Тр		SignalMatching	Δ
0.0054	0.8694	15.9519		0.0100		100.2616
0.0046	0.8668	13.9571		0.0098		137.8549
0.0047	0.8687	13.9969		0.0093		138.2714
0.0049	0.8545	13.9865		0.0099		192.0046
0.0052	0.8529	10 7515	1	0.0102		196.9189
0.0053	0.8503	Create new evaluatio	n result	0.0094		215.8378
0.0052	0.8447			0.0109		222.4661
0.0055	0.8412	Plot all result data		0.0095		266.2807
0.0048	0.8406	16.4884		0.0092		268.3061
0.000	0.8186		_	0.0101		451.3620
	0.8136					and the second second

a Select the relevant rows of parameter values in the **Evaluation Result** table of the app.

b Right-click, and select **Create new evaluation result**. A new variable is created in the **Results** area of the app.



c When exporting to a Response Optimizer app session on page 4-76, choose this variable to specify the uncertain variables.

Export Sensitivity Analysis Results

You can export results from **Sensitivity Analyzer** to the **Parameter Estimator** and **Response Optimizer** apps. To do so, in the **Sensitivity Analyzer**, click **Optimize**. In the drop-down menu, choose the app to export to.

				<u>- □ ×</u> - • • • • • • • • • • •
t for uation	Store Intermediate Data Options	Evaluate Model	Optimiz	e
Result X	OPTIONS Evaluation Result: Eva Optimization or Parameter E			Create Response Optimization Session Use sensitivity analysis data to create a response optimization session.
Tc 0.8694 0.8668	15.9519 13.9571	Тр		Create Parameter Estimation Session Use sensitivity analysis data to create a parameter estimation session.
Laconternation			-	0.0093 138.2714

Alternatively, if you have an open **Parameter Estimator** or **Response Optimizer** session, in these apps, click **Sensitivity Analysis**. In the drop-down menu, choose **Import from Open Session** or **Import from Session file**. The latter option loads results from a previously saved **Sensitivity Analyzer** session.

📣 Parame	ter Estimatio	n - sdoVOR								
PARAN	METER ESTIMA		VALIDATI	ON	VIEW			a de la compañía de la	the second s	$\langle \rangle \rangle$
								Cost Function:	Sum Squared	Error
Open Session 👻	Save Session 💌	New Experiment	Select Experiments	Select Parameters	Sensitivity Analysis 💌	Add Plot	Plot Model Response	🎯 More Opti	ions	
FI	LE	EXPER	IMENTS	PARAM	EXPORT TO	SENSITIVI	TY ANALYSIS			
Data Brows			•		Use se estima	nsitivity an tion prob NSITIVITY A t from Ope	ANALYSIS RES	ct model paran	neters to tune.	
▼ Experim	ents				to cor	figure the t from Ses	nalysis result : estimation sion File nalysis result	problem.		/
						-	estimation			7

Note

- Only signal matching requirements are exported from Sensitivity Analysis to a **Parameter Estimator** session. In the **Parameter Estimator**, they are referred to as experiments.
- Only requirements other than signal matching requirements are exported from Sensitivity Analysis to a **Response Optimizer** session.

See Also

Related Examples

- "What is Sensitivity Analysis?" on page 4-2
- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112
- "Identify Key Parameters for Estimation (GUI)" on page 4-131

Interact with Plots in the Sensitivity Analyzer

This topic shows how to interact with and interpret plots generated in the **Sensitivity Analyzer** app.

Parameter Set Plots

After you have generated parameter values for sensitivity analysis, you can plot the generated parameter set. For information about parameter generation, see "Generate Parameter Samples for Sensitivity Analysis" on page 4-8.

The app displays the generated parameter set and the corresponding parameter set table. The number of rows in the parameter set table correspond to the number of samples you specified. To plot the generated parameters in the app:

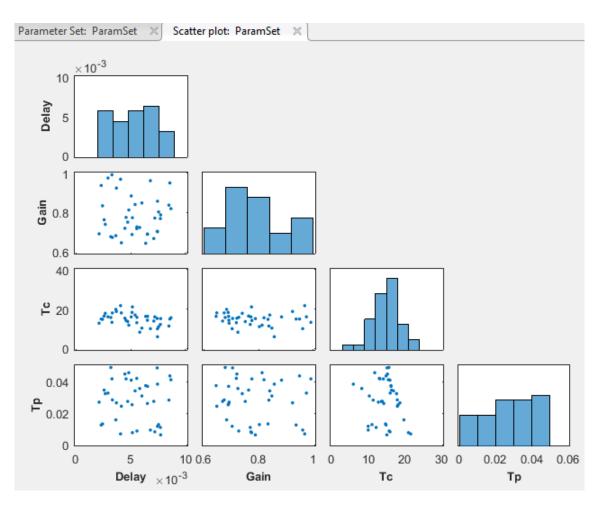
📣 Sensitivity Analysis* - sdoVOR - Parameter Set: ParamSet							
SENSITIVITY ANALY	YSIS STATISTI	CS PLOTS	PARAMETER SET VIE	ew 🖪 🛃 🚣	1 1 5 c 🗗 🖓 🗖		
ParamSet	Scatter plot	editor			-		
SELECTED VARIABLE	L3		PLOTS				
Data Browser	$\overline{\mathbf{v}}$	Parameter Set: Para	mSet				
➡ Parameter Sets							
ParamSet		Delay	Gain	Tc	Тр		
		0.004	9 0.8029	14.0276	0.0099 🔺		
		0.004	6 0.8710	15.6653	0.0103		
		0.005	0.8220	14.9205	0.0105		
		0.005	0.8732	13.9582	0.0097		
		0.005	2 0.7585	14.5234	0.0103		
 Requirements 		0.005	2 0.8282	15.3222	0.0098		
SignalMatching		0.005	1 0.7663	14.0752	0.0107		
		0.004	5	15.7153	0.0107		
					0.000		

1 Select the generated parameter set in the **Parameter Sets** area of the app.

2 On the **Plots** tab, select **Scatter Plot**.

Alternatively, right-click the parameter set, and select **Plot** in the drop-down menu.

The diagonal subplots display the histograms of generated parameter values. The off-diagonal subplots are pair-wise scatter plots of the parameters. The number of data points in each scatter plot equals the number of rows in the parameter set table.



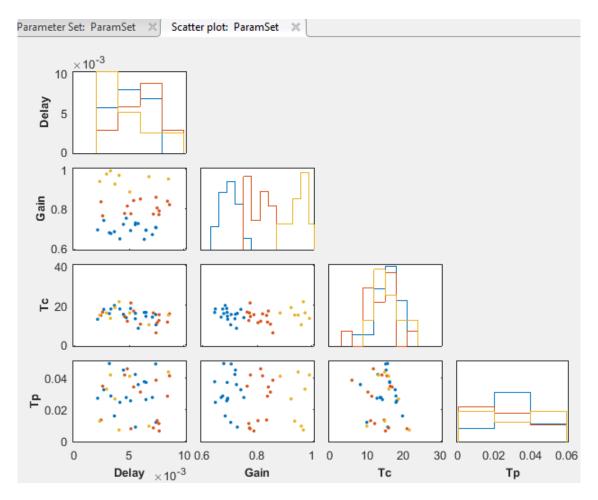
You can inspect the histograms to ensure that the generated parameter values match the desired parameter distributions within the constraints of a finite sample size. Inspect the off-diagonal scatter plots to ensure that any specified correlations between parameters are present. For more information, see "Inspect the Generated Parameter Set" on page 4-96.

3 To access additional plot features, right-click in the white area of any scatter plot.

You can choose from the following options:

- Variables Select the parameters to plot.
- **Groups** Select grouping variables for the plots, and configure how the groups are displayed.

To select a parameter as a grouping variable, click **Groups > New Grouping Variable**. For example, the following plot is generated when the grouping variable is Gain.



The app creates three groups based on low, medium, and high values of the grouping variable. The app computes these grouping values, but you can change them in the **Manage Groups** dialog box. The second diagonal plot shows the distribution of the gain values in the low (blue), medium (red), and high (yellow) groups. The other diagonal plots show the distribution of the remaining parameters when the corresponding gain value is low, medium, or high. The off-diagonal scatter plots show points belonging to the same group using the same color.

You can demarcate the groups based on marker size and marker type instead of color, add more groups corresponding to the grouping variable, and change the grouping values. You can also add more grouping variables. To do so, click **Groups > Manage Groups**.

Grouping V		Label	Туре		yle	Active	Remove
Gain	Gain		Continuous	Color	•		×
Create Groupin	g Variable:	Create Gr	ouping Variab	e		•	
			-				
ain (3 Gro	upc)						
aiii (5 010	ups)						
	Dia (Value	Cal	en Ch		Damager		
Group Label		Col		ow	Remove	:	
Group Label Low	Bin/Value 0.7736 0.8265	Col		/ X	:		
	0.7736	Col			; ;		
Group Label Low Medium	0.7736 0.8265	Col			; ;		

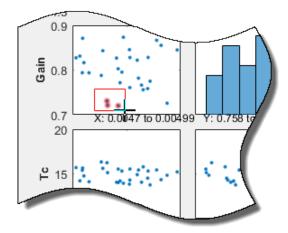
In the **Manage Groups** dialog box, you configure how the groups are displayed. You can perform tasks such as:

- Select the plotting **Style** as either **Color**, **MarkerSize**, or **MarkerType**. In the plots, the app uses the selected style to demarcate the groups corresponding to a grouping variable.
- Select whether a grouping variable is **Active**. If a grouping variable is inactive, the scatter plot points are not demarcated in groups corresponding to that variable. To delete a grouping variable, click **X** in the corresponding **Remove** column.
- Add more grouping variables using the **Create Grouping Variable** drop-down list.
- For a grouping variable, specify the range of values for each group in the **Bin/Value** column. For example, currently the dialog box shows that the Gain values in the groups are:
 - **Low** below 0.7736
 - **Medium** 0.7736-0.8265
 - **High** above 0.8265

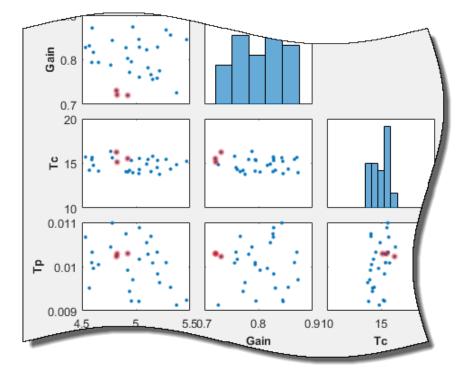
To change the **Low** group values to be 0.79 or lower, type 0.79 in the corresponding row of the **Bin/Value** column.

- Add more groups corresponding to a grouping variable. For example, to add a group with values from 0.8265 through 0.9, type 0.9 in New Bin/Value, and click Add Group.
- **Upper triangle plot** Plot the off-diagonal subplots above the diagonal in addition to the existing plots.
- **Marginal Box Plots** Requires Statistics and Machine Learning Toolbox software. Plot box plots for each of the parameters in the parameter set, and choose the position of the plots.

- **Histograms** Plot the probability distribution of the parameters, and choose the position of the plots.
- **Kernel Density Plots** Requires Statistics and Machine Learning Toolbox software. Plot the probability distribution of the parameters using a kernel density estimator, and choose the position of the plots. For more information, see "Kernel Distribution" (Statistics and Machine Learning Toolbox).
- **Overlay linear fit** Plot the best-fit line on the scatter subplots. You can choose to plot the best-fit lines for one, all, a row, or a column of scatter subplots.
- Enable brushing/data selection Enable selection of data points in the scatter subplots.



When you highlight parameter values in one plot, the values corresponding to other parameters from the same row in the parameter set table are also highlighted. In addition, the rows in the parameter set table that correspond to these values are highlighted.



Delay		Gain	Tc	Тр
	0.0049	0.8029	14.0276	0.0099
	0.0046	0.8710	15.6653	0.0103
	0.0051	0.8220	14.9205	0.0105
	0.0050	0.8732	13.9582	0.0097
	0.0052	0.7585	14.5234	0.0103
	0.0052	0.8282	15.3222	0.0098
	0.0051	0.7663	14.0752	0.0107
	0.0045	0.8275	15.7153	0.0107
	0.0046	0.8312	14.2285	0.0095
	0.0048	0.7309	16.2523	0.0102
	0.0050	0.7608	14.3072	0.0102
	0.0052	0.7558	15.7965	0.0101
	0.0049	0.8269	14.0660	0.0107
	0.0053	0.8551	14.3625	0.0095
	0.0052	0.7751	13.7733	0.0096
	0.0055	0.8449	15.2286	0.0092
	0.0050	0.8281	15.5501	0.0109
	0.0048	0.7211	15.1398	0.0103
	0.0046	0.8163	14.7772	0.0100
	0.0051	0.7819	15.4333	0.0103
	0.0053	0.8666	15.4429	0.0101
	0.0049	0.7202	15.5371	0.0103
	0.0046	0.7940	15.4074	0.0101
	0.0048	0.7879	16.3355	0.0104

To remove the highlighting, invert the selection to all other data points in the plot, or disable the feature, right-click the highlighted data points, and choose from the context-menu.

• **Pop-out plot** — View a subplot in a new window.

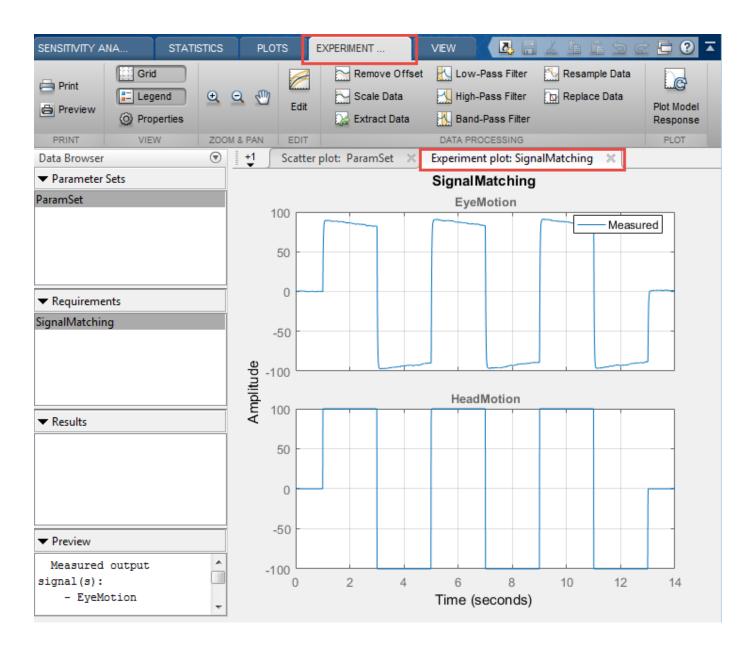
Requirement Plots

After you have specified design requirements, you can plot the requirements and associated model response. For information about specifying the requirements, see "Specify Time-Domain Requirements" on page 4-21 and "Specify Frequency-Domain Requirements" on page 4-48.

The specified requirements are displayed in the **Requirements** area of the app. To plot the requirement in the app, right-click the requirement, and select **Plot**.

Data Browser		$\overline{\mathbf{v}}$	+ <u>+</u>
➡ Parameter Sets			
ParamSet			
 Requirements 			
SignalMatching	✓ Use	in evalua	tion
	Plot	t N	
	Plot	t & Simula	ite
	Edit	t	
 Results 	Cop	у	
	Ren	ame	
	Del	ete	

Alternatively, select the requirement, and in the **Plots** tab of the app, select the plot type. A plot is generated and a new tab associated with the plot appears in the app. In the new tab, you can perform additional tasks such as preprocessing imported data on page 1-13 (for signal matching requirement only), zooming, and plotting the associated model response. The model response is the signal or system on which the requirement is applied. The response is plotted using the parameter values specified in the model workspace and is not updated during evaluation.

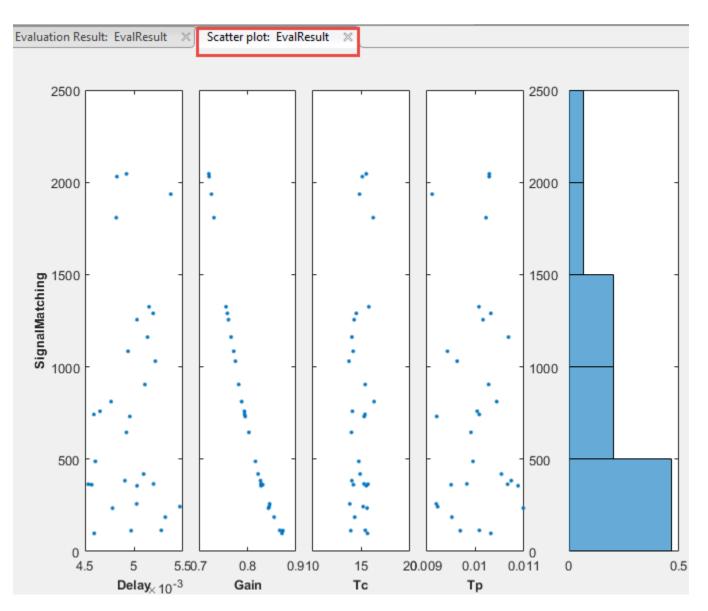


Evaluated Result Scatter Plots

After you have evaluated your design requirements on page 4-60, an evaluation results table lists the samples in the parameter set and the corresponding evaluated requirement (cost function) values. For requirements that involve a bound, a positive requirement value indicates that your requirement was violated, while a negative value indicates that the requirement was satisfied for that sample of parameter values.

			n or Parameter Estimation.	
Delay	Gain	Tc	Тр	SignalMatching
0.0049	0.8029	14.0276	0.0099	644.8298
0.0046	0.8710	15.6653	0.0103	97.8802
0.0051	0.8220	14.9205	0.0105	419.7438
0.0050	0.8732	13.9582	0.0097	112.4360
0.0052	0.7585	14.5234	0.0103	1.2900e+03
0.0052	0.8282	15.3222	0.0098	365.7245
0.0051	0.7663	14.0752	0.0107	
0.0045	0.8275	15.7153	0.0107	364.4182
0.0046	0.8312	14.2285	0.0095	361.4330
	0.7309	16-25-	0.0102	1.8083e+03
	0.7608			1.2555

An evaluation result plot is also generated. The scatter subplots display the evaluated requirement (cost function value) as a function of each parameter in the parameter set. The number of points in each scatter plot equals the number of rows in the parameter set. The last column of subplots displays histograms of the probability distribution of the evaluated cost function values.



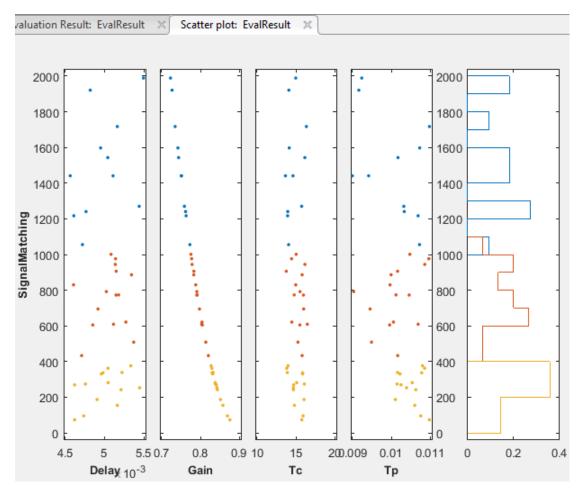
Use this plot to visually analyze the relation between parameters and requirements on page 4-67. For example, in this case, the SignalMatching requirement looks monotonically related to the Gain parameter.

You can also plot best-fit lines on the scatter subplots. To do so, and to access additional plot features, right-click in the white area of any scatter subplot.

You can choose from the following options:

- **X-Variables** Select the parameters and requirements to use as x-variables in the scatter subplots.
- **Y-Variables** Select the parameters and requirements to use as y-variables in the scatter subplots.
- **Grouping** Select grouping variables for the subplots, and configure how the groups are displayed.

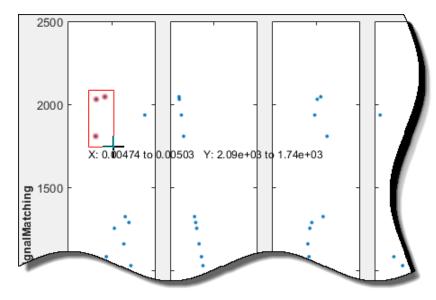
To select a parameter or evaluated requirement as a grouping variable, click **Groups > New Grouping Variable**. For example, the following plot is generated when the grouping variable is Gain. The app creates three groups based on low, medium, and high values of the grouping variable. The app computes these grouping values, but you can change them in the **Manage Groups** dialog box. The scatter subplots display the evaluated requirement values when the corresponding gain value is low (blue), medium (red), and high (yellow). The histogram plots the probability distribution of the evaluated requirement corresponding to the groups.



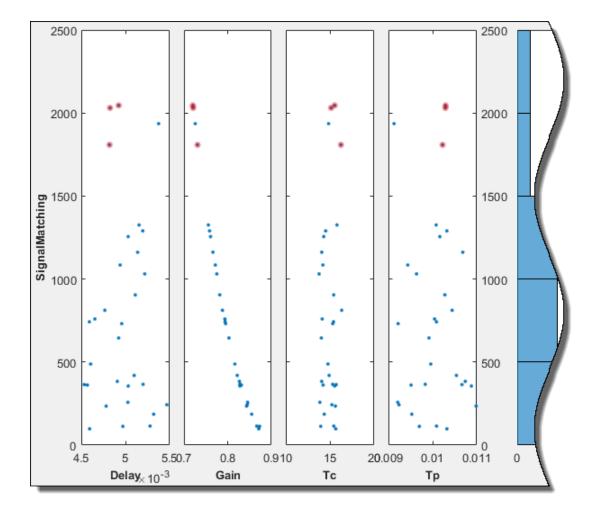
You can demarcate the groups based on marker size and marker type instead of color, add more groups corresponding to the grouping variable, and change the grouping values. You can also add more grouping variables. To do so, click **Groups** > **Manage Groups**. For more information, see "Parameter Set Plots" on page 4-79.

- **Marginal Box Plots** Requires Statistics and Machine Learning Toolbox software. Plot box plots for each of the parameters in the parameter set, and choose the position of the plots.
- **Histograms** Plot the probability distribution of the parameters, and choose the position of the plots.
- **Kernel Density Plots** Requires Statistics and Machine Learning Toolbox software. Plot the probability distribution of the parameters using a kernel density estimator, and choose the position of the plots. For more information, see "Kernel Distribution" (Statistics and Machine Learning Toolbox).

- **Overlay linear fit** Plot the best-fit line on the scatter subplots. You can choose to plot the best-fit lines for one, all, a row, or a column of scatter plots.
- Enable brushing/data selection Enable selection of data points in the scatter subplots.



When you highlight parameter values in one scatter subplot, the values corresponding to other parameters from the same row in the evaluated results table are also highlighted. In addition, the rows in the evaluated results table that correspond to these values are highlighted.



Delay	Gain	Tc	Тр	SignalMatching
0.0049	0.8029	14.0276	0.0099	644.8298
0.0046	0.8710	15.6653	0.0103	
0.0051	0.8220	14.9205	0.0105	419.7438
0.0050	0.8732	13.9582	0.0097	
0.0052	0.7585	14.5234	0.0103	1.2900e+03
0.0052	0.8282	15.3222	0.0098	365.7245
0.0051	0.7663	14.0752	0.0107	
0.0045	0.8275	15.7153	0.0107	364.4182
0.0046	0.8312	14.2285	0.0095	
0.0048	0.7309	16.2523	0.0102	1.8083e+03
0.0050	0.7608	14.3072	0.0102	1.2555e+03
0.0052	0.7558	15.7965	0.0101	1.3255e+03
0.0049	0.8269	14.0660	0.0107	
0.0053	0.8551	14.3625	0.0095	185.9608
0.0052	0.7751	13.7733	0.0096	
0.0055	0.8449	15.2286	0.0092	242.5879
0.0050	0.8281	15.5501	0.0109	355.8483
0.0048	0.7211	15.1398	0.0103	2.0309e+03
0.0046	0.8163	14.7772	0.0100	487.9127
0.0051	0.7819	15.4333	0.0103	
0.0053	0.8666	15.4429	0.0101	114.2653
0.0049	0.7202	15.5371	0.0103	2.0458e+03
0.0046	0.7940	15.4074	0.0101	742.1060
0.0048	0.7879	16.3355	0.0104	811.8904
0.0017	0 7007	4 4 4 9 7 9	0.0400	750 5103

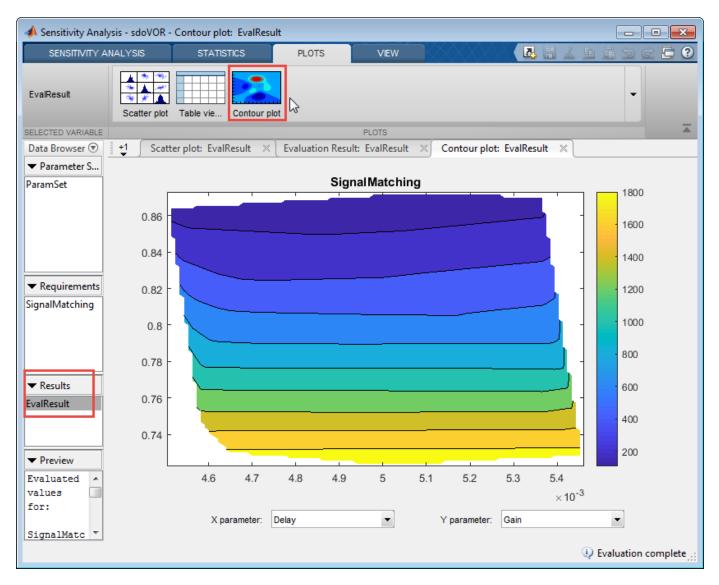
To remove the highlighting, invert the selection to all other data points in the plot, or disable the feature, right-click the highlighted data points, and choose from the context-menu.

• **Pop-out plot** — View a subplot in a new window.

Evaluated Result Contour Plots

After you have evaluated your design requirements, an evaluation results table lists the samples in the parameter set and the corresponding evaluated requirement (cost function) values. For information about evaluation, see "Evaluate Design Requirements" on page 4-60.

You can plot a contour plot of the evaluated results. To do so, select the evaluated result in the **Results** area of the app, and choose a contour plot in the **Plots** tab of the app.



Use this plot to visually analyze the relation between parameters and design requirements. Select the parameters to plot in the **X parameter** and **Y parameter** drop-down lists. The evaluated requirement value is plotted as a function of these parameters.

Statistical Analysis Tornado Plots

After you have evaluated the design requirements for each parameter, you can perform statistical analysis to analyze how the parameters of your Simulink model influence the requirements.

To generate a tornado plot ranking the influence of parameters on requirements:

1 In the **Statistics** tab of the app, select the evaluation results you want to analyze in the **Evaluation Results to Analyze** list.

📣 Sensitivity Analysis* - sdoVOI	R - Evaluation Result: Ev	alResult			
SENSITIVITY ANALYSIS	STATISTICS	PLOTS	VIEW	· .	
Evaluation Result to Analyze:	thod: Correlation		 Linear Ranked 		1
	Partial Correlatio	n	🗖 Kendall	Compute Statistics	
		S & TYPES		CALCULATE	<u> </u>

2 Specify the statistical analysis methods.

You can choose to calculate a correlation coefficient, standardized regression coefficient, and partial correlation coefficient (requires Statistics and Machine Learning Toolbox software).

For more information, see "Analyze Relation Between Parameters and Design Requirements" on page 4-67.

3 For each of these methods, specify what data to use for the analysis. You can choose from **Linear** (Pearson), **Ranked** (Spearman), and **Kendall** analysis types. **Kendall** is applicable when the analysis method is **Correlation**, and requires Statistics and Machine Learning Toolbox software.

You can compute all applicable combinations of analysis methods and types.

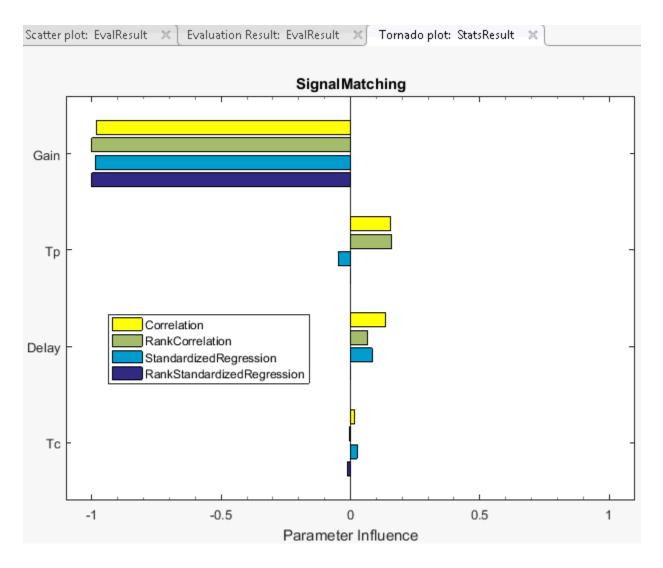
4 Calculate the coefficients, and generate a tornado plot.



The resulting tornado plot displays the calculated coefficients for each specified analysis method and type. The coefficients are plotted in order of influence of parameters on the cost function. The parameter with the greatest magnitude of influence on the cost function is displayed on the top, giving the plot a tornado shape. When more than one type of coefficient is calculated, the tornado plot sorts the parameters based on the first calculated coefficient. The coefficients are calculated in the following order:

- Correlation
- Rank correlation
- Kendall correlation
- Partial correlation
- Rank partial correlation
- Standardized regression
- Rank Standardized Regression

In this tornado plot, the parameters are sorted based on the Correlation coefficient. For all calculated coefficients, the Gain parameter has the most influence on the design requirement cost function.



See Also

Related Examples

- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Specify Time-Domain Requirements" on page 4-21
- "Specify Frequency-Domain Requirements" on page 4-48
- "Evaluate Design Requirements" on page 4-60
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67

Validate Sensitivity Analysis

You can validate sensitivity analysis by checking generated parameter values, evaluation results, and analysis results.

Inspect the Generated Parameter Set

To perform sensitivity analysis, you select model parameters for evaluation, and generate a representative set of parameter values to explore the design space. You create the parameter set by specifying parameter distributions such as normal or uniform. You can also specify correlations between parameters. For more information, see "Generate Parameter Samples for Sensitivity Analysis" on page 4-8. After generating the parameter values, plot them to check if generated parameter values match the desired specifications. This is particularly important if you generate a small number of random samples for each parameter set.

If you see a discrepancy in the generated parameters and the specified distribution and correlations, you can try one of the following:

- Generate the random samples again, until you achieve the specified distributions and correlations.
- Increase the sample size at the expense of increasing the evaluation time.
- Specify different sampling methods. Use Latin hypercube sampling method for a more systematic space-filling approach than random sampling. If you have Statistics and Machine Learning Toolbox software, use the Sobol and Halton quasirandom sampling methods for a more space-filling approach than the Latin hypercube method.

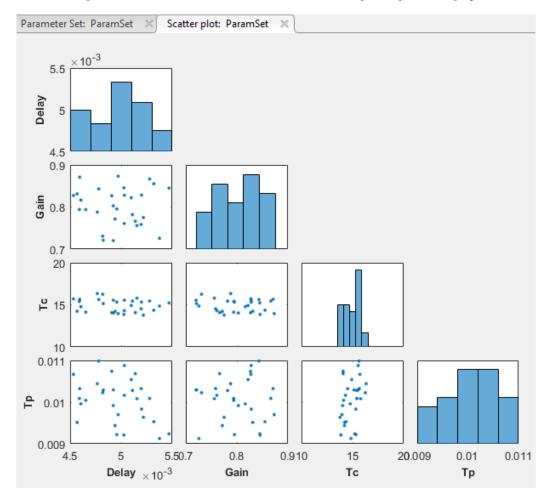
To plot the generated parameters in the **Sensitivity Analyzer**:

1 Select the generated parameter set in the **Parameter Sets** area of the app.

Sensitivity Analysis						
SENSITIVITY ANALY	SIS STATIS	TICS PLOTS	PARAM	ETER SET VI		1 i d d 🗗 🕐 🤅
ParamSet	Scatter plot	e ditor				-
SELECTED VARIABLE	6			PLOTS		
Data Browser		Parameter	Set: ParamSet	×		
 Parameter Sets 						
ParamSet		Delay	Gair	١	Tc	Тр
			0.0049	0.8029	14.0276	0.0099 🔺
			0.0046	0.8710	15.6653	0.0103
			0.0051	0.8220	14.9205	0.0105
			0.0050	0.8732	13.9582	0.0097
		-	0.0052	0.7585	14.5234	0.0103
 Requirements 			0.0052	0.8282	15.3222	0.0098
SignalMatching			0.0051	0.7663	14.0752	0.0107
			0.0045		15.7153	0.0107
			0.00			0.0000

2 In the **Plots** tab, select **Scatter Plot**.

The generated plot displays histograms of generated values for each parameter on the diagonal, and the pairwise scatter plot of the parameters on the off-diagonals. For more information about the scatter plot, see "Interact with Plots in the Sensitivity Analyzer" on page 4-79.



3 Inspect the histograms to ensure that the generated parameter values match the intended parameter distributions. Inspect the off-diagonal scatter plots to ensure that any specified correlations between parameters are present.

To plot the generated parameter values at the command line, use sdo.scatterPlot. Use functions such as mean to check the sample statistics.

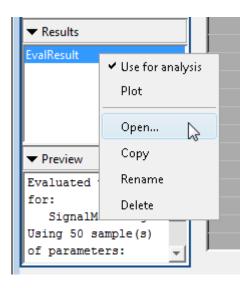
Check Evaluation Results

After generating a parameter set, you define a cost function by creating a design requirement on the model signals. You then evaluate the cost function at each set of parameter values on page 4-60. To validate the evaluation results, inspect the evaluated cost function values. If the cost function evaluations contain NaN values, that could indicate an issue.

To check for NaN values in the **Sensitivity Analyzer** after the evaluation is complete:

1 Open the evaluation results table if it is not already open.

In the **Results** area of the app, right-click the evaluated result, and select **Open** in the menu.



In the **Evaluation Results** table, each row of the table lists the parameter set values and the corresponding evaluated design requirement cost function values.

2 Sort the evaluated requirement values in descending order. To do so, click twice on the evaluated requirement column. Any NaN values are listed at the top of the evaluated requirement column.

Data Browser 💿	t2 Sc:	atter plot: EvalF	lesult 🗶 Ev	valuation Res	ult: EvalResult 🛛 🗶
▼ Parameter Sets	Select ar	id right-click a rov	v to create an in	itial guess for F	Response Optimization or
ParamSet	Delay	Gain	Tc	Тр	SignalMatching 🛐
	0.0	053 0.773	9 14.2199	0.0106	NaN
	0.0	052 0.721	9 14.0517	0.0106	2.0227e+03
	0.0	049 0.728	6 15.3662	0.0106	1.8622e+03
	0.0	049 0.732	1 16.0591	0.0109	1.7803e+03
 Requirements 	0.0	051 0.739	0 13.9969	0.0100	1.6656e+03
SignalMatching	0.0	050 0.740	8 13.7279	0.0096	1.6389e+03
	0.0	053 0.743	9 13.7345	0.0102	1.5719e+03
	0.0	054 0.742	2 16.4884	0.0109	1.5718e+03
	0.0	054 0.746	0 15.0856	0.0109	1.5100e+03
	0.0	052 0.751	5 13.7533	0.0103	1.4276e+03
✓ Results	0.0	055 0.755	8 15.7445	0.0092	1.3312e+03
EvalResult	0.0	054 0.759	0 15.8247	0.0090	1.2768e+03
	0.0	055 0.760	8 13.7515	0.0098	1.2625e+03
	0.0	049 0.769		0.0105	1.2598e+03
	Q.O.	055			17474

3 Inspect the parameter values that resulted in the NaN values for evaluated requirements. If you do not expect an NaN result for that row of parameter values, investigate your model further.

To view the evaluated results at the command line, inspect the cost function evaluation output of sdo.evaluate.

Perform Sensitivity Analysis with Different Parameter Set

After evaluation, you analyze the effect of the parameters on the design requirements, and identify most influential parameters. For more information, see "Analyze Relation Between Parameters and Design Requirements" on page 4-67. To validate the analysis results, generate a different parameter set and reevaluate the design requirements. If the analysis results are not consistent, consider increasing the number of samples in your parameter set.

See Also

Related Examples

- "What is Sensitivity Analysis?" on page 4-2
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67

Store Intermediate Data in the App

This topic shows how to speed up successive evaluations of design requirements in the **Sensitivity Analyzer** by storing intermediate data during evaluation. Use this option when memory usage is not a concern. Choosing this option results in a memory-time trade-off.

To store intermediate data during evaluation, in the **Sensitivity Analysis** tab, click **Store Intermediate Data**.

📣 Sensitiv	ity Analysis -	sdoVOR								
SENSI	TIVITY ANALY	sis	STATISTICS	PLOTS		VIEW			Z 8 .	k B E D
Open Session -	Save Session -	Select Parameters	Generate	'arameter Set to Evaluate: No Parameter Set Define 	_	New Requirement		Store Intermediate Data	Evaluate Model	Optimize
						REQU	JIREMENTS	OPTIONS	EVALUATE	

After creating a parameter set and specifying design requirements, you can evaluate your model. When you click **Evaluate Model**, the app saves the logged signals that are used to compute the design requirements at each combination of parameter values in your parameter set. The logged signal values are stored in the RAM. After the evaluation is complete, you can view the number of logged entries by hovering over **Store Intermediate Data**.

												_ 8	×
N									16 B	É 9	¢ E	?	≖
	Ŵ	🖾 Storę Intermediate Data											
New Requirement 🔻		el signals in a log, for faster	Eveluate r re-evaluat		ame parameter set.	The log currently	/ has 50 ei	ntries.	To clear	the log,	unchec	k this b	20X,
REQUIRE		OPTIONS	EVALUATE	OPTIMIZE									
					<u></u>								

If you now evaluate your model for a different design requirement that uses the same logged signals and parameter set values, the app uses the stored signal values. As a result, successive evaluations take less time.

If your system is running out of memory during evaluation, stop the evaluation, and clear **Store Intermediate Data**. Doing so will delete the stored data from memory.

See Also

Related Examples

- "Specify Parameters for Design Exploration" on page 4-4
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Specify Time-Domain Requirements" on page 4-21
- "Specify Frequency-Domain Requirements" on page 4-48

- "Evaluate Design Requirements" on page 4-60
- "Ways to Speed Up Design Optimization Tasks"

Specify Steady-State Operating Point for Sensitivity Analysis

What is a Steady-State Operating Point?

An *operating point* of a dynamic system defines the states and root-level input signals of the model at a specific time. For example, in a car engine model, variables such as engine speed, throttle angle, engine temperature, and surrounding atmospheric conditions typically describe the operating point.

A *steady-state operating point* of a model, also called an equilibrium or *trim* condition, includes state variables that do not change with time.

A model can have several steady-state operating points. For example, a hanging damped pendulum has two steady-state operating points at which the pendulum position does not change with time. A *stable steady-state operating point* occurs when a pendulum hangs straight down. When the pendulum position deviates slightly, the pendulum always returns to equilibrium. In other words, small changes in the operating point do not cause the system to leave the region of good approximation around the equilibrium value.

When using optimization search to compute operating points for nonlinear systems, your initial guesses for the states and input levels must be near the desired operating point to ensure convergence.

When linearizing a model with multiple steady-state operating points, it is important to have the right operating point. For example, linearizing a pendulum model around the stable steady-state operating point produces a stable linear model, whereas linearizing around the unstable steady-state operating point produces an unstable linear model.

For more information on operating points, see "What Is an Operating Point?" (Simulink Control Design) and "What Is a Steady-State Operating Point?" (Simulink Control Design).

Setting up a Steady-State Operating Point

This topic shows how to setup a steady-state operating point in the **Sensitivity Analyzer**. To improve the fit between the model and measured data, the model must be set to steady-state when parameters are evaluated.

- **1** Open the **Sensitivity Analyzer** and setup your requirements using the steps outlined in "Identify Key Parameters for Estimation (GUI)" on page 4-131.
- 2 In the toolstrip, click **Options** and select **Operating Point Options** from the drop down menu.



3 The following **Operating Point** dialog box opens.



The **Evaluate at steady-state** option is checked by default when you open the operating point dialog. Select the appropriate requirement to change the parameters for from the Requirement: drop down menu. Use the **States**, **Inputs** and **Outputs** tabs to specify the known parameters, bounds and deviations. For instance, there is one state in the above figure. Use the operating point dialog to specify that this state should be treated as an unknown, and it should be set to steady state. During sensitivity analysis, the operating point computation will vary this state to set it at steady-state.



You can also sync operating point specifications from your Simulink model or another requirement using the Sync with specification from: drop-down list. After you make your

selection, click on the selection to copy the parameters.

4 The Simulink Design Optimization software uses optimization methods to search for operating points in a model. Use the **Options** tab of the dialog to specify these optimization methods. These options specify the optimization algorithm, tolerances, and stopping conditions. For instance, the option **Gradient descent with projection** is often used to find the operating point for systems that use physical modeling. For more information, click on the methods.

 Sync with specification 	lon Smith Model	
ant descant with elimination 🤍 🗠	Algorithms: Active-Set.	
	Maximum has evalu-	2000
	Mainter Instance	400
14-06	Pasendarbóeance	14-05
14-04		
or	-	
hardfox.		
	14-06 74-08	la Mathema Kanada Mathema Kanada wati Mathema Kanada Mathema Kanada Mathem

5 Having specified the operating point parameters, continue with the sensitivity analysis workflow as described in "Identify Key Parameters for Estimation (GUI)" on page 4-131.

See Also

More About

- "What Is an Operating Point?" (Simulink Control Design)
- "What Is a Steady-State Operating Point?" (Simulink Control Design)
- "Set Model to Steady-State When Estimating Parameters (Code)" on page 2-97
- "Set Model to Steady-State When Estimating Parameters (GUI)" on page 2-117

Use Parallel Computing for Sensitivity Analysis

Configure Your System for Parallel Computing

To perform global sensitivity analysis, you sample the model parameters and states, define a cost function by creating a design requirement on the model signals, and evaluate the cost function for each sample. Evaluating the model for many samples can be time consuming. You can speed up evaluation in the **Sensitivity Analyzer**, or at the command line, using parallel computing on multicore processors or multiprocessor networks.

When you evaluate the cost function with the parallel computing option enabled, the software uses the available parallel pool. If none is available, and **Automatically create a parallel pool** is selected in your Parallel Computing Toolbox preferences, the software starts a parallel pool using the settings in those preferences. To open a parallel pool that uses a specific cluster profile, use:

parpool(MyProfile);

MyProfile is the name of a cluster profile.

For information regarding creating a cluster profile, see "Add and Modify Cluster Profiles" (Parallel Computing Toolbox).

Model Dependencies

Model dependencies are any referenced models, data such as model variables, S-functions, or additional files necessary to run the model. Before starting the parallel model evaluation, verify that the model dependencies are complete. Otherwise, you may get unexpected results.

Making Model Dependencies Accessible to Remote Workers

When you use parallel computing, the Simulink Design Optimization software helps you identify model dependencies. To do so, the software uses the Dependency Analyzer. The dependency analysis may not find all the files required by your model. To learn more, see "Dependency Analyzer Scope and Limitations". If your model has dependencies that are undetected or inaccessible by the parallel pool workers, then add them to the list of model dependencies.

The dependencies are made accessible to the parallel pool workers by specifying one of the following:

- File dependencies: the model dependency files are copied to the parallel pool workers.
- Path dependencies: the paths to the model dependencies are added to the paths of the parallel pool workers. If you are working in a multi-platform scenario, ensure that the paths are compatible across platforms.

Using file dependencies is recommended, however, in some cases it can be better to choose path dependencies. For example, if parallel computing is set up on a local multi-core computer, using path dependencies is preferred as using file dependencies creates multiple copies of the dependent files on the local computer.

Perform Sensitivity Analysis Using Parallel Computing (GUI)

To perform sensitivity analysis using parallel computing in the **Sensitivity Analyzer**:

1 Ensure that the software can access parallel pool workers that use the appropriate cluster profile.

For more information, see "Configure Your System for Parallel Computing" on page 4-104.

- 2 Open the **Sensitivity Analyzer** for the Simulink model.
- **3** Specify the parameter set, generate parameter samples, and specify the requirements for sensitivity analysis. For example, see "Design Exploration Using Parameter Sampling (GUI)" on page 4-112 and "Identify Key Parameters for Estimation (GUI)" on page 4-131.
- 4
- On the **Sensitivity Analysis** tab, click **Options** to open the **Evaluation Options** dialog box.
- 5 Select the **Parallel Options** tab.

Evaluation Options			×
General Options	Parallel Options	Linearization Options	
Use the paralle	el pool during eva	aluation	
Model file depen	dencies Model	path dependencies	
	ependencies spec		
		2	
	A	Add file dependency Sync file dependencies from mode	9
<u> </u>			
		OK Cancel Help	

6 Select the Use the parallel pool during evaluation check box.

This option checks for dependencies in your Simulink model. The file dependencies are displayed in the **Model file dependencies** list box, and corresponding path to the files in **Model path dependencies**. The files listed in **Model file dependencies** are copied to the remote workers.

Note The automatic dependencies check may not detect all the dependencies in your model.

For more information, see "Model Dependencies" on page 4-104. In this case, add the undetected dependencies manually.

Evaluation Options	×
General Options Parallel Options Linearization Options	
✓ Use the parallel pool during evaluation	
Model file dependencies Model path dependencies	
C:\matlab\toolbox\parallelModelDependecies\file.mat	
Add file dependency	Sync file dependencies from model
OK Cancel I	Help

7 Add any file dependencies that the automatic check does not detect.

Specify the files in the **Model file dependencies** list box separated by semicolons or on separate lines.

Evaluation Options	х
General Options Parallel Options Linearization Options	
✓ Use the parallel pool during evaluation	
Model file dependencies Model path dependencies	
C:\matlab\toolbox\parallelModelDependecies\file.mat; C:\myproject\file2.mat C:\matlab\work\file3.mat	
Add file dependency Sync file dependencies from model	
OK Cancel Help	

Alternatively, click Add file dependency to open a dialog box, and select the file to add.

Note If you do not want to copy the files to the remote workers, delete all entries in the **Model file dependencies** list box. Populate the **Model path dependencies** list box by clicking the **Sync path dependencies from model**, and add any undetected path dependencies. In addition, in the list box, update the paths on local drives to make them accessible to remote workers. For example, change C:\ to \\\hostname\\C\$\\.

8 If you modify the Simulink model, resync the dependencies to ensure that any new dependencies are detected. Click **Sync file dependencies from model** in the **Parallel Options** tab to rerun the automatic dependency check for your model.

This action updates the **Model file dependencies** list box with any new file dependency found in the model.

- 9 Click OK.
- **10** In the **Sensitivity Analyzer**, click **Evaluate** to perform sensitivity analysis using parallel computing. The design requirements are evaluated for each combination of parameter values in your parameter set.

This functionality is not supported in Simulink Online.

Perform Sensitivity Analysis Using Parallel Computing (Code)

To evaluate a model using parallel computing:

1 Ensure that the software can access parallel pool workers that use the appropriate cluster profile.

For more information, see "Configure Your System for Parallel Computing" on page 4-104.

- **2** Open the model.
- **3** Specify the cost function and generate parameter samples for sensitivity analysis. For example, see "Design Exploration Using Parameter Sampling (Code)" on page 4-157.
- **4** Enable parallel computing using an evaluation option set.

```
opt = sdo.EvaluateOptions;
opt.UseParallel = true;
```

5 Find the model dependencies.

```
[dirs,files] = sdo.getModelDependencies(modelname)
```

Note sdo.getModelDependencies may not detect all the dependencies in your model. For more information, see "Model Dependencies" on page 4-104. In this case, add the undetected dependencies manually.

6 Modify files to include any file dependencies that sdo.getModelDependencies does not detect.

```
files = vertcat(files, 'C:\matlab\work\filename.m')
```

Note If you do not want to copy the files to the remote workers, add any undetected path dependencies to dirs and update the paths on local drives to make them accessible to remote workers. See sdo.getModelDependencies for more details.

7 Add the file dependencies for evaluation.

```
opt.ParallelFileDependencies = files;
```

8 Specify the name of the model to be evaluated in parallel.

opt.EvaluatedModel = modelname;

9 Evaluate the model.

[pOpt,opt_info] = sdo.evaluate(fcn,samples,opt);

Troubleshooting

Why Don't I See the Evaluation Speed up I Expected Using Parallel Computing?

When you evaluate a model that does not require a large number of evaluations or does not take long to simulate, you might not see a speedup in the evaluation time. In such cases, the overhead associated with creating and distributing the parallel tasks outweighs the benefits of running the evaluation in parallel.

See Also

sdo.evaluate | sdo.EvaluateOptions | sdo.getModelDependencies | parpool

More About

- "Ways to Speed Up Design Optimization Tasks"
- "Store Intermediate Data in the App" on page 4-100

Use Fast Restart Mode During Sensitivity Analysis

This topic shows how to speed up sensitivity analysis using Simulink fast restart. You can use the fast restart feature to speed up sensitivity analysis of tunable parameters of a model.

Fast restart enables you to perform iterative simulations without compiling a model or terminating the simulation each time. Using fast restart, you compile a model only once. You can then tune parameters and simulate the model again without spending time on compiling. Fast restart associates multiple simulation phases with a single compile phase to make iterative simulations more efficient. You see a speedup of design optimization tasks using fast restart in models that have a long compilation phase. See "How Fast Restart Improves Iterative Simulations".

When you enable fast restart, you can only change tunable properties of the model during simulation. For more information about the limitations, see "Limitations".

You can perform sensitivity analysis using fast restart in the **Sensitivity Analyzer** or at the command line on page 4-109.

Sensitivity Analyzer Workflow for Fast Restart

To evaluate a model using fast restart in the **Sensitivity Analyzer**:

- **1** Open the Simulink model.
- **2** Enable fast restart in the model.

Click Fast Restart window.

- **3** Open the **Sensitivity Analyzer** for the model.
- 4 Specify the parameter set, generate parameter samples, and specify the requirements for sensitivity analysis. Optionally, specify evaluation settings. For example, see "Design Exploration Using Parameter Sampling (GUI)" on page 4-112 and "Identify Key Parameters for Estimation (Code)" on page 4-169.
- **5** Click **Evaluate** to evaluate the model in fast restart mode. The design requirements are evaluated for each combination of parameter values in your parameter set.
- **6** Disable fast restart.

In the model window, click Fast Restart

This functionality is not supported in Simulink Online.

Command-Line Workflow for Fast Restart

You can use sensitivity analysis to evaluate which model parameters most influence a cost function. You can use these parameters during parameter estimation or response optimization. Suppose that you want to use sensitivity analysis to reduce the number of parameters that you need to estimate to fit a model.

To evaluate the model in fast restart mode using a cost function aimed at parameter estimation:

- **1** Open the Simulink model.
- 2 Specify the model parameter values, params, to estimate and generate parameter samples, params_samples. For an example, see "Identify Key Parameters for Estimation (Code)" on page 4-169.
- **3** Create an experiment object, Exp.

```
Exp = sdo.Experiment('model');
```

Store the measured input-output data in Exp. For an example, see "Identify Key Parameters for Estimation (Code)" on page 4-169.

4 Create a model simulator from the experiment.

```
Simulator = createSimulator(Exp);
```

Simulator is an sdo.SimulationTest object.

Note You must create a simulation scenario with logging information before configuring the model for fast restart. You cannot modify logging information once the model has been compiled for fast restart.

5 Configure the model and simulator for fast restart.

Simulator = fastRestart(Simulator, 'on');

6 Create a cost function, myCostfcn, and pass Simulator to the cost function as an input. For more information, see "Write a Cost Function" on page 2-49. In the cost function, the simulator configured for fast restart is used to update the model parameters, simulate the model, and log signals.

Use an anonymous function with one argument that calls myCostfcn.

evalfcn = @(param) myCostfcn(param,Simulator,Exp);

7 Evaluate the model.

[param_opt,opt_info] = sdo.evaluate(evalfcn,param,param_samples);

8 Restore the simulator fast restart settings.

```
Simulator = fastRestart(Simulator, 'off');
```

The fast restart workflow is similar for sensitivity analysis that identifies design variables using a cost function aimed at response optimization. See "Use Fast Restart Mode During Response Optimization" on page 3-189.

Troubleshooting

Why Don't I See the Evaluation Speedup I Expected Using Fast Restart?

You see a speedup of design optimization tasks using fast restart in models that have a long compilation phase. If the compilation phase of your model is not long, you do not see a significant change in estimation speed.

See Also

sdo.SimulationTest|sdo.evaluate|fastRestart

More About

- "Ways to Speed Up Design Optimization Tasks"
- "Store Intermediate Data in the App" on page 4-100

Design Exploration Using Parameter Sampling (GUI)

This example shows how to sample and explore a design space using the **Sensitivity Analyzer**. You explore the design of a Continuously Stirred Tank Reactor (CSTR) to minimize product concentration variation and production cost. The design must also account for the uncertainty in the temperature and concentration of the input feed to the reactor.

You explore the CSTR design by characterizing model parameters using probability distributions. You use the distributions to generate random samples and perform Monte-Carlo evaluation of the design at these sample points. You then create plots to visualize the design space and select the best design. You then use the best design as an initial guess for optimization of the design.

Continuously Stirred Tank Reactor (CSTR) Model

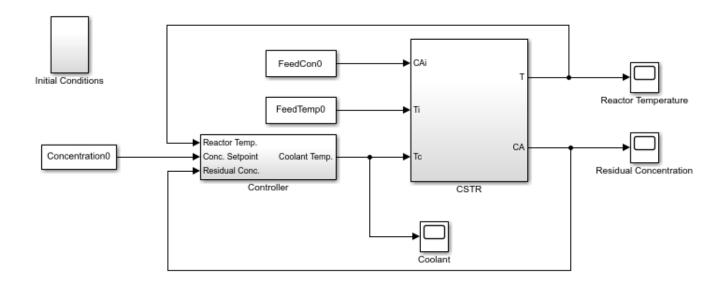
Continuously Stirred Tank Reactors (CSTRs) are common in the process industry. The Simulink® model, sdoCSTR, models a jacketed diabatic (i.e., non-adiabatic) tank reactor described in [1]. The CSTR is assumed to be perfectly mixed, with a single first-order exothermic and irreversible reaction, $A \rightarrow B$. A, the reactant, is converted to B, the product.

In this example, you use the following two-state CSTR model, which uses basic accounting and energy conservation principles:

$$\begin{aligned} \frac{dC_A}{dt} &= \frac{F}{A*h}(C_{feed} - C_A) - r*C_A\\ \frac{dT}{dt} &= \frac{F}{A*h}(T_{feed} - T) - \frac{H}{c_p\rho}r - \frac{U}{c_p*\rho*h}(T - T_{cool})\\ r &= k_0*e^{\frac{-E}{R*T}} \end{aligned}$$

- C_A, and C_{feed} Concentrations of A in the CSTR and in the feed [kgmol/m³]
- T, T_{feed} , and T_{cool} CSTR, feed, and coolant temperatures [K]
- F and ρ Volumetric flow rate [m^3/h] and the density of the material in the CSTR [1/m^3]
- *h* and *A* Height [m] and heated cross-sectional area [m²] of the CSTR.
- k_0 Pre-exponential non-thermal factor for reaction $A \rightarrow B$ [1/h]
- E and H Activation energy and heat of reaction for A
 ightarrow B [kcal/kgmol]
- *R* Boltzmann's gas constant [kcal/(kgmol * K)]
- G_P and U Heat capacity [kcal/K] and heat transfer coefficients [kcal/(m² * K * h)] Open the Simulink model.

open_system('sdoCSTR');



Copyright 2012-2015 The MathWorks, Inc.

CSTR Design Problem

Assume that the CSTR is cylindrical, with the coolant applied to the base of the cylinder. Tune the CSTR cross-sectional area, A, and CSTR height, h, to meet the following design goals:

- Minimize the variation in residual concentration, C_A . Variations in the residual concentration negatively affect the quality of the CSTR product. Minimizing the variations also improves CSTR profit.
- Minimize the mean coolant temperature T_{cool} . Heating or cooling the jacket coolant is expensive. Minimizing the mean coolant temperature improves CSTR profit.

The quality of the feed to the reactor can differ amongst suppliers. Thus, the design must allow for variations in the supply feed concentration, C_{feed} , and feed temperature, T_{feed} . The quality of the feed differs from supplier to supplier and also varies within each supply batch.

Specify Design Variables

Open the **Sensitivity Analyzer**. In the Simulink model from the **Apps** tab, click **Sensitivity Analyzer** under **Control Systems**. The app opens with an empty Sensitivity Analysis session.

SENSITIVITY ANALY	(SIS STA	TISTICS	PLOTS	VIEW			3 6	<u>/ 665660</u> -
	=		ameter Set to Evaluate:		W	Store Intermediate Data		
Open Save Session ▼ Session ▼	Select Ger Parameters Val		Parameter Set Defined 👻	New Requirement 💌	Select for Evaluation	Options	Evaluate Model	Optimize
FILE		PARAMETER	SET	REQUIREM	ENTS	OPTIONS	EVALUATE	OPTIMIZE
Data Browser	(Ð						
▼ Parameter Sets								
 Requirements 								
▼ Results								
		_						
▼ Preview								
								.:

Create a parameter set that includes the CSTR design variables A and h and the feed variation parameters FeedConc0 and FeedTemp0. You randomly generate multiple values for these parameters to evaluate the CSTR design.

- In the **Sensitivity Analysis** tab, in the **Select Parameters** drop-down menu, select **New**.
- In the dialog box, select A, FeedCon0, FeedTemp0, and h.
- Click **OK**. An empty parameter set, **ParamSet** is created in the **Parameter Set** area of the app browser.

•	Variable	Current value	Used By
V	A	0.5	sdoCSTR/CSTR/F//(A*h) + U//(RhoCph) sdoCSTR/CSTR/F//V sdoCSTR/CSTR/F//V 2 and 4 more blocks
	Concentration0	2	sdoCSTR/CA Setpoint sdoCSTR/CSTR/CA dot sdoCSTR/Initial Conditions/ T0 sdoCSTR/Initial Conditions/DeltaH*(FeedCon0-Concentration0)
	DeltaH	-11.92	sdoCSTR/CSTR/H//(RhoCp) sdoCSTR/Initial Conditions/DeltaH*(FeedCon0-Concentration0)

Specify the parameter distributions and correlations. ParamSet will be populate with parameter values selected randomly from the specified distributions:

- Sample A from a uniform distribution with lower bound 0.2 m² and upper bound 2 m².
- Sample h from a uniform distribution with lower bound 0.5 m and upper bound 3 m.
- Sample FeedConc0 from a normal distribution with mean 10 kgmol/m^3 and standard deviation 0.5 kgmol/m^3.
- Sample FeedTemp0 from a normal distribution with mean 295 K and standard deviation 3 K.
- Specify FeedCon0 and FeedTemp0 as negatively correlated with covariance 0.6.

To generate 100 parameter values using the above distribution and correlation information, click **Generate Values**, and select **Generate Random Values**. For repeatability of the example reset the random number generator.

rng('default')

In the Generate Random Parameter Values dialog box, specify the following:

- Set the number of samples to 100
- For parameter A select **Uniform** distribution, set the lower bound to 0.2 and upper bound to 2.
- For parameter FeedCon0 select Normal distribution, set mu to 10 and sigma to 0.5, and check cross-correlated.
- For parameter FeedTemp0 select Normal distribution, set mu to 295 and sigma to 3, and check cross-correlated.
- For parameter h select **Uniform** distribution, set the lower bound to 0.5 and upper bound to 3.
- In the **Correlation Matrix** tab, set the FeedCon0, FeedTemp0 covariance to -0.6.
- Click **OK** to generate the parameter values.

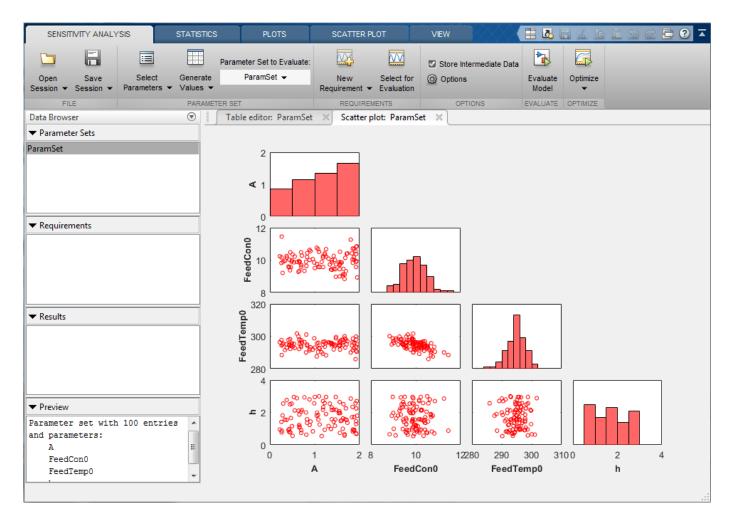
Number of Samples:	100				
Overwrite previous values in parameter set when generating new values					
Append to previous values in parameter set when generating new values					
Sampling Method:	Random				
Parameter	Distribution				Cross-Correlated
FeedCon0	Normal 🔻	mu: 10	sigma: 0.5	1	× .
FeedTemp0	Normal 🔻	mu: 295	sigma: 3		
h	Uniform 🔻	Lower: 0.5	Upper: 3		
	Correlation Matrix FeedCon0 FeedTem 1 -0.60 -0.6000				
				ОК	Apply Help

The ParamSet table is updated with the generated parameter values. Note that you can manually edit the generated parameter values in the ParamSet table.

To plot the parameter set click ParamSet in the **Parameter Sets** area of the app browser. In the **Plots** tab, select **Scatter Plot** in the plots gallery. The plot shows the histogram of the generated parameters on the diagonal and pair-wise parameter values on the off-diagonal.

Note that due to the random number generator the specific values in the plots and tables below may differ from what you get when running the example.

Each marker on the plot represents one row of the ParamSet table, with each row being simultaneously displayed on all the plots. The correlation between FeedCon0 and FeedTemp0 can be seen on the plot.



Specify Requirements for Evaluation

The CSTR design is required to minimize the variation in residual concentration and minimize the mean coolant temperature. Select **New Requirement** and click **Signal Property** to create a requirement to minimize the residual concentration variation.

In the **Create Requirement** dialog box, specify the following fields:

- In the **Property** drop-down list, select **Signal Variance**.
- In the **type** drop-down list, select **Minimize**.
- In the **Select Signals** area, select a logged signal to apply the requirement to. To do so click +. A Create Signal Set dialog box opens where you specify the logged signal. In the Simulink model, click the signal at the CA output of the CSTR block. The dialog box now displays this signal. Add the signal to the signal set and click **OK**.

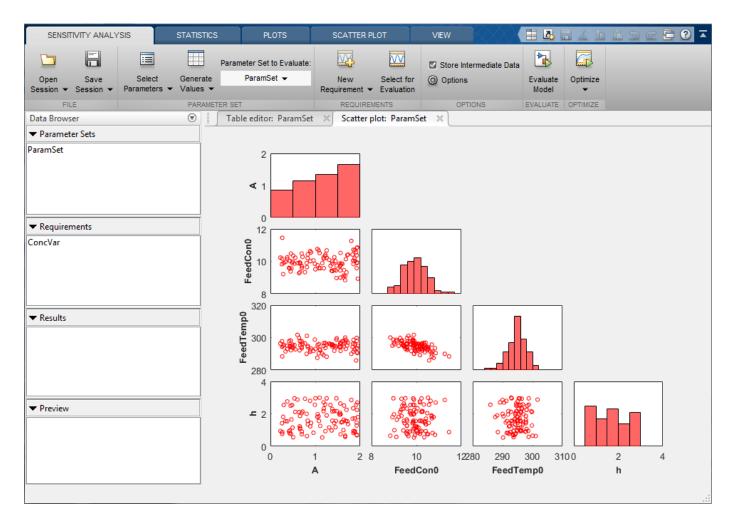
Create Signal Set		×
Signal set: Sig		
Currently selected signals	Signal Signal	
	OK Cancel	Help

Create Requ	irement	×
	Signal Property	
	Specify a requirement on a signal property such as the mear	i value.
Name: Sid	InalProperty	
 Specify 	/ Signal Property	
Property:	Signal variance \checkmark	Time-Weighted
Туре:	Minimize the property	~
Bound:	0	
▼ Select	Signals	
	Signal	
	ig (sdoCSTR/CSTR:2)	

• Close the Signal Property requirement dialog by clicking the x in the upper right dialog corner.

Close the **Create Requirement** dialog box. A new requirement, **SignalProperty** is listed in the **Requirements** area of the app browser

• Right-click SignalProperty, select **Rename**; rename the requirement to ConcVar.



Select **New Requirement** and click **Signal Property** to create a requirement to minimize the coolant mean (output of block sdoCSTR/Controller) temperature.

In the **Create Requirement** dialog box, specify the following fields:

- In the **Property** drop-down list, select **Signal Mean**.
- In the **Type** drop-down list, select **Minimize**.
- In the **Select Signals** area, add the sdoCSTR/Controller signal to the requirement.

Create Req	uirement	×
	Signal Property	
	Specify a requirement on a signal property such as the mean value.	
Name: Si	gnalProperty	
 Specif 	y Signal Property	
Property:	Signal mean V 🗌 Time-V	Neighted
Type:	Minimize the property	~
Bound:	0	
▼ Select	Signals	
	Signal	
	Sig (sdoCSTR/CSTR:2)	+
	Sig1 (sdoCSTR/Controller:1)	
		1
]

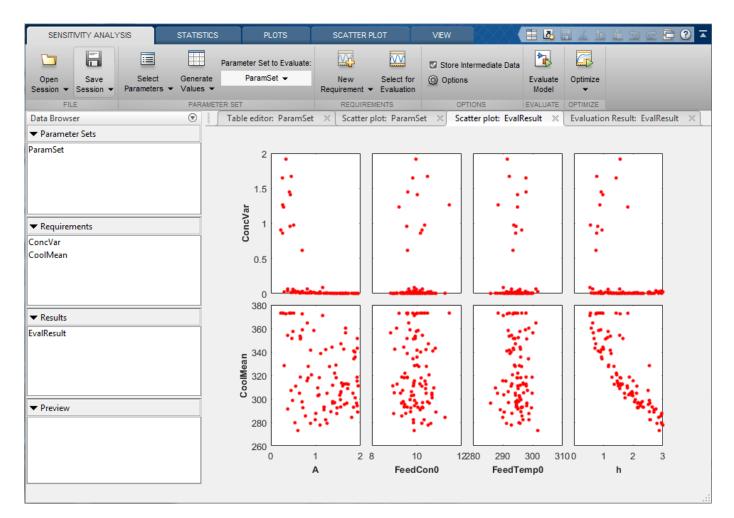
Close the **Create Requirement** dialog box. A new requirement, **SignalProperty** is created in the **Requirements** area of the app browser. Rename the requirement **CoolMean**.

Evaluate

In the **Sensitivity Analysis** tab, click **Select for Evaluation**. By default, all requirements are selected to be evaluated. Click **Evaluate Model** to evaluate the ConcVar and CoolMean requirements for each row of parameter values in ParamSet. Note you can speed up evaluation by using parallel computing if you have the Parallel Computing Toolbox (TM), or by using fast restart. For more information, see "Use Parallel Computing for Sensitivity Analysis" and "use Fast Restart Mode During Sensitivity Analysis" in the Simulink Design Optimization[™] documentation.

A results scatter plot showing each parameter vs each requirement is updated during model evaluation. At the end of evaluation a table with the evaluation results is created, each row in the evaluation result table contains values for A, FeedCon0, FeedTemp0, h and the resulting requirement values ConcVar and CoolMean. The evaluation results are stored in the EvalResult variable in the **Results** area of the app.

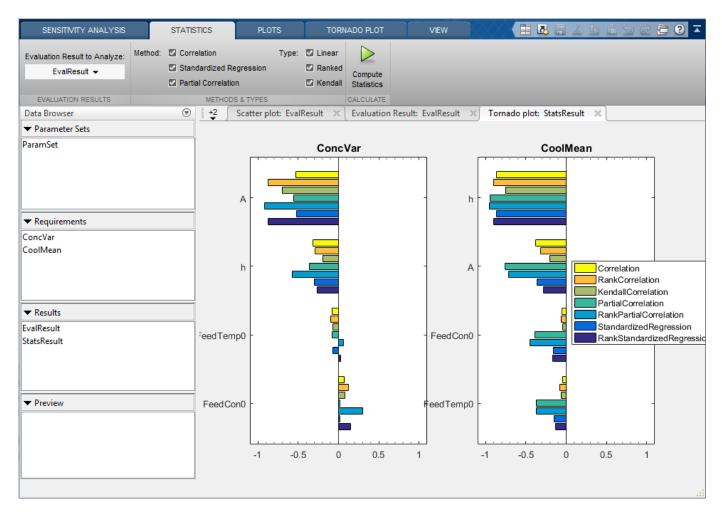
SENSITIVITY ANALY	SIS	STATIST	ics	PLOTS	VIEW				H 🕹 🗄	16 B	1 2 2 5 ?
	=		Para	meter Set to Eval	uate:	W	E Sto	re Intermediate Data			
Open Save	Select	Generat		ParamSet 👻	New	Select for	(a) 0p			otimize	
Session Session	Parameters .				Requirement		(O) Ob	uons	Model		
FILE		PARA	METER SE	ET	REQUIRE	MENTS		OPTIONS E		PTIMIZE	
Data Browser		$\overline{\mathbf{v}}$	Ta	ble editor: Para	mSet 🛛 Scatter	plot: ParamS	et X	Scatter plot: EvalRe	sult ≍∫ E	valuation	Result: EvalResult 🛛 🛛
 Parameter Sets 		(Sele	ct and right-click	a row to create an initi	al guess for Re	sponse O	ptimization or Paramet	er Estimation.		
'aramSet			A		FeedCon0	FeedTemp0)	h	ConcVar		CoolMean
				1.6665	9.1027		298.7040			0.0078	360.1810
				1.8304	10.4202		294.1778			0.0017	304.7684
				0.4286	9.5560		294.4233	0.8015	5		
				1.8441	10.0500		296.2681	2.3446	5		292.8644
 Requirements 				1.3382	9.7277		296.0506	2.5454	ļ		
ConcVar				0.3756	10.1518		295.0374	2.7658	3		291.3596
CoolMean				0.7013	9.6998		297.5851	0.9037			358.7545
				1.1844	10.2450		294.7875	2.1729)		292.9212
				1.9235	10.3697	:	291.0346	1.4615	5		315.1435
				1.9368	10.8559		290.4815	2.6991			288.7562
 Results 				0.4837	9.9029		294.8352	2.5439			298.6608
valResult				1.9471	8.9308		295.0001	1.7965	5		311.0358
				1.9229	9.5802	:	299.8118	1.2818	5		318.4055
				1.0737	10.6773		290.3048	1.8478	3		303.4633
				1.6405	9.4639		294.5618				295.2404
				0.4554	10.4805		292.1669		-		373
Preview				0.9592	10.0620		291.7999				358.2039
- Treview				1.8483	10.7183		291.3615		-		340.7548
				1.6260	9.0196		295.9212				342.2727
				1.9271	9.9024		298.0002				294.1091
				1.3803	9.3961		295.2950		-		297.6694
				0.2643	11 4540	l	288 4227	0 8747		1 2637	373



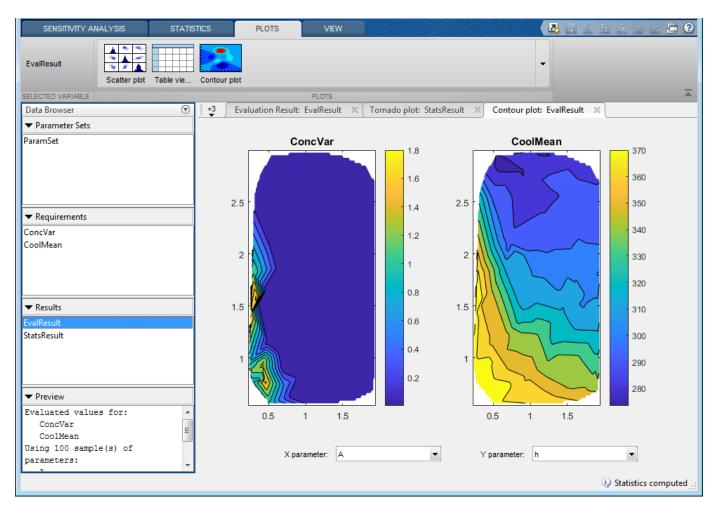
Analyze the Evaluation Results

The results scatter plot for EvalResult shows that CoolMean is inversely correlated with h (increasing h decreases CoolMean) and that low values of h can result in high values for ConcVar. The plot shows that low values of A can result in high values for ConcVar, but it is not clear from the plot how A is correlated with ConcVar or CoolMean or which parameter influences ConcVar the most. To investigate further, in the **Statistics** tab, select all the analysis methods and types and click **Compute Statistics**. This performs analysis on the evaluation results and creates a tornado plot. The tornado plot shows the influence of each parameter on each requirement:

- h is inversely correlated with CoolMean, and is the parameter that influences CoolMean the most.
- A is inversely correlated with CoolMean.
- FeedCon0 and FeedTemp0 are inversely correlated with CoolMean.
- A is inversely correlated with ConcVar, and is the parameter that influences ConcVar the most.
- h is inversely correlated with ConcVar.
- FeedCon0 and FeedTemp0 have mixed correlation with ConVar, but have minimal correlation with ConcVar.



The analysis shows that choosing a large h, to reduce CoolMean and choosing a large A to reduce CoolVar would appear to be a good design choice. You can confirm this by creating a contour plot of CoolMean and CoolVar versus h and A. Select EvalResult from the **Results** area of the app browser, and in the **Plots** tab, in the plots gallery click **Contour plot**. On the contour plot select h for the **Y parameter**, note that large h medium values of A give low values for both ConcVar and CoolMean.



Choose an Initial Guess for Optimization

Sort the Evaluation Result table by decreasing h, and select a row that has low ConcVar and CoolMean values. Right-click the selected row and click **Extract Parameter Values**. The extracted parameter values are saved in the ParamValues variable in the **Results** area of the app browser. These parameter values are used as the initial guess for optimization.

SENSITIVITY AN	IALYSIS	STATE	STICS	PLOTS	VIEW							É D	c 🗗 ?
			Par	rameter Set to Evalua	ite:	<u>vv</u>	🖾 Sto	re Intermediat	e Data				
Open Save Session - Session				ParamSet 👻	New Requirement 👻	Select for Evaluation	() Op	tions		Evaluate Model	Optimize		
FILE		PAF	RAMETER	SET	REQUIRE	MENTS		OPTIONS	E	EVALUATE	OPTIMIZE		
Data Browser			+ 3	Evaluation Resu	lt: EvalResult 🛛 🛛	Tornado pl	ot: Stats	Result 🛛 🕹	Conto	ur plot: Ev	alResult 🔅	×	
 Parameter Sets 			Sel	lect and right-click a	row to create an initia	al guess for Re	sponse Op	otimization or	Paramete	er Estimatio	n.		
ParamSet			A		eedCon0	FeedTemp0		h		ConcVar		CoolMea	an
				1.0971	10.5093	1	292.8640		2.9977		0.0228		277.6852
				1.0021	9.5104	1	293.0473		2.9602	2			
				1.4709	8.9741	1	298.7273		2.9576	5			
				1.2535	10.2275	2	295.7842		2.9567				
 Requirements 				0.5364	9.8572	1	295.4055		2.9541				
ConcVar				0.7708	9.3332	2	296.4149		2.9324	ł			
CoolMean				0.6029	9.8876	3	301.6883		2.8791		0.0268		273.0267
				1.8036	11.2630	2	289.9894		2.8568				
				0.8848	10.6303	1	291.2480		2.8008				
				0.3756	10.1518	2	295.0374		2.7658				291.3596
 Results 				1.1850	10.1538	1	289.1535		2.7047				290.7058
EvalResult			i 🔔	1.9368	10.8559	2	290.4815		2.6991				288.7562
ParamValues				1.5567	9.4412	2	294.5130		2.6388				293.1237
StatsResult				1.2221	10.3301	2	296.7435		2.5784				282.2222
				1.9104	10.5637	1	285.9125		2.5633				296.3316
				0.9897	9.8505		298.7508		2.5609	-			282.2279
Preview			ī —	1.3382	9.7277		296.0506		2.5454	-			289.0413
				0.2971	9.9012		295.5474		2.5444				328.2981
				0.4837	9.9029		294.8352		2.5439				298.6608
				1.6405	9.4639		294.5618		2.5084	-			295.2404
				0.4495	9.3714		298.2479		2.5008				307.1411
				1 8441	10 0500		296 2681		2 3446		0.0015		292 8644

Optimize

Use the data in the **Sensitivity Analyzer** to create an optimization problem to optimize A and h. In the **Sensitivity Analysis** tab click **Optimize**, and select **Create Response Optimization Session**. This opens a dialog to import data from Sensitivity Analysis to Response Optimizer.

- Select both the ConcVar and CoolMean requirements to import.
- Select ParamValues to import as design variables for optimization.
- Select EvalResult to import as uncertain variables to use during optimization.
- Click OK to import the data to the Response Optimizer

	ConcVar CoolMean	Minimize signal 'var'. Minimize signal 'mean'.
V (CoolMean	Minimize signal 'mean'
	meter values to import as o	-
Import	Parameter Values	Description
V	ParamValues	Values for - A, FeedCon0, FeedTemp0, h
elect parar	meter values to import as u	uncertain variables:
• •	Parameter Values	Description
Import		
Import E	EvalResult	100 values for - A, FeedCon0, FeedTemp0, h

		105144				
RESPONSE OPTIM	IZATION	VIEW		SSA		9250
	New 👻	S	Design Variables Set: 😰 ParamValues 👻 🧷		Data to Plot:	
Open Save Session ▼ Session ▼	Select	Evaluate Requirements	Uncertain Variables Set: 📉 None 👻 📿	Sensitivity Analysis 👻	 ☑ No data selected ▼ Add Plot Plot Model ▼ Response 	Options Opti
FILE	REQUI	REMENTS	VARIABLES		PLOTS	OPTIONS OPTIM
Data Browser		\odot				
Search workspace var	riables	- ۹				
▼ MATLAB Workspa	ce					
Name 🔺	Value					
🕂 ct	5					
{} err	1x5 cell					
▼ Model Workspace	(sdoCSTR)					
Name 🔺	Value					
A	0.5000					
Concentration0	2					
🛨 DeltaH	-11.9200					
E div R	5.9636e+03	T				
▼ Data						
Name 🔺	Value					
🗊 ConcVar	1x1 Signal					
Concentration	1x1 Signal					
CoolMean	1x1 Signal	_				
Coolant	1x1 Signal					
 Variable Preview 						

Configure the **Response Optimizer** to optimize the CSTR design:

• Click the pencil icon to edit the ParamValues design variable set, and remove the FeedCon0 and FeedTemp0 variables from the design variable set.

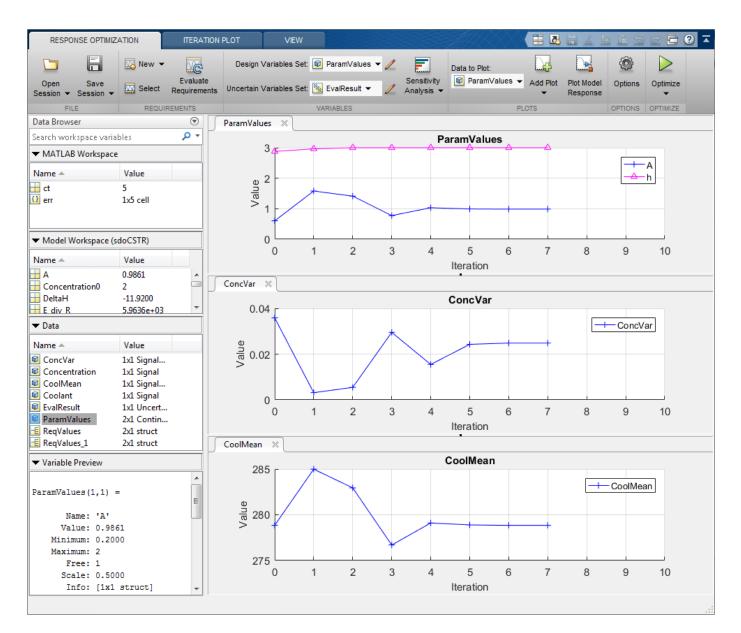
E	dit: Para	mValues					×
		Variable	Value	Minimum	Maximum	Scale	
	1	A	0.602861	-Inf	Inf	0.5	+
	1	h	2.879076	-Inf	Inf	2	
		ble Detail		Up	date model	variables	
	r _i vana	DIE Detaii					?

• Select EvalResult as the uncertain variable set, click the pencil icon to edit EvalResult, and remove A and h from the uncertain variable set.

Ed	it: Evall	Result			×
		Variable	Nominal Value	Uncertain Values	1
Γ	1	FeedCon0	10	[9.102660579272438;10.42018776	+
	1	FeedTemp0	295	[298.7040374406723;294.1777893	1
	M		nbinations only (
					?

Add iteration plots to see how the variables ParamValues (A and h), and optimization requirements ConcVar and CoolMean change during optimization.

- Select the variables in the Data to Plot drop-down list, and select Iteration Plot in the Add Plot drop-down list.
- Click **Optimize**.



The optimization minimizes CoolMean and ConcVar in the presence of varying FeedCon0 and FeedTemp0.

Related Examples

To learn how to explore the CSTR design space using the sdo.evaluate command, see "Design Exploration Using Parameter Sampling (Code)" on page 4-157.

References

[1] Bequette, B.W. *Process Dynamics: Modeling, Analysis and Simulation.* 1st ed. Upper Saddle River, NJ: Prentice Hall, 1998.

Close the model

bdclose('sdoCSTR')

See Also

More About

- "Design Exploration Using Parameter Sampling (Code)" on page 4-157
- "Use Parallel Computing for Sensitivity Analysis" on page 4-104
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67
- "Validate Sensitivity Analysis" on page 4-96

Identify Key Parameters for Estimation (GUI)

This example shows how to use sensitivity analysis to narrow down the number of parameters that you need to estimate when fitting a model. This example uses a model of the vestibulo-ocular reflex, which generates compensatory eye movements.

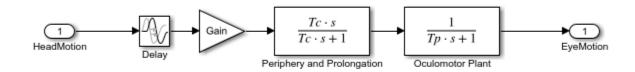
Model Description

The vestibulo-ocular reflex (VOR) enables the eyes to move at the same speed and in the opposite direction as the head, so that vision is not blurred when the head moves during normal activity. For example, if the head turns to the right, the eyes turn to the left at the same speed. This happens even in the dark. In fact, the VOR is most easily characterized by measurements in the dark, to ensure that eye movements are predominantly driven by the VOR.

Head rotation is sensed by organs in the inner ears, known as semicircular canals. These detect head motion and transmit signals about head motion to the brain, which sends motor commands to the eye muscles, so that eye movements compensate for head motion. We would like to use eye movement data to estimate model parameters for these various stages. The model we will use is shown below. There are four parameters in the model: Delay, Gain, Tc, and Tp.

open_system('sdoVOR')

Vestibulo-Ocular Reflex



Copyright 2013 The MathWorks, Inc.

The file sdoVOR_Data.mat contains uniformly sampled data of stimulation and eye movements. If the VOR were perfectly compensatory, then a plot of eye movement data, when flipped vertically, would overlay exactly on top of a plot of head motion data. Such a system would be described by a gain of 1 and a phase of 180 degrees. However, real eye movements are close, but not perfectly compensatory.

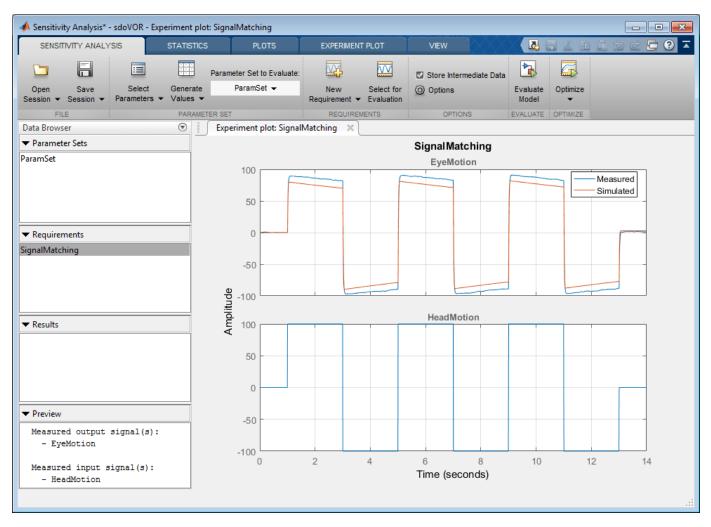
load sdoVOR_Data.mat; % Column vectors: Time HeadData EyeData

We will use Sensitivity Analysis UI to see how well the model output fits the data, and explore which model parameters have the most influence on the goodness of fit. To open Sensitivity Analysis UI, in the **Apps** tab, click **Sensitivity Analyzer** under **Control Systems** to launch the **Sensitivity Analyzer**.

To associate the data with the model, click **New Requirement** and select a Signal Matching requirement. This specifies an objective function consisting of the sum of squared error between the data and model output. In the Signal Matching dialog, specify the output as [Time EyeData], and specify the input as [Time HeadData].

Edit Signal Matching: SignalMatching 🛛 🛛 🗙
Outputs Specify measured output signals for this requirement. sdoVOR/Oculomotor Plant:1 (EyeMotion)
[Time EyeData] 🔹 🕁 🕹 🗴
Select Measured Output Signals
Inputs Optionally specify input signals for this requirement. sdoVOR/HeadMotion:1 (HeadMotion)
<1x1 Signal, 1 points> 🔹 🖶 🛃
Select Inputs
Initial States Optionally define initial states for this requirement.
There are currently no initial states defined for this requirement.
Select Initial States
🙀 Plot & Simulate 🛛 Plot 💉 OK 📀 Help

To view the eye movement data, navigate to the data browser on the left side of the UI, right-click the SignalMatching requirement, and select Plot & Simulate. The bottom plot shows the stimulation, consisting of a series of pulses. The top plot shows eye movement data, which resembles but does not exactly match the stimulation. It also shows that the model simulated output does not match the eye movement data, because model parameters need to be estimated.



Explore the Design Space

The model attempts to capture the phenomena which cause the difference between head movements and eye movements. Here we will explore the design space formed by the model parameters. To specify the parameters to explore in Sensitivity Analysis UI, click **Select Parameters** and create a new parameter set. Select all model parameters: Delay, Gain, Tc and Tp.

Select P	arameters				×
Filter	by variable na	ame			- م
•	Variable	Current va	Used By		
1	Delay	0.005	sdoVOR/Delay		
1	Gain	0.8	sdoVOR/Gain		
1	Tc	15	sdoVOR/Periphery and Prolongation		
	Тр	0.01	sdoVOR/Oculomotor Plant		
► Spe	cify expressio	on indexing if	necessary (e.g., a(3) or s.x.)		
		2		🔀 Cancel	(?) Help

Explore the design space by generating parameter values. Click **Generate Values** and select random values. For repeatability of the example, reset the random number generator.

rng('default')

Since there are 4 parameters, we will generate 40 samples.

The Delay parameter models the fact that there is some delay in communicating the signals from the inner ear to the brain and eyes. This delay is due to the time needed for chemical neurotransmitters to traverse the synaptic clefts between nerve cells. Based on the number of synapses in the vestibulo-ocular reflex, this delay is expected to be around 5 ms. We will model it with a uniform distribution with a lower bound of 2 ms and an upper bound of 9 ms.

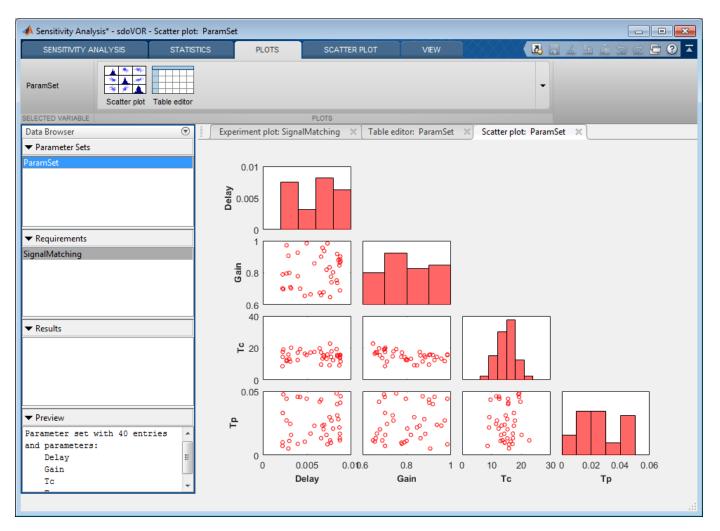
The Gain parameter models the fact that in the dark, the eyes do not move quite as much as the head. We will model it with a uniform distribution with a lower bound of 0.6 and an upper bound of 1.

The Tc parameter models the dynamics associated with the semicircular canals, as well as some additional neural processing. The canals are high-pass filters, because after a subject has been put into rotational motion, the neurally active membranes in the canals slowly relax back to resting position, so the canals stop sensing motion. Thus, after the stimulation undergoes transition edges, eye movements tend to depart from the stimulation over time. Based on mechanical characteristics of the canals, combined with additional neural processing which prolongs this time constant to improve the accuracy of the VOR, we will model Tc with a normal (i.e., bell curve) distribution with a mean of 15 seconds and a standard deviation of 3 seconds.

Finally, the Tp parameter models the dynamics of the oculomotor plant, i.e. the eye and the muscles and tissues attached to it. The plant can be modeled by two poles, however it is believed that the pole with the larger time constant is cancelled by precompensation in the brain, to enable the eye to make quick movements. Thus in the plot, when the stimulation undergoes transition edges, the eye movements follow with only a little delay. We will model Tp with a uniform distribution with a lower bound of 0.005 seconds and an upper bound of 0.05 seconds.

承 Generate Random	Parameter Values	- • •
Number of Samples:	40	
Overwrite previou	us values in parameter set when generating new values	
Append to previous	us values in parameter set when generating new values	
Sampling Method:	Random	
Parameter	Distribution	Cross-Correlated
Delay	Uniform V Lower: 0.002 Upper: 0.009	
Gain	Uniform V Lower: 0.6 Upper: 1	E
Tc	Normal 🔻 mu: 15 sigma: 3	— —
Distribution Plot	Correlation Matrix	
	Probability Distribution for Tc	
Probability Density		Mean: 15 rd Deviation: 3
	ОК	Apply Help

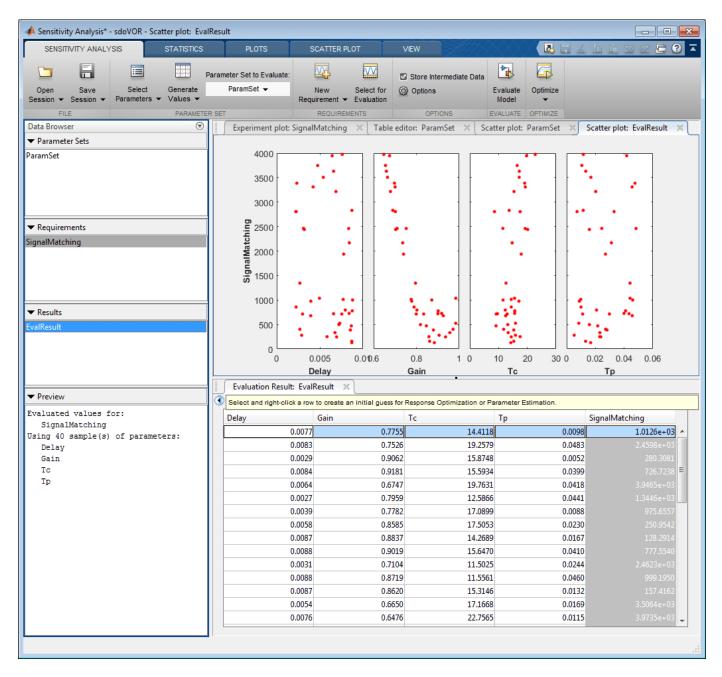
When the sample values are generated, they appear in a table in the Sensitivity Analysis UI. To plot them, select ParamSet in the data browser, click the **Plots** tab, and make a scatter plot. The sampling above used default options, and these are reflected in the scatter plot. For parameters modeled by a uniform distribution, the histograms appear approximately uniform. However, parameter Tc was modeled by a normal distribution, and its histogram has a bell curve profile. If Statistics and Machine Learning Toolbox[™] is available, many other distributions may be used, and sampling can be done using Sobol or Halton low-discrepancy sequences. The off-diagonal plots show scatter plots between pairs of different variables. Since we did not specify cross-correlations between parameters, the scatter plots appear uncorrelated. However, if parameters were believed to be correlated, this can be specified using Correlation Matrix tab in the dialog for generating random parameter values.



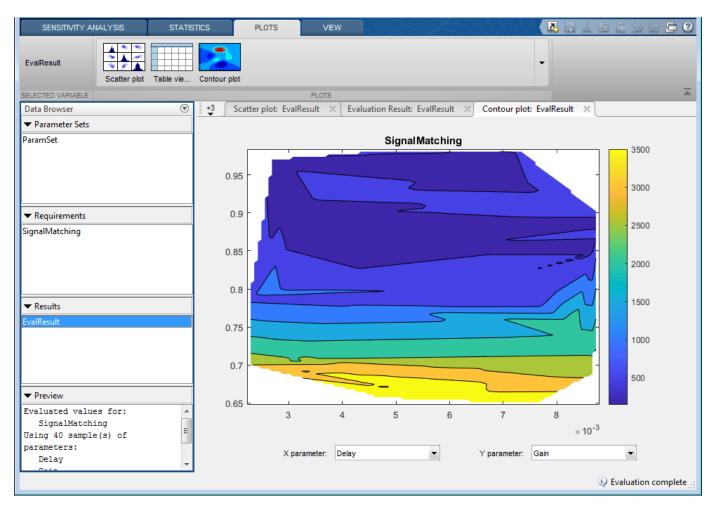
Evaluate the Model

Now that we have generated values for the parameter set and specified a requirement (SignalMatching), we can evaluate the model. In the **Sensitivity Analysis** tab, click **Evaluate Model**.

The model is run once for each set of parameter values, and the results scatter plot is updated as new computations become available. Evaluation could also be sped up using parallel computing. After evaluation is complete, all results are also displayed in a table.

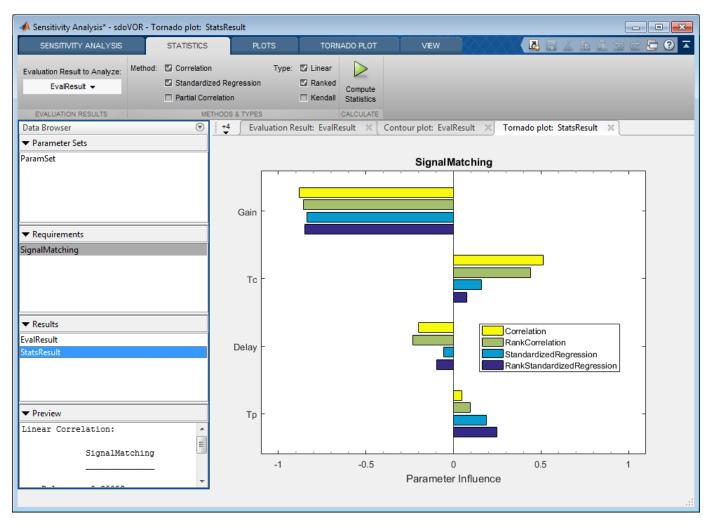


From the scatter plot of evaluation results, the SignalMatching requirement seems to vary systematically as a function of Gain and Tc, but not Delay or Tp. Something similar can be seen in a contour plot. Select the EvalResults variable in the data browser, click the Plots tab, and make a contour plot. The requirement does not vary systematically from left to right as a function of Delay, but it does vertically as a function of Gain.



Statistical Analysis

We can use statistical analysis to quantify how much each parameter influences the requirement. Click the **Statistics** tab and select both correlation and standardized regression; and both linear and ranked analysis types. If Statistics and Machine Learning Toolbox is available, partial correlation and Kendall correlation can also be selected. Click **Compute Statistics** to carry out the calculations and show a tornado plot. The tornado plot displays results from top to bottom in order of which parameter most influences the requirement. The statistical values range from -1 to 1, where the magnitude indicates how much the parameter influences the requirement, and the sign indicates whether an increase in the parameter value corresponds to an increase or decrease in the requirement value. By most measures, this SignalMatching requirement is more sensitive to Gain and Tc, and less sensitive to Delay and Tp.



Select Parameters for Estimation

For parameter estimation, we need to specify starting values for the parameters. Click the evaluation results table and click the SignalMatching column header to sort results. Select the row of parameter values that minimizes the SignalMatching requirement. Right-click on the row and extract these parameter values. A new variable, ParamValues, is shown in the data browser.

SENSITIVITY ANALYSIS		STATISTICS		PLOTS	VIEW					1 i 9 c 🗗 ?
	=		Parame	eter Set to Evaluate:		W	Store Intermediate D	ata 🛅	G	
Open Save Session ▼ Session ▼ Par	Select ameters 💌	Generate Values 👻	-	ParamSet 👻	New Requirement 💌	Select for Evaluation	Options	Evaluate Model	Optimiz •	e
FILE		PARAMET	1 -		REQUIREN	1	OPTIONS	EVALUATE		
Data Browser		$\overline{\mathbf{v}}$	<u>+</u> 4		ult: EvalResult		our plot: EvalResult 🛛 🛛	Tornado plot		esult 🛛
 Parameter Sets 			🔇 s	elect and right-click a	a row to create an	initial guess	for Response Optimization	or Parameter Esti	mation.	
aramSet			De	elay	Gain		Tc	Тр		SignalMatching
				0.00	77	0.7755	14.4118		0.0098	1.0126e+03
				0.00	83	0.7526	19.2579		0.0483	2.4598e+03
				0.00	29	0.9062	15.8748		0.0052	
				0.00	84	0.9181	15.5934		0.0399	
 Requirements 				0.00	64	0.6747	19.7631		0.0418	
ignalMatching				0.00	27	0.7959	12.5866		0.0441	
				0.00	39	0.7782	17.0899		0.0088	975.6557
				0.00	58	0.8585	17.5053		0.0230	
				0.00	87	0.8837	14.2689		0.0167	
				0.00	88	0.9019	15.6470		0.0410	
Results				0.00	31	0.7104	11.5025		0.0244	
valResult			i L	0.00	88	0.8719	11.5561		0.0460	999.1950
aramValues				0.00		0.8620	15.3146		0.0132	
tatsResult				0.00	54	0.6650	17.1668		0.0169	3.5064e+03
				0.00	76	0.6476	22.7565		0.0115	3.9735e+03
				0.00		0.7993	12.9993		0.0111	713.5724
Preview			í	0.00		0.9839	15.5620		0.0441	1.0352e+03
				0.00		0.7362	14.7525		0.0311	2.1674e+03
				0.00		0.8341	9.2009		0.0297	714.3145
				0.00		0.6895	13.6831		0.0115	
				0.00		0.9005	9.6160		0.0434	719.8047
				0.00	22	0 7020	17 5211		0.0330	2 8054e+03

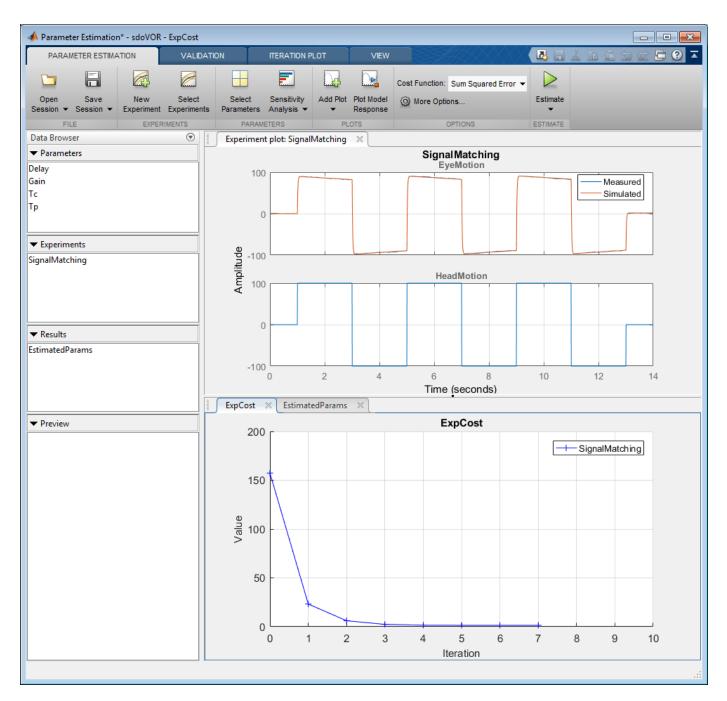
To transition from sensitivity analysis to parameter estimation, navigate to the **Sensitivity Analysis** tab, click **Optimize**, and open a parameter estimation session. In the dialog that appears, specify that you want to use the parameter values in ParamValues, and the SignalMatching requirement.

📣 Paramet	er Estimation - Import from Sensitivity	y Analysis 📃 🔳 🗮	×
Select para	ameter values to import as initial guess:	ParamValues (Values for - Delay, Gain, Tc, Tp)	•]
Select exp	eriments to Import:		
Import		Experiment	
V	SignalMatching with data for signal(s) -	OculomotorPlant:1	
		OK Cancel Help]

Since we found above that parameters Gain and Tc have the most influence on the value of SignalMatching, we would like to estimate only these two parameters, since the time for estimation increases with the number of parameters being estimated. In Parameter Estimation UI, click **Select Parameters** and select only Gain and Tc for estimation.

Edit: Estimated Parameters			×
Parameters Tuned for all Experiment	nts		
Delay			
0.00870016863770062	- 🚽	<u>х</u> П Е	stimate
<u>Gain</u>			
0.862039201589536	- 🖶	X. 🖾 E	stimate
Tc			
15.3146241480495	- 🚽	X 🖾 E	stimate
Tp			
0.0131831162736284	- 🖶	<u>х</u> П е	stimate
Select parameters			
Parameters and Initial States Tuned	per Experimen	t	
Experiment: SignalMatching 💌			
Select experiment initial states for estimat	ion.		
There are no initial states defined for this	experiment.		
Select experiment parameters for estimat	ion.		
There are no parameters defined for this	experiment.		
Edit experiment			
ł	😼 Update Model	🖋 ок	(?) Help

Since the experiment definition has been imported from SignalMatching and the parameter values have been imported from ParamValues, we have everything needed for estimation. Click Estimate to carry out parameter estimation for Gain and Tc. Because we are only estimating the two most influential parameters, estimation converges quickly and the model output closely matches the data. As was the case with model evaluations in sensitivity analysis, parallel computing could be used to speed up estimation.



In summary, Sensitivity Analysis UI was used to explore the parameter design space and determine that two parameters, Gain and Tc, were substantially more influential than the others. A start point for estimation was also determined. This start point and the requirement of obtaining a good fit to experimental data were imported into Parameter Estimation UI. Estimation completed quickly because only two parameters needed to be estimated, and the model output fit the data with very little residual error.

Close the model.

bdclose('sdoVOR')

See Also

Related Examples

- "Identify Key Parameters for Estimation (Code)" on page 4-169
- "Use Parallel Computing for Sensitivity Analysis" on page 4-104
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67
- "Validate Sensitivity Analysis" on page 4-96

Explore Design Reliability Using Parameter Sampling (GUI)

This example shows how to use the **Sensitivity Analyzer** to explore the behavior of a PI controller for a DC motor. The controller is susceptible to variations caused by component tolerances, and the impact on controller reliability is explored.

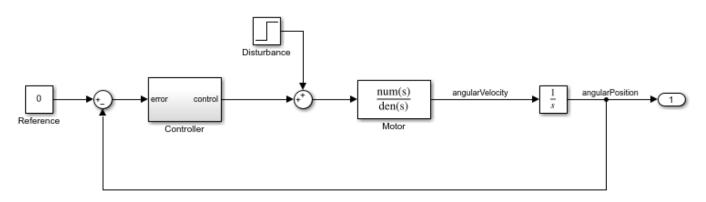
You explore the controller reliability by characterizing the components using probability distributions. You use the distributions to generate random samples and perform Monte-Carlo evaluation of the controller design at these sample points. You evaluate the impact of the component tolerances on the controller behavior, and use statistical analysis to determine which components have the most influence on whether the controller meets its requirements. This analysis guides the selection of component tolerances.

This example requires Statistics and Machine Learning Toolbox™.

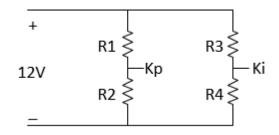
Implementation of Controller for DC Motor

The controller enables the DC motor's angular position to match a desired reference value. The load on the motor is subject to disturbances, and the controller needs to reject these disturbances. The Simulink® model can be used to probe how well the controller rejects a step disturbance at 1 second.

open_system('sdoMotorPosition');



The gains of the PI controller, Kp and Ki, are set using resistors in the circuit below:



The resistances R1 through R4 are 47 kOhm, 180 kOhm, 10 kOhm and 10 kOhm respectively. These were chosen to set Kp and Ki to values that enable the controller to meet the requirements for disturbance rejection. However, in practice the actual resistor values will differ from the nominal ones, within a tolerance. This raises concern about whether the actual controller will still satisfy the requirements. To explore the effect of different resistance values, use the **Sensitivity Analyzer**. In

the Simulink model from the **Apps** tab, click **Sensitivity Analyzer** under **Control Systems** to open the app.

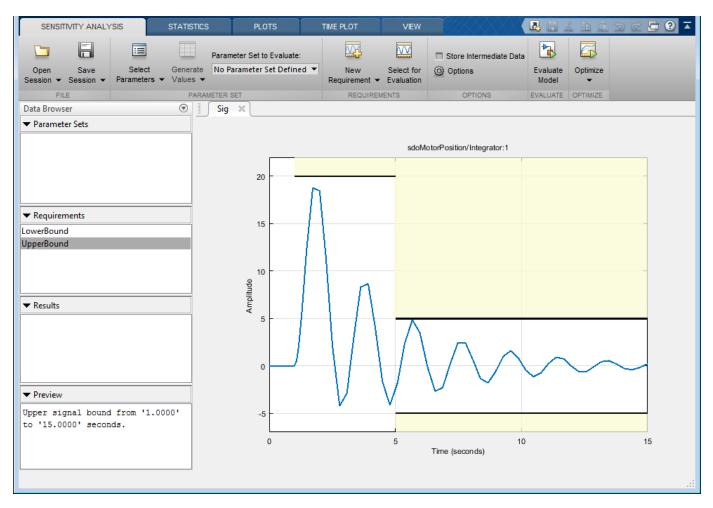
Design Requirements

The controller needs to maintain the motor at a reference position in the presence of disturbances. If a step disturbance occurs, the motor needs to deviate no more than 20 degrees, and needs to settle back to within 5 degrees of the reference position by 4 seconds after the disturbance.

Load previously specified design requirements for disturbance rejection. In the app, click **Open Session** and select **Open from model workspace** in the drop-down menu.

Open Session	х
Open from Simulink model workspace	
Existing Session Variables	
sdoMotorPosition_sasession	
Session: sdoMotorPosition_sasession	
OK Cancel Help	

You can plot the requirements and verify that they are met when the resistances have the nominal values. In the **Requirements** area in the data browser, right-click on the LowerBound requirement, and select **Plot and Simulate**. Do the same for the UpperBound requirement.



Parameter Sampling

The motor position satisfies the disturbance rejection requirements, when the resistances are at their nominal values. However, in practice the actual resistor values will differ from the nominal ones, and we need to determine whether the controller will still meet the requirements. Click **Select Parameters** and make a new parameter set. This creates **ParamSet** in the **Parameter Sets** area of the app. Specify that R1, R2, R3, and R4 are in the parameter set, and click **OK**.

lter l	by variable na	ame		Q					
•	Variable	Current va	Used By						
1	R1	47	sdoMotorPosition/Controller/Proportional Gain						
1	R2	180	sdoMotorPosition/Controller/Proportional Gain						
1	R3	10	sdoMotorPosition/Controller/Integral Gain						
V	R4	10	sdoMotorPosition/Controller/Integral Gain						
	g	1	sdoMotorPosition/Motor						
	tau	0.85	sdoMotorPosition/Motor						

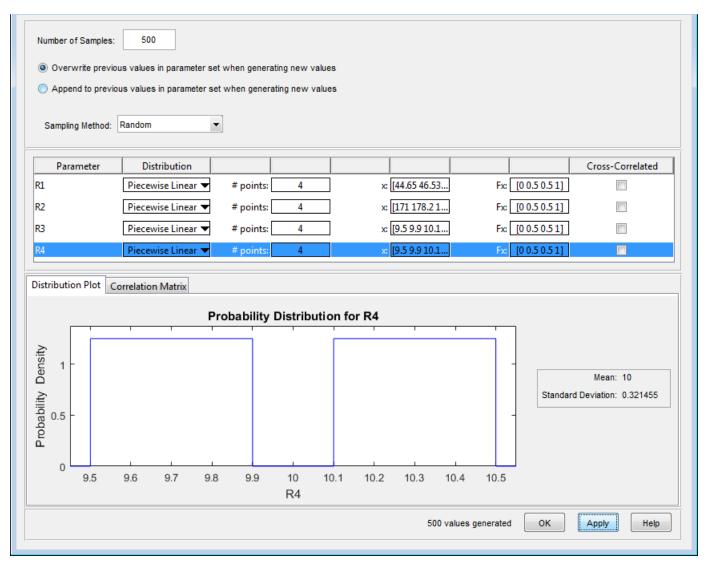
Click **Generate Values** and generate random values. For repeatable results, reset the state of the random number generator in MATLAB®.

rng('default')

In the Generate Random Parameters dialog box, specify 500 samples to generate.

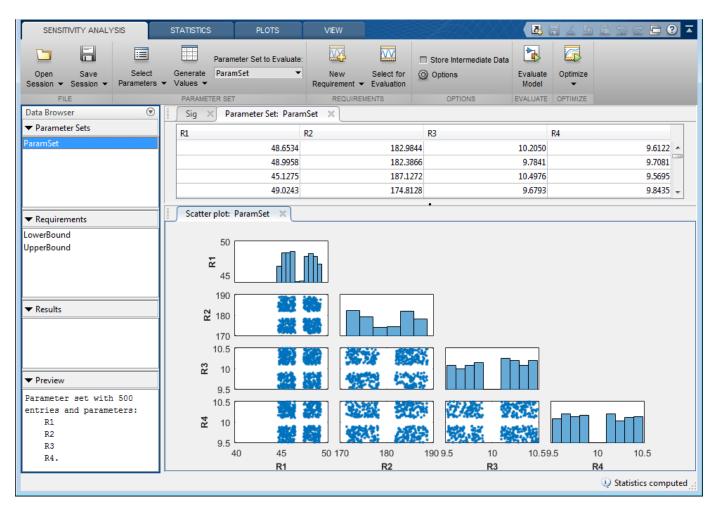
Specify the probability distribution for each parameter. Standard precision resistors match their nominal component value within a tolerance of 5%. This could be modeled using a uniform probability distribution. However, because resistors that measure within 1% of the nominal value are separated out and sold as higher-priced precision resistors, the 5% resistors can be more accurately modeled by a probability distribution with a well that excludes values within 1% of nominal. This can be modeled using a piecewise linear probability distribution if Statistics and Machine Learning Toolbox[™] is available.

Specify the distribution of R1 as piecewise linear with 4 points. Specify the x values as [0.95 0.99 1.01 1.05] times 47 (the nominal value of the resistor). Specify the Fx values as [0 0.5 0.5 1]; these are the values of the cumulative distribution function corresponding to each x value. Similarly, set the distributions of R2, R3 and R4 to piecewise linear with 4 points, the x values as [0.95 0.99 1.01 1.05] times the nominal values (180, 10, and 10, respectively), and the Fx values as [0 0.5 0.5 1].



Click **OK** to generate parameter values. The generated values are stored in the **ParamSet** variable in the **Parameter Set** area of the app. (Note that due to the random number generator, the specific values in the table below may differ from what you get when running the example.)

To plot the parameter set click ParamSet in the **Parameter Sets** area of the app browser. In the **Plots** tab, select **Scatter Plot** in the plot gallery. The plot shows the histogram of the generated parameters on the diagonal and pair-wise parameter scatter plots off the diagonal. Each marker on the plot represents one row of the ParamSet table, with each row being simultaneously displayed on all the scatter plots. You can use the **View** tab to arrange the layout of the table and plot so they are both visible.



Evaluate Requirements with 5% Components

Evaluate the requirements for each row of parameter values in the table to see if the requirements are satisfied. In the **Sensitivity Analysis** tab, click **Select for Evaluation**. By default, all requirements are selected to be evaluated. Click **Evaluate Model** to evaluate the UpperBound and LowerBound requirements for each row of parameter values in ParamSet. Note you can speed up evaluation by using parallel computing if you have the Parallel Computing Toolbox[™], or by using fast restart. For more information, see "Use Parallel Computing for Sensitivity Analysis" on page 4-104 and "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109.

A results scatter plot showing each requirement vs. each parameter is updated during model evaluation. At the end of evaluation a table with the evaluation results is shown. Each row in the evaluation result table contains values for R1, R2, R3, R4 and the resulting requirement values UpperBound and LowerBound. The evaluation results are stored in the EvalResult variable in the **Results** area of the app. You can use the **View** tab to arrange the layout of the table and plot so they are both visible.

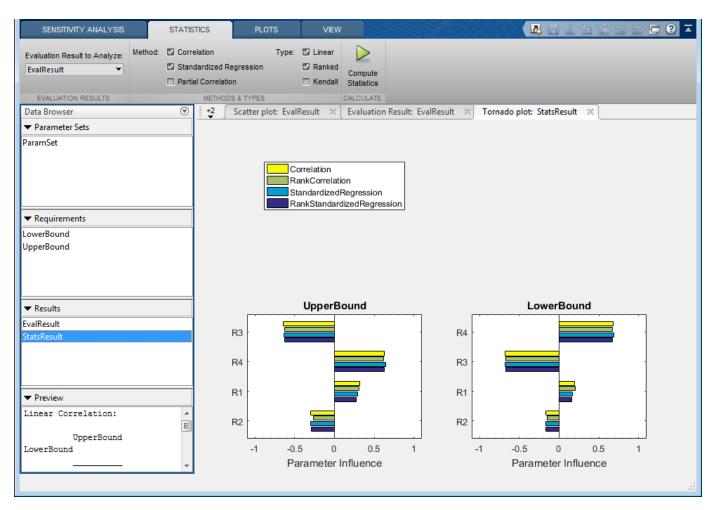
You can sort the evaluation results table by clicking on the column headers in the table. The LowerBound requirement is still met, as indicated by the fact that all evaluation results for the signal bound requirement are negative. That is not the case for the UpperBound requirement, which has several positive values. By selecting the rows of the table with these positive values, you can also see the corresponding points highlighted in the scatter plot.

SENSITIVITY ANALYSIS STAT	TISTICS PLOTS	VIEW							h i o	2 🗄 🕐 🔺
	Parameter Set to Evaluate:		NV	Store Intermediate Data						
	erate ParamSet 💌	New	Select for	Options		ptimize				
	jes 🔻	Requirement -		-	Model	•				
	ARAMETER SET	REQUIREN		OPTIONS	EVALUATE O					
Data Browser 💿	+2 Scatter plot: ParamS	et 🛛 Scatter	r plot: EvalRes			wer: EvalRes	ult ×			
▼ Parameter Sets				0	Ð					
ParamSet										
	0.02	1 6 1	0.0	2	R1	R2	R3	R4	UpperBo	LowerBo
					49.0395	174.6776	9.5806	10.3857	0.0179	-0.3994 🔺
		5			45.6369	172.0289	9.5370	10.4962	0.0123	-0.3981 😑
	0 🎦 🌅	1 🌠 📋	0		48.9842	171.6142	9.7746	10.3696	0.0115	-0.4114
▼ Requirements	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1				49.0341	173.3554	9.5157	10.1184		-0.4112
LowerBound	물 -0.02		-0.0	12	47.6171	174.8330	9.6970	10.4709		-0.4066
UpperBound	0.02 ·		-0.0		49.0840	177.3165	9.7694	10.4651		-0.4094
	E 🕺 🚼		8 2		49.2810	184.6286	9.5484	10.4074		-0.4036
	🚊 -0.04 - 🚺 🐰		-0.0	4	46.4303	172.7307	9.5037	10.3042		-0.4060
	57 23				47.5020	177.7258	9.6109	10.4293		-0.4063
▼ Results	44 S S				48.8835	184.1444	9.5338	10.3622		-0.4064
EvalResult	-0.06		-0.0	16	48.2697	172.2748	9.6654	10.2294		-0.4157
	5 73		7		48.0954	173.6994	9.7599	10.3858		-0.4138
	0.00		-0.0		49.2112	177.5672	9.7288	10.3472		-0.4140
	-0.08 - 1 -		E		48.7827	176.2678	9.5700	10.1791		-0.4142
	-0.35] []]	-0.3	15	48.9924	175.8362	9.7653	10.3441		-0.4159
▼ Preview					48.2026	176.6658	9.8066	10.4692		-0.4139
					48.1160	177.5936	9.5125	10.1878		-0.4129
Evaluated values for: UpperBound	-0.4		1 -0.4		46.4956	175.0589	9.5998	10.3771		-0.4099
LowerBound		2	3		49.0619 49.3465	176.1787 174.9303	9.5976 9.8167	10.1417 10.3097		-0.4176
Using 500 sample(s) of	P 🤾 籠	c 🏭 🗌			49.5405	176.6838	9.5536	10.3097		-0.4198 -0.4149
parameters:	punoguous -0.45		1 5 🕅		48.1437	170.0050	9.3330	10.1809		-0.4149
R1	H -0.45		-0.4		49.0241	184,5043	9.5697	10.4772		-0.4170
R2 R3	E 2	14 SX			48.1420	183.6417	9.6445	10.3831		-0.4098
R5 R4					49.1380	171.3518	9.8119	10.4313		-0.4104
	-0.5		-0.5	; -	49.2484	185.1834	9.6973	10.1040		-0.4148
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		5		44.6752	172.4187	9.5448	10.4017		-0.4111
			2		47.5858	172.6086	9.7772	10.2981		-0.4213
	-0.55		-0.5	5	45.7642	171.9130	9.6843	10.3640		-0.4162
	40 45 501701801	90 10	10	0 0.1 0.2	48.0143	172.3299	9.7594	10.2067	-2.0420e-04	-0.4244
	R1 R2	R3	R4		46.4624	185.0101	9.5763	10.4996		-0.4107
					45 8258	175 513/	0 7060	10 / 276	_0 0010	-0 /160

Analyze the Evaluation Results

Using 5% tolerance components resulted in violation of the UpperBound requirement. Precision components with 1% tolerance would satisfy the design requirements, but they are more costly, so it is desirable to use only as many as necessary. You can use statistical analysis to identify the components that most influence the design requirements.

In the **Statistics** tab, select a variety of analyses to be done, including **Correlation** and **Standardized Regression** methods, and **Linear** and **Ranked** types of processing. Click **Compute Statistics**. The analysis result is stored in **StatsResult** in the **Results** area of the app, and a tornado plot shows the analysis results. For each requirement, the tornado plot shows the most influential parameters at the top, and the others in decreasing order of the magnitude of their influence on the requirement. For the UpperBound requirement, R3 and R4 have the most influence, so we will try replacing these by higher precision 1% components.



Evaluate Requirements with Mixed Components

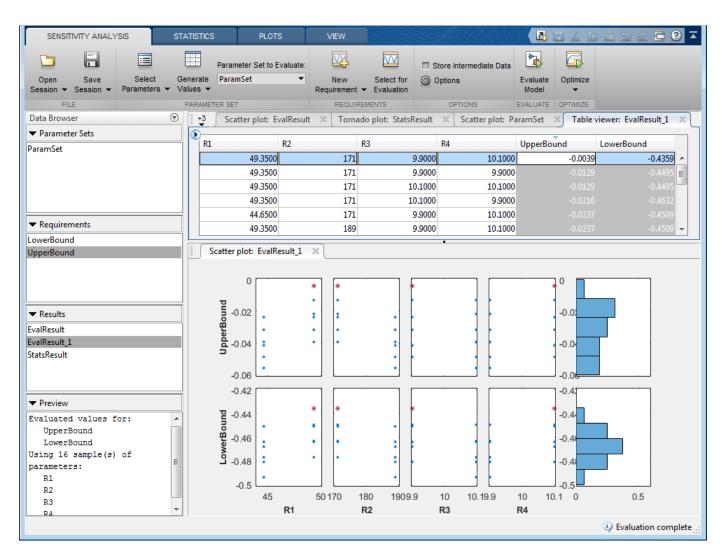
Explore the use of 1% component tolerances only for resistors R3 and R4. In the **Sensitivity Analysis** tab, click **Generate Values** and generate gridded values. For R1 and R2, specify that the nominal value is to be perturbed by plus-and-minus 5%. For R3 and R4, specify that the nominal value is to be perturbed by plus-and-minus 1%.

Generate Gridded Parameter Value	s X								
Gridding Method: All Combinations (16 samples) Min-Max Combinations (16 samples) Pair-Wise Combinations (2 samples) Starting Values:									
Parameter	Values								
R1	47 * [0.95 1.05]								
R2	180 * [0.95 1.05]								
R3	10 * [0.99 1.01]								
R4	10 * [0.99 1.01]								
₩ c	Iverwrite 🚟 Append 🕐 Help								

Click **Overwrite** to generate the new parameter values. To plot the parameter set click ParamSet in the **Parameter Sets** area of the app browser. In the **Plots** tab, select **Scatter Plot** in the plot gallery.

SENSITIVITY ANALYSIS STATIST		TICS PLOTS		METER SET	VIEW			É D G	2 🔁 🕐 🔼
	Par	ameter Set to Evalu	ate:		Store Intermediate Data		G		
Open Save Select Session - Session - Parameter	Generate Pa s ▼ Values ▼	ramSet	 New Requirem 			Evaluate Model	Optimize ▼		
FILE			REC	QUIREMENTS	OPTIONS	EVALUATE	OPTIMIZE		
Data Browser		Sig × Para	meter Set: Pa	ramSet 🛛 🗶	Scatter plot: ParamSet	×			
✓ Parameter Sets									
ParamSet	R1	R2	R3	R4	50				
	4	44.6500 1			50				
		49.3500 1			₩ 45				
		44.6500 1			2 40				
		49.3500 1							
▼ Requirements		44.6500 1			40				
LowerBound		49.3500 1			190 • •				
UpperBound			39 10.1000		N				
		49.3500 1			2 180				
		44.6500 1							
		49.3500 1		-	170 - 170				
▼ Results		44.6500 1			10.1	•			
		49.3500 1							
EvalResult StatsResult		44.6500 1			≌ 10				
Statsresult		49.3500 1							
		44.6500 1			9.9				
		49.3500 1	10.1000	10.1000	10.1	•			
▼ Preview					₩ 10				
Linear Correlation:	<u>^</u>								
UpperBound					9.9		-		
LowerBound					45 50170		09.9 10		10 10.1
	_				R1	R2	R3	I	R4
								① Statisti	cs computed .:

In the **Sensitivity Analysis** tab, click **Evaluate Model**. The requirements are evaluated for each row in the table of parameter values, and results are stored in EvalResults_1 shown in the **Results** area of the app. The evaluation results scatter plot and the evaluation results table show that both requirements are met for all combinations of component values.



The **Sensitivity Analyzer** was used to explore the effect of standard precision components on the design requirements of a PI controller. With standard precision components, some requirements were found to be violated. Statistical analysis was used to identify which parameters most influence the requirements. The analysis resulted in replacement of only two of the four components with most costly high-precision components.

Close the model.

bdclose('sdoMotorPosition')

See Also

Related Examples

- "Specify Parameters for Design Exploration" on page 4-4
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67

- "Validate Sensitivity Analysis" on page 4-96
- "Interact with Plots in the Sensitivity Analyzer" on page 4-79

Design Exploration Using Parameter Sampling (Code)

This example shows how to sample and explore a design space. You explore the design of a Continuously Stirred Tank Reactor to minimize product concentration variation and production cost. The design includes feed stock uncertainty.

You explore the CSTR design by characterizing design parameters using probability distributions. You use the distributions to generate random samples in the design space and perform Monte-Carlo evaluation of the design at these sample points. You then create plots to visualize the design space and select the best design. You can then use the best design as an initial guess for optimization of the design.

You can also use the sampled design space and Monte-Carlo evaluation output to analyze the influence of design parameters on the design; see "Identify Key Parameters for Estimation (Code)" on page 4-169.

Continuously Stirred Tank Reactor (CSTR) Model

Continuously Stirred Tank Reactors (CSTRs) are common in the process industry. The Simulink® model, sdoCSTR, models a jacketed diabatic (i.e., non-adiabatic) tank reactor described in [1]. The CSTR is assumed to be perfectly mixed, with a single first-order exothermic and irreversible reaction, $A \rightarrow B$. A, the reactant, is converted to B, the product.

In this example, you use the following two-state CSTR model, which uses basic accounting and energy conservation principles:

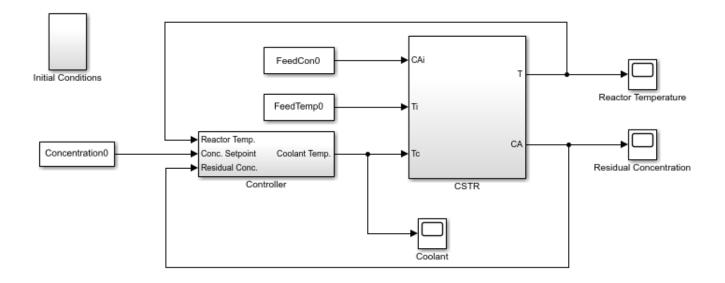
$$\frac{dC_A}{dt} = \frac{F}{A*h}(C_{feed} - C_A) - r*C_A$$
$$\frac{dT}{dt} = \frac{F}{A*h}(T_{feed} - T) - \frac{H}{c_p\rho}r - \frac{U}{c_p*\rho*h}(T - T_{cool})$$

$$r = k_0 * e^{\frac{-E}{R*I}}$$

- C_A, and C_{feed} Concentrations of A in the CSTR and in the feed [kgmol/m³]
- T, T_{feed} , and T_{cool} CSTR, feed, and coolant temperatures [K]
- F and ρ Volumetric flow rate [m³/h] and the density of the material in the CSTR [1/m³]
- h and A Height [m] and heated cross-sectional area [m²] of the CSTR.
- k_0 Pre-exponential non-thermal factor for reaction $A \rightarrow B$ [1/h]
- E and H Activation energy and heat of reaction for $A \rightarrow B$ [kcal/kgmol]
- *R* Boltzmann's gas constant [kcal/(kgmol * K)]
- C_p and U Heat capacity [kcal/K] and heat transfer coefficients [kcal/(m² * K * h)]

Open the Simulink model.

open_system('sdoCSTR');



Copyright 2012-2015 The MathWorks, Inc.

CSTR Design Problem

Assume that the CSTR is cylindrical, with the coolant applied to the base of the cylinder. Tune the CSTR cross-sectional area, A, and CSTR height, h, to meet the following design goals:

- Minimize the variation in residual concentration, C_A . Variations in the residual concentration negatively affect the quality of the CSTR product. Minimizing the variations also improves CSTR profit.
- Minimize the mean coolant temperature T_{cool} . Heating or cooling the jacket coolant temperature is expensive. Minimizing the mean coolant temperature improves CSTR profit.

The design must allow for variations in the quality of supply feed concentration, C_{feed} , and feed temperature, T_{feed} . The CSTR is fed with feed from different suppliers. The quality of the feed differs from supplier to supplier and also varies within each supply batch.

Specify Design Variables

Select the following model parameters as design variables:

- Cylinder cross-sectional area A
- Cylinder height *h*

```
p = sdo.getParameterFromModel('sdoCSTR', {'A', 'h'});
```

Limit the cross-sectional area to a range of [0.2 2] m².

p(1).Minimum = 0.2; p(1).Maximum = 2;

Limit the height to a range of [0.5 3] m.

p(2).Minimum = 0.5; p(2).Maximum = 3;

Sample the Design Space

Create a parameter space for the design variables. The parameter space characterizes the allowable parameter values and combinations of parameter values.

pSpace = sdo.ParameterSpace(p);

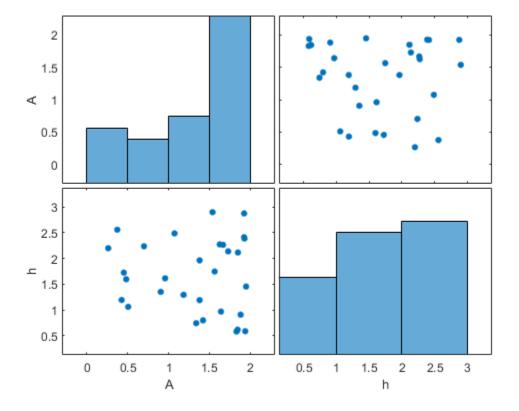
The parameter space uses default uniform distributions for the design variables. The distribution lower and upper bounds are set to the design variable minimum and maximum value respectively.

Use the **sdo.sample** function to generate samples from the parameter space. You use the samples to evaluate the model and explore the design space.

```
rng default; % For reproducibility
pSmpl = sdo.sample(pSpace,30);
```

Use the sdo.scatterPlot command to visualize the sampled parameter space. The scatter plot shows the parameter distributions on the diagonal subplots and pairwise parameter combinations on the off diagonal subplots.

```
figure, sdo.scatterPlot(pSmpl)
```



Specify Uncertain Variables

Select the feed concentration and feed temperature as uncertain variables. You evaluate the design using different values of feed temperature and concentration.

pUnc = sdo.getParameterFromModel('sdoCSTR', {'FeedCon0', 'FeedTemp0'});

Create a parameter space for the uncertain variables. Use normal distributions for both variables. Specify the mean as the current parameter value. Specify a variance of 5% of the mean for the feed concentration and 1% of the mean for the temperature.

```
uSpace = sdo.ParameterSpace(pUnc);
uSpace = setDistribution(uSpace, 'FeedCon0', makedist('normal', pUnc(1).Value, 0.05*pUnc(1).Value));
uSpace = setDistribution(uSpace, 'FeedTemp0', makedist('normal', pUnc(2).Value, 0.01*pUnc(2).Value))
```

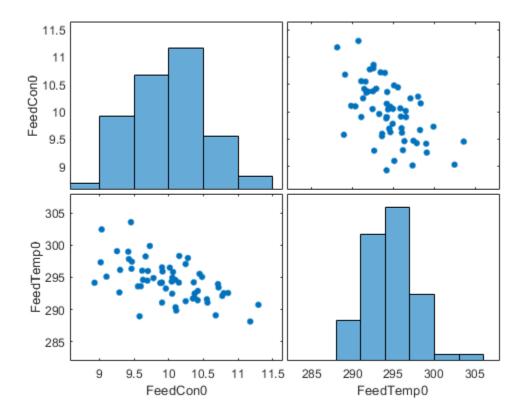
The feed concentration is inversely correlated with the feed temperature. Add this information to the parameter space.

uSpace.RankCorrelation = [1 -0.6; -0.6 1];

The rank correlation matrix has a row and column for each parameter with the (i,j) entry specifying the correlation between the i and j parameters.

Sample the parameter space. The scatter plot shows the correlation between concentration and temperature.

```
uSmpl = sdo.sample(uSpace,60);
sdo.scatterPlot(uSmpl)
```



Ideally you want to evaluate the design for every combination of points in the design and uncertain spaces, which implies 30*60 = 1800 simulations. Each simulation takes around 0.5 sec. You can use parallel computing to speed up the evaluation. For this example you instead only use the samples that have maximum & minimum concentration and temperature values, reducing the evaluation time to around 1 min.

```
[~,iminC] = min(uSmpl.FeedCon0);
[~,imaxC] = max(uSmpl.FeedCon0);
[~,iminT] = min(uSmpl.FeedTemp0);
[~,imaxT] = max(uSmpl.FeedTemp0);
uSmpl = uSmpl(unique([iminC,imaxC,iminT,imaxT]) ,:)
```

4x2 table

FeedCon0	FeedTemp0
9.4555	303.58
11.175	288.13
11.293	290.73
8.9308	294.16

Create Evaluation Function

Create a function that evaluates the design for a given sample point in the design space. The design is evaluated on how well it minimizes the variation in residual concentration and mean coolant temperature.

Specify Design Requirements

Evaluating a point in the design space requires logging model signals. Logged signals are used to evaluate the design requirements.

Log the following signals:

CSTR concentration, available at the second output port of the sdoCSTR/CSTR block

```
Conc = Simulink.SimulationData.SignalLoggingInfo;
Conc.BlockPath = 'sdoCSTR/CSTR';
Conc.OutputPortIndex = 2;
Conc.LoggingInfo.NameMode = 1;
Conc.LoggingInfo.LoggingName = 'Concentration';
```

• Coolant temperature, available at the first output of the sdoCSTR/Controller block

```
Coolant = Simulink.SimulationData.SignalLoggingInfo;
Coolant.BlockPath = 'sdoCSTR/Controller';
Coolant.OutputPortIndex = 1;
Coolant.LoggingInfo.NameMode = 1;
Coolant.LoggingInfo.LoggingName = 'Coolant';
```

Create and configure a simulation test object to log the required signals.

```
simulator = sdo.SimulationTest('sdoCSTR');
simulator.LoggingInfo.Signals = [Conc,Coolant];
```

Evaluation Function

Use an anonymous function with one argument that calls the sdoCSTR_design function.

evalDesign = @(p) sdoCSTR_design(p,simulator,pUnc,uSmpl);

The evalDesign function:

- Has one input argument that specifies the CSTR dimensions
- Returns the optimization objective value

The sdoCSTR_design function uses a for loop that iterates through the sample values specified for the feed concentration and temperature. Within the loop, the function:

- Simulates the model using the current design point, feed concentration, and feed temperature values
- · Calculates the residual concentration variation and coolant temperature costs

To view the objective function, type edit sdoCSTR_design.

```
type sdoCSTR_design
```

```
function design = sdoCSTR design(p,simulator,pUnc,smplUnc)
%SDOCSTR DESIGN
% The sdoCSTR design function is used to evaluate a CSTR design.
%
% The |p| input argument is the vector of CSTR dimensions.
%
% The [simulator] input argument is a sdo.SimulinkTest object used to
% simulate the |sdoCSTR| model and log simulation signals.
%
% The |pUnc| input argument is a vector of parameters to specify the CSTR
% input feed concentration and feed temperature. The |smplUnc| argument is
% a table of different feed concentration and temperature values.
%
% The IdesignI return argument contains information about the design
% evaluation that can be used by the |sdo.optimize| function to optimize
% the design.
%
% see also sdo.optimize, sdoExampleCostFunction
%
% Copyright 2012-2013 The MathWorks, Inc.
%% Model Simulations and Evaluations
%
% For each value in |smplUnc|, configure and simulate the model. Use
\% the logged concentration and coolant signals to compute the design cost.
%
costConc
          = 0:
costCoolant = 0;
```

```
for ct=1:size(smplUnc,1)
    %Set the feed concentration and temperature values
    pUnc(1).Value = smplUnc{ct,1};
   pUnc(2).Value = smplUnc{ct,2};
   %Simulate model
    simulator.Parameters = [p; pUnc];
    simulator = sim(simulator);
    logName = get param('sdoCSTR','SignalLoggingName');
    simLoa
             = get(simulator.LoggedData,logName);
   %Compute Concentration cost based on the standard deviation of the
   %concentration from a nominal value.
    Sig = find(simLog, 'Concentration');
    costConc = costConc+10*std(Sig.Values-2);
   %Compute coolant cost based on the mean deviation from room
   %temperature.
    Sig = find(simLog, 'Coolant');
    costCoolant = costCoolant+abs(mean(Sig.Values - 294))/30;
end
%% Return Total Cost
%
% Compute the total cost as a sum of the concentration and coolant costs.
%
design.F = costConc + costCoolant;
%%
\% Add the individual cost terms to the return argument. These are not used
% by the optimizer, but included for convenience.
design.costConc = costConc;
design.costCoolant = costCoolant;
end
```

Evaluate

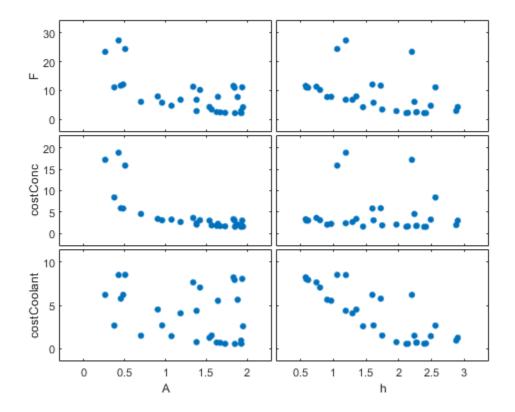
Use the sdo.evaluate command to evaluate the model at the sample design points.

```
y = sdo.evaluate(evalDesign,p,pSmpl);
```

```
Model evaluated at 30 samples.
```

View the results of the evaluation using a scatter plot. The scatter plot shows pairwise plots for each design variable (A,h) and design cost. The plot includes the total cost, F, as well as coolant and concentration costs, costCoolant and costConc respectively.

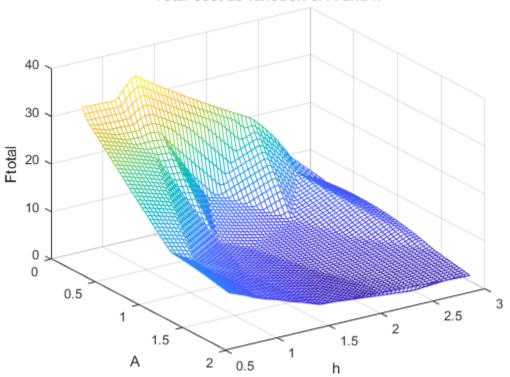
```
sdo.scatterPlot(pSmpl,y);
```



The plot shows that larger cross-sectional areas result in lower total costs. However it is difficult to tell how the height influences the total cost.

Create a mesh plot showing the total cost as a function of A and h.

```
Ftotal = scatteredInterpolant(pSmpl.A,pSmpl.h,y.F);
xR = linspace(min(pSmpl.A),max(pSmpl.A),60);
yR = linspace(min(pSmpl.h),max(pSmpl.h),60);
[xx,yy] = meshgrid(xR,yR);
zz = Ftotal(xx,yy);
mesh(xx,yy,zz)
view(56,30)
title('Total cost as function of A and h')
zlabel('Ftotal')
xlabel(p(1).Name), ylabel(p(2).Name);
```



Total cost as function of A and h

The plot shows that high values of A and h result in lower costs. The best design in the sampled space corresponds to the design with the lowest cost value.

```
[~,idx] = min(y.F);
pBest = [y(idx,:), pSmpl(idx,:)]
pBest =
1x5 table
F costConc costCoolant A h
2.106 1.5505 0.55552 1.9271 2.3867
```

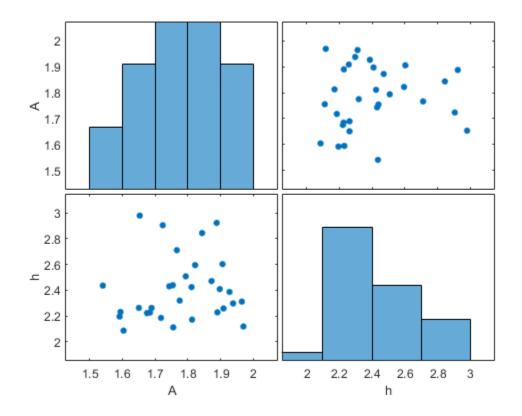
Refine the Design Space

The total cost surface plot shows that low cost designs are designs with A in the range [1.5 2] and h in the range [2 3]. Modify the parameter space distributions for A and h and resample the design space to focus on this region.

```
pSpace = setDistribution(pSpace, 'A', makedist('uniform', 1.5, 2));
pSpace = setDistribution(pSpace, 'h', makedist('uniform', 2, 3));
pSmpl = sdo.sample(pSpace, 30);
```

Add the pBest found earlier to the new samples so that it is part of the refined design space.

pSmpl = [pSmpl;pBest(:,4:5)]; sdo.scatterPlot(pSmpl)



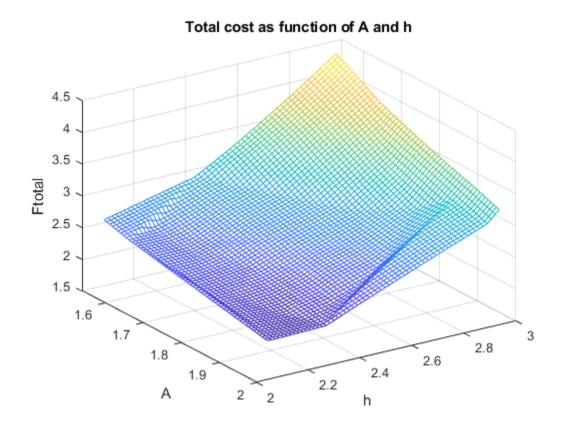
Evaluate Using Refined Design Space

```
y = sdo.evaluate(evalDesign,p,pSmpl);
```

```
Model evaluated at 31 samples.
```

Create a mesh plot for this section of the design space. The surface indicates that better designs are near the A = 1.9, h = 2.1 point.

```
Ftotal = scatteredInterpolant(pSmpl.A,pSmpl.h,y.F);
xR = linspace(min(pSmpl.A),max(pSmpl.A),60);
yR = linspace(min(pSmpl.h),max(pSmpl.h),60);
[xx,yy] = meshgrid(xR,yR);
zz = Ftotal(xx,yy);
mesh(xx,yy,zz)
view(56,30)
title('Total cost as function of A and h')
zlabel('Ftotal')
xlabel(p(1).Name), ylabel(p(2).Name);
```



Find the best design from the new design space and compare with the best design point found earlier.

```
[~,idx] = min(y.F);
pBest = [pBest; [y(idx,:), pSmpl(idx,:)]]
```

pBest =

2x5 table

F	costConc	costCoolant	A	h
2.106	1.5505	0.55552	1.9271	2.3867
1.9754	1.4824	0.49295	1.9695	2.1174

The best design in the refined design space is better than the design found earlier. This indicates that there may be better designs in the same region and warrants refining the design space further. Alternatively you can use the best design point as an initial guess for optimization.

Related Examples

To learn how to explore the CSTR design space using the **Sensitivity Analyzer**, see "Design Exploration Using Parameter Sampling (GUI)" on page 4-112.

To learn how to optimize the CSTR design using the sdo.optimize command, see "Design Optimization with Uncertain Variables (Code)" on page 3-152.

To learn how to analyze the influence of design parameters on the design using the sdo.analyze command, see "Identify Key Parameters for Estimation (Code)" on page 4-169

References

[1] Bequette, B.W. *Process Dynamics: Modeling, Analysis and Simulation.* 1st ed. Upper Saddle River, NJ: Prentice Hall, 1998.

Close the model

bdclose('sdoCSTR')

See Also

More About

- "Design Exploration Using Parameter Sampling (GUI)" on page 4-112
- "Use Parallel Computing for Sensitivity Analysis" on page 4-104
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67
- "Validate Sensitivity Analysis" on page 4-96

Identify Key Parameters for Estimation (Code)

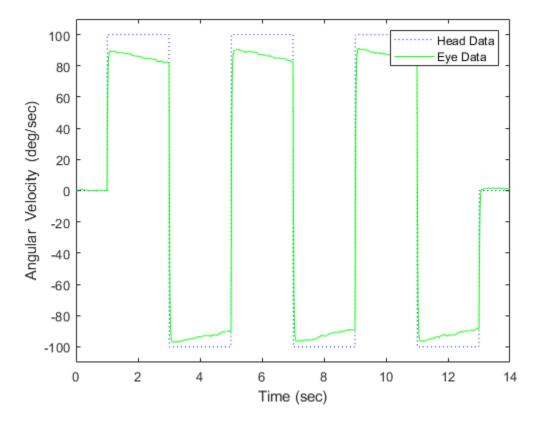
This example shows how to use sensitivity analysis to narrow down the number of parameters that you need to estimate to fit a model. This example uses a model of the vestibulo-ocular reflex, which generates compensatory eye movements.

Model Description

The vestibulo-ocular reflex (VOR) enables the eyes to move at the same speed and in the opposite direction as the head, so that vision is not blurred when the head moves during normal activity. For example, if the head turns in one direction, the eyes turn in the opposite direction, with the same speed. This happens even in the dark. In fact, the VOR is most easily characterized by measurements in the dark, to ensure that eye movements are predominantly driven by the VOR.

The file sdoVOR_Data.mat contains uniformly sampled data of stimulation and eye movements. If the VOR were perfectly compensatory, then a plot of eye movement data, when flipped vertically, would overlay exactly on top of a plot of head motion data. Such a system would be described by a gain of 1 and a phase of 180 degrees. However, when we plot the data in the file sdoVOR_Data.mat, the eye movements are close, but not perfectly compensatory.

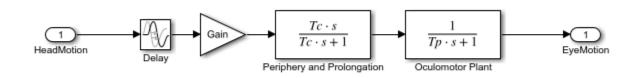
```
load sdoVOR_Data.mat; % Column vectors: Time HeadData EyeData
figure
plot(Time, HeadData, ':b', Time, EyeData, '-g')
xlabel('Time (sec)')
ylabel('Angular Velocity (deg/sec)')
ylim([-110 110])
legend('Head Data', 'Eye Data')
```



The eye movement data does not perfectly overlay the head motion data, and this can be modeled by several factors. Head rotation is sensed by organs in the inner ears, known as semicircular canals. These detect head motion and transmit signals about head motion to the brain, which sends motor commands to the eye muscles, so that eye movements compensate for head motion. We would like to use this eye movement data to estimate the parameters in the models for these various stages. The model we will use is shown below. There are four parameters in the model: Delay, Gain, Tc, and Tp.

model_name = 'sdoVOR';
open_system(model_name)

Vestibulo-Ocular Reflex



Copyright 2013 The MathWorks, Inc.

The Delay parameter models the fact that there is some delay in communicating the signals from the inner ear to the brain and the eyes. This delay is due to the time needed for chemical neurotransmitters to traverse the synaptic clefts between nerve cells. Based on the number of synapses involved in the vestibulo-ocular reflex, this delay is expected to be around 5 ms. For estimation purposes, we will assume it is between 2 and 9 ms.

```
Delay = sdo.getParameterFromModel(model_name, 'Delay');
Delay.Value = 0.005; % seconds
Delay.Minimum = 0.002;
Delay.Maximum = 0.009;
```

The Gain parameter models the fact that the eyes do not move quite as much as the head does. We will use 0.8 as our initial guess, and assume it is between 0.6 and 1.

```
Gain = sdo.getParameterFromModel(model_name, 'Gain');
Gain.Value = 0.8;
Gain.Minimum = 0.6;
Gain.Maximum = 1;
```

The Tc parameter models the dynamics associated with the semicircular canals, as well as some additional neural processing. The canals are high-pass filters, because after a subject has been put into rotational motion, the neurally active membranes in the canals slowly relax back to resting position, so the canals stop sensing motion. Thus in the plot above, after the stimulation undergoes transition edges, the eye movements tend to depart from the stimulation over time. Based on mechanical characteristics of the canals, combined with additional neural processing which prolongs this time constant to improve the accuracy of the VOR, we will estimate the Tc parameter to be 15 seconds, and assume it is between 10 and 30 seconds.

```
Tc = sdo.getParameterFromModel(model_name, 'Tc');
Tc.Value = 15;
Tc.Minimum = 10;
Tc.Maximum = 30;
```

Finally, the Tp parameter models the dynamics of the oculomotor plant, i.e. the eye and the muscles and tissues attached to it. The plant can be modeled by two poles, however it is believed that the pole with the larger time constant is cancelled by precompensation in the brain, to enable the eye to make quick movements. Thus in the plot, when the stimulation undergoes transition edges, the eye movements follow with only a little delay. For the Tp parameter, we will use 0.01 seconds as our initial guess, and assume it is between 0.005 and 0.05 seconds.

```
Tp = sdo.getParameterFromModel(model_name, 'Tp');
Tp.Value = 0.01;
Tp.Minimum = 0.005;
Tp.Maximum = 0.05;
```

Collect these parameters into a vector.

v = [Delay Gain Tc Tp];

Compare Measured Data to Initial Simulated Output

Create an Experiment object. Specify HeadData as input.

```
Exp = sdo.Experiment(model_name);
Exp.InputData = timeseries(HeadData, Time);
```

Associate eye movement data with model output.

```
EyeMotion = Simulink.SimulationData.Signal;
EyeMotion.Name = 'EyeMotion';
EyeMotion.BlockPath = [model_name '/Oculomotor Plant'];
EyeMotion.PortType = 'outport';
EyeMotion.PortIndex = 1;
EyeMotion.Values = timeseries(EyeData, Time);
```

Add EyeMotion to the experiment.

Exp.OutputData = EyeMotion;

Use the data's timing characteristics in the model.

```
stop_time = Time(end);
set_param(gcs, 'StopTime', num2str(stop_time));
dt = Time(2) - Time(1);
set_param(gcs, 'FixedStep', num2str(dt))
```

Create a simulation scenario using the experiment, and obtain the simulated output.

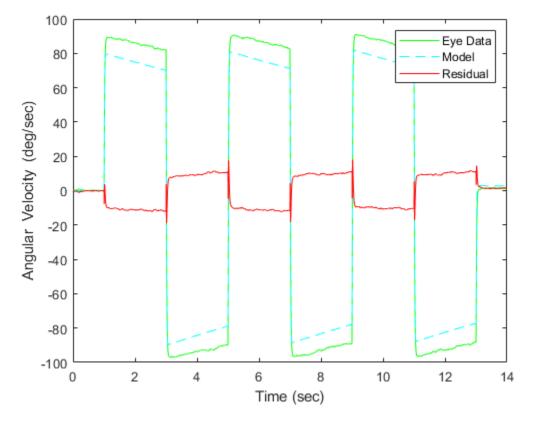
```
Exp = setEstimatedValues(Exp, v); % use vector of parameters/states
Simulator = createSimulator(Exp);
Simulator = sim(Simulator);
```

Search for the model_residual signal in the logged simulation data.

```
SimLog = find(Simulator.LoggedData, ...
get_param(model_name, 'SignalLoggingName') );
EyeSignal = find(SimLog, 'EyeMotion');
```

The model output does not match the data very well, as shown by the residual, which we can compute by calling the objective function.

```
estFcn = @(v) sdoVOR_Objective(v, Simulator, Exp, 'Residuals');
Model_Error = estFcn(v);
plot(Time, EyeData, '-g', ...
EyeSignal.Values.Time, EyeSignal.Values.Data, '--c', ...
Time, Model_Error.F, '-r');
xlabel('Time (sec)');
ylabel('Angular Velocity (deg/sec)');
legend('Eye Data', 'Model', 'Residual');
```



The objective function used above is defined in the file "sdoVOR Objective.m".

```
type sdoVOR_Objective.m
```

```
function vals = sdoVOR_Objective(v, Simulator, Exp, Method)
% Compare model output with data
%
%
     Inputs:
%
        v - vector of parameters and/or states
%
        Simulator - used to simulate the model
%
        Exp - Experiment object
        Method - 'SSE' for scalar output, 'Residuals' for vector of residuals
%
% Copyright 2014-2015 The MathWorks, Inc.
% Requirement setup
req = sdo.requirements.SignalTracking;
req.Type = '==';
req.Method = Method;
% If Residuals requested, keep on same scale as signals, for plotting
switch Method
    case 'Residuals'
        req.Normalize = 'off';
end
% Simulate the model
```

```
Exp = setEstimatedValues(Exp, v); % use vector of parameters/states
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
% Compare model output with data
SimLog = find(Simulator.LoggedData, ...
get_param(Exp.ModelName, 'SignalLoggingName') );
OutputModel = find(SimLog, 'EyeMotion');
Model_Error = evalRequirement(req, OutputModel.Values, Exp.OutputData.Values);
vals.F = Model_Error;
```

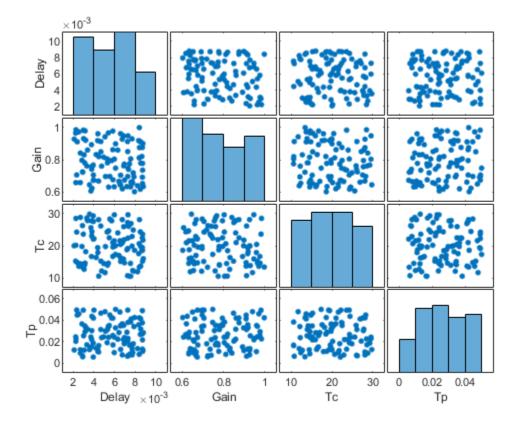
Sensitivity Analysis

Create an object to sample the parameter space.

```
ps = sdo.ParameterSpace([Delay ; Gain ; Tc ; Tp]);
```

Generate 100 samples from the parameter space.

```
rng default; % for reproducibility
x = sdo.sample(ps, 100);
sdo.scatterPlot(x);
```



The sampling above used default options, and these are reflected in the plots above. Parameter values were selected at random from distributions that were uniform over the range of each parameter. Consequently, the histogram plots along the diagonal appear approximately uniform. If Statistics and Machine Learning Toolbox[™] is available, many other distributions may be used, and sampling can be done using Sobol or Halton low-discrepancy sequences.

The off-diagonal plots above show scatter plots between pairs of different variables. Since we did not specify a RankCorrelation matrix in ps, the scatter plots do not indicate correlations. However, if parameters were believed to be correlated, this can be specified using the RankCorrelation property of ps.

For sensitivity analysis, it is simpler to use a scalar objective, so we will specify the sum of squared errors, "SSE":

```
estFcn = @(v) sdoVOR_Objective(v, Simulator, Exp, 'SSE');
y = sdo.evaluate(estFcn, ps, x);
```

```
Model evaluated at 100 samples.
```

Evaluation could also be sped up using parallel computing.

Obtain the standardized regression coefficients.

```
opts = sdo.AnalyzeOptions;
opts.Method = 'StandardizedRegression';
sensitivities = sdo.analyze(x, y, opts);
```

Other types of analysis include correlation and, if Statistics and Machine Learning Toolbox is available, partial correlation.

We can view the analysis results.

disp(sensitivities)

	F
Delay	0.01303
Gain	-0.90873
Tc	-0.044395
Tp	0.19919

For standardized regression, parameters that highly influence the model output have sensitivity magnitudes close to 1. On the other hand, less influential parameters have smaller sensitivity magnitudes. We see that this objective function is sensitive to changes in the Gain and Tp parameters, but much less sensitive to changes in the Delay and Tc parameters.

You can validate sensitivity analysis results by resampling and reevaluating the objective function for the samples. You can also use engineering intuition for a quick analysis. For example, in this model, the time constant Tc ranges from 10 to 30 seconds. Even the minimum value of 10 seconds is large compared to the 2-second duration for which the head motion stimulation is held at constant velocity. Therefore, Tc is not expected to affect the output greatly. However, even when this kind of intuition is not readily available in other models, sensitivity analysis can help highlight which parameters are influential.

Based on the results of sensitivity analysis, designate the Delay and Tc parameters as fixed when optimizing. This reduction in the number of free parameters speeds up optimization.

Delay.Free = false; Tc.Free = false;

Optimization

We can use the minimum from sensitivity analysis as the initial guess for optimization.

```
[fval, idx_min] = min(y.F);
Delay.Value = x.Delay(idx_min);
Gain.Value = x.Gain(idx_min);
Tc.Value = x.Tc(idx_min);
Tp.Value = x.Tp(idx_min);
%
v = [Delay Gain Tc Tp];
opts = sdo.OptimizeOptions;
opts.Method = 'fmincon';
```

As was the case with model evaluations in sensitivity analysis, parallel computing could be used to speed up the optimization.

```
vOpt = sdo.optimize(estFcn, v, opts);
disp(vOpt)
```

```
Optimization started 02-Sep-2021 00:37:17
```

			max		First-order
Iter	F-count	f(x)	constraint	Step-size	optimality
0	5	13.4798	Θ		
1	18	12.2052	Θ	0.129	305
2	30	11.1441	Θ	0.0648	790
3	41	10.0493	Θ	0.0843	290
4	46	9.23607	Θ	0.0758	286
5	51	8.76122	Θ	0.0183	10.1
6	56	8.75862	Θ	0.00184	0.476
7	57	8.75862	Θ	8.41e-05	0.476
		nearthle Co	notrointo col	Licfied	

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than the value of the step size tolerance and constraints are satisfied to within the value of the constraint tolerance.

```
(1,1) =
```

```
Name: 'Delay'
Value: 0.0038
Minimum: 0.0020
Maximum: 0.0090
Free: 0
Scale: 0.0078
Info: [1x1 struct]
```

(1,2) =

Name:	'Gain'
Value:	0.9012
Minimum:	0.6000
Maximum:	1
Free:	1
Scale:	1

```
Info: [1x1 struct]
(1,3) =
      Name: 'Tc'
      Value: 16.6833
   Minimum: 10
   Maximum: 30
       Free: 0
      Scale: 16
      Info: [1x1 struct]
(1, 4) =
      Name: 'Tp'
      Value: 0.0157
   Minimum: 0.0050
   Maximum: 0.0500
       Free: 1
      Scale: 0.0156
      Info: [1x1 struct]
```

1x4 param.Continuous

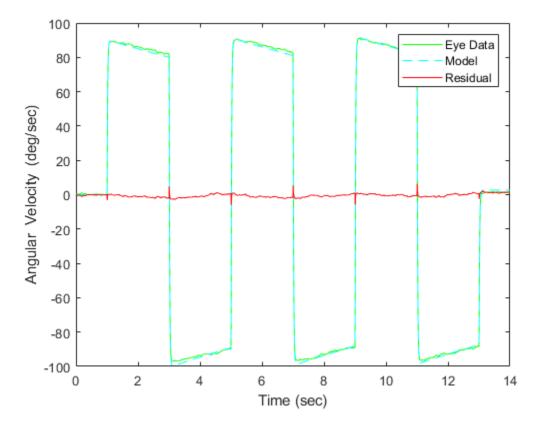
Visualizing Result of Optimization

Obtain the model response after estimation. Search for the model_residual signal in the logged simulation data.

```
Exp = setEstimatedValues(Exp, vOpt);
Simulator = createSimulator(Exp,Simulator);
Simulator = sim(Simulator);
SimLog = find(Simulator.LoggedData, ...
get_param(model_name, 'SignalLoggingName') );
EyeSignal = find(SimLog, 'EyeMotion');
```

Comparing the measured eye data with the optimized model response shows that the residuals are much smaller.

```
estFcn = @(v) sdoVOR_Objective(v, Simulator, Exp, 'Residuals');
Model_Error = estFcn(vOpt);
plot(Time, EyeData, '-g', ...
EyeSignal.Values.Time, EyeSignal.Values.Data, '--c', ...
Time, Model_Error.F, '-r');
xlabel('Time (sec)');
ylabel('Angular Velocity (deg/sec)');
legend('Eye Data', 'Model', 'Residual');
```



Close the model.

bdclose(model_name)

See Also

Related Examples

- "Identify Key Parameters for Estimation (GUI)" on page 4-131
- "Use Parallel Computing for Sensitivity Analysis" on page 4-104
- "Use Fast Restart Mode During Sensitivity Analysis" on page 4-109
- "Generate Parameter Samples for Sensitivity Analysis" on page 4-8
- "Analyze Relation Between Parameters and Design Requirements" on page 4-67
- "Validate Sensitivity Analysis" on page 4-96

Generate MATLAB Code for Sensitivity Analysis Statistics to Identify Key Parameters (GUI)

This example shows how to automatically generate a MATLAB function to solve a Sensitivity Analysis statistics problem. You use the **Sensitivity Analyzer** to define a sensitivity statistics problem for a model of the body's vestibulo-ocular reflex, and generate MATLAB® code to solve this statistics problem.

Vestibulo-Ocular Reflex Sensitivity Statistics Problem

The "Identify Key Parameters for Estimation (GUI)" on page 4-131 example shows how to use the **Sensitivity Analyzer** to compute sensitivity statistics for different parameter values in a model of the body's vestibulo-ocular reflex. In this example, we load a preconfigured **Sensitivity Analyzer** session based on that example.

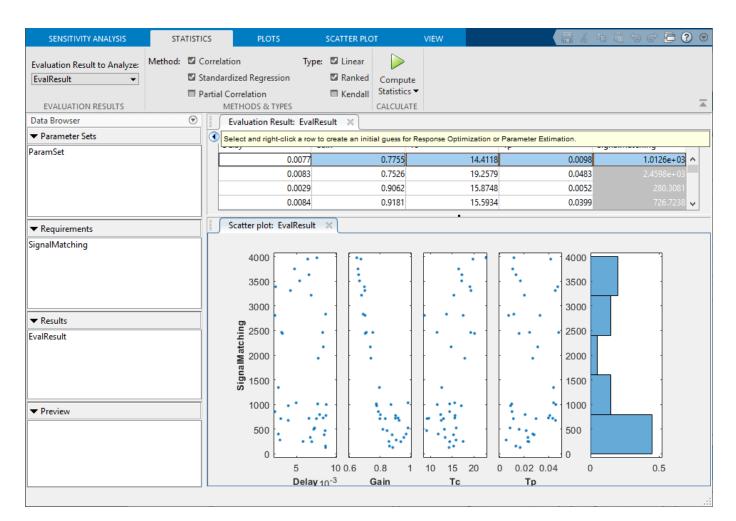
Open the **Sensitivity Analyzer** for the sdoVOR model:

ssatool('sdoVOR')

In the **Sensitivity Analyzer**, click **Open Session** and **Open from model workspace**. Open session sdoVOR_sasessionForSensitivityStatistics.

Open Session ×	1
Open from Simulink model workspace	
Existing Session Variables	
sdoVOR_sasessionForEvaluation	
sdoVOR_sasessionForSensitivityStatistics	
Session: sdoVOR_sasessionForSensitivityStatistics	
OK Cancel Help	
	;

This opens a preconfigured session in the **Sensitivity Analyzer**.



Generate MATLAB Code

From the **Compute Statistics** list, select Generate MATLAB Function.

SENSITIVITY ANALYSIS	STATIS	TICS	PLOTS	SCATTER PLO	от	VIEW				t i t d	. 🔁 🕄 🖉
Evaluation Result to Analyze:	Method:	Correlatio	on	Type: 🗹 Linear							
EvalResult 💌		Standard	ized Regression	🗹 Ranked	Comp	ute					
		Partial Co	orrelation	🗖 Kendall	Statisti	cs 👻			_		_
EVALUATION RESULTS		ME	ETHODS & TYPES			Compute Stati	istics				<u> </u>
Data Browser 💿		Σ Ε	Evaluation Result: EvalResult 🛛 📄 Compute influer			eters					
▼ Parameter Sets		Se Se	elect and right-click	a row to create an ini	tia	on requirements and store results.		on.			
ParamSet						c					
			0.0	077		Generate MAT		-	0.0098	1.01	26e+03 ^
			0.0	083		Generate a MAT the defined sen			0.0483		98e+03
			0.0	029		The defined sen	servicy analysi	is storistics.	0.0052		80.3081
			0.0	084	0.9181		15.5934		0.0399		26.7238 🗸

The generated code is added to the MATLAB editor as an unsaved MATLAB function.

```
Editor - Untitled2*
                                                                                                       Θ×
        function sensitivity = sensitivityStatistics(x,y)
1
                                                                                                        ~
2
      SENSITIVITYSTATISTICS
3
 4
        % Compute sensitivity analysis statistics for the model.
 5
        8
 6
        % The function returns sensitivity analysis statistics, sensitivity,
7
        % indicating which parameters, x, have the most influence on the
8
        % requirements, y.
9
        2
10
         The input argument, x, defines the parameters. If omitted, the
11
        % parameters specified in the function body are used.
12
        8
13
       % Modify the function to change the analysis techniques.
14
        $
15
        % Auto-generated by SSATOOL on 09-Jul-2019 16:26:25.
16
       00
17
18
        %% Specify Analysis Variables
19
        2
20
        % Specify parameters and requirements.
21
        if (nargin < 1) || isempty(x)</pre>
22
           x = getData('x');
23
        end
24
25
        if (nargin < 2) || isempty(y)
           y = getData('y');
26
27
        end
28
29
        %% Statistics Options
30
31
        2
32
        % Specify options for statistical analysis
33
        opts = sdo.AnalyzeOptions;
        opts.Method = getData('opts Method');
34
        opts.MethodOptions = {'Linear', 'Ranked'};
35
36
37
        %% Compute Statistics
38
39
        % Call sdo.analyze with the parameters and requirements, to determine which
40
        % parameters most influence the requirements.
41
        sensitivity = sdo.analyze(x,y,opts);
  Untitled2* × +
```

Examine the generated code. Significant portions are:

- **Statistics Analysis Variables** Specify the inputs and outputs, to determine which inputs have the most influence on outputs.
- Statistics Options Specify the types of analyses to be computed.
- **Compute Statistics** Solve the sensitivity statistics problem using the sdo.analyze command.

Select Save from the MATLAB editor to save the generated function.

Run Generated Code

Run the generated function.

```
Command Window
  >> sensitivity = sensitivityStatistics
  sensitivity =
    struct with fields:
                      Correlation: [4×1 table]
                  RankCorrelation: [4×1 table]
           StandardizedRegression: [4×1 table]
      RankStandardizedRegression: [4×1 table]
  >> sensitivity.Correlation
  ans =
    4×1 table
                SignalMatching
      Delay
                   -0.20052
      Gain
                   -0.88175
                    0.51264
      Тс
                   0.046374
      Tр
f_{\underline{x}} >>
```

The computation shows the result of analyzing which inputs have the most influence on the output. For example, the Correlation field shows that the Gain parameter has the largest magnitude correlation with the output, and in a negative direction, meaning that when Gain increases, the output decreases.

 \odot

Modify the Generated Code

You can:

- Modify the specified input and output variables.
- Modify the options to change the types of analyses computed.

See Also

More About

• "Generate MATLAB Code for Sensitivity Analysis for Design Space Exploration and Evaluation (GUI)" on page 4-183

Generate MATLAB Code for Sensitivity Analysis for Design Space Exploration and Evaluation (GUI)

This example shows how to automatically generate a MATLAB function to solve a Sensitivity Analysis evaluation problem. You use the **Sensitivity Analyzer** to define an evaluation problem for a model of the body's vestibulo-ocular reflex, and generate MATLAB® code to solve this evaluation problem.

Vestibulo-Ocular Reflex Evaluation Problem

The "Identify Key Parameters for Estimation (GUI)" on page 4-131 example shows how to use the **Sensitivity Analyzer** to evaluate a cost function for different parameter values in a model of the body's vestibulo-ocular reflex. In this example we load a pre-configured **Sensitivity Analyzer** session based on that example.

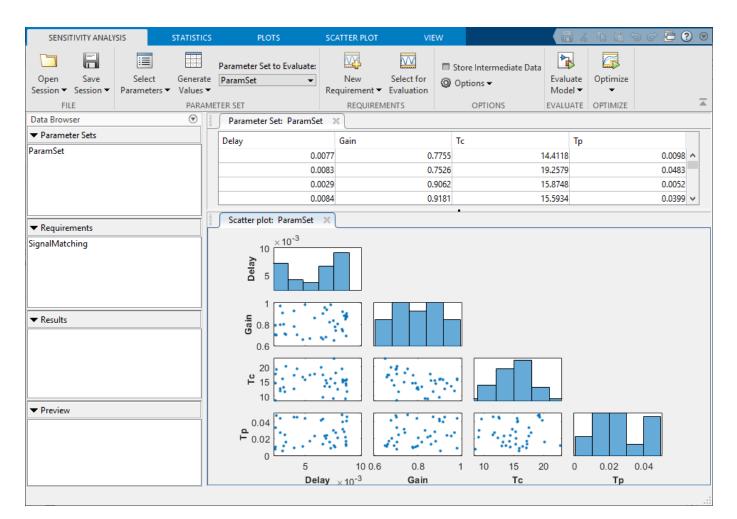
Open the **Sensitivity Analyzer** for the sdoVOR model:

ssatool('sdoVOR')

In the **Sensitivity Analyzer**, click **Open Session** and **Open from model workspace**. Open session sdoVOR_sasessionForEvaluation.

Open Session ×
Open from Simulink model workspace
Existing Session Variables
sdoVOR_sasessionForEvaluation
sdoVOR_sasessionForSensitivityStatistics
Session: sdoVOR_sasessionForEvaluation
OK Cancel Help

This opens a preconfigured session in the **Sensitivity Analyzer**.



Generate MATLAB Code

From the Evaluate Model list, select Generate MATLAB Function.

SENSIT	FIVITY ANALY	SIS	STATISTICS	PLOTS	SCATTER PLOT	VIE	w		8.	(覧 ਛੇ ↔) ¢ 🗗 🕐 🖲	
Open Session ▼	Save Session 🕶	Select Parameters 🗸	Generate Values 🕶	Parameter Set to Evaluate: ParamSet 🔹	New Requirement •	Select for Evaluation	■ Store Interme Ø Options •	E	valuate 1odel 🕶			
FILE PAR			PARAM	ETER SET	REQUIRE	MENTS	OPTIONS		F	valuate Mode	'	
Data Browser 💿			Parameter Set: ParamSet 🛛 🗶				Evaluate model and requirements for each			or each		
▼ Parameter Sets			Delay	Gain		Tc		row in the parameter set, and store		result.		
ParamSet					0077	0.7	7755	14	G	ienerate MA1	FLAB Function	
			0.	0.0083 0.7526		19 •	19 📣 Generate a MATLAB function to					
			0.	0029 0.9062		15	S	olve the defin	ed model evaluations	s.		
			0.	0084	0.9	9181	15.5	934		0.0399 🗸		

The generated code is added to the MATLAB editor as an unsaved MATLAB function.

📝 Editor -	Untitled*
1	function [EvalResult, Info] = sensitivityEvaluationSdoVOR(p)
2	*SENSITIVITYEVALUATIONSDOVOR
3	8
4	% Solve a design exploration problem for the sdoVOR model.
5	8
6	% The function returns model results, EvalResult,
7	% and model evaluation information, Info.
8	8
9	% The input argument, p, defines the model parameters to vary,
10	% if omitted the parameters specified in the function body are varied.
11	8
12	% Modify the function to include or exclude new design requirements, or
13	% to change the evaluation options.
14	8
15	% Auto-generated by SDOTOOL on 09-Jul-2019 13:37:12.
16	- %
17	
18	%% Open the model.
19	open_system('sdoVOR')
20	
21	%% Specify Design Variables
22	90 · · · · · · · · · · · · · · · · · · ·
23	% Specify model parameters as design variables to vary.
24	if nargin < 1 isempty(p)
25	<pre>p = sdo.getParameterFromModel('sdoVOR', {'Delay', 'Gain', 'Tc', 'Tp'});</pre>
26	p(1).Value = 0.0055;
27	p(1).Minimum = 0.002;
28 29	p(1).Maximum = 0.00899999999999999; p(2).Minimum = 0.6;
	p(2).mmmum = 0.6;
<	
Untitle	id* × +

Examine the generated code. Significant code portions are:

- Specify Design Variables Definition of the model parameters being varied.
- **Define the Experiments (Signal Matching Requirements)** Definition of the measured and expected signal data to use for signal matching requirements. In this case the signal matching requirement is the only requirement. In other cases, there may be other requirements such as signal bounds.
- **Create the Objective Function** Creation of an anonymous function that calls the subfunction sdoVOR_evalFcn, which evaluates the model using each experiment and compares simulation and measured experiment outputs. This anonymous function is called by sdo.evaluate at each iteration of the evaluation problem to evaluate the model at all combinations of parameters.
- Evaluate the Model Solve the evaluation problem using the sdo.evaluate command.

Select ${\bf Save}$ from the MATLAB editor to save the generated function.

Run Generated Code

Run the generated function.

```
📣 Command Window
                                                                                                   \times
                                                                                                         •
  >> [EvalResult, Info] = sensitivityEvaluationSdoVOR
  EvalResult =
    40×1 table
      SignalMatching
          1012.6
          2459.8
          280.31
          726.72
          3946.5
          1344.6
          975.66
          250.95
          128.29
          777.55
          2462.3
fx
           999.2
```

The first output argument, EvalResult, contains the result of evaluating the model at each combination of parameter values. The second output argument, Info, contains information about each evaluation.

Modify the Generated Code

You can:

- Modify the generated sensitivityEvaluationSdoVOR function to include or exclude new experiments or other requirements, or change evaluation options.
- Call the generated sensitivityEvaluationSdoVOR function with a different set of parameter values to evaluate.

For details on how to write an objective/constraint function to use with the sdo.evaluate command, type help sdoExampleCostFunction at the MATLAB command prompt.

Close the model.

See Also

More About

• "Generate MATLAB Code for Sensitivity Analysis Statistics to Identify Key Parameters (GUI)" on page 4-179

Optimization-Based Control Design

- "Time-Domain Design Requirements in Simulink" on page 5-2
- "Frequency-Domain Design Requirements in Simulink" on page 5-9
- "Time- and Frequency-Domain Requirements in Control System Designer App" on page 5-22
- "Time-Domain Simulations in Control System Designer App" on page 5-25
- "Design Optimization-Based Controllers for LTI Systems" on page 5-26
- "Optimize LTI System to Meet Frequency-Domain Requirements" on page 5-27
- "Design Linear Controllers for Simulink Models" on page 5-46
- "Enforcing Time and Frequency Requirements on a Single-Loop Controller Design" on page 5-48
- "Airframe Controller Tuning" on page 5-63
- "DC Motor Controller Tuning" on page 5-65
- "Hydraulic Piston Regulator Tuning" on page 5-67

Time-Domain Design Requirements in Simulink

Specify Piecewise-Linear Lower and Upper Bounds

To specify upper and lower bounds on a signal:

- 1 In the **Response Optimizer**, select **Signal Bound** in the **New** drop-down list. A window opens where you specify upper or lower bounds on a signal.
- 2 Specify a requirement name in the **Name** box.
- **3** Select the requirement type using the **Type** list.
- 4 Specify the edge start and end times and corresponding amplitude in the **Time (s)** and **Amplitude** columns.
- 5 Click to specify additional bound edges.

Select a row and click **to** delete a bound edge.

6 In the **Select Signals to Bound** area, select a logged signal to apply the requirement to.

If you have already selected signals, as described in "Specify Signals to Log" on page 3-10, they appear in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal	1
		OK Cancel	Help
			:

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **i** to add it to the signal set.
- d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

7 Click OK.

A variable with the specified requirement name appears in the **Data** area of the app. A graphical display of the requirement also appears in the **Response Optimizer** app window.

- 8 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can add a Check Custom Bounds block to your model to specify piecewise-linear bounds.

Specify Step Response Characteristics

To apply a step response requirement to a signal in your model, specify the step response characteristics as follows:

1 Select a step response requirement from the **Response Optimizer**.

In the **New** drop-down menu of the app, in the **New Time Domain Requirement** section, select **Step Response Envelope**.

A Create Requirement dialog box opens where you specify the step response requirements on a signal.

- 2 Specify a requirement name in the **Name** field of the dialog box.
- **3** Specify the step response characteristics:
 - Initial value Input level before the step occurs
 - Step time Time at which the step takes place
 - Final value Input level after the step occurs
 - **Rise time** The time taken for the response signal to reach a specified percentage of the step range. The step range is the difference between the final and initial values.
 - % **Rise** The percentage of the step range used with **Rise time** to define the overall rise time characteristics.
 - **Settling time** Time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
 - % **Settling** The percentage of the final value that defines the settling range of settling time characteristic specified in **Settling time**.
 - **% Overshoot** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
 - % Undershoot The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- 4 Specify the signal to be bound.

To apply this requirement to a model signal, in the **Select Signals to Bound** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . The Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

Alternatively, you can use the Check Step Response Characteristics block to specify step response bounds for a signal.

See Also

"Design Optimization to Meet Step Response Requirements (GUI)"

Track Reference Signals

Use reference tracking to force a model signal to match a desired signal. To track a reference signal:

- 1 In the **Response Optimizer**, select **Signal Tracking** in the **New** drop-down list. A window opens where you specify the reference signal to track.
- 2 Specify a requirement name in the **Name** box.
- **3** Define the reference signal by entering vectors, or variables from the workspace, in the **Time vector** and **Amplitude** fields.

Click **Update reference signal data** to use the new amplitude and time vector as the reference signal.

4 Specify how the optimization solver minimizes the error between the reference and model signals using the **Tracking Method** list:

- SSE Reduces the sum of squared errors
- SAE Reduces the sum of absolute errors
- 5 In the **Specify Signal to Track Reference Signal** area, select a logged signal to apply the requirement to.

If you already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, they appear in the list. Select the corresponding check-box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			×
Signal set: Sig]		
No simple have a smaller have a locked		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

e Select the check-box corresponding to the signal and click **OK**.

A variable with the specified requirement name appears in the **Data** area of the app. A graphical display of the signal bound also appears in the **Response Optimizer** app window.

Note When tracking a reference signal, the software ignores the maximally feasible solution option. For more information on this option, in the **Response Optimization** tab, click **Options** > **Optimization Options**, and click **Help**.

Alternatively, you can use the Check Against Reference block to specify a reference signal to track.

See Also

"Design Optimization to Track Reference Signal (GUI)"

Specify Custom Requirements

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

- 1 In the **Response Optimizer**, in **New** drop-down menu, select **Custom Requirement**. The Create Requirement dialog box opens where you specify the custom requirement.
- 2 Specify a requirement name in Name.
- **3** Specify the requirement type in the **Type** drop-down menu.
- **4** Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click



to review or edit the function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

5 (Optional) To prevent the solver from considering specific parameter combinations, select Error if constraint is violated. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next iteration.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current iteration. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

For more information, see "Skip Model Simulation Based on Parameter Constraint Violation (GUI)" on page 3-163.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

6 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

^a Click . A Create Signal Set dialog box opens where you specify the logged signal.

b In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			Х
Signal set: Sig			
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•	Signal	1
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In Signal set field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**. A graphical display of the requirement also appears in the **Response Optimizer** app window.

See Also

- "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103
- "Design Optimization to Meet a Custom Objective (Code)" on page 3-118

Edit Design Requirements

The Edit Design Requirement dialog box allows you to exactly position constraint segments and to edit other properties of these constraints. The dialog box has two main components:

- An upper panel to specify the constraint you are editing
- A lower panel to edit the constraint parameters

The upper panel of the Edit Design Requirement dialog box resembles the image in the following figure.

Design requirement:	Upper time response bound from 0 to 10 sec	▼
---------------------	--------------------------------------------	---

In the **Control System Designer** app in Control System Toolbox, you can edit design requirements from the analysis plots. The **Design requirement** drop-down list will contain all the requirements on that plot.

Edit Design Requirement Dialog Box Parameters

The particular parameters shown within the lower panel of the Edit Design Requirement dialog box depend on the type of constraint/requirement. In some cases, the lower panel contains a grid with one row for each segment and one column for each constraint parameter. The following table summarizes the various constraint parameters.

Edit Design Requirement Dialog Box Parameters

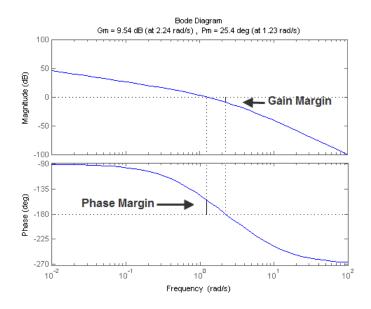
Parameter	Found in	Description			
Time	Upper and lower time response bounds on step and impulse response plots	Defines the time range of a segment within a constraint/requirement.			
Amplitude	Upper and lower time response bounds on step and impulse response plots	Defines the beginning and ending amplitude of a constraint segment.			
bounds segment. It is an alternat specifying the magnitude new Slope value changes		Defines the slope, in 1/s, of a constraint segment. It is an alternative method of specifying the magnitude values. Entering a new Slope value changes any previously defined magnitude values.			
Final value Step response bounds		Defines the input level after the step occurs.			
Rise time	Step response bounds	Defines a constraint segment for a particular rise time.			
% Rise	Step response bounds	The percentage of the step's range used to describe the rise time.			
Settling time	Step response bounds	Defines a constraint segment for a particular settling time.			
% Settling	Step response bounds	The percentage of the final value that defines the settling region used to describe the settling time.			
% Overshoot	Step response bounds The percentage amount by which can exceed the final value before				
% Undershoot	Step response bounds	Defines the constraint segments for a particular percent undershoot.			

Frequency-Domain Design Requirements in Simulink

Specify Lower Bounds on Gain and Phase Margin

To specify lower bounds on the gain and phase margin of a linear system:

- 1 In the **Response Optimizer**, select **Gain and Phase Margin** in the **New** list. A window opens where you specify lower bounds on the gain and phase margin of your linear system.
- 2 Specify a requirement name in Name.
- **3** Specify bounds on the gain margin or phase margin, or both.



- **Gain margin** Amount of gain increase or decrease required to make the loop gain unity at the frequency where the phase angle is -180°.
- **Phase margin** Amount of phase increase or decrease required to make the phase angle 180° when the loop gain is 1.0

To specify a lower bound on the gain margin or phase margin, or both, select the corresponding check box and enter the lower bound value.

4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- a Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A variable with the specified requirement name appears in the **Data** area of the app. A graphical display of the requirement also appears in the **Response Optimizer** app window.

- **6** (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Gain and Phase Margins block to specify bounds on the gain and phase margin. (Requires Simulink Control Design.)

Specify Piecewise-Linear Lower and Upper Bounds on Frequency Response

To specify upper or lower bounds on the magnitude of a system response:

- 1 In the **Response Optimizer**, select **Bode Magnitude** in the **New** list. A window opens where you specify the lower or upper bounds on the magnitude of the system response.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify the requirement type using the **Type** list.
- 4 Specify the edge start and end frequencies and corresponding magnitude in the **Frequency** and **Magnitude** columns.
- **5** Insert or delete bound edges.

Click to specify additional bound edges.

Select a row and click **to** delete a bound edge.

6 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

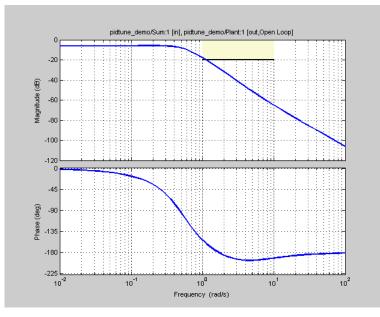
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app window. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 8 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Bode Characteristics block to specify bounds on the magnitude of the system response. (Requires Simulink Control Design.)

Specify Bound on Closed-Loop Peak Gain

To specify an upper bound on the closed-loop peak response of a system:

- 1 In the **Response Optimizer**, select **Closed-Loop Peak Gain** in the **New** list. A window opens where you specify an upper bound on the closed-loop peak gain of the system.
- 2 Specify a requirement name in the Name box.
- 3 Specify the upper bound on the closed-loop peak gain in the **Closed-Loop peak gain** box.
- 4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

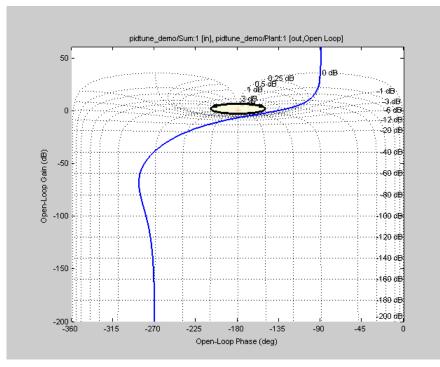
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app window. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- **6** (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Nichols Characteristics block to specify bounds on the magnitude of the system response. (Requires Simulink Control Design.)

Specify Lower Bound on Damping Ratio

To specify a lower bound on the damping ratio of the system:

- 1 In the **Response Optimizer**, select **Damping Ratio** in the **New** list. A window opens where you specify a lower bound on the damping ratio of the system.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify the lower bound on the damping ratio in the **Damping ratio** box.
- 4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

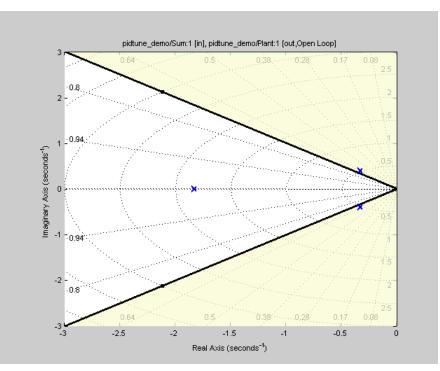
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 6 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Pole-Zero Characteristics block to specify a bound on the damping ratio. (Requires Simulink Control Design.)

Specify Upper and Lower Bounds on Natural Frequency

To specify a bound on the natural frequency of the system:

- 1 In the **Response Optimizer**, select **Natural Frequency** in the **New** list. A window opens where you specify a bound on the natural frequency of the system.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify a lower or upper bound on the natural frequency in the **Natural frequency** box.
- **4** In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

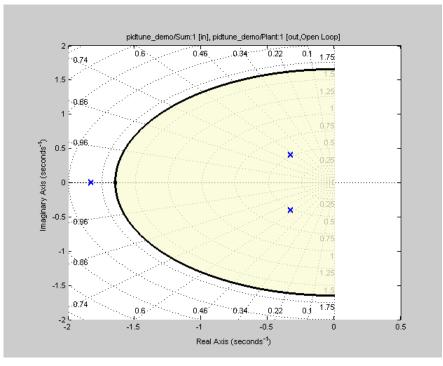
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 6 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Pole-Zero Characteristics block to specify a bound on the natural frequency. (Requires Simulink Control Design.)

Specify Upper Bound on Approximate Settling Time

To specify an upper bound on the approximate settling time of the system:

- 1 In the **Response Optimizer**, select **Settling Time** in the **New** list. A window opens where you specify an upper bound on the approximate settling time of the system.
- 2 Specify a requirement name in the Name box.

- 3 Specify the upper bound on the approximate settling time in the **Settling time** box.
- 4 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- a Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

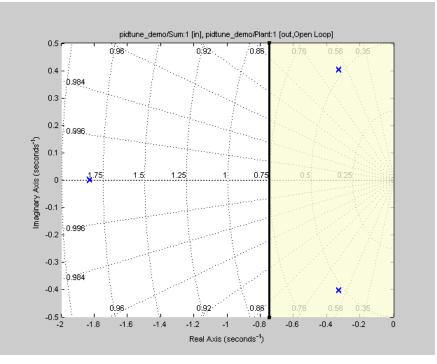
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

5 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



6 (Optional) In the graphical display, you can:

- "Move Constraints Graphically" on page 3-14
- "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Pole-Zero Characteristics block to specify the approximate settling time. (Requires Simulink Control Design.)

Specify Piecewise-Linear Upper and Lower Bounds on Singular Values

To specify piecewise-linear upper and lower bounds on the singular values of a system:

- 1 In the **Response Optimizer**, select **Singular Values** in the **New** list. A window opens where you specify the lower or upper bounds on the singular values of the system.
- 2 Specify a requirement name in the **Name** box.
- **3** Specify the requirement type using the **Type** list.
- 4 Specify the edge start and end frequencies and corresponding magnitude in the **Frequency** and **Magnitude** columns, respectively.
- **5** Insert or delete bound edges.

Click to specify additional bound edges.

Select a row and click to delete a bound edge.

6 In the **Select Systems to Bound** section, select the linear systems to which this requirement applies.

Linear systems are defined by snapshot times at which the model is linearized and sets of linearization I/O points defining the system inputs and outputs.

- **a** Specify the simulation time at which the model is linearized using the **Snapshot Times** box. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

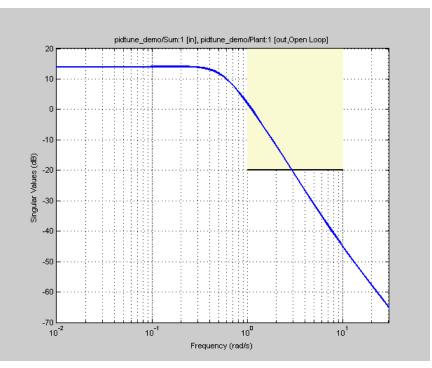
If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable with the specified name appears in the **Data** area of the **Response Optimizer** app. A graphical display of the requirement also appears in the **Response Optimizer** app window.



- 8 (Optional) In the graphical display, you can:
 - "Move Constraints Graphically" on page 3-14
 - "Position Constraints Exactly" on page 3-15

Alternatively, you can use the Check Singular Value Characteristics block to specify bounds on the singular value. (Requires Simulink Control Design).

Specify Step Response Characteristics

To apply a step response requirement to a linearization of your model (requires Simulink Control Design), specify the step response characteristics as follows:

1 Select a step response requirement from the **Response Optimizer**.

In the **New** drop-down menu of the app, in the **New Frequency Domain Requirement** section, select **Step Response Envelope**.

A Create Requirement dialog box opens where you specify the step response requirements.

- 2 Specify a requirement name in the **Name** field of the dialog box.
- **3** Specify the step response characteristics:
 - Initial value Input level before the step occurs
 - Step time Time at which the step takes place
 - Final value Input level after the step occurs
 - **Rise time** The time taken for the response signal to reach a specified percentage of the step range. The step range is the difference between the final and initial values.

- % **Rise** The percentage of the step range used with **Rise time** to define the overall rise time characteristics.
- **Settling time** Time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
- % **Settling** The percentage of the final value that defines the settling range of settling time characteristic specified in **Settling time**.
- **% Overshoot** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **% Undershoot** The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step range. The step range is the difference between the final and initial values.
- **4** Specify the systems to be bound.

To apply this requirement to a linearization of your Simulink model:

- **a** In the **Select Systems to Bound** area, specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
- **b** Select the linearization input/output set from the Linearization I/O area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box.

For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

Alternatively, you can use the Check Step Response Characteristics block to specify step response bounds for a signal.

Specify Custom Requirements

You can specify custom requirements, such as minimizing system energy. To specify custom requirements:

- 1 In the **Response Optimizer**, in **New** drop-down menu, select **Custom Requirement**. The Create Requirement dialog box opens where you specify the custom requirement.
- 2 Specify a requirement name in Name.
- **3** Specify the requirement type in the **Type** drop-down menu.
- 4 Specify the name of the function that contains the custom requirement in **Function**. The field must be specified as a function handle using @. The function must be on the MATLAB path. Click

1

to review or edit the function.

If the function does not exist, clicking opens a template MATLAB file. Use this file to implement the custom requirement. The default function name is myCustomRequirement.

5 (Optional) To prevent the solver from considering specific parameter combinations, select **Error if constraint is violated**. Use this option for parameter-only constraints.

During an optimization iteration, the solver first evaluates requirements with this option selected.

- If the constraint is violated, the solver skips evaluating any remaining requirements and proceeds to the next iteration.
- If the constraint is *not* violated, the solver evaluates the remaining requirements for the current iteration. If any of the remaining requirements bound signals or systems, then the solver simulates the model.

For more information, see "Skip Model Simulation Based on Parameter Constraint Violation (GUI)" on page 3-163.

Note If you select this check box, then do not specify signals or systems to bound. If you *do* specify signals or systems, then this check box is ignored.

6 (Optional) Specify the signal or system, or both, to be bound.

You can apply this requirement to model signals, or a linearization of your Simulink model (requires Simulink Control Design), or both.

Click **Select Signals and Systems to Bound (Optional)** to view the signal and linearization I/O selection area.

• To apply this requirement to a model signal:

In the **Signal** area, select a logged signal to which you will apply the requirement.

If you have already selected a signal to log, as described in "Specify Signals to Log" on page 3-10, it appears in the list. Select the corresponding check box.

If you have not selected a signal to log:

- ^a Click . A Create Signal Set dialog box opens where you specify the logged signal.
- **b** In the Simulink model window, click the signal to which you want to add a requirement.

Create Signal Set			Х
Signal set: Sig			
		Signal	
No signals have currently been selected. Please go back to the model and click on a signal to select it.	•		
		OK Cancel	Help

The Create Signal Set dialog box updates and displays the name of the block and the port number where the selected signal is located.

- **c** Select the signal and click **b** to add it to the signal set.
- d In **Signal set** field, enter a name for the selected signal set.

Click **OK**. A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**.

- To apply this requirement to a linear system:
 - **a** Specify the simulation time at which the model is linearized in **Snapshot Times**. For multiple simulation snapshot times, specify a vector.
 - **b** Select the linearization input/output set from the **Linearization I/O** area.

If you have already created a linearization input/output set, it appears in the list. Select the corresponding check box.

If you have not created a linearization input/output set, click to open the Create linearization I/O set dialog box. For more information on using this dialog box, see "Create Linearization I/O Sets" on page 3-64.

For more information on linearization, see "What Is Linearization?" (Simulink Control Design).

7 Click OK.

A new variable, with the specified name, appears in the **Data** area of the **Response Optimizer**. A graphical display of the requirement also appears in the **Response Optimizer** app window.

See Also

- "Design Optimization to Meet a Custom Objective (GUI)" on page 3-103
- "Design Optimization to Meet a Custom Objective (Code)" on page 3-118

Time- and Frequency-Domain Requirements in Control System Designer App

Root Locus Diagrams

Settling Time

If you specify a settling time in the continuous-time root locus, a vertical line appears on the root locus plot at the pole locations associated with the value provided (using a first-order approximation). In the discrete-time case, the constraint is a curved line.

It is required that $\operatorname{Re}\left\{pole\right\} < -4.6/T_{settling}$ for continuous systems and $\log(\operatorname{abs}(pole))/T_{discrete} < -4.6/T_{settling}$ for discrete systems. This is an approximation of the settling time based on second-order dominant systems.

Percent Overshoot

Specifying percent overshoot in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the percent value (using a second-order approximation). In the discrete-time case, the constraint appears as two curves originating at (1,0) and meeting on the real axis in the left-hand plane.

The percent overshoot *p.o* constraint can be expressed in terms of the damping ratio, as in this equation:

$$p.o. = 100e^{-\pi\zeta/\sqrt{1-\zeta^2}}$$

where ζ is the damping ratio.

Damping Ratio

Specifying a damping ratio in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the damping ratio. In the discrete-time case, the constraint appears as curved lines originating at (1,0) and meeting on the real axis in the left-hand plane.

The damping ratio defines a requirement on $-\text{Re}\{pole\}/\text{abs}(pole)$ for continuous systems and on

$$r = abs(pSys)$$

$$t = angle(pSys)$$

$$c = -\log(r)/\sqrt{(\log(r))^{2} + t^{2}}$$

for discrete systems.

Natural Frequency

If you specify a natural frequency, a semicircle centered around the root locus origin appears. The radius equals the natural frequency.

The natural frequency defines a requirement on *abs(pole)* for continuous systems and on

r = abs(pSys) t = angle(pSys) $c = \sqrt{(log(r))^{2} + t^{2}}/Ts_{model}$

for discrete systems.

Region Constraint

Specifies an exclusion region in the complex plane, causing a line to appear between the two specified points with a shaded region below the line. The poles must not lie in the shaded region.

Open-Loop and Prefilter Bode Diagrams

Gain and Phase Margins

Specify a minimum phase and or a minimum gain margin.

Upper Gain Limit

You can specify an upper gain limit, which appears as a straight line on the Bode magnitude curve. You must select frequency limits, the upper gain limit in decibels, and the slope in dB/decade.

Lower Gain Limit

Specify the lower gain limit in the same fashion as the upper gain limit.

Open-Loop Nichols Plots

Phase Margin

Specify a minimum phase amount.

While displayed graphically at only one location around a multiple of -180 degrees, this requirement applies to phase margin regardless of actual phase (i.e., it is interpreted for all multiples of -180).

Gain Margin

Specify a minimum gain margin.

While displayed graphically at only one location around a multiple of -180 degrees, this requirement applies to gain margin regardless of actual phase (i.e., it is interpreted for all multiples of -180).

Closed-Loop Peak Gain

Specify a peak closed-loop gain at a given location. The specified value can be positive or negative in dB. The constraint follows the curves of the Nichols plot grid, so it is recommended that you have the grid on when using this feature.

While displayed graphically at only one location around a multiple of -180 degrees, this requirement applies to gain margin regardless of actual phase (i.e., it is interpreted for all multiples of -180).

Gain-Phase Requirement

Specifies an exclusion region for the response on the Nichols plot. The response must not pass through the shaded region.

This only applies to the region (phase and gain) drawn.

Step/Impulse Response Plots

Upper Time Response Bound

You can specify an upper time response bound for step and impulse responses.

Lower Time Response Bound

You can specify a lower time response bound for step and impulse responses.

Step Response Bound

For a step response plot, you can also specify a step response bound design requirement.

To define a step response bound requirement, specify the following step response parameters:

- Final value Final steady-state value
- Rise time Time required to reach the specified percentage, % Rise, of the Final value.
- Settling time Time at which the response enters and stays within the settling percentage, % Settling, of the Final value.
- % Overshoot Maximum percentage overshoot above the Final value.
- % Undershoot Maximum percentage undershoot below the Initial value.

In the **Control System Designer** app, step response plots always use an **Initial value** and a **Step time** of 0.

See Also

Related Examples

- "Design Optimization-Based Controllers for LTI Systems" on page 5-26
- "Optimize LTI System to Meet Frequency-Domain Requirements" on page 5-27
- "Design Optimization-Based PID Controller for Linearized Simulink Model (GUI)"

Time-Domain Simulations in Control System Designer App

When performing optimization-based tuning in the **Control System Designer** app, Simulink Design Optimization software automatically sets the model simulation start and stop times; you cannot directly change them. By default, the simulation starts at 0 and continues until the app determines that the dynamics of the model have settled out. In addition, when the design requirements extend beyond this point, the simulation continues to the extent of the design requirements. Although you cannot directly adjust the start or stop time of the simulation, you can adjust the design requirements to extend further in time and thus force the simulation to continue to a certain point.

See Also

Related Examples

• "Optimize LTI System to Meet Frequency-Domain Requirements" on page 5-27

Design Optimization-Based Controllers for LTI Systems

This topic shows how to design optimization-based linear controllers for an LTI model.

To design an optimization-based linear controller:

- 1 Create and import a linear model into the **Control System Designer** app. You can create an LTI model at the MATLAB command line, as described in "Create an LTI Plant Model" on page 5-27.
- 2 Create a Control System Designer session, as described in "Open the Control System Designer App" on page 5-28.
- **3** In the **Tuning Methods** drop-down list, select **Optimization Based Tuning** to open the Response Optimization window. For more information, see "Open Optimization Based Tuning Method" on page 5-31.
- 4 In the Response Optimization window, select the **Compensators** tab to select and configure the compensator elements you want to tune during the response optimization. For more information, see "Select Tunable Compensator Elements" on page 5-33.
- 5 In the **Design requirements** tab, select the design requirements you want the system to satisfy. For more information, see "Add Design Requirements" on page 5-34.
- **6** In the Response Optimization window, click **Start Optimization**. The optimization progress results are displayed in the **Optimization** tab. The **Compensators** pane contains the new, optimized compensator element values. For more information, see "Optimize the System Response" on page 5-42.

See Also

Related Examples

• "Optimize LTI System to Meet Frequency-Domain Requirements" on page 5-27

Optimize LTI System to Meet Frequency-Domain Requirements

This example shows how to use frequency-domain design requirements to optimize the response of an LTI system in the **Control System Designer** app.

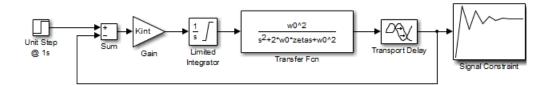
When used with Control System Toolbox software, you can place Simulink Design Optimization design requirements or constraints on plots in the **Control System Designer** app. You can include design requirements for response optimization in the frequency-domain and time-domain.

You can specify frequency-domain design requirements to optimize response signals for any model that you design in the **Control System Designer** app, such as:

- Command-line LTI models created with the Control System Toolbox commands
- Simulink models that have been linearized using Simulink Control Design software

Design Requirements

In this example, you use a linearized version of the Simulink model, srotut1.



You use optimization methods to design a compensator so that the closed-loop system meets the following design specifications when you excite the system with a unit step input:

- Maximum 30-second settling time
- Maximum 10% overshoot
- Maximum 10-second rise time
- Limit of ± 0.7 on the actuator signal

Create an LTI Plant Model

In the srotut1 model, the plant model is composed of a gain, a limited integrator, a transfer function, and a transport delay block.

Design the compensator for the open-loop transfer function of the linearized srotut1 model. The linearized srotut1 plant model is composed of the gain, an unlimited integrator, the transfer function, and a Padé approximation to the transport delay.

To create an open-loop transfer function based on the linearized srotut1 model, enter the following commands.

```
w0 = 1;
zeta = 1;
Kint = 0.5;
Tdelay = 1;
[delayNum,delayDen] = pade(Tdelay,1);
integrator = tf(Kint,[1 0]);
```

```
transfer_fcn = tf(w0^2,[1 2*w0*zeta w0^2]);
delay_block = tf(delayNum,delayDen);
open_loopTF = integrator*transfer_fcn*delay_block;
```

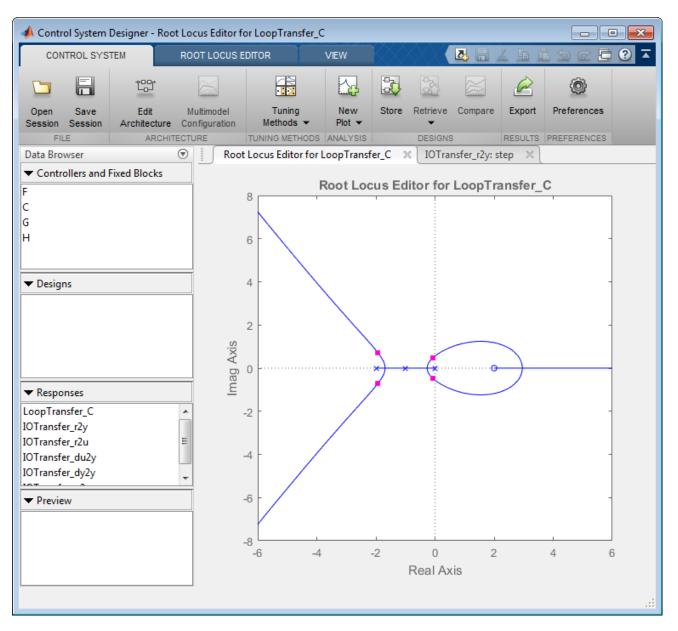
If the plant model is an array of models (Control System Toolbox), the controller is designed for a nominal model only. You can also analyze the control design for the remaining models in the array. For more information, see "Multimodel Control Design" (Control System Toolbox).

Tip You can directly linearize the Simulink model using Simulink Control Design software.

Open the Control System Designer App

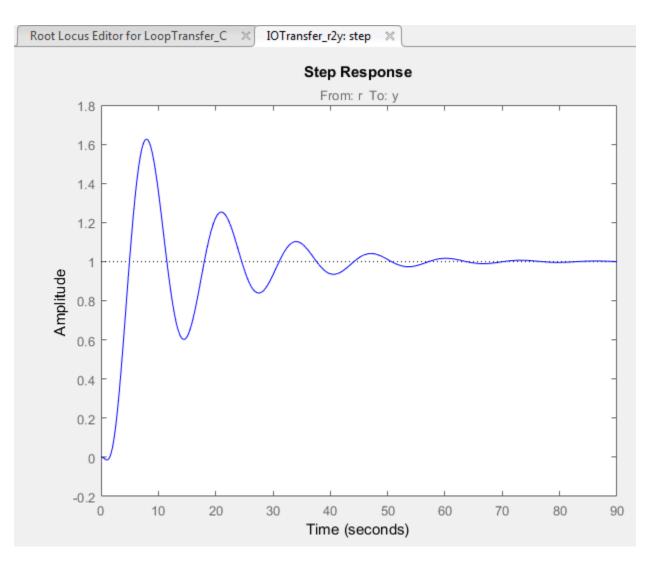
This example uses a root locus diagram to design the response of the open-loop transfer function, open_loopTF. To create a **Control System Designer** app session with a root locus plot for the open-loop transfer function, use the following command:

controlSystemDesigner('rlocus', open_loopTF)



The **Control System Designer** app opens, and a **Root Locus Editor** is displayed. The app lets you design controllers for single-input, single-output (SISO) systems in MATLAB and Simulink. For more information, see the "Classical Control Design" (Control System Toolbox) category.

The app also displays the step response plot of the system. The plot shows the response of the closed-loop system from r (input to the prefilter, F) to y (output of the plant model, G).



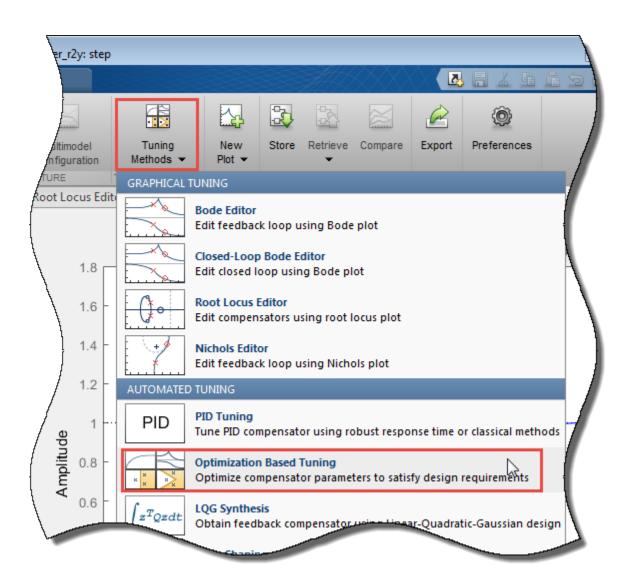
To choose the architecture for the control system you are designing, in the app click **Edit Architecture**. This example uses the default architecture. In this system, the plant model, *G*, is the open-loop transfer function open_loopTF. The prefilter, *F*, and the sensor, *H*, are set to 1, and the compensator, *C*, is the compensator that is designed using response optimization methods.

Edit Architecture - C	onfiguratio	on 1	>
Select Control Architecture:			du dy
╺━╍⋧ҹ━╍ᢕ┯╸	′ ►	F → _ +	C →Č [™] → G →Č [™] → ^y
╺━╍ <mark>┍</mark> ┓╴		ym	ұ⊷Н
			n
╶╬╋╋╬	Blocks	Loop Signs	
└─┸┍┙	Identifier	Block Name	Value
-■q∎ <mark>-</mark>	с	C	<1x1 zpk>
	F	F	<1x1 zpk>
	G	G	<1x1 tf>
	н	Н	<1x1 tf>
			OK Cancel Help

Open Optimization Based Tuning Method

There are several possible methods for designing a SISO system; this example uses an automated approach that uses response optimization methods.

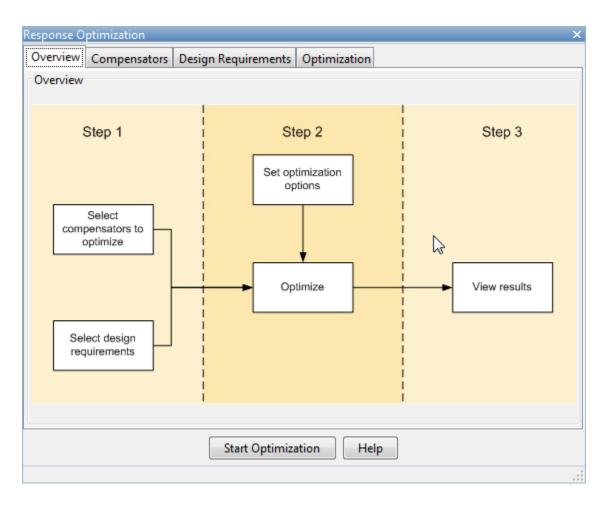
To create a response optimization task, in the **Tuning Methods** drop-down list, select Optimization Based Tuning.



The Response Optimization window has four tabs. Except for the first tab, each tab corresponds to a step in the response optimization process:

- **Overview** Schematic diagram of the response optimization process.
- **Compensators** Select and configure the compensator elements that you want to tune. See "Select Tunable Compensator Elements" on page 5-33.
- **Design requirements** Select the design requirements that you want the system to meet after tuning the compensator elements. See "Add Design Requirements" on page 5-34.
- **Optimization** Configure optimization options, and view the progress of the response optimization. See "Optimize the System Response" on page 5-42.

Note When optimizing responses in the app, you cannot add uncertainty to parameters or compensator elements.



Select Tunable Compensator Elements

You can tune compensator elements or parameters within compensators in your system to meet the design requirements you specify.

To specify the compensator elements to tune:

- **1** In the Response Optimization window, select the **Compensators** tab.
- 2 In the **Compensators** tab, select the check boxes in the **Optimize** column that correspond to the compensator elements to tune.

In this example, select **Gain** in the compensator **C**.

Response Optimi	zation						X
Overview Compensators Design Requirements Optimization							
Select compensators to optimize							
Optimize	ptimize Compensator elements Value Initial guess Minimum Maximum Typical value						
	F						
	Gain		1	1	-Inf	Inf	1
	C						
	Gain		1	1	-Inf	Inf	1
2							
						Use Value as I	initial guess
		Start C	Optimization	Help			

Add Design Requirements

You can use both frequency-domain and time-domain design requirements to tune parameters in a control system.

This example uses the design specifications described in "Design Requirements" on page 5-27. Create design requirements to meet these specifications:

- "Settling Time Design Requirement" on page 5-35
- "Overshoot Design Requirement" on page 5-36
- "Rise Time Design Requirement" on page 5-37
- "Actuator Limit Design Requirement" on page 5-39

After you add the design requirements, you can select a subset of requirements for controller design, as described in "Select the Design Requirements to Use During Response Optimization" on page 5-41. In the **Design requirements** tab of the Response Optimization window, you can create design requirements and select the requirements you want to use for optimization.

Settling Time Design Requirement

The first design requirement is to have a settling time of 30 seconds or less. This specification can be represented on a root locus diagram as a constraint on the real parts of the poles of the open-loop system.

To add the settling time design requirement:

1 In the **Design requirements** tab, click **Add new design requirement**. A New Design Requirement dialog box opens.

In this dialog box, you can specify new design requirements, and add them to a new or existing plot.

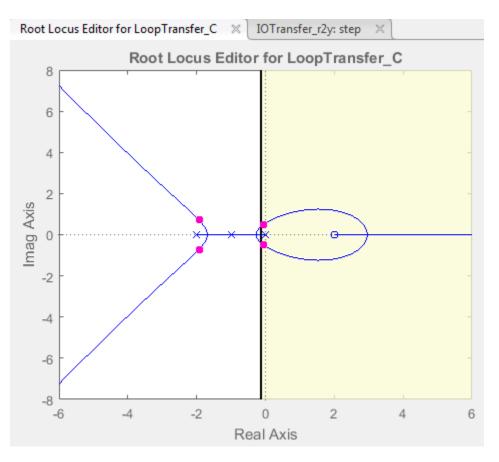
- 2 Add a design requirement to the existing root locus diagram.
 - a In the **Design requirement type** drop-down list, select Settling time.
 - **b** In the **Requirement for response** drop-down list, select LoopTransfer_C.
 - c Specify Settling time as 30 seconds.
 - d Click OK.

📣 New Design Requirement	t 🗖 🗖 🗾
Design requirement type:	Settling time
Requirement for response:	LoopTransfer_C
Design requirement param	neters
Settling time <	30 seconds
	OK Cancel Help

The settling time design requirement is listed in the **Design Requirements** tab of the Response Optimization window.

Response Optimization						
Overview Compensators Design Requirements Optimization						
ement to satisfy						
Optimize Response plot Design requirement						
LoopTransfer_C						
Root Locus Editor for LoopTransfer_C Settling time < 30 (seconds)						
	sators Design Requirements Optimization ement to satisfy Response plot LoopTransfer_C					

In the app, the design requirement appears on the root locus plot as a vertical line.



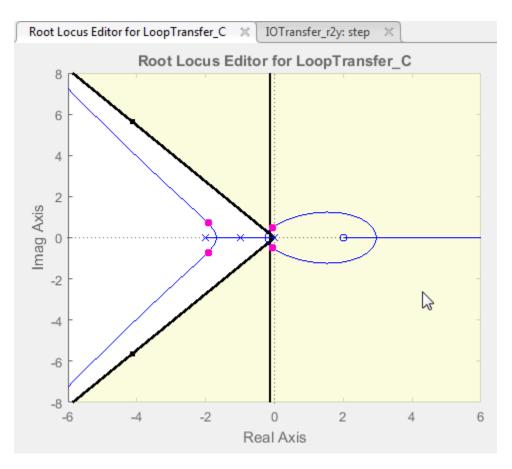
Overshoot Design Requirement

The second design requirement is to have a percentage overshoot of 10% or less. This requirement is related to the damping ratio on a root locus diagram. In addition to adding a design requirement with the **Add new design requirement** button, you can also right-click directly on the plots to add the requirement.

To add this design requirement:

- 1 In the **Control System Designer** app, right-click within the white space of the root-locus diagram. Select **Design Requirements** > **New** to open the New Design Requirement dialog box.
- 2 In the **Design requirement type** drop-down list, select Percent overshoot.
- **3** Specify **Percent overshoot** as **10**.
- 4 Click OK.

In the app, the design requirement appears on the root-locus plot as two lines radiating at an angle from the origin.

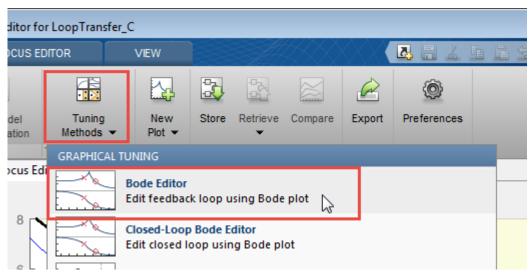


Rise Time Design Requirement

The third design requirement is to have a rise time of 10 seconds or less. This requirement corresponds to a lower limit on a Bode Magnitude diagram.

To add this design requirement:

1 In the app, in the **Tuning Methods** drop-down list, select Bode Editor.

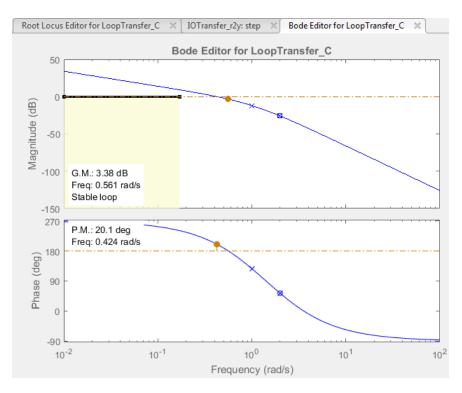


2 In the Select Response to Edit dialog box, specify **Select Response to Edit** as LoopTransfer_C, and click **Plot**.

A Bode plot is displayed in a **Bode Editor**.

- **3** Right-click within the white space of the open-loop Bode plot, and select **Design Requirements** > **New**, to open the New Design Requirement dialog box.
- 4 Specify the design requirement to represent the rise time, and add it to the new Bode plot.
 - a In the **Design requirement type** drop-down list, select Lower gain limit.
 - **b** Specify the **Frequency** range as 1e-2 to 0.17.
 - c Specify the **Magnitude** range as 0 to 0.
 - d Click OK.

The design requirement appears on the plot as a horizontal line.

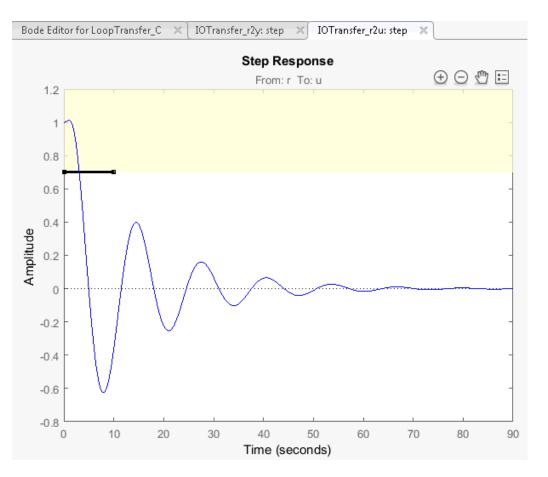


Actuator Limit Design Requirement

The fourth design requirement is to limit the actuator signal to within ± 0.7 .

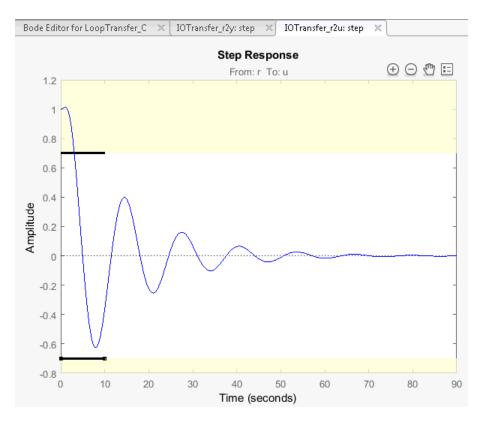
To add this design requirement:

- 1 In the Response Optimization window, in the **Design requirements**, click **Add new design requirement**. A New Design Requirement dialog box opens.
- 2 Create a time-domain design requirement to represent the upper limit on the actuator signal, and add it to a new step response plot:
 - a In the **Design requirement type** drop-down list, select Step response upper amplitude limit.
 - **b** In the **Requirement for response** drop-down list, select **IOTransfer_r2u**.
 - c Specify the **Time** range as 0 to 10.
 - d Specify the Amplitude range as 0.7 to 0.7.
 - e Click **OK**. A second step response plot for the closed-loop response from r to u is generated in the app. The plot contains a horizontal line representing the upper limit on the actuator signal.
 - **f** To extend this limit for all times (to $t = \infty$), right-click in the yellow shaded area, and select **Extend to inf**.



To add the corresponding design requirement for the lower limit on the actuator signal:

- 1 In the Response Optimization window, in the **Design requirements**, click **Add new design requirement**. A New Design Requirement dialog box opens.
- 2 Create a time-domain design requirement to represent the lower limit on the actuator signal, and add it to the step response plot:
 - a In the **Design requirement type** drop-down list, select Step response lower amplitude limit.
 - **b** In the **Requirement for response** drop-down list, select **IOTransfer_r2u**.
 - c Specify the **Time** range as 0 to 10.
 - **d** Specify the **Amplitude** range as -0.7 to -0.7.
 - e Click **OK**. The step response plot now contains a second horizontal line representing the lower limit on the actuator signal.
 - **f** To extend this limit for all times (to $t = \infty$), right-click in the yellow shaded area of the design requirement, and select **Extend to inf**.



Select the Design Requirements to Use During Response Optimization

The table in the **Design requirements** tab lists all the specified design requirements. Select the design requirements you want to use in the response optimization. This example uses all the current design requirements.

Overview Compensators Design Requirements Optimization Select design requirement to satisfy					
Optimize Response plot Design requirement					
	IOTransfer_r2u	*			
v	Step Response	Upper time response bound from 0 to 10 (se			
v	Step Response Lower time response bound from 0 to 10 (se				
	LoopTransfer_C				
~	Root Locus Editor for LoopTransfer_C	Settling time < 30 (seconds)			
v	Root Locus Editor for LoopTransfer_C	Percent overshoot < 10%			
v	Bode Editor for LoopTransfer_C	Lower gain limit from 0.01 to 0.17 (rad/s)			

Optimize the System Response

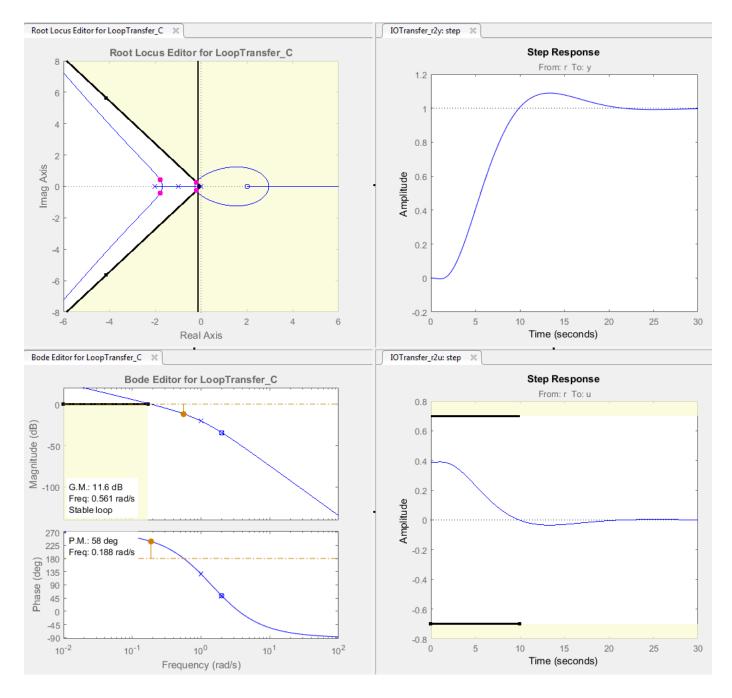
After you select the compensator elements to tune and add design requirements, you can optimize the system response.

To optimize the response of the system, in the **Optimization** tab of the Response Optimization window, click **Start Optimization**.

The **Optimization** tab displays the progress of the optimization.

Response Optimi:	zation						×
Overview Compensators Design Requirements Optimization							
Optimization Progress							
Iteration	Eval-Count	Cost function	Constraint vi	Step size	Procedure	\Box	Optimization options
0	6	0	0.4485			<u> </u>	
1	14	0	0.1001	0.525	infeasible		Display options
2	20	0	0	0.0865			
						-	
Constructing on	timization problem					-	
Optimization st	arted 01-Dec-2015	20:23:07				_	
	hished 01-Dec-2015	5 20:23:26					
Successful term Found a feasibl	ination. e or optimal solutio	n within the speci	fied tolerances.				
						+	
1							
Start Optimization Help							

The status message indicates that the optimization solver found a solution that meets the design requirements within the tolerances. Verify that the design requirements are satisfied.



Create and Display the Closed-Loop System

After designing a compensator, you can export it to the MATLAB workspace and create a model of the full closed-loop system. To export the tuned compensator:

1 In the app, select **Export**.

🙏 Contro	ol System I	Designer - Bod	e Editor for Loc	pTransfer_C						
CONT	TROL SYST	ТЕМ	BODE EDITOR	VIEW						
		î COJ			N	3	5		\swarrow	٢
Open Session	Save Session	Edit Architecture	Multimodel Configuration	Tuning Methods 👻	New Plot ▼	Store	Retrieve	Compare	Export	Preferences
		ARCHIT	ECTURE	TUNING METHODS	A14					PREFEREN

2 In the Export Model dialog box, select C, the compensator you designed, and click Export.

xport m	odels to MATLAB Work	space	
Export	Models	Export as	
	F	F	
V	С	С	
	LoopTransfer_C	LoopTransfer_C	=
	IOTransfer_r2y	IOTransfer_r2y	
	IOTransfer_r2u	IOTransfer_r2u	
	IOTransfor du?v	IOTransfor du?v	-

At the command line, enter the following command to create the closed-loop system, CL, from the open-loop transfer function, <code>open_loopTF</code>, and the compensator, C:

CL = feedback(C*open_loopTF,1)

The following model is returned:

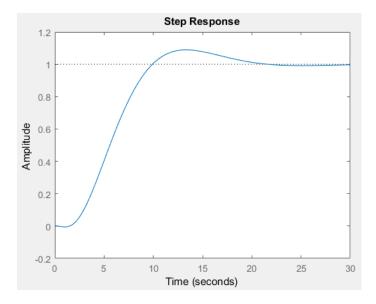
CL =

-0.19414 (s-2) (s^2 + 0.409s + 0.1136) (s^2 + 3.591s + 3.418)

Continuous-time zero/pole/gain model.

To create a step response plot of the closed-loop system, enter the following command.

step(CL);



See Also

More About

- "Design Optimization-Based Controllers for LTI Systems" on page 5-26
- "Time-Domain Simulations in Control System Designer App" on page 5-25

Design Linear Controllers for Simulink Models

When you have Control System Toolbox and Simulink Control Design software, you can perform frequency-domain optimization of Simulink models.

You can use Simulink Control Design software to configure the **Control System Designer** app with compensators, inputs, outputs, and loops computed from a Simulink model. For more information, see topics in the "Classical Control Design" (Simulink Control Design) category.

After you configure the **Control System Designer** app, use Simulink Design Optimization software to optimize the controller parameters of the linearized Simulink model. For an example of optimization-based control design for a model linearized using Simulink Control Design software, see "Design Optimization-Based PID Controller for Linearized Simulink Model (GUI)".

When tuning compensators derived from Simulink Control Design software, the tuning of compensators from a Simulink model is done through the masks of the Simulink blocks representing each compensator. When selecting parameters to optimize, you can tune the compensator in the pole, zero, or gain format, or in a format consistent with the Simulink block mask.

The second secon
Discrete Transfer Fcn
Implement a z-transform transfer function. Specify the numerator and denominator coefficients in descending powers of z. The order of the denominator must be greater than or equal to the order of the numerator.
Main Data Types State Attributes
Data
Source Value
Numerator: Dialog [100.109745431442 -99.1097454314419]
Denominator: Dialog 🔻 [1 -0.888934462738605]
Initial states: Dialog 🔻 0
External reset: None
Input processing: Elements as channels (sample based)
Optimize by skipping divide by leading denominator coefficient (a0)
Sample time (-1 for inherited): 0.01
OK Cancel Help Apply

Mask of a Simulink Compensator Block

Overview Com	pensators Design Requirements Optim	mization				
Select compens	ators to optimize					
🔲 Optimize	Compensator elements	Value	Initial guess	Minimum	Maximum	Typical value
	airframe_demo/az Control/az DTF					
	Numerator	Pole/Zero	ormat .	[-Inf -Inf]	[Inf Inf]	[11]
	Denominator	 Parameteri 	zed format	[-Inf -Inf]	[Inf Inf]	[11]
	airframe_demo/q Control/q Gain		13			
	Gain	2.7718	2.7718	-Inf	Inf	2.7718
	npensator name to change its representation.					
ight click on a cor	npensator name to change its representation.				Use Value as 1	Initial guess
	Start	Optimization	Help			
			<u> </u>			

Response Optimization Compensators Tab — Parameterized Format

Response Optimiz	ation						×
Overview Com	pensators Design Requirements Opt	imiza	ation				
Select compens	ators to optimize						
Optimize	Compensator elements		Value	Initial guess	Minimum	Maximum	Typical value
	airframe_demo/az Control/az DTF						
	Gain	✓	Pole/Zero f	ormat	-Inf	Inf	9.0037
	Real Zero		Parameteriz	ed format	-Inf	Inf	0.99001
	Real Pole		0.88893	0.88893	-Inf	Inf	0.88893
	airframe_demo/q Control/q Gain						
	Gain		2.7718	2.7718	-Inf	Inf	2.7718
Right click on a cor	npensator name to change its representation.					Use Value as 1	Initial guess
	Star	t Opt	timization	Help			

Response Optimization Compensators Tab — Pole/Zero Format

Note You cannot change the compensator format if the compensator is not a Simulink block.

See Also

Related Examples

• "Design Optimization-Based PID Controller for Linearized Simulink Model (GUI)"

Enforcing Time and Frequency Requirements on a Single-Loop Controller Design

This example shows how to use Simulink® Design Optimization[™] to tune a compensator in a Simulink model. You will add performance requirements to further refine and optimize an initial compensator design performed with Simulink® Control Design[™] (see "Single Loop Feedback/ Prefilter Compensator Design" (Simulink Control Design)).

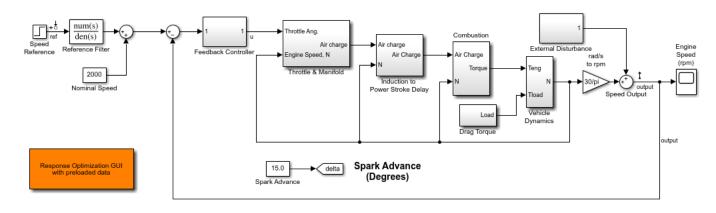
Using Simulink Design Optimization software you can graphically specify design and performance requirements for your system by positioning bounds on response plots such as Bode, Nichols, Pole/ Zero, Step, or Impulse. Then, using optimization-based methods you can automatically tune compensator elements to satisfy the design requirements. Compensator elements that are tunable via optimization-based tuning include gains, poles, and zeros.

This example requires Simulink[®] Control Design[™].

Opening the Model

Open the model using the command below, and double click on the orange block to launch the **Control System Designer** app.





Copyright 2004-2016 The MathWorks Inc.

Design Overview

This example designs a single feedback loop for the speed control of an engine. A preliminary PI controller design has been created using Simulink Control Design (see "Single Loop Feedback/ Prefilter Compensator Design" (Simulink Control Design)) and is used as a starting point to further refine the design using response optimization. This example will tune the controller to satisfy the following time-domain and frequency-domain performance specifications:

Requirement 1. A lower amplitude limit on the step response output of -0.1 and a 3 second rise time to reach 95% of the set-point value.

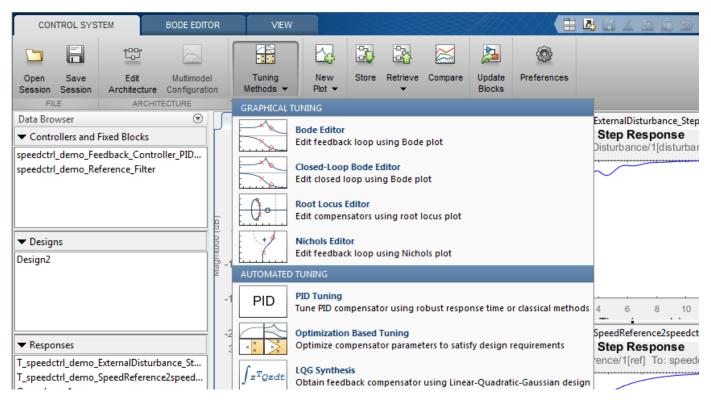
Requirement 2. A maximum overshoot of 1% for the unit step response from Speed Reference to Speed Output.

Requirement 3. A minimum loop gain of 10db over the frequency range 1e-4 to 1 rad/sec to ensure good output disturbance rejection and reference tracking over this frequency range.

Requirement 4. A maximum loop gain of -10db over the frequency range 10 to 1e4 rad/sec to ensure adequate high frequency noise rejection, and together with the low frequency requirement, to ensure a loop bandwidth of between 1 and 10 rad/sec.

Launching Simulink Design Optimization

Both time- and frequency-domain response optimization are integrated into the Control System Designer app. In the **Control System** tab, in the **Tuning Methods** drop-down list, select **Optimization Based Tuning**.



Configuring an Optimization

The first step in configuring an optimization is to select the compensator elements to tune. For this example select the Gain and Real Zero of the PID controller; the reference filter is not tuned.

Response O	ptimiz	zation							x
Overview	Com	pensators	Design Requirements	Optin	nization				
Select cor	mpens	ators to op	timize						
🔲 Opti	mize	C	ompensator elements		Value	Initial guess	Minimum	Maximum	Typical value
		speedctrl_	demo/Feedback Contro	oller/	PID Controller				
V		Gain			0.0014181	0.0014181	-Inf	Inf	0.0014181
		Real Zero			-2.4639	-2.4639	-Inf	Inf	-2.4639
		speedctrl_	demo/Reference Filter						
		Gain			1	1	-Inf	Inf	1
		Real Pole	:		-10	-10	-Inf	Inf	-10
Right click o		monstor na	me to change its representati						
Ngrit click o		inpensator na	me to change is representati	UN.				Use Value as I	initial guess
				Start (Optimization	Help			

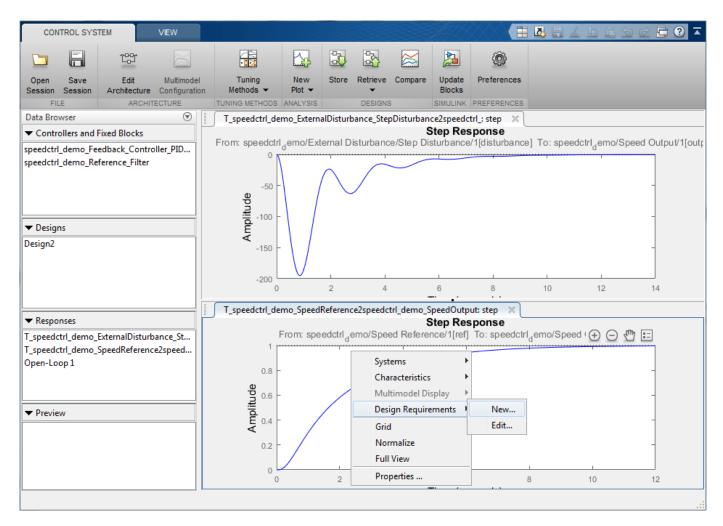
Adding Design Requirements

The next step is to create the design requirements that the optimization should satisfy. Design requirements are visualized on system response plots. You can add response plots by using the **Graphical Tuning** or the **New Plot** drop down lists in the Control System Designer app. The "Getting Started with the Control System Designer" (Control System Toolbox) example shows how to use the Control System Designer.

There are two ways to add requirements; you can add them using the **Add new design requirement** button on the **Design Requirements** tab in the **Response Optimization** window or by right clicking on a response plot and selecting **Design Requirements->New**.

To add **Requirement 1** to limit the lower amplitude of the output resulting from a step input,

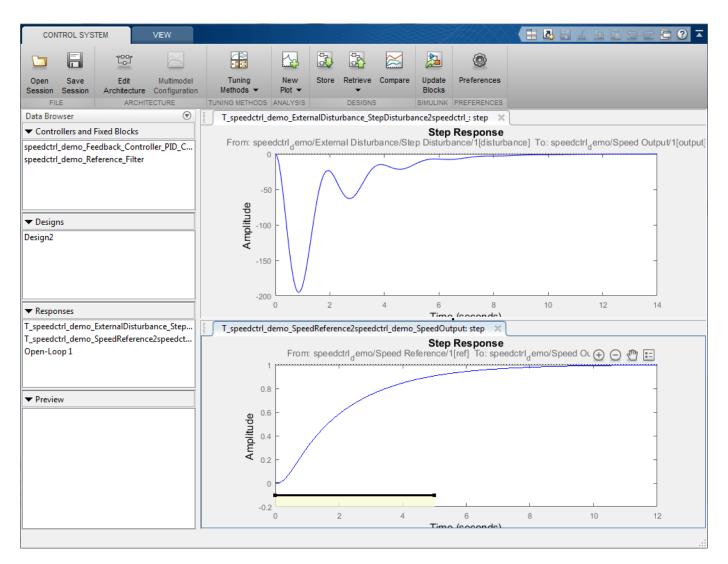
1. Right click on the lower step response plot and select **Design Requirements->New** .



2. Specify the lower limit as -0.1 over the time range 0 to 5 seconds.

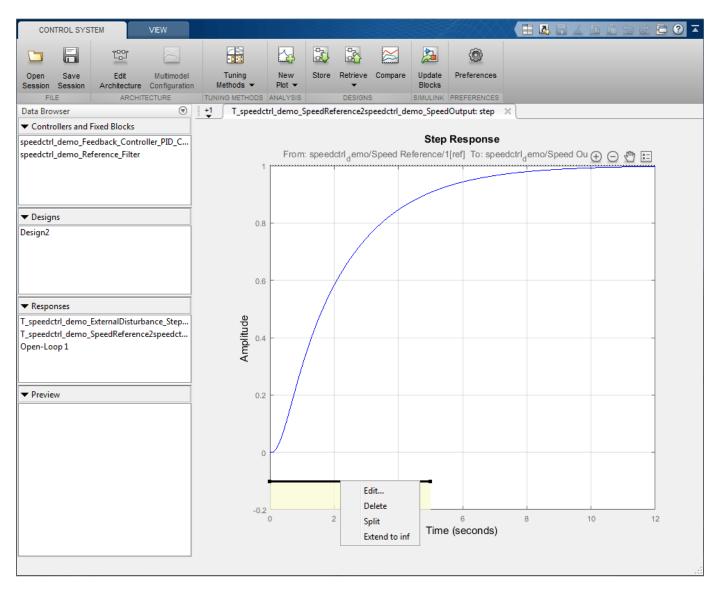
D	esign require	ement type: [ower time re	esponse bour	nd		•
٢I	Design requir	rement paran	neters				
	Type: Const	rain signal to	be >= the b	ound 👻			
	Edge	Start	Edge	End			
	Time (s)	Amplitude	Time (s)	Amplitude	Slope (1/s)	Weight	+
	0	-0.1000	5	-0.1000	0	1	
L]
					Ok	Close	Help

This creates the lower amplitude limit on the step response plot as shown in the next figure.

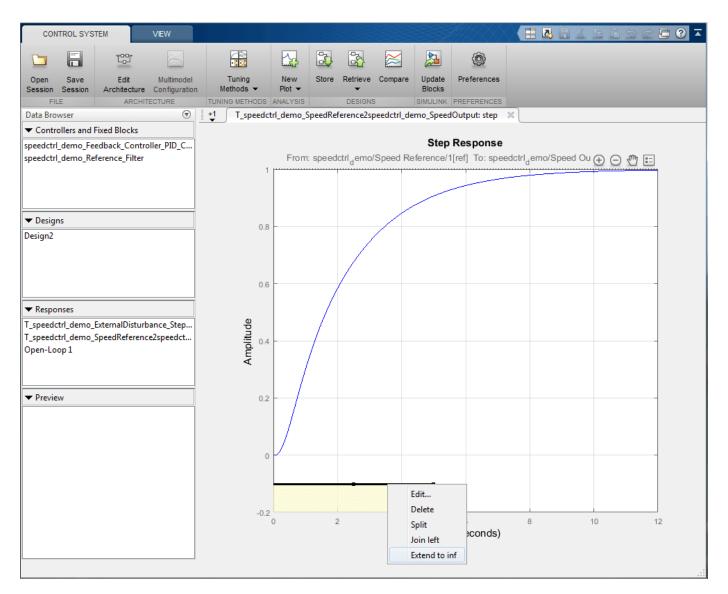


To add the rise time requirement to the step response, you can graphically manipulate the lower amplitude requirement on the step response plot.

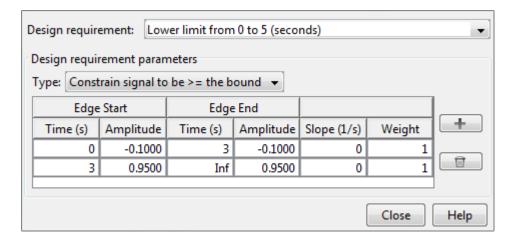
1. Right click the lower amplitude limit requirement and select **Split** to split a segment into two pieces.

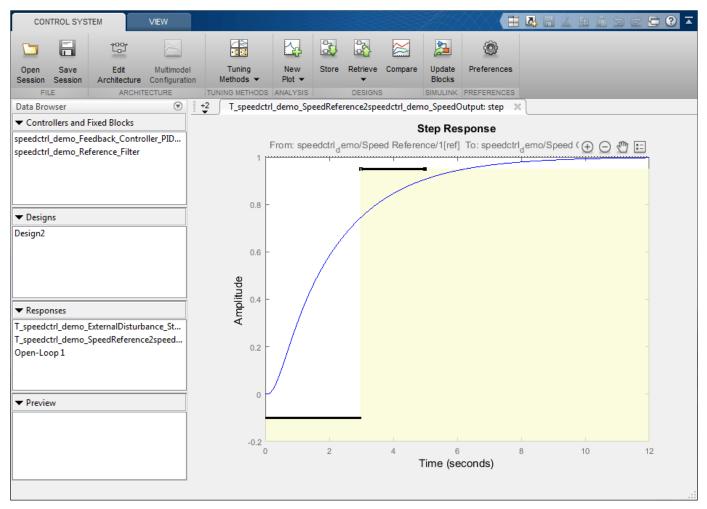


2. Right click the second segment of the requirement and select Extend to inf to extend it to infinity.



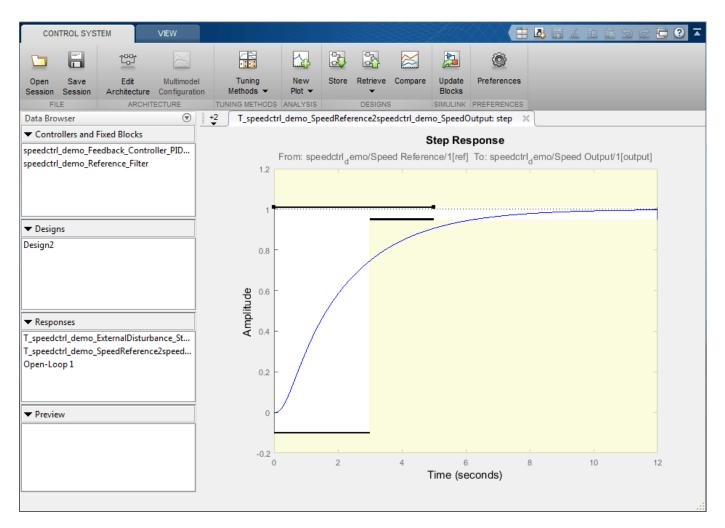
3. Right click the second segment of requirement, select **Edit**, and set the values to represent a 95% rise time at 3 seconds.





Alternatively you can left click the second segment of the requirement and drag it into position.

Next add **Requirement 2** for maximum overshoot to the step response plot. The time-domain constraints on the step response plot are shown in the next figure.



The plot shows the lower amplitude limit of -0.1, maximum overshoot and 95% of the unit step response value of 1.01 and 0.95 respectively.

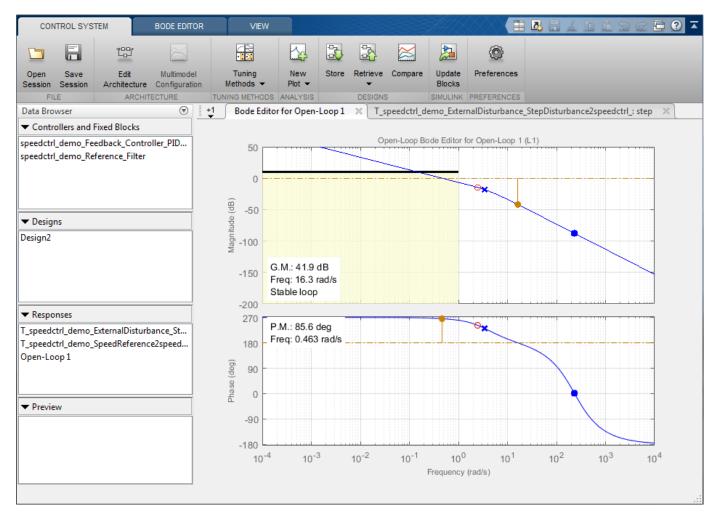
To add Requirement 3 for minimum loop gain,

1. Click **Add new design requirement** in the **Design Requirements** tab of the **Response Optimization** window.

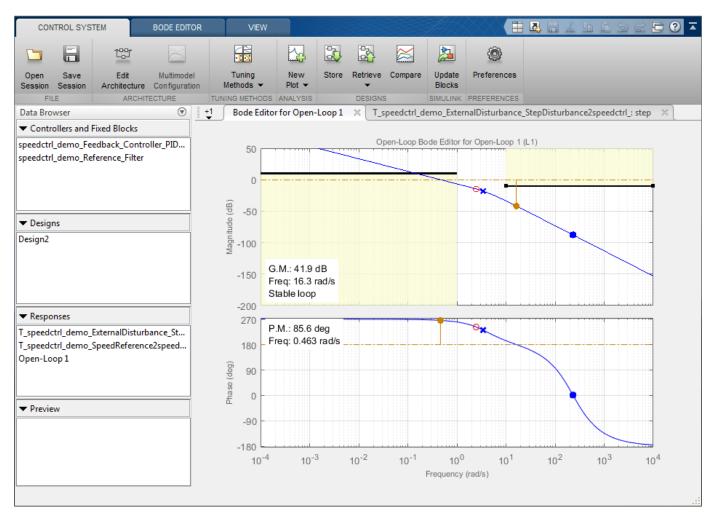
 ${\bf 2.}$ Specify the Bode magnitude lower limit for the open loop as 10db over the frequency range 1e-4 to 1 rad/sec.

D	esign require	ment type:		E	Bode magnitu	ude lower lin	nit 🔻
R	equirement f	or response:			Open-Loop 1		•
	Design requir Type: Const			bound 👻			
	Edge	Start	Edge	End			
	Freq. (rad	Mag. (db)	Freq. (rad	Mag. (db)	Slope (dB	Weight	+
	1.0000e-04	10	1	10	0	1	
					ОК	Cancel	Help

This creates the minimum loop gain constraint on the Bode magnitude plot as shown in the next figure.



Add **Requirement 4** for the maximum loop gain to the Bode magnitude plot to satisfy the overall design specifications. The Bode magnitude plot shows the minimum and maximum loop gain over the specified frequency range.



Select the design requirements for optimization from the **Design Requirements** tab. After you have selected the requirements, the **Design Requirements** table appears as shown next:

Response Optimizatio	n	x
Overview Compen	sators Design Requirements Optimization	
Select design requir	ement to satisfy	
Optimize	Response plot	Design requirement
	Open-Loop 1	
v	Open-Loop Bode Editor for Open-Loop 1 (L1)	Lower gain limit from 0.0001 to 1 (rad/s)
V	Open-Loop Bode Editor for Open-Loop 1 (L1)	Upper gain limit from 10 to 1e+04 (rad/s)
	T_speedctrl_demo_SpeedReference2speedctrl_demo_Spee	dOutput
V	Step Response	Lower time response bound from 0 to 5 (sec
	Step Response	Upper time response bound from 0 to 10 (se
Click the "Add new desig	gn requirement" button or right click on a plot to add a new design requirem	nent
		Add new design requirement
	Start Optimization Help]

Running an Optimization

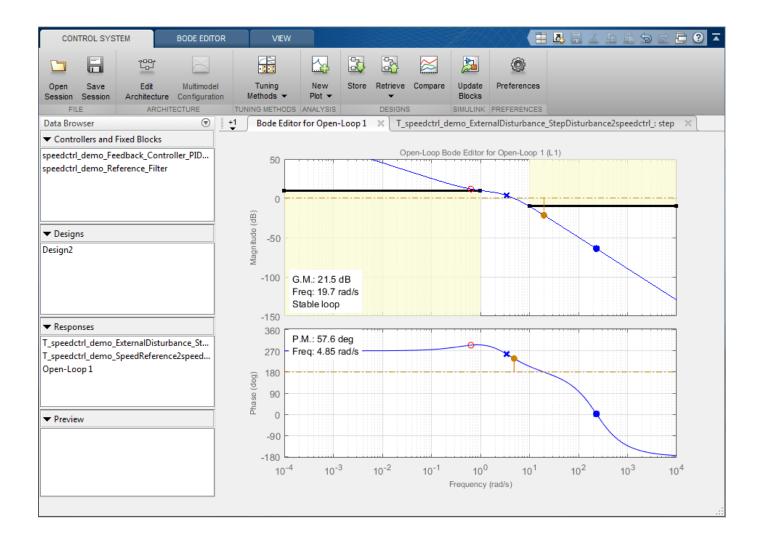
After defining the design requirements and selecting the compensator elements to tune, the optimization is ready to run. Select the **Optimization** tab and click the **Start Optimization** button. During optimization the response plots update and numerical progress data is displayed in the **Optimization** tab.

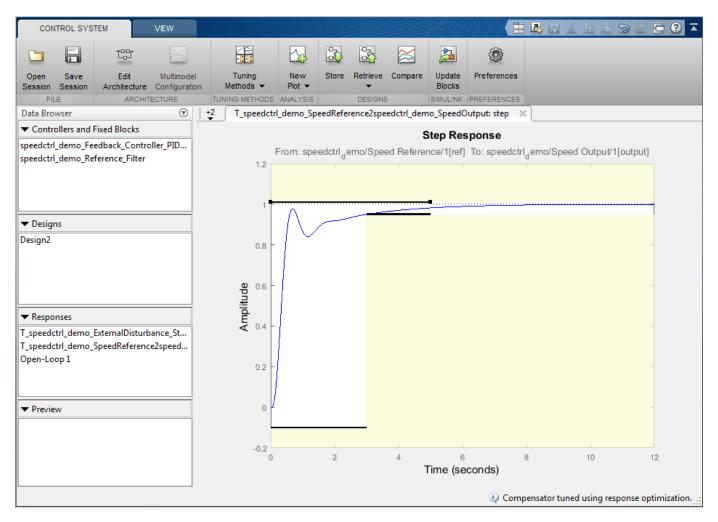
Response Optimization	×
Overview Compensators Design Requirements Optimization	
Optimization Progress	
Iteration Eval-Count Cost function Constraint vi Step size Procedure Optimiz	zation options
0 10 0 1.661	
1 22 0 0.586 1.04e+03 infeasible Disp	lay options
2 32 0 0.1502 1.75e+03 Hessian modi	
3 42 0 0.01917 1.09e+03	
4 52 0 0.0002952 293	
· · · · · · · · · · · · · · · · · · ·	
Constructing optimization problem	
Optimization started 18-Feb-2016 09:58:00	
Optimization finished 18-Feb-2016 09:58:19 Successful termination.	
Found a feasible or optimal solution within the specified tolerances.	
Ψ	
Start Optimization Help	
	.::

Inspecting and Verifying the Final Design

You can check how well the optimized design meets the specified design requirements by viewing the optimized responses (shown below).

To verify the compensator design on the full non-linear Simulink model, return to the **Control System Designer** and click the **Update Simulink Block Parameters** button to write the compensator back to the Simulink model. You can now simulate the Simulink model with the newly designed compensator.





bdclose('speedctrl_demo')

See Also

Related Examples

• "Design Optimization to Meet Time-Domain and Frequency-Domain Requirements (GUI)" on page 3-76

More About

- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43

Airframe Controller Tuning

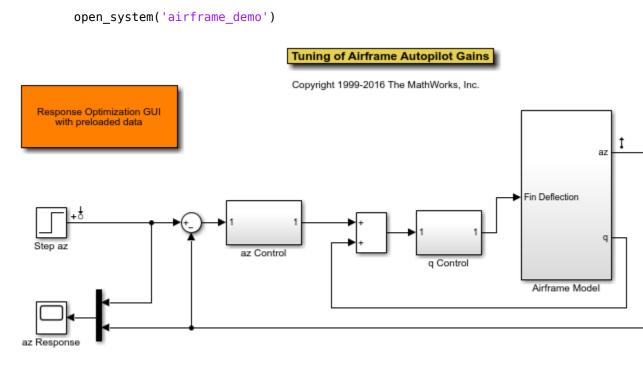
This example shows how to design two feedback loops in a cascaded control system to track reference signals. The design uses the body rate (q) as an inner feedback loop and the acceleration (az) as an outer feedback signal. This example is based on the Simulink® Control Design[™] example "Cascaded Multiloop Feedback Design" (Simulink Control Design).

After loading the model and pre-configured **Control System Designer** app session, you can design a new controller using Response Optimization.

Requires Simulink[®] Control Design[™].

Opening the Model

Open the model using the following command, and double click on the orange block to launch the **Control System Designer**.



Design Overview

The goal of the design is to have an overall rise time of under 0.5 seconds for the outer feedback loop. A preliminary design is done using Simulink Control Design (see "Cascaded Multiloop Feedback Design" (Simulink Control Design).) and is used as a starting point for optimization. The controller must satisfy the following requirements:

- A Gain Margin >= 10db and Phase Margin >= 50 degrees for the inner feedback loop.
- A Gain Margin >= 10db and Phase Margin >= 60 degrees for the outer feedback loop.
- An overshoot of at most 1%, a 80% rise time of 0.5 seconds, and a 99% rise time of 0.6 seconds for the step response of the outer loop.

These design requirements have been added to the Control System Designer app. To complete the design using response optimization, in the **Control System** tab, in the **Tuning Methods** drop-down

list, select **Optimization Based Tuning**. In the **Response Optimization** window, click **Start Optimization**.

bdclose('airframe_demo')

See Also

More About

- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43

DC Motor Controller Tuning

This example shows how to design a PI control system to control the speed of a DC motor. It is based on the Control System Toolbox[™] "DC Motor Control" (Control System Toolbox) example.

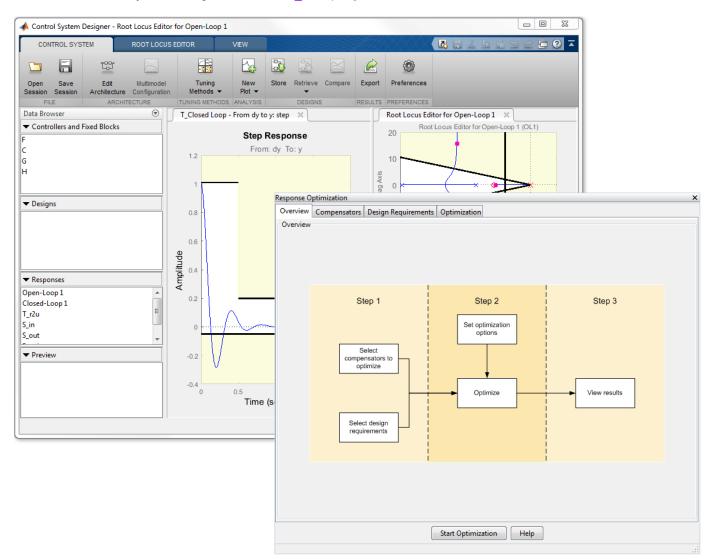
After loading the model and pre-configured **Control System Designer**, you can design a new controller using Response Optimization.

Requires Control System Toolbox[™].

Opening the Model and Control System Designer

Open the model and pre-configured Control System Designer using the command

controlSystemDesigner('dcmotor_demoproject')



Design Overview

The goal of the overall design is to track a reference change in speed with minimal overshoot and to reject output disturbances. The controller must satisfy the following requirements:

- The closed-loop poles of the control loop are restricted to a region on the root locus plot that implies less than a 5% overshoot.
- The closed-loop poles of the control loop are restricted to a region on the root locus plot that implies a settling time less than 2 seconds.
- The output (y) of a unit step output disturbance is reduced by 80% within 0.5 seconds and by 95% within 1 second.

These design requirements have been added to the Control System Designer. To complete the design, using response optimization, click the **Start Optimization** button within the **Response Optimization** dialog.

See Also

More About

- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43

Hydraulic Piston Regulator Tuning

This example shows how to tune a lead-lag regulator for a simplified hydraulic piston. The piston model is given by:

PlantModel = zpk(10,[0,0,-10],-1);

An initial controller design using the Control System Toolbox[™] yields a controller:

Controller = zpk(-0.15, -3.5, 0.15);

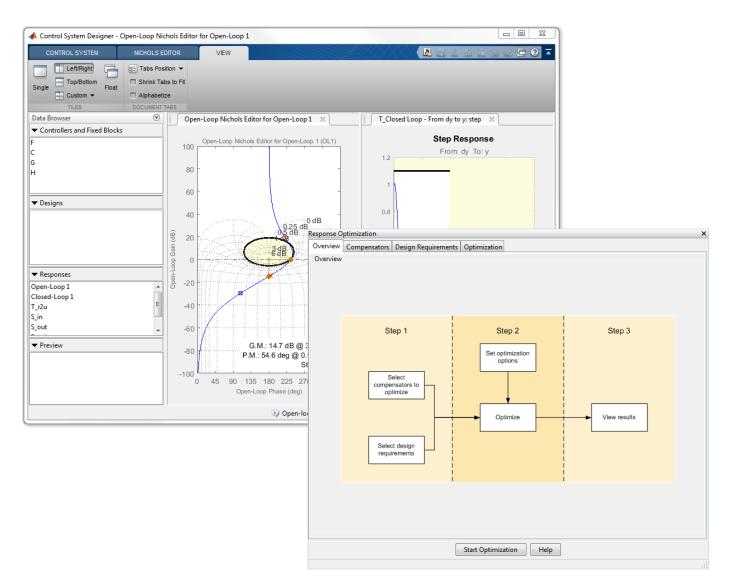
After loading the model and pre-configured Control System Designer, you can design a new controller using Response Optimization.

Requires Control System Toolbox.

Opening the Model and Control System Designer

Open the model and pre-configured Control System Designer using the command

controlSystemDesigner('piston_demoproject');



Design Overview

The goal of the design is to fine tune the designed regulator to better reject disturbances. The controller must satisfy the following requirements:

- The closed-loop peak gain must be less than 1db; this ensures good regulation with minimum overshoot.
- The output (y) resulting from a unit step output disturbance (dy) must be reduced by 99% within 10 seconds and no more than a 20% overshoot is allowed.

These design requirements have been added to the Control System Designer. To complete the design, using response optimization, click **Start Optimization** button in the **Response Optimization** window.

See Also

More About

- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Frequency-Domain Design Requirements in the App" on page 3-43

Lookup Tables

- "What are Adaptive Lookup Tables?" on page 6-2
- "How to Estimate Lookup Table Values" on page 6-4
- "Estimate Constrained Values of a Lookup Table" on page 6-5
- "Estimate Lookup Table Values from Data" on page 6-17
- "Building Models Using Adaptive Lookup Table Blocks" on page 6-28
- "Selecting an Adaptation Method" on page 6-31
- "Model Engine Using n-D Adaptive Lookup Table" on page 6-33
- "Using Adaptive Lookup Tables in Real-Time Environment" on page 6-43
- "Design Optimization Using Lookup Table Requirements for Gain Scheduling (Code)" on page 6-44
- "Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)" on page 6-59
- "2-D Adaptive Lookup Table Generation" on page 6-70
- "Engine Volumetric Efficiency Surface Matching" on page 6-71

What are Adaptive Lookup Tables?

Lookup Tables

Lookup tables store numeric data in a multidimensional array format. In the simpler two-dimensional case, lookup tables can be represented by matrices. Each element of a matrix is a numerical quantity, which can be precisely located in terms of two indexing variables. At higher dimensions, lookup tables can be represented by multidimensional matrices, whose elements are described in terms of a corresponding number of *indexing variables*.

Lookup tables provide a means to capture the dynamic behavior of a physical (mechanical, electronic, software) system. The behavior of a system with M inputs and N outputs can be approximately described by using N lookup tables, each consisting of an array with M dimensions.

You usually generate lookup tables by experimentally collecting or artificially creating the input and output data of a system. In general, you need as many indexing parameters as the number of input variables. Each indexing parameter may take a value within a predetermined set of data points, which are called the *breakpoints*. The set of all breakpoints corresponding to an indexing variable is called a *grid*. Thus, a system with M inputs is gridded by M sets of breakpoints. The software uses the breakpoints to locate the array elements, where the output data of the system are stored. For a system with N outputs, the software locates the N array elements and then stores the corresponding data at these locations.

After you create a lookup table using the input and output measurements as described previously, you can use the corresponding multidimensional array of values in applications without having to remeasure the system outputs. In fact, you need only the input data to locate the appropriate array elements in the lookup table because the software reads the approximate system output from the data stored at these locations. Therefore, a lookup table provides a suitable means of capturing the input-output mapping of a *static* system in the form of numeric data stored at predetermined array locations.

Adaptive Lookup Tables

Statically defined lookup tables, as described in "Lookup Tables" on page 6-2, cannot accommodate the *time-varying* behavior (characteristics) of a physical plant. Static lookup tables establish a permanent and static mapping of input-output behavior of a physical system. Conversely, the behavior of actual physical systems often varies with time due to wear, environmental conditions, and manufacturing tolerances. With such variations, the static mapping of input-output behavior of a plant described by the lookup table may no longer provide a valid representation of the plant characteristics.

Adaptive lookup tables incorporate the time-varying behavior of physical plants into the lookup table generation and maintenance process while providing all of the functionality of a regular lookup table.

The adaptive lookup table receives the input and output measurements of a plant's behavior, which are then used to dynamically create and update the content of the underlying lookup table. In addition to requiring the input data to create the lookup table, the adaptive lookup table also uses the output data of the plant to recalculate the table values. For example, you can collect the output data of the plant by placing sensors at appropriate locations in a physical system.

The software uses the input measurements to locate the array elements by comparing these input values with the breakpoints defined for each indexing variable. Next, it uses the output

measurements to recalculate the numeric value stored at these array locations. However, unlike a regular table, which only stores the array data before the actual use of the lookup table, the adaptive table continuously improves the content of the lookup table. This continuous improvement of the table data is referred to as the adaptation process or learning process.

The adaptation process involves statistical and signal processing algorithms to recapture the inputoutput behavior of the plant. The adaptive lookup table always tries to provide a valid representation of the plant dynamics even though the plant behavior may be time varying. The underlying signal processing algorithms are also robust against reasonable measurement noise and they provide appropriate filtering of noisy output measurements.

See Also

Adaptive Lookup Table (1D Stair-Fit) | Adaptive Lookup Table (2D Stair-Fit) | Adaptive Lookup Table (nD Stair-Fit)

Related Examples

• "Model Engine Using n-D Adaptive Lookup Table" on page 6-33

More About

- "About Lookup Table Blocks"
- "Building Models Using Adaptive Lookup Table Blocks" on page 6-28

How to Estimate Lookup Table Values

You can use lookup table Simulink blocks to approximate a system's behavior, as described in "About Lookup Table Blocks". After you build your system using lookup tables, you can use Simulink Design Optimization software to estimate the table values from measured I/O data.

Estimating lookup table values is an example of estimating parameters which are matrices or multidimensional arrays. The workflow for estimating parameters of a lookup table consist of the following tasks:

- **1** Creating a Simulink model using lookup table blocks.
- 2 Importing the measured input and output (I/O) data from which you want to estimate the table values.
- **3** Analyzing and preparing the I/O data for estimation.
- 4 Estimating the lookup table values.
- **5** Validating the estimated table values using a validation data set.

See Also

Related Examples

- "Estimate Lookup Table Values from Data" on page 6-17
- "Estimate Constrained Values of a Lookup Table" on page 6-5

Estimate Constrained Values of a Lookup Table

Objectives

This example shows how to estimate constrained values of a lookup table in the **Parameter Estimator**. Apply monotonically increasing constraints to the lookup table output values, and use the **Parameter Estimator** to estimate the table values.

About the Data

In this example, use lookup_increasing.mat, which contains the measured I/O data for estimating the lookup table values. The MAT-file includes the following variables:

- xdata1 Input data consisting of 602 uniformly sampled data points in the range [-5,5].
- ydata1 Output data corresponding to the input data samples.
- time1 Time vector.

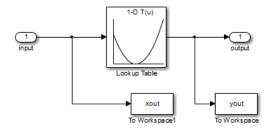
Use the I/O data to estimate monotonically increasing output values of the lookup table in the lookup_increasing Simulink model.

Lookup Table Output

1 Open the lookup table model by typing the following command at the MATLAB prompt:

lookup_increasing

This command opens the Simulink model, and loads the estimation data in the MATLAB workspace.



2 View the table output values by double-clicking the Lookup Table block.

🔁 Function Block Parameters: L	pokup Table
Lookup Table (n-D)	
	olated table lookup including index searches. The table is a sampled n N variables. Breakpoint sets relate the input values to positions in the table. The o the top (or left) input port.
Table and Breakpoints Ale	jorithm Data Types
Number of table dimensions:	1
Table data:	cumsum(ydelta)
Breakpoints specification:	Explicit values
Breakpoints 1:	[-5:5]
Edit table and breakpoints	
0	OK Cancel Help Apply

The table contains 11 output values at breakpoints [-5:5], specified in the Function Block Parameters dialog box. To learn more about how to specify the table values, see "Enter Breakpoints and Table Data".

The **Table data** field shows that the table output values are the cumulative sum of the values stored in variable ydelta. Thus, if y_n are the 11 table output values, ydelta is $(y_1, y_2-y_1, y_3-y_2, ..., y_{11}-y_{10})$. The initial ydelta values are loaded from lookup_increasing.mat.

The initial table output values are not monotonically increasing. To ensure monotonically increasing table output values, the difference between adjacent table output values should be positive. To do so, estimate ydelta in the **Parameter Estimator** using the measured I/O estimation data, and constrain ydelta(2:end) to be positive during estimation.

Estimate the Monotonically Increasing Table Values Using Default Settings

1 Open a parameter estimation session.

In the Simulink model, select **Parameter Estimator** from the **Apps** tab, in the gallery, under **Control Systems** to open a session with the name **lookup_increasing** in the **Parameter Estimator**.

PARAMETER EST	IMATION	VALIDA	ATION	VIEW				1 5 e 🗄 🖸
] Open Session ▼] Save Session ▼	Select	New	Select	Add Plot	Plot Model	Cost Function: Sum Squared Error 💌	Estimate	
	Parameters		Experiments	•	Response	-	•	
FILE	PARAMETERS	EXPER	RIMENTS	PL	.OTS	OPTIONS	ESTIMATE	
Data Browser								
 Parameters 								
 Experiments 								
▼ Results								
▼ Results								
▼ Results								
▼ Results								
▼ Results								
▼ Results								
▼ Results								
▼ Results								
Results Preview								

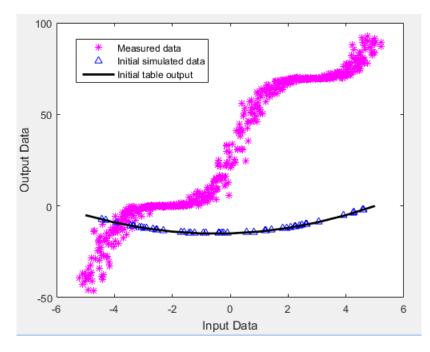
2 Create an experiment and import the I/O data.

On the **Parameter Estimation** tab, click **New Experiment**. Type [time1,ydata1] in **Outputs** and [time1,xdata1] in **Inputs** of the Edit Experiment dialog box. Click **OK**. A new experiment with name Exp is created in the **Experiments** area of the app. Rename the experiment EstimationData by right-clicking the default experiment name, Exp, and selecting Rename. For more information, see "Import Data for Parameter Estimation" on page 1-5.

dit Experiment: Exp	×
Outputs	
Specify measured output signals for this experiment.	
/Lookup Table:1 (output)	
<1x1 Signal, 602 points> 🔹 🕁 🐇	
Select Measured Output Signals	
Inputs	
Optionally specify input signals for this experiment.	
lookup_increasing/input:1 (input)	
<1x1 Signal, 602 points> 🔹 🖶 🐇	
Select Inputs	
Initial States	
Optionally define initial states for this experiment.	
There are currently no initial states defined for this experiment.	
Select Initial States	
Parameters	
Optionally define parameters for this experiment.	
There are currently no parameters defined for this experiment.	
Select Parameters	
🛺 Plot & Simulate 🛛 Dot 🖌 OK 🕜 He	lp

3 Run an initial simulation to view the measured data, simulated model values, and the initial table values by typing the following commands at the MATLAB prompt.

```
sim('lookup_increasing')
figure(1); plot(xdata1,ydata1,'m*',xout,yout,'b^')
hold on; plot(-5:5,cumsum(ydelta),'k','LineWidth',2)
xlabel('Input Data'); ylabel('Output Data');
legend('Measured data','Initial simulated data','Initial table output')
```



The initial table output values and simulated data do not match the measured data.

4 Select parameter for estimation.

On the **Parameter Estimation** tab, click **Select Parameters**. The Edit: Estimated Parameters dialog box opens. In the **Parameters Tuned for all Experiments** panel, click **Select parameters** to open the Select Model Variables dialog box. Check the box next to ydelta, and click OK.

Select	model variabl	es				×
Filter	by variable n	ame				+ م
•	Variable	Current va	Used	Ву		
	time1	[0;0.01663				
	xdata1	[-2.899131				
	ydelta	[-5 -4 -3 -2	lookup_increasing/Lookup_Table	<u>e</u>		
► Sp	ecify expression	on indexing if	necessary (e.g., a(3) or s.x)			
	, 1			ок	🔀 Cancel	(?) Help

The ydelta values are selected for estimation by default in the Edit: Estimated Parameters dialog box.

dit: Estimated Parameters ×
Parameters Tuned for all Experiments
<u>ydelta</u>
[-5 -4 -3 -2 -1 0 1 2 3 4 5]
Select parameters
Parameters and Initial States Tuned per Experiment
Experiment: EstimationData 💌
Select experiment initial states for estimation.
There are no initial states defined for this experiment.
Select experiment parameters for estimation.
There are no parameters defined for this experiment.
Edit experiment
🔚 Update Model 🛛 🔗 OK 📀 Help

5 Apply a monotonically increasing constraint on the table output values. For more details about the table, see "Lookup Table Output" on page 6-5.

In the Edit: Estimated Parameters dialog box, click the arrow next to the ydelta values. In the expanded menu, set **Minimum** ydelta values to [-Inf,zeros(1,10)]. Thus, while the first value in ydelta can by anything, subsequent values which are the difference between adjacent table output values, must be positive.

Edit: Estimated Pa	arameters		×
Parameters Tu	ned for all Experiments		
<u>ydelta</u>			
[-5 -4 -3 -2	-1 0 1 2 3 4 5]	- 🕁	🗙 🗹 Estimate
Minimum:	[-Inf 0 0 0 0 0 0 0 0 0 0 0]	- 🖶	
Maximum:	<1x11 double>	- 🖶	
Scale:	[8 4 4 2 1 1 1 2 4 4 8]	- 🖶	
	oeters		
÷-			

6 Select EstimationData experiment for estimation.

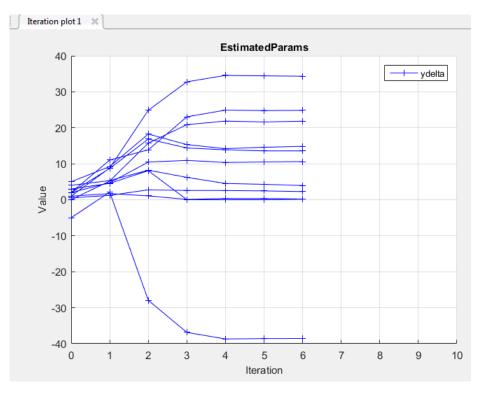
On the **Parameter Estimation** tab, click **Select Experiment**. By default, **EstimationData** is selected for estimation. If not, check the box under the **Estimation** column, and click **OK**.

elect experin	nents to includ	le for estimation or validation	
Estimation	Validation	Experiment	
1		EstimationData	
			÷
			÷
			4
			 4

7 Estimate the table values using default settings.

On the **Parameter Estimation** tab, click **Estimate**.

The **Parameter Trajectory** plot shows the change in the parameter values at each iteration.



The Estimation Progress Report shows the iteration number, number of times the objective function is evaluated, and value of the cost function at the end of each iteration.

teration	F-count	EstimationData
		(Minimize)
0		135.683
1	45	37.301
2		3.336
3		1.195
4	114	1.161
5	137	1.134
6	160	1.133
Dotimizati	on started 1	9-May-2015 13:34:55
stimation	converged	19-May-2015 13:37:02
		dated with estimated parameter values

The estimated parameters are saved in a new variable, EstimatedParams, in the **Results** area of the app. To view the estimated parameters, right-click EstimatedParams and select **Open**.

View Result : EstimatedParams ×
Estimation result(s): ydelta = [-38.537 24.846 13.579 0.2107 2.273 21.836 34.322 10.596 0.1599 3.9537 14.855]
Parameters estimated using experiments: EstimationData, cost = 1.1337
Solver output:
Cost: 1.1337 ExitFlag: 1 FCount: 160 Date: 19-May-2015 13:37
Solver termination message:
Local minimum found.
Optimization completed because the size of the gradient is less than the selected value of the function tolerance.
Stopping criteria details:
Optimization completed: The first-order optimality measure, 2.353798e-04, is less than options.TolFun = 1.000000e-03.
Optimization Metric Options relative first-order optimality = 2.35e-04 TolFun = 1e-03 (selected)
🔛 Use as initial guess 🛛 😹 Update Model 🛛 🛫 OK

The estimated ydelta(2:end) values are positive. Thus, the output of the table, which is the cumulative sum of the values stored in ydelta, is monotonically increasing.

Validate the Estimation Results

After you estimate the table values, as described in "Estimate the Monotonically Increasing Table Values Using Default Settings" on page 6-6, you use another measured data set to validate and check

that you have not over-fit the model. You can plot and examine the following plots to validate the estimation results:

- Residuals plot
- Measured and simulated data plots
- 1 Create an experiment to use for validation and import the validation I/O data.

On the **Parameter Estimation** tab, click **New Experiment**. Type [time2,ydata2] in **Outputs** and [time2,xdata2] in **Inputs** of the Edit Experiment dialog box. Name the experiment ValidationData by right-clicking the default experiment name, Exp, in the **Experiments** area of the app, and selecting Rename. For more information, see "Import Data for Parameter Estimation" on page 1-5.

2 Select the experiment for validation.

Click **Select Experiments** on the **Parameter Estimation** tab. The ValidationData experiment is selected for estimation by default. Clear **Estimation** and select the box for **Validation**.

Select Experime	ents		×
Select experim	ents to includ	le for estimation or validation	
Estimation	Validation	Experiment	
v		EstimationData	
	V	ValidationData	
			<u> </u>
		🖋 ок	(2) Help

3 Select the results to validate.

On the **Validation** tab, click **Select Results to Validate**. Clear Use current parameter values, select EstimatedParams, and click **OK**.

Se	lect Re	sults	×
		Select results to include in validation	
		Results	
		Use current parameter values	
	V	EstimatedParams	
L			
		🖋 ок	(?) Help
			.:

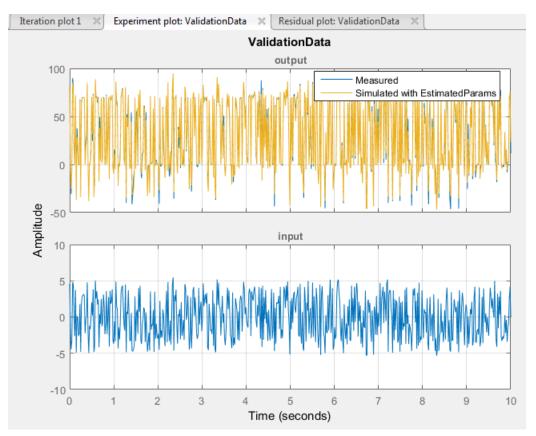
4 Select the plots to display during validation.

The **Parameter Estimator** displays the experiment plot after validation by default. Add the residuals plot by selecting the corresponding box on the **Validation** tab.

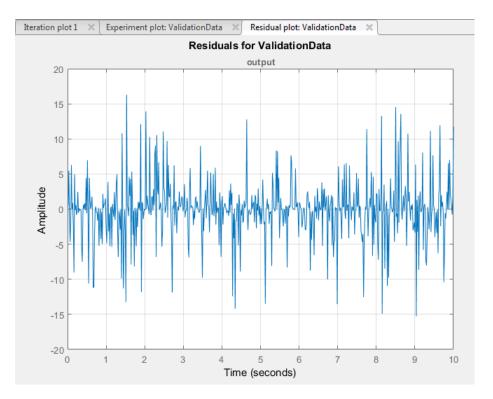


Click Validate.

- **5** Examine the plots.
 - **a** The experiment plot shows the data simulated using estimated parameters agrees with the measured validation data.



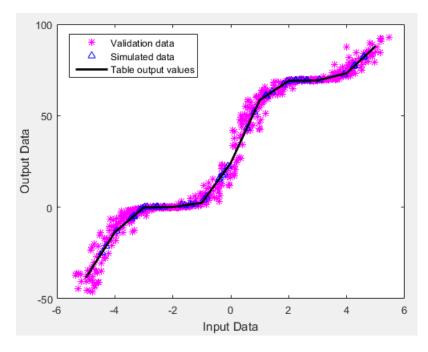
b To view the residuals plot, click **Residual plot: ValidationData** tab.



The residuals, which show the difference between the simulated and measured data, lie within 15% of the maximum output variation. This indicates a good match between the measured and simulated table data values.

c Plot and examine the validation data, simulated data, and estimated table values.

```
sim('lookup_increasing')
figure(2); plot(xdata2,ydata2,'m*',xout,yout,'b^')
hold on; plot(-5:5,cumsum(ydelta),'k','LineWidth', 2)
xlabel('Input Data'); ylabel('Output Data');
legend('Validation data','Simulated data','Table output values');
```



The table output values match both the measured data and the simulated table values. The table output values cover the entire range of input values, which indicates that all the lookup table values have been estimated.

See Also

Related Examples

• "Estimate Lookup Table Values from Data" on page 6-17

Estimate Lookup Table Values from Data

Objectives

This example shows how to estimate lookup table values from time-domain input-output (I/O) data in the **Parameter Estimator**.

About the Data

In this example, use the I/O data in lookup_regular.mat to estimate the values of a lookup table. The MAT-file includes the following variables:

- xdata1 Consists of 63 uniformly-sampled input data points in the range [0,6.5]
- ydata1 Consists of output data corresponding to the input data samples
- time1 Time vector

Use the I/O data to estimate the lookup table values in the lookup_regular Simulink model. The lookup table in the model contains ten values, which are stored in the MATLAB variable table. The initial table values comprise a vector of 0s. To learn more about how to model a system using lookup tables, see "Guidelines for Choosing a Lookup Table".

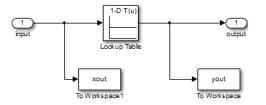
Open a Parameter Estimation Session

To estimate the lookup table values, open a Parameter Estimation session.

1 Open the lookup table model by typing the following command at the MATLAB prompt:

lookup_regular

This command opens the Simulink model, and loads the estimation data into the MATLAB workspace.



2 In the Simulink model, select **Parameter Estimator** from the **Apps** tab, in the gallery, under **Control Systems** to open a new session with name **lookup_regular** in the **Parameter Estimator**.

🔺 Parameter Estimat	ion - lookup_i	regular								
PARAMETER EST	IMATION	VALID	ATION	VIEW					i te	- 🕐 -
Copen Session ▼ Copen Session ▼	Select Parameters	New Experiment	Select Experiments	Add Plot	Plot Model Response	Cost Function:	Sum Squared Error 💌	Estimate		
FILE	PARAMETERS	EXPE	RIMENTS	PL	.OTS	C	OPTIONS	ESTIMATE		
Data Browser		۲								
▼ Experiments										
Results										

Estimate the Table Values Using Default Settings

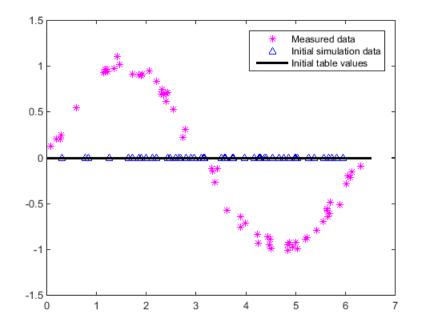
Use the following steps to estimate the lookup table values.

1 Create a new experiment by clicking New Experiment on the Parameter Estimation tab. Name it EstimationData. Then import the I/O data, xdatal and ydatal, and the time vector, timel, into the experiment. To do this open the experiment editor by right-clicking EstimationData and selecting Edit.... Type [timel, ydatal] in the output dialog box and [timel, xdatal] in the input dialog box in the experiment editor. For more information, see "Import Data for Parameter Estimation" on page 1-5. After you import the data the experiment looks as follows:

Edit Experiment: EstimationData	X
Define measured output signals for this experiment. <u>lookup_regular/Lookup Table:1 (output)</u>	^
<1x1 Signal, 63 points>	
Inputs Optionally define input signals for this experiment. lookup regular/input:1 (input) <1x1 Signal, 63 points> Select Inputs	
Initial States Optionally define initial states for this experiment. There are currently no initial states defined for this experiment. E Select Initial States	Ш
Parameters Optionally define parameters for this experiment. There are currently no parameters defined for this experiment. Select Parameters	
🙀 Plot & Simulate 🛛 Plot 🖌 OK 🕜 Help	-

2 Run an initial simulation to view the I/O data, simulated output, and the initial table values. To do so, type the following commands at the MATLAB prompt:

```
sim('lookup_regular')
figure(1); plot(xdata1,ydata1, 'm*', xout, yout,'b^')
hold on; plot(linspace(0,6.5,10), table, 'k', 'LineWidth', 2);
legend('Measured data','Initial simulation data','Initial table values');
```



The x-axis and y-axis of the figure represent the input and output data, respectively. The figure shows the following plots:

- Measured data Represented by the magenta stars (*).
- Initial table values Represented by the black line.
- Initial simulation data Represented by the blue deltas (Δ).

You can see that the initial table values and simulated data do not match with the measured data.

3 To select the table values to estimate, on the Parameter Estimation tab, click the Select Parameters button to open the Edit:Estimated Parameters dialog. In the Parameters Tuned for all Experiments panel, click Select parameters to launch the Select Model Variables dialog. Check the box next to table, and click OK.

Current va [0 0 0 0 0 0	Used B lookup_regular/Lookup_Table	,	
	reekup regului/ ceekup rubie		
[0;0.16129			
[1.7185120			
sion indexing if r	ecessary (e.g., a(3) or s.x)		
	[1.7185120		[1.7185120

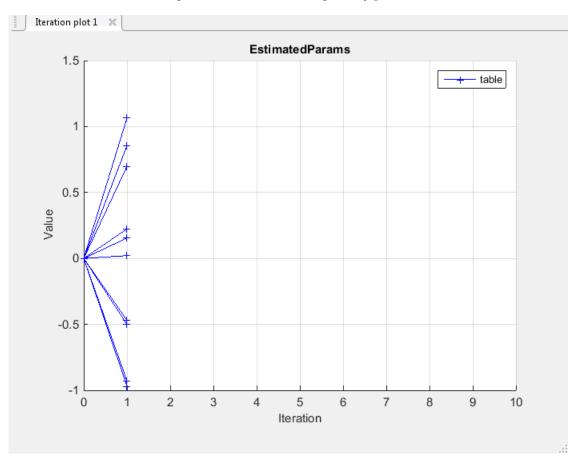
The **Edit:Estimated Parameters** window now looks as follows. The table values are selected for estimation by default.

Edit: Estimated Parameters ×
Parameters Tuned for all Experiments
table_
🕨 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Select parameters
Parameters and Initial States Tuned per Experiment
Experiment: EstimationData 💌
Select experiment initial states for estimation.
There are no initial states defined for this experiment.
Select experiment parameters for estimation.
There are no parameters defined for this experiment.
Edit experiment
🚵 Update Model 🛛 🔗 OK 📀 Help
.:

4 On the **Parameter Estimation** tab, click **Select Experiment**. EstimationData is selected for estimation by default. If not, check the box under the **Estimation** column, and click **OK**.

S	elect Experime	ents		×
	Select experim	ents to includ	e for estimation or validation	
	Estimation	Validation	Experiment	
	V		EstimationData	
				4
			🖋 ок (? Help
				:

5 To estimate the table values using the default settings, on the Parameter Estimation tab, click Estimate to open the Parameter Trajectory plot and Estimation Progress Report window. The Parameter Trajectory plot shows the change in the parameter values at each iteration.



After the estimation converges, the **Parameter Trajectory** plot looks like this:

The **Estimation Progress Report** shows the iteration number, number of times the objective function is evaluated, and the value of the cost function at the end of each iteration. After the estimation converges, the **Estimation Progress Report** looks like this:

承 Estimati	ion Progre	ss Report
Iteration	F-count	EstimationData (Minimize)
0	21	27.1233
1	42	0.1478
1		0-Jun-2014 17:14:37
'lookup_reg	gular' updat	ted with estimated parameter values
		Save Iteration Display Options Estimate

The estimated parameters are saved in EstimatedParams in the **Results** section of the **Data Browser** pane on the left. To view the results, right-click on EstimatedParams and then select **Open**. The report resembles the following.

View Result : EstimatedParams	Х
Estimation result(s): table = [0.021165 0.69751 1.0635 0.85461 0.22361 -0.46882 -0.93019 -0.96984 -0.50022 0.15432]	
Parameters estimated using experiments: EstimationData, cost = 0.14785	
Solver output:	
Cost: 0.14785 ExitFlag: 1 FCount: 43 Date: 10-Jun-2014 17:14	
Solver termination message:	
Local minimum found.	
Optimization completed because the size of the gradient is less than the selected value of the function tolerance.	
Stopping criteria details:	
Optimization completed: The first-order optimality measure, 1.181520e-14, is less than options.ToIFun = 1.00000e-03.	
Optimization Metric Options relative first-order optimality = 1.18e-14 TolFun = 1e-03 (selected)	
🔛 Use as initial guess 🛛 🔊 🖉 Update Model 🛛 🖉	ок

This report includes the estimated parameter values, the final value of the cost function, and other optimization results. You can see that the optimization stopped when the size of the gradient, 1.18e-14 was less than the criteria value, 1e-3.

Validate the Estimation Results

After you estimate the table values, as described in "Estimate the Table Values Using Default Settings" on page 6-18, you must use another data set to validate that you have not over-fitted the model. You can plot and examine the following plots to validate the estimation results:

- Residuals plot
- Measured and simulated data plots

To validate the estimation results:

- Create a new experiment to use for validation. Name it ValidationData. Import the validation I/O data, xdata2 and ydata2, and time vector, time2 in the ValidationData experiment. To do this open the experiment editor by right-clicking ValidationData and selecting Edit.... Then, type [time2, ydata2] in the output dialog box and [time2, xdata2] in the input dialog box in the experiment editor. For more information, see "Import Data for Parameter Estimation" on page 1-5.
- 2 To select the experiment for validation, on the Parameter Estimation tab, click Select Experiments. The ValidationData experiment is selected for estimation by default. Deselect the box for estimation and check it for validation.

Select Exp	eriment	s			×
Select exp	perimen	ts to includ	le for estimation or validation		
Estimat	ion V	alidation	Experiment		
V			EstimationData		₽
		V	ValidationData		<u> </u>
					
			🖋 ОК	?	Help

3 To select results to use, on the **Validation** tab, click **Select Results to Validate**. Deselect Use current parameter values and select EstimatedParams, and click **OK**.

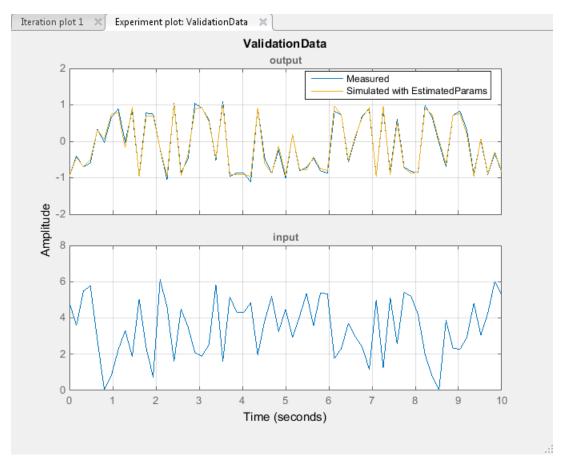
Sel	ect Re	sults	×
		Select results to include in validation	
		Results	
		Use current parameter values	
	1	EstimatedParams	
L			
		🖋 ок	(?) Help

4 The **Parameter Estimator**, by default, displays the experiment plot after validation. Add the residuals plot by checking the corresponding box on the **Validation** tab.

 Plot Measured & Simulated Data Plot Residuals
PLOT OPTIONS

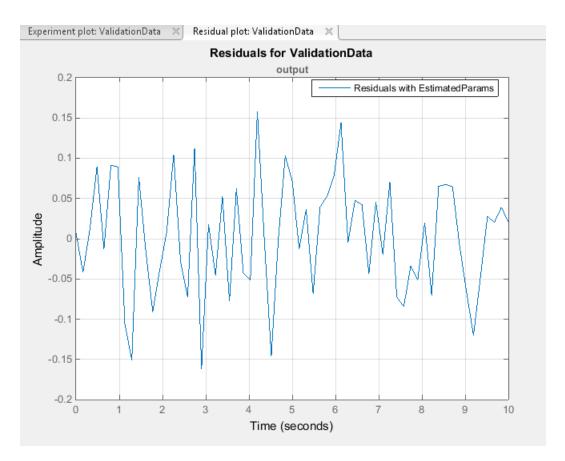
To start validation, on the **Validation** tab, click **Validate**.

- **5** Examine the plots
 - a Experiment plot



You can see that the data simulated using the estimated parameters agrees with the measured validation data.

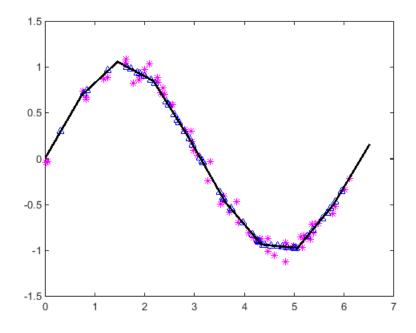
b Click Residual plot: ValidationData to open the residuals plot.



The residuals, which show the difference between the simulated and measured data, lie in the range [-0.15,0.15]— within 15% of the maximum output variation. This indicates a good match between the measured and the simulated table data values.

c Plot and examine the estimated table values against the validation data set and the simulated table values by typing the following commands at the MATLAB prompt.

```
sim('lookup_regular')
figure(2); plot(xdata2,ydata2, 'm*', xout, yout,'b^')
hold on; plot(linspace(0,6.5,10), table, 'k', 'LineWidth', 2)
```



The plot shows that the table values, displayed as the black line, match both the validation data and the simulated table values. The table data values cover the entire range of input values, which indicates that all the lookup table values have been estimated.

See Also

Related Examples

• "Estimate Constrained Values of a Lookup Table" on page 6-5

Building Models Using Adaptive Lookup Table Blocks

Simulink Design Optimization software provides blocks for modeling systems as adaptive lookup tables. You can use the adaptive lookup table blocks to create lookup tables from measured or simulated data. You build a model using the adaptive lookup table blocks, and then simulate the model to adapt the lookup table values to the time-varying I/O data. During simulation, the software uses the input data to locate the table values, and then uses the output data to recalculate the table values. The updated table values are stored in the adaptive lookup table block. For more information, see "What are Adaptive Lookup Tables?" on page 6-2.

The Adaptive Lookup Table library has the following blocks:

- Adaptive Lookup Table (1D Stair-Fit) One-dimensional adaptive lookup table
- Adaptive Lookup Table (2D Stair-Fit) Two-dimensional adaptive lookup table
- Adaptive Lookup Table (nD Stair-Fit) Multidimensional adaptive lookup table

Note Use the Adaptive Lookup Table (nD Stair-Fit) block to create lookup tables of three or more dimensions.

To access the Adaptive Lookup Tables library:

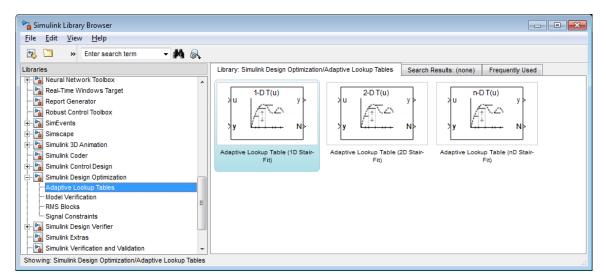
1 Open the Simulink Library Browser.

At the MATLAB prompt, enter slLibraryBrowser.

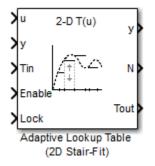
2 Open the Simulink Design Optimization library.

In the **Libraries** pane, expand the **Simulink Design Optimization** node.

3 In the Simulink Design Optimization library tree, click Adaptive Lookup Tables.



By default, the Adaptive Lookup Table blocks have two inputs and outputs. You can display additional inputs and outputs in a block by selecting the corresponding options in the Function Block Parameters dialog box. To learn more about the options, see the block reference pages.



Adaptive Lookup Table Block Showing Inputs and Outputs

The 2-D Adaptive Lookup Table block has the following inputs and outputs:

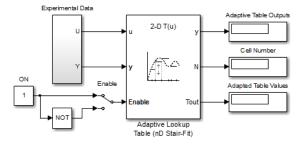
• u and y — Input and output data of the system being modeled, respectively.

For example, to model an engine's efficiency as a function of engine rpm and manifold pressure, specify u as the rpm, y as the pressure, and y as the efficiency signals.

- Tin The initial table data.
- Enable Signal to enable, disable, or reset the adaptation process.
- Lock Signal to update only specified cells in the table.
- y Value of the cell currently being adapted.
- N Number of the cell currently being adapted.
- Tout Values of the adapted table data.

For more information on how to use adaptive lookup tables, see "Model Engine Using n-D Adaptive Lookup Table" on page 6-33.

A typical Simulink diagram using an adaptive lookup table block is shown in the next figure.



Simulink Diagram Using an Adaptive Lookup Table

In this figure, the Experiment Data block imports a set of experimental data into Simulink through MATLAB workspace variables. The initial table is specified in the block mask parameters. When the simulation runs, the initial table begins to adapt to new data inputs and the resulting table is copied to the block's output.

See Also

Adaptive Lookup Table (1D Stair-Fit) | Adaptive Lookup Table (2D Stair-Fit) | Adaptive Lookup Table (nD Stair-Fit)

Related Examples

• "Model Engine Using n-D Adaptive Lookup Table" on page 6-33

More About

- "What are Adaptive Lookup Tables?" on page 6-2
- "Selecting an Adaptation Method" on page 6-31

Selecting an Adaptation Method

You specify the algorithm using the **Adaptation Method** drop-down list in the Function Block Parameters dialog box of an adaptive lookup table block. This section discusses the details of these algorithms.

Sample Mean

Sample mean provides the average value of *n* output data samples and is defined as:

$$\widehat{y}(n) = \frac{1}{n} \sum_{i=1}^{n} y(i)$$

where y(i) is the *i*th measurement collected within a particular *cell*. For each input data *u*, the sample mean at the corresponding cell is updated using the output data measurement, *y*. Instead of accumulating *n* samples of data for each cell, a recursive relation is used to calculate the sample mean. The recursive expression is obtained by the following equation:

$$\widehat{y}(n) = \frac{1}{n} \left[\sum_{i=1}^{n-1} y(i) + y(n) \right] = \frac{n-1}{n} \left[\frac{1}{n-1} \sum_{i=1}^{n-1} y(i) \right] + \frac{1}{n} y(n) = \frac{n-1}{n} \widehat{y}(n-1) + \frac{1}{n} y(n)$$

where y(n) is the n^{th} data sample.

Defining *a priori estimation error* as $e(n) = y(n) - \hat{y}(n-1)$, the recursive relation can be written as:

$$\widehat{y}(n) = \widehat{y}(n-1) + \frac{1}{n}e(n)$$

where $n \ge 1$ and the initial estimate $\hat{y}(0)$ is arbitrary.

In this expression, only the number of samples, n, for each cell— rather than n data samples—is stored in memory.

Sample Mean with Forgetting

The adaptation method "Sample Mean" on page 6-31 has an *infinite memory*. The past data samples have the same weight as the final sample in calculating the sample mean. Sample mean (with forgetting) uses an algorithm with a *forgetting factor* or Adaptation gain that puts more weight on the more recent samples. This algorithm provides robustness against initial response transients of the plant and an adjustable speed of adaptation. Sample mean (with forgetting) is defined as:

$$\widehat{y}(n) = \frac{1}{\sum_{i=1}^{n} \lambda^{n-i}} \sum_{i=1}^{n} \lambda^{n-i} y(i)$$
$$= \frac{1}{\sum_{i=1}^{n} \lambda^{n-i}} \left[\sum_{i=1}^{n-1} \lambda^{n-i} y(i) + y(n) \right] = \frac{s(n-1)}{s(n)} \widehat{y}(n-1) + \frac{1}{s(n)} y(n)$$

where $\lambda \in [0, 1]$ is the **Adaptation gain** and $s(k) = \sum_{i=1}^{k} \lambda^{n-i}$.

Defining a priori estimation error as $e(n) = y(n) - \hat{y}(n-1)$, where $n \ge 1$ and the initial estimate $\hat{y}(0)$ is arbitrary, the recursive relation can be written as:

$$\widehat{y}(n) = \widehat{y}(n-1) + \frac{1}{s(n)}e(n) = \widehat{y}(n-1) + \frac{1-\lambda}{1-\lambda^n}e(n)$$

A small value of λ results in faster adaptation. A value of 0 indicates short memory (last data becomes the table value), and a value of 1 indicates long memory (average all data received in a cell).

Model Engine Using n-D Adaptive Lookup Table

Objectives

In this example, you learn how to capture the time-varying behavior of an engine using an n-D adaptive lookup table. You accomplish the following tasks using the Simulink software:

- Configure an adaptive lookup table block to model your system.
- Simulate the model to update the lookup table values dynamically.
- Export the adapted lookup table values to the MATLAB workspace.
- Lock a specific cell in the table during adaptation.
- Disable the adaptation process and use the adaptive lookup table as a static lookup table.

About the Data

In this example, you use the data in vedata.mat which contains the following variables measured from an engine:

- X 10 input breakpoints for intake manifold pressure in the range [10,100]
- Y 36 input breakpoints for engine speed in the range [0,7000]
- Z 10x36 matrix of table data for engine volumetric efficiency

To learn more about breakpoints and table data, see "Anatomy of a Lookup Table".

The output volumetric efficiency of the engine is time varying, and a function of two inputs—intake manifold pressure and engine speed. The data in the MAT-file is used to generate the time-varying input and output (I/O) data for the engine.

Building a Model Using Adaptive Lookup Table Blocks

In this portion of the tutorial, you learn how to build a model of an engine using an Adaptive Lookup Table block.

1 Open a preconfigured Simulink model by typing the model name at the MATLAB prompt:

enginetable1_data

The Experimental Data subsystem in the Simulink model generates time-varying I/O data during simulation.

🍋 en	iginetab	le1_dat	a								_		x
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>D</u> isplay	Dia <u>gr</u> am	<u>S</u> imulatio	on <u>A</u> na	alysis <u>(</u>	ode	<u>T</u> ools	<u>H</u> elp			
▶	•		<₽ ₽		i 🖗 י	-	4) 🕪			• » (⊘	**** •
engir	netable 1	_data											
۲	🎦 eng	inetable	e1_data 🕨										•
Q													
K 7 K 2		Ex	perimental D	ata									
⇒													
EA			Y										
>>													
Read	у					100%				F	ixedSte	epDis	crete 🔡

This command also loads the variables X, Y and Z into the MATLAB workspace. To learn more about this data, see "About the Data" on page 6-33.

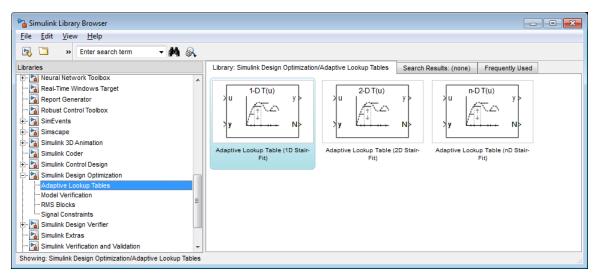
- 2 Add an Adaptive Lookup Table block to the Simulink model.
 - a Open the Simulink Library Browser.

At the MATLAB prompt, enter slLibraryBrowser.

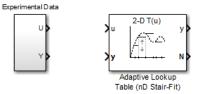
b Open the Simulink Design Optimization library.

In the Libraries pane, expand the Simulink Design Optimization node.

c In the Simulink Design Optimization library tree, click Adaptive Lookup Tables.



d Drag and drop the Adaptive Lookup Table (nD Stair-Fit) block from the Adaptive Lookup Tables library to the Simulink model window.



3 Double-click the Adaptive Lookup Table (nD Stair-Fit) block to open the Function Block Parameters: Adaptive Lookup Table (nD Stair-Fit) dialog box.

🚡 Function Block Parameters: Adaptive Lookup Table (nD Stair-Fit)						
Adaptive Lookup Table (nD) (mask) (link)						
Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.						
Parameters						
Number of table dimensions:						
2						
Table breakpoints (cell array):						
{[10,22,31,40], [10,22,31,40]}						
Make initial table an input						
Table data (initial):						
[4 5 6;16 19 20;10 18 23]						
Table numbering data:						
[1 2 3; 4 5 6; 7 8 9]						
Adaptation method: Sample mean (with forgetting)						
Adaptation gain (0 to 1):						
0.9						
Make adapted table an output						
Add adaptation enable/disable/reset port						
Add cell lock enable/disable port						
Action for out-of-range input Ignore						
OK Cancel Help Apply						

- 4 In the Function Block Parameters dialog box:
 - **a** Specify the following block parameters:
 - Table breakpoints (cell array) Enter {[X; 110], [Y; 7200]} to specify the range of input breakpoints.
 - **Table data (initial)** Enter rand(10,36) to specify random numbers as the initial table values for the volumetric efficiency.
 - **Table numbering data** Enter reshape(1:360,10,36) to specify a numbering scheme for the table cells.
 - **b** Verify that Sample mean (with forgetting) is selected in the **Adaptation method** drop-down list.

c Enter 0.98 in the Adaptation gain (0 to 1) field to specify the *forgetting factor* for the Sample mean (with forgetting) adaptation algorithm.

An adaptation gain close to 1 indicates high robustness of the lookup table values to input noise. To learn more about the adaptation gain, see "Sample Mean with Forgetting" on page 6-31 in "Selecting an Adaptation Method" on page 6-31.

d Select the Make adapted table an output check box.

This action adds a new port named **Tout** to the Adaptive Lookup Table block. You use this port to plot the table values as they are being adapted.

e Select the Add adaptation enable/disable/reset port check box.

This action adds a new port named **Enable** to the Adaptive Lookup Table block. You use this port to enable or disable the adaptation process.

f Select the Add cell lock enable/disable port check box.

This action adds a new port named Lock to the Adaptive Lookup Table block. You use this port to lock a cell during the adaptation process.

g Verify that Ignore is selected in the Action for out-of-range drop-down list.

This selection specifies that the software ignores any time-varying inputs outside the range of input breakpoints during adaptation.

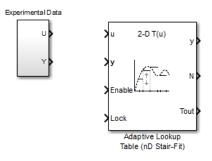
Tip To learn more, see Adaptive Lookup Table (nD Stair-Fit) block reference page.

After you configure the parameters, the block parameters dialog box looks like the following figure.

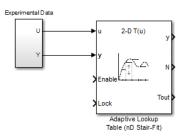
🚡 Function Block Parameters: Adaptive Lookup Table (nD Stair-Fit)						
Adaptive Lookup Table (nD) (mask) (link)						
Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.						
Parameters						
Number of table dimensions:						
2						
Table breakpoints (cell array):						
{[X; 110], [Y; 7200]}						
Make initial table an input						
Table data (initial):						
rand(10,36)						
Table numbering data:						
reshape(1:360,10,36)						
Adaptation method: Sample mean (with forgetting)						
Adaptation gain (0 to 1):						
0.98						
Make adapted table an output						
Add adaptation enable/disable/reset port						
Add cell lock enable/disable port						
Action for out-of-range input Ignore						
OK Cancel Help Apply						

h Click **OK** to close the Function Block Parameters dialog box.

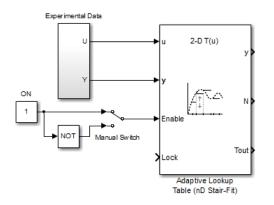
The Simulink model now looks similar to the following figure.



5 Assign the input and output data to the engine model by connecting the U and Y ports of the Experimental Data block to the u and y ports of the Adaptive Lookup Table block, respectively.

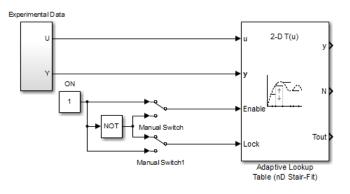


6 Design a logic using Simulink blocks to enable or disable the adaptation process. Connect the logic to the Adaptive Lookup Table block, as shown in the following figure.

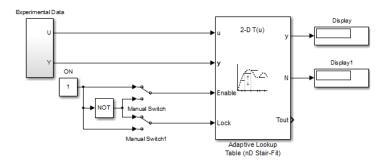


This logic outputs an initial value of 1 which enables the adaptation process.

7 Design a logic to lock a cell during adaptation. Connect the logic to the Adaptive Lookup Table block, as shown in the following figure.



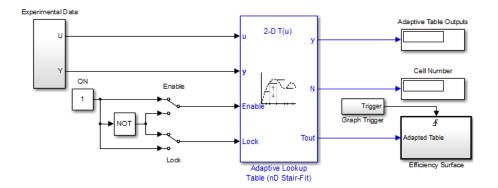
8 In the Simulink Library Browser, select the **Simulink** > **Sinks** library, and drag Display blocks to the model window. Connect the blocks, as shown in the following figure.



During simulation, the Display blocks show the following:

- Display block Shows the value of the current cell being adapted.
- Display1 block Shows the number of the current cell being adapted.
- **9** Write a MATLAB function to plot the lookup table values as they adapt during simulation.

Alternatively, type enginetable at the MATLAB prompt to open a preconfigured Simulink model. The Efficiency Surface subsystem contains a function to plot the lookup table values, as shown in the next figure.



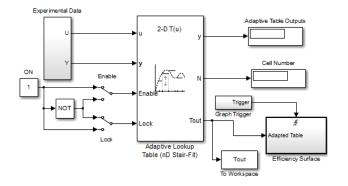
- **10** Connect a To Workspace block to export the adapted table values:
 - a In the Simulink Library Browser, select the **Simulink** > **Sinks** library, and drag the To Workspace block to the model window.

To learn more about this block, see the To Workspace block reference page in the Simulink documentation.

b Double-click the To Workspace block to open the Sink Block Parameters dialog box, and type Tout in the **Variable name** field.

🚡 Sink Block Parameters: To Workspace						
To Workspace						
Write input to specified timeseries, array, or structure in a workspace. For menu-based simulation, data is written in the MATLAB base workspace. Data is not available until the simulation is stopped or paused. For command-line simulation using the sim command, the workspace is specified using DstWorkspace field in the option structure.						
To log a bus signal, use "Timeseries" save format.						
Parameters						
Variable name:						
Tout						
Limit data points to last:						
inf						
Decimation:						
1						
Sample time (-1 for inherited):						
-1						
Save format: Structure						
Log fixed-point data as a fi object						
OK Cancel Help Apply						

- c Click OK.
- **d** Connect the To Workspace block to the adaptive lookup table output signal **Tout**, as shown in the next figure.



You have now built the model for updating and viewing the adaptive lookup table values. You must now simulate the model to start the adaptation, as described in "Adapting the Lookup Table Values Using Time-Varying I/O Data" on page 6-40.

Adapting the Lookup Table Values Using Time-Varying I/O Data

In this portion of the tutorial, you learn how to update the lookup table values to adapt to the timevarying input and output values.

You must have already built the Simulink model, as described "Building a Model Using Adaptive Lookup Table Blocks" on page 6-33.

To perform the adaptation:

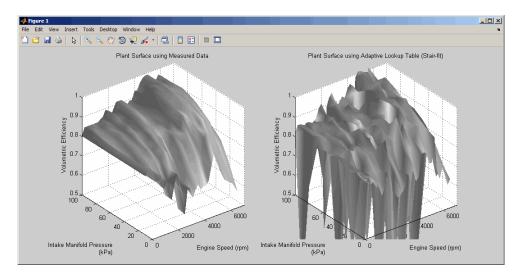
1 In the Simulink Editor, specify the simulation time as inf.

The simulation time of infinity specifies that the adaptation process continues as long as the input and output values of the engine change.

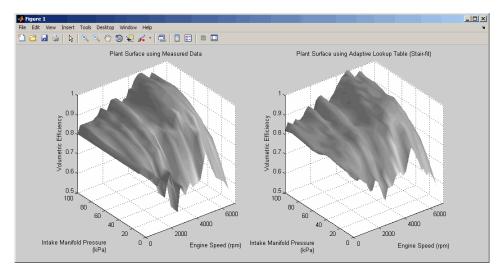
2 In the Simulink Editor, click **Run** under **Simulation** to start the adaptation process.

A figure window opens that shows the volumetric efficiency of the engine as a function of the intake manifold pressure and engine speed:

- The left plot shows the measured volumetric efficiency as a function of intake manifold pressure and engine speed.
- The right plot shows the volumetric efficiency as it adapts with the time-varying intake manifold pressure and engine speed.



During simulation, the lookup table values displayed on the right plot adapt to the variations in the I/O data. The left and the right plots resemble each other after a few seconds, as shown in the next figure.



Tip During simulation, the Cell Number and Adaptive Table Outputs blocks in the Simulink model display the cell number, and the adapted lookup table value in the cell, respectively.

3 Pause the simulation by clicking **Pause** under **Simulation**.

This action also exports the adapted table values Tout to the MATLAB workspace.

Note After you pause the simulation, the adapted table values are stored in the Adaptive Lookup Table block.

- **4** Examine that the left and the right plots match. This resemblance indicates that the table values have adapted to the time-varying I/O data.
- 5 Lock a table cell so that only one cell adapts. You may find this feature useful if a portion of the data is highly erratic or otherwise difficult for the algorithm to handle.

- a Click **Run** under **Simulation** to restart the simulation.
- b Double-click the Lock block. This action toggles the switch and feeds the output of the ON block to theLock input port of the Adaptive Lookup Table(nD Stair-Fit) block.

You can view the number of the locked cell in the Cell Number block in the Simulink model.

- 6 After the table values adapt to the time-varying I/O data, you can continue to use the Adaptive Lookup Table block as a static lookup table:
 - **a** In the Simulink model window, double-click the Enable block. This action toggles the switch, and disables the adaptation.
 - **b** Click **Run** under **Simulation** to restart the simulation, if it is not already running.

During simulation, the Adaptive Lookup Table block works like a static lookup table, and continues to estimate the output values as the input values change. You can see the current lookup table value in the Adaptive Table Outputs block in the Simulink model window.

Note After you disable the adaptation, the Adaptive Lookup Table block does not update the stored table values, and the figure that displays the table values does not update.

See Also

Adaptive Lookup Table (nD Stair-Fit)

More About

• "What are Adaptive Lookup Tables?" on page 6-2

Using Adaptive Lookup Tables in Real-Time Environment

You can use experimental data from sensor measurements collected by running various tests on a system in real time. The measured data is then sent to the adaptive table block to generate a lookup table describing the relation between the system inputs and output.

You can also use the Adaptive Lookup Table block in a real-time environment, where some timevarying properties of a system need to be captured. To do so, generate C code using Simulink Coder[™] code generation software that can then be run in Simulink Real-Time[™] or dSPACE[®] software. Because you can start, stop, or reset the adaptation if you want, use logic to enable the adaptation of the table data only when it is desired. The cell number output N, and the Enable and Lock inputs facilitate this process. Use the Enable input to start and stop the adaptation and the Lock input to update only one of the table cells. The Lock input combined with some logic using the cell number output N provide the means for updating only the desired table cells during a simulation run.

See Also

Adaptive Lookup Table (1D Stair-Fit) | Adaptive Lookup Table (2D Stair-Fit) | Adaptive Lookup Table (nD Stair-Fit)

Related Examples

• "Model Engine Using n-D Adaptive Lookup Table" on page 6-33

More About

• "What are Adaptive Lookup Tables?" on page 6-2

Design Optimization Using Lookup Table Requirements for Gain Scheduling (Code)

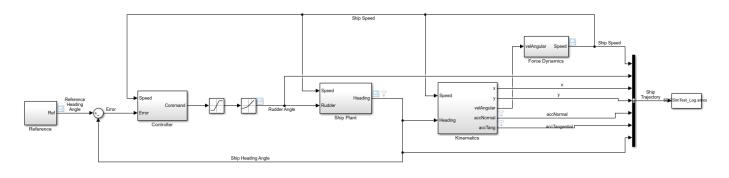
This example shows how to tune parameters in a lookup table in a model that uses gain scheduling to adjust the controller's response to a plant that varies. Model tuning uses the sdo.optimize command.

Ship Steering Model

Open the Simulink® Model.

mdl = 'sdoShipSteering';
open_system(mdl)

Ship Steering



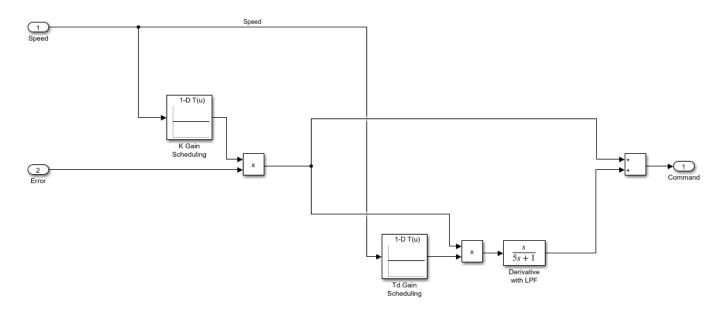
Copyright 2016 The MathWorks, Inc

This model implements the Nomoto model which is commonly used for ship steering. The dynamic characteristics of a ship vary significantly with factors such as the ship speed. Therefore the rudder controller should also vary with speed, in order to meet requirements for steering the ship.

To keep the ship on course, a control loop compares the ship's heading angle with the reference heading angle, and a PD controller sends command signals to the rudder. The Ship Plant block implements the Nomoto model, a second order system whose parameters vary with the ship's speed. The ship is initially traveling at its maximum speed of 15 m/s, but it will slow down when the reference trajectory specifies a turn in the water. This turning, along with the force of the engine, is used by the Force Dynamics block to compute the speed of the ship over time. The Kinematics block computes the trajectory of the ship.

Open the Controller block.

open_system([mdl '/Controller'])



When the speed changes, the ship plant also changes. Therefore the PD controller gains need to change, and speed is used as the scheduling variable. The controller is in the form K(1 + sTd) where K is the overall gain and Td is the time constant associated with the derivative term. Gain scheduling is implemented via lookup tables, and the table data are specified by K and Td. These are vectors which specify different values for different speeds. The different speeds are specified in the lookup table breakpoint vectors bpK and bpTd.

Design Problem

The reference specifies that at 200 seconds, the ship should turn 180 degrees and reverse course. One requirement is that the ship heading angle needs to match the reference heading angle within an envelope. For the safety and comfort of passengers, a second requirement is that the total acceleration of the ship needs to stay within a bound of 0.25 g's, where 1 g is the acceleration of gravity at Earth's surface, 9.8 m/s/s.

The controller parameter vectors K and Td will be design variables, and will be tuned to try to meet the requirements. If it is not possible to meet both requirements, then the lookup table breakpoints bpK and bpTd will also be used as design variables. In that case, we will need to specify an additional requirement that bpK and bpTd must be monotonically strictly increasing, because this is required for breakpoint vectors in Simulink lookup tables.

Specify Design Requirements

Specify the requirements which must be satisfied. First, the ship should follow the reference trajectory. Since the reference is essentially a step change from 0 to 180 degrees, you specify a step response envelope for the ship heading angle.

```
Requirements = struct;
Requirements.StepResponseEnvelope = sdo.requirements.StepResponseEnvelope(...
'StepTime', 200, ... (seconds)
'RiseTime', 75, ...
'SettlingTime', 200, ...
'PercentRise', 85, ... (%)
'PercentOvershoot', 5, ...
'PercentSettling', 1, ...
```

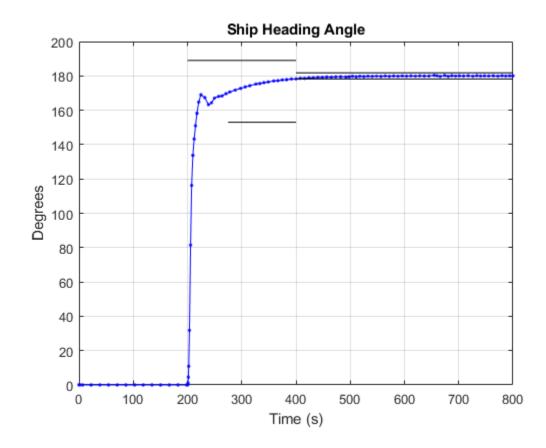
'FinalValue', pi ... (radians)
);

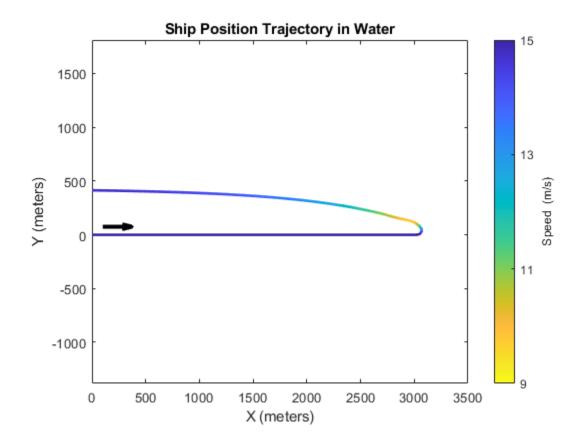
The second requirement is that for the safety and comfort of passengers, the total acceleration should not exceed 0.25 g's at any time. The total acceleration is composed of two components, a tangential component along the direction of the ship's motion, and a normal (horizontal) component. The requirement that total acceleration not exceed 0.25 g's corresponds to requiring that in the phase plane of tangential and normal acceleration, this ship's trajectory remain within a circle of radius 0.25*9.8.

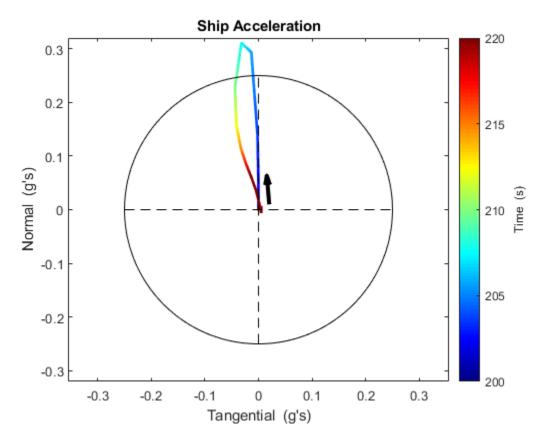
```
accelGravity = 9.8;
Requirements.Comfort = sdo.requirements.PhasePlaneEllipse(...
'Radius', 0.25*[1 1] * accelGravity );
```

Examine the behavior of the ship with the initial, untuned controller parameters, to see whether the requirements are satisfied. Use the function sdoShipSteeringPlots to plot the ship's behavior. The first plot below shows that the ship's heading angle stays within the tolerance bounds of the step response envelope, satisfying the first requirement. The second plot shows the trajectory of the ship in the water. The black arrow indicates the starting position and direction of motion. The ship turns 180 degrees and the diameter is approximately 415 meters. The third plot shows the ship acceleration in the phase plane of tangential and normal acceleration. The black arrow indicates the starting point and direction of evolution over time. The total acceleration exceeds the bound near 208 seconds, so the requirement for passenger safety and comfort is not satisfied.

```
hPlots = sdoShipSteeringPlots(mdl, Requirements);
```







Specify Design Variables

Specify the design variables to be tuned by the optimization routine in order to satisfy the requirements. Specify the gains of the PD controller, K and Td, as design variables. For initial values, use -0.1 for all entries in the K vector, and 50 for all entries in the Td vector. If the requirements can't all be satisfied, then later the breakpoint vectors bpK and bpTd can also be design variables.

```
DesignVars = sdo.getParameterFromModel(mdl, {'K', 'Td'});
DesignVars(1).Value = -0.1*[1 1 1 1];
DesignVars(2).Value = 50*[1 1 1 1];
sdo.setValueInModel(mdl, DesignVars);
```

Create Optimization Objective Function

Create an objective function that will be called at each optimization iteration to evaluate the design requirements as the design variables are tuned. This cost function has input arguments for the model, design variables, simulator (defined below), and design requirements. The cost function uses the maximum of the Comfort requirements at all times when it is computed, in order to consolidate requirement evaluation results to a scalar so the number of elements is the same regardless of time steps taken by the Simulink solver.

type sdoShipSteering_Design

```
function Vals = sdoShipSteering_Design(mdl, P, Simulator, Requirements)
%SD0SHIPSTEERING_DESIGN Objective function for ship steering
%
```

```
% Function called at each iteration of the optimization problem.
%
% The function is called with the model named mdl, a set of parameter
% values, P, a Simulator, and the design Requirements to evaluate. It
% returns the objective value and constraint violations, Vals, to the
% optimization solver.
%
% See the sdoExampleCostFunction function and sdo.optimize for a more
% detailed description of the function signature.
%
% See also sdoShipSteering cmddemo
% Copyright 2016 The MathWorks, Inc.
%% Model Evaluation
% Simulate the model.
Simulator.Parameters = P;
Simulator = sim(Simulator):
% Retrieve logged signal data.
SimLog = find(Simulator.LoggedData,get param(mdl, 'SignalLoggingName'));
Heading = find(SimLog, 'Heading');
NormalAccel = find(SimLog, 'NormalAccel');
TangenAccel = find(SimLog, 'TangenAccel');
% Evaluate the step response envelope requirement
Cleg StepResponseEnvelope = evalRequirement(Requirements.StepResponseEnvelope, Heading.Values);
% Evaluate the safety/comfort requirement on total acceleration.
Cleq Comfort = evalRequirement(Requirements.Comfort, NormalAccel.Values, TangenAccel.Values);
%% Return Values.
% Collect the evaluated design requirement values in a structure to return
% to the optimization solver.
Vals.Cleg = Cleg StepResponseEnvelope(:);
Vals.Cleg = [Vals.Cleg ; Cleg Comfort];
% Evaluate monotonic variable requirement
if isfield(Requirements, 'Monotonic')
    Cleg bpK = evalRequirement(Requirements.Monotonic, P(3).Value);
    Clea bpTd = evalRequirement(Requirements, Monotonic, P(4), Value);
    Vals.Cleg = [Vals.Cleg ; Cleg bpK ; Cleg bpTd];
end
end
```

The cost function requires a Simulator to run the model. Create the simulator and add model signals to log, so their values are available to the cost function.

```
% Ship Heading Angle
Heading = Simulink.SimulationData.SignalLoggingInfo;
Heading.BlockPath = 'sdoShipSteering/Ship Plant';
Heading.LoggingInfo.LoggingName = 'Heading';
Heading.LoggingInfo.NameMode = 1;
```

Simulator = sdo.SimulationTest(mdl);

Heading.OutputPortIndex = 1;

```
% Normal Acceleration
NormalAccel = Simulink.SimulationData.SignalLoggingInfo;
NormalAccel.BlockPath = 'sdoShipSteering/Kinematics';
NormalAccel.LoggingInfo.LoggingName = 'NormalAccel';
NormalAccel.LoggingInfo.NameMode = 1;
NormalAccel.OutputPortIndex = 4;
```

```
% Tangential Acceleration
TangenAccel = Simulink.SimulationData.SignalLoggingInfo;
TangenAccel.BlockPath = 'sdoShipSteering/Kinematics';
TangenAccel.LoggingInfo.LoggingName = 'TangenAccel';
TangenAccel.LoggingInfo.NameMode = 1;
TangenAccel.OutputPortIndex = 5;
```

```
% Collect logged signals
Simulator.LoggingInfo.Signals = [...
Heading ; ...
NormalAccel ; ...
TangenAccel ];
```

Optimize Lookup Table Data

Before optimizing, set up the simulator for fast evaluation by enabling Simulink Fast Restart.

```
Simulator = fastRestart(Simulator, 'on');
```

During optimization, the Simulink solver may generate a warning if the size of the time step becomes too small. Temporarily suppress this warning.

```
warnState = warning('query', 'Simulink:Engine:SolverMinStepSizeWarn');
warnState1 = warning('query', 'Simulink:Solver:Warning');
warnState2 = warning('query', 'SimulinkExecution:DE:MinStepSizeWarning');
warning('off', 'Simulink:Engine:SolverMinStepSizeWarn');
warning('off', 'Simulink:Solver:Warning');
warning('off', 'SimulinkExecution:DE:MinStepSizeWarning');
```

To optimize, define a handle to the cost function that uses the Simulator and Requirements defined above. Use an anonymous function that takes one argument (the design variables) and calls the objective function. Finally, call sdo.optimize to optimize the design variables to try to meet the requirements.

```
optimfcn = @(P) sdoShipSteering_Design(mdl, P, Simulator, Requirements);
[Optimized_DesignVars, Info] = sdo.optimize(optimfcn, DesignVars);
```

Optimization started 02-Sep-2021 00:17:02

			max		First-order
Iter F	-count	f(x)	constraint	Step-size	optimality
0	17	Θ	0.6127		
1	34	Θ	0.3215	0.865	0.321
2	57	Θ	0.3079	0.174	0.308
3	103	Θ	0.3079	0.00127	0.308
Canadana	ad to ar	infoncible.	noint		

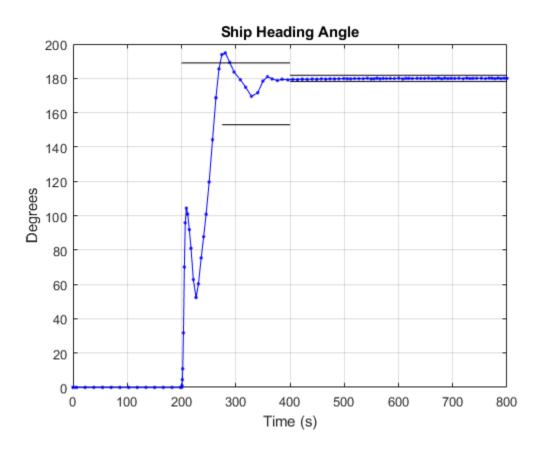
Converged to an infeasible point.

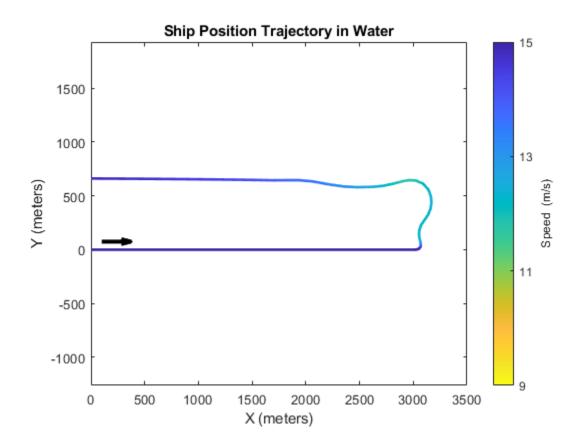
fmincon stopped because the size of the current step is less than

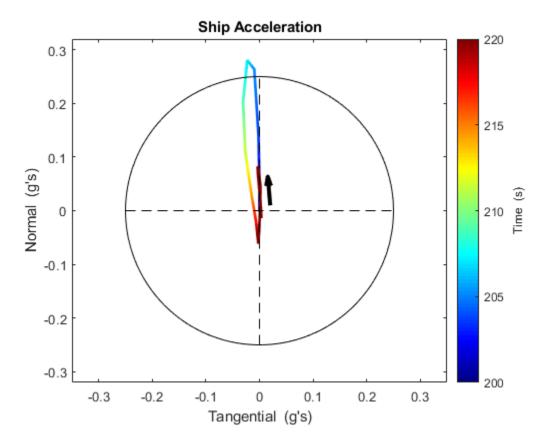
the value of the step size tolerance but constraints are not satisfied to within the value of the constraint tolerance.

The display indicates that the optimizer was not able to satisfy all requirements. Try the optimized design variables in the model and plot the results. The heading angle is not within the required step response envelope, and the total acceleration still exceeds the allowable level during part of the turn. In addition, the turning diameter has increased to 660 meters, so the turn is not as tight as with untuned gains.

sdo.setValueInModel(mdl, Optimized_DesignVars); hPlots = sdoShipSteeringPlots(mdl, Requirements, hPlots);







Optimize Lookup Table Data and Breakpoints

To try to meet the design requirements, use the optimization result from above as the start point, and tune additional variables. Add breakpoints bpK and bpTd as design variables. The ship's maximum speed is 15 m/s, and during turning it may slow to 60% of the maximum speed, or 9 m/s. Set the breakpoint initial values to be equally spaced between 9 and 15 m/s. Constrain the breakpoint minimum values to 9 m/s, and constrain the breakpoint maximum values to 15 m/s.

```
DesignVars = Optimized_DesignVars;
DesignVars(3:4) = sdo.getParameterFromModel(mdl, {'bpK','bpTd'});
% Set initial values
DesignVars(3).Value = [9 11 13 15];
DesignVars(4).Value = [9 11 13 15];
% Constrain min and max values
DesignVars(3).Minimum = 9;
DesignVars(3).Maximum = 15;
DesignVars(4).Minimum = 9;
DesignVars(4).Maximum = 15;
% Set values in the model
sdo.setValueInModel(mdl, DesignVars);
```

Breakpoints in the Simulink lookup table block must be strictly monotonically increasing. Add this to the design requirements.

Requirements.Monotonic = sdo.requirements.MonotonicVariable; optimfcn = @(P) sdoShipSteering_Design(mdl, P, Simulator, Requirements);

Optimize the model by tuning all four design variables, to see if all requirements can be met.

```
[Optimized_DesignVars, Info] = sdo.optimize(optimfcn, DesignVars);
```

Optimization started 02-Sep-2021 00:17:30

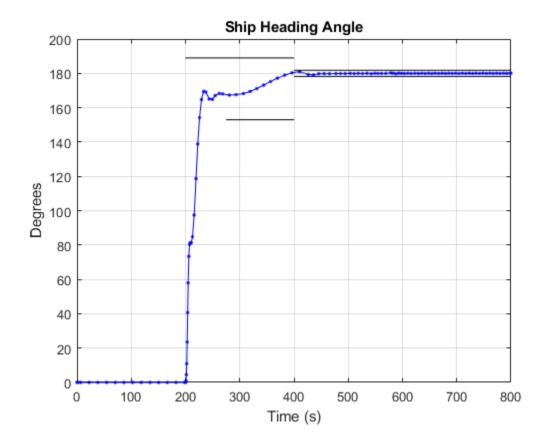
				max		First-order
Iter	F-count		f(x)	constraint	Step-size	optimality
0	29		0	0.3079		
1	61		0	0.1432	0.148	1.01
2	94		0	0.03626	0.0858	0.597
3	123		0	0.01911	0.0548	0.0859
4	153		Θ	0.007837	0.0341	0.00627
5	183		0	Θ	0.0256	0.000903
Local	minimum	found	that a	satisfies the	constraints	

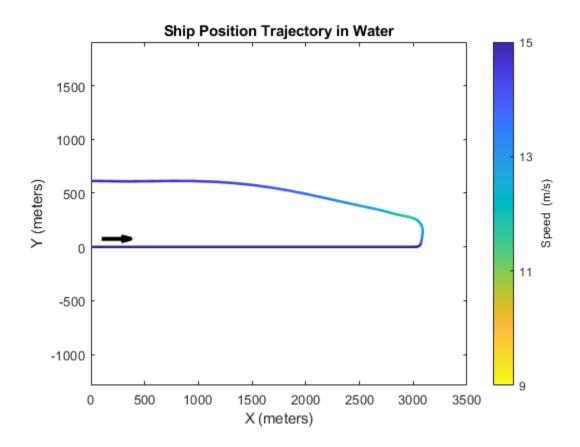
Local minimum found that satisfies the constraints.

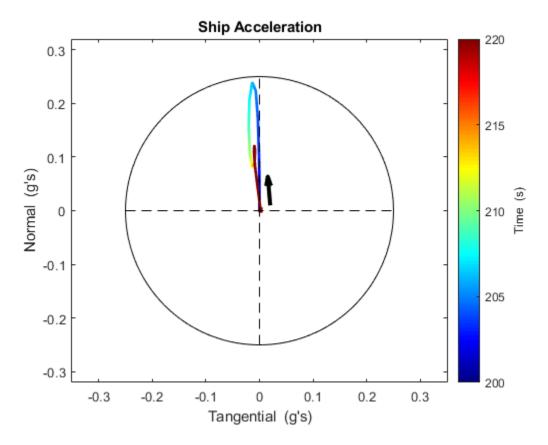
Optimization completed because the objective function is non-decreasing in feasible directions, to within the value of the optimality tolerance, and constraints are satisfied to within the value of the constraint tolerance.

The display indicates that the optimizer was able to satisfy all requirements. Try the optimized design variables in the model and plot the results. In the plots below, the heading angle is within the step response envelope, and the total acceleration is within the allowed range of 0.25 g's. In addition, the turning diameter is 615 meters, which is tighter than when breakpoints were not tuned.

sdo.setValueInModel(mdl, Optimized_DesignVars); hPlots = sdoShipSteeringPlots(mdl, Requirements, hPlots);







In this example, the ship plant varied with the ship speed, so the controller gains also needed to vary. Gain scheduling was implemented using lookup tables. By tuning the gains and breakpoint values in the controller, the ship was able to follow the reference heading angle, while also constraining total acceleration to ensure a safe and comfortable ride for passengers.

Related Examples

To learn how to optimize the lookup tables in the gain scheduled controller using the **Response Optimizer**, see "Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)" on page 6-59.

```
% Close the model and figures, and restore state of warnings.
fastRestart(Simulator, 'off'); % restore fast restart settings
bdclose(mdl)
close(hPlots)
warning(warnState); % restore state of warning
warning(warnState1); % restore state of warning
warning(warnState2); % restore state of warning
```

See Also

sdo.requirements.FunctionMatching|sdo.requirements.MonotonicVariable| sdo.requirements.PhasePlaneEllipse|sdo.requirements.PhasePlaneRegion| sdo.requirements.RelationalConstraint|sdo.requirements.SmoothnessConstraint

Related Examples

- "Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)" on page 6-59
- "Estimate Lookup Table Values from Data" on page 6-17

Design Optimization Using Lookup Table Requirements for Gain Scheduling (GUI)

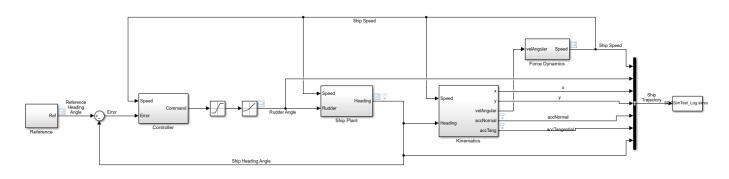
This example shows how to tune parameters in a lookup table in a model that uses gain scheduling to adjust the controller's response to a plant that varies. Model tuning uses the **Response Optimizer** app.

Ship Steering Model

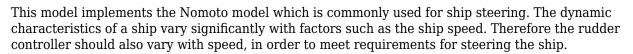
Open the Simulink® model.

open_system('sdoShipSteering')

Ship Steering

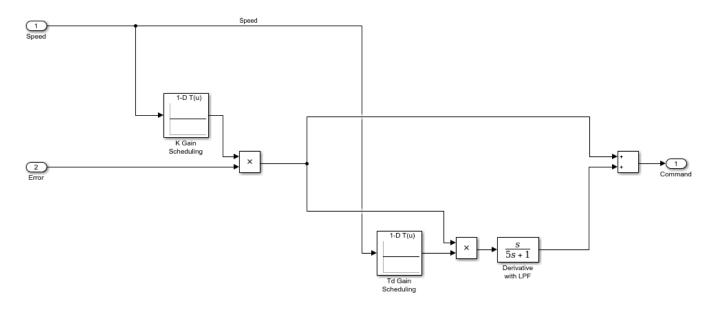


Copyright 2016 The MathWorks, Inc



To keep the ship on course, a control loop compares the ship's heading angle with the reference heading angle, and a PD controller sends command signals to the rudder. The Ship Plant block implements the Nomoto model, a second order system whose parameters vary with the ship's speed. The ship is initially traveling at its maximum speed of 15 m/s, but it will slow down when the reference trajectory specifies a turn in the water. This turning, along with the force of the engine, is used by the Force Dynamics block to compute the speed of the ship over time. The Kinematics block computes the trajectory of the ship.

Open the Controller block by double clicking it.



When the speed changes, the ship plant also changes. Therefore the PD controller gains need to change, and speed is used as the scheduling variable. The controller is in the form K(1 + sTd) where K is the overall gain and Td is the time constant associated with the derivative term. Gain scheduling is implemented using lookup tables, and the table data are specified by K and Td. These are vectors which specify different values for different speeds. The different speeds are specified in the lookup table breakpoint vectors bpK and bpTd.

Design Problem

The reference specifies that at 200 seconds, the ship should turn 180 degrees and reverse course. One requirement is that the ship heading angle needs to match the reference heading angle within an envelope. For the safety and comfort of passengers, a second requirement is that the total acceleration of the ship needs to stay within a bound of 0.25 g's, where 1 g is the acceleration of gravity at Earth's surface, 9.8 m/s^2 .

The controller parameter vectors K and Td will be design variables, and will be tuned to try to meet the requirements. If it is not possible to meet both requirements, then the lookup table breakpoints bpK and bpTd will also be used as design variables. In that case, we will need to specify an additional requirement that bpK and bpTd must be monotonically strictly increasing, because this is required for breakpoint vectors in Simulink lookup tables.

Open the Response Optimizer

In the Apps tab, click Response Optimizer under Control Systems.

Specify Design Requirements

Specify the requirements which must be satisfied. First, the ship should follow the reference trajectory. Since the reference is essentially a step change from 0 to 180 degrees, you specify a step response envelope for the ship heading angle. In the toolstrip, click **New** and select **Step Response Envelope**. Set the initial value to 0 and the final value to pi radians. Set the step time to 200 seconds. Set the rise time as 75 seconds and the rise percent to 85%. Set the settling time to 200 seconds and the settling percent to 1%. Set the percent overshoot to 5%. To specify that this requirement applies to the ship heading, click +.

Create Requirem	ent				×
	Step I	Response	Envelope		
	Specify a step	response e	nvelope on a signa	al.	
Name CL. D	5 1				
Name: StepRes	spEnvelope				
▼ Specify Ste	p Response Characteris	tics			
Initial value:	0		Final value:		3.1416
Step time :	200	seconds			
Rise time :	75	seconds	% Rise:		85
Settling time :	200	seconds	% Settling:		1.0000
% Overshoot:	5]	% Undershoot:		1.0000
- Coloct Sign	als to Round				
▼ Select Sign					1
V Headi	ingAngle (sdoShipSteerir	Signal			
M Head	ingAngle (soosnipsteenr	ig/ship Pia	nt:1)		
Create Plot			OK	Cancel	Help

In the Simulink model click the ship heading signal, which is the output of the **Ship Plant** block. Select this signal in the Create Signal Set dialog, and click the arrow button to make it the designated signal, and click OK.

Create Signal Set		×
Signal set: HeadingAngle		
E Currently selected signals	Signal	
	sdoShipSteering/Ship Plant:1 (Ship Heading Angle)	
	OK Cancel	Help

The second requirement is that for the safety and comfort of passengers, the total acceleration should not exceed 0.25 g's at any time. The total acceleration is composed of two components, a tangential component along the direction of the ship's motion, and a normal (horizontal) component. The requirement that total acceleration not exceed 0.25 g's corresponds to requiring that in the phase plane of tangential and normal acceleration, this ship's trajectory remain within a circle of radius 0.25*9.8.

In the toolstrip, click **New** and select **Ellipse Region Constraint**. Specify the name as SafeAccel, and the semi-axis length for both signals as 0.25*9.8 = 2.45. To specify that the requirement applies to the tangential acceleration of the ship, click the **Select** button. In the Simulink model click the tangential acceleration signal, which is output from the **Kinematics** block. Select this signal in the Create Signal Set dialog, and click the arrow button to make it the designated signal, and click OK. Similarly, to specify that the requirement applies to the normal acceleration of the ship, in the Ellipse Region Constraint dialog click the other Select button, and use the Create Signal Set dialog to specify the normal acceleration signal.

Create Requir	ement		x
	Ellipse Region	Constraint	
Constrain	a pair of signals to an ellipt	ical region in signa	al phase plane.
Name: Safe	Accel		
Specify e	ellipse bound		
Signal		Semi-axis length	Center
Select	sdoShipSteering/Kinemati	2.45	0
Select	sdoShipSteering/Kinemati	2.45	0
		Angle (rad): 0	
Туре:	Signal <= bound 👻		
Create P	lot	OK Canc	el Help

Create Signal Set			X
Signal set: Sig_1			
Currently selected signals		Signal	
/Kinematics : 5(accTangential)		sdoShipSteering/Kinematics:5 (accTangential)	
	Þ		1
		OK Cancel	Help

Create Signal Set		X
Signal set: Sig_2		
Currently selected signals	Signal	
sdoShipSteering/Kinematics : 4 (accNorr	sdoShipSteering/Kinematics:4 (accNormal)	
	OK Cancel I	Help

Specify Design Variables

Specify the design variables to be tuned by the optimization in order to satisfy the requirements. In the toolstrip, click the selection box next to **Design Variables Set** and then click **New**. Select the gains of the PD controller, K and Td, and click the arrow button to designate them as design variables. Use -0.1 for all entries in the value of the K vector, and use 50 for all entries in the value of the Td vector, and click OK. If the requirements can't all be satisfied, then later the breakpoint vectors bpK and bpTd can also be tried as design variables.

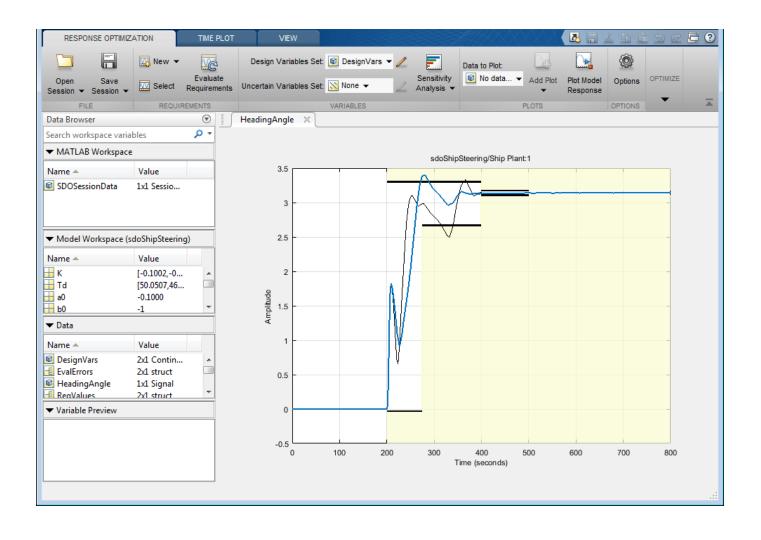
reate De	sign Variables Set									X
Create D	esign Variables set:	DesignVars								
	Variable	Value	Minimum	Maximum	Scale		Variable	Current val	Used By	
1	к	[-0.1 -0.1	[-Inf -Inf	[Inf Inf I	[0.125 0		a0	-0.1	sdoShipSteering/Ship Plant/Gain2	-
1	Td	[50 50 50	[-Inf -Inf	[Inf Inf I	[64 64 64		b0	-1	sdoShipSteering/Ship Plant/Gain	
							bpK	[9 11 13 15]	sdoShipSteering/Controller/K Gain Scheduling	Ξ
							bpTd	[9 11 13 15]	sdoShipSteering/Controller/Td Gain Scheduling	
							len	40	sdoShipSteering/Ship Plant/Gain,	
			Up	date model	variables]	mass	250000	sdoShipSteering/Force Dynamics/Gain1	Ŧ
Varia	ble Detail						Specify exp	ression indexi	ng if necessary (e.g., a(3) or s.x)	
	·								OK Cancel He	lp

Optimize Lookup Table Data

During optimization, the Simulink solver may generate a warning if the size of the time step becomes too small. Temporarily suppress this warning.

```
warnState = warning('query', 'Simulink:Engine:SolverMinStepSizeWarn');
warning('off', 'Simulink:Engine:SolverMinStepSizeWarn');
```

In the **Response Optimizer**, click **Optimize**. The ship heading angle does not meet the required step response envelope, as can be seen in the step response plot in the **Response Optimizer** app and in the Optimization Progress dialog, where the value at the last iteration is still positive, which indicates violation of the requirement. The requirement for safe acceleration is also not met, as seen in the Optimization Progress dialog, where the value at the last iteration is also positive.



	F-count	StepRespEnvelope (<=0)	SafeAccel (<=0)	
0	17	-1.1269e-04	0.612	7 🔺
1	34	0.0665	0.321	5
2	57	0.0307	0.307	9
3	103	0.0307	0.307	9
				E
				-
Optimizat The optin	ion failed to	30-Dec-2016 16:14:35 o converge, 30-Dec-2016 16:15:01 ntered 2 errors during the optimization. Deta sign Optimization workspace.	ils of the errors have been written to	^

Optimize Lookup Table Data and Breakpoints

To try to meet the design requirements, use the optimization result from above as the start point, and tune additional variables. Add breakpoints bpK and bpTd as design variables. The ship's maximum speed is 15 m/s, and during turning it may slow to 60% of the maximum speed, or 9 m/s. Set the breakpoint initial values to be equally spaced between 9 and 15 m/s. Constrain the breakpoint minimum values to 9 m/s, and constrain the breakpoint maximum values to 15 m/s.

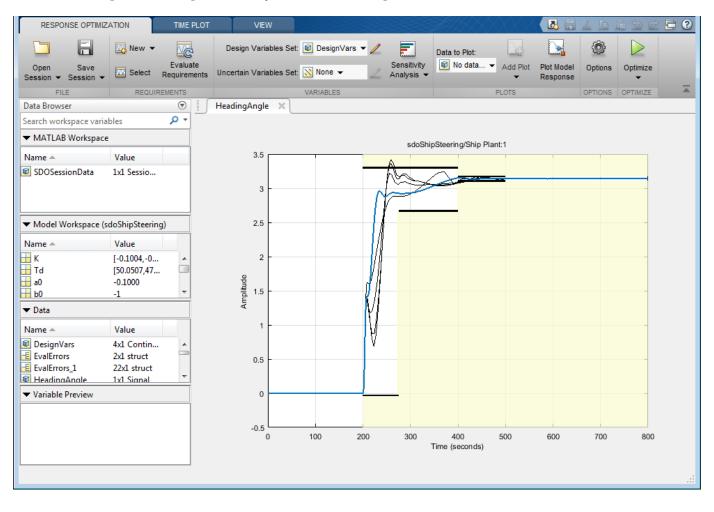
dit: Desig	nVars								
	Variable	Value	Minimum	Maximum	Scale		Variable	Current value	Used By
V	к	[-0.1002	[-Inf -Inf	[Inf Inf I	[0.125 0		a0	-0.1	sdoShipSteering/Ship Plant/Gain2
	Td	[50.0506	[-Inf -Inf	[Inf Inf I	[64 64 6		b0	-1	sdoShipSteering/Ship Plant/Gain
	bpК	[9 11 13		[15 15 15	-		len	40	sdoShipSteering/Ship Plant/Gain,
V	bpTd	[9 11 13	[9999]	[15 15 15	[16 16 1	Ē	mass	250000	sdoShipSteering/Force Dynamics/Gain1
							speed_0	15	sdoShipSteering/Force Dynamics/Fmotor,
			Upo	date model	variables]			
Varial	ble Detail					·	Specify expression	ession indexing	if necessary (e.g., a(3) or s.x)
									3

Breakpoints in the Simulink lookup table block must be strictly monotonically increasing. Add this to the design requirements.

Create Requirement	t	×
	Monotonic Varial Specify a requirement that a vector, matrix	
Name: Monotoni	c_bpK	
▼ Specify Monot	tonic Constraint	
Variable	ьрК - ▼	<1-dimensional> Show in Model
Direction		
Dimension 1:	Strictly increasing 👻	
Create Plot		OK Cancel Help
Create Requirement	+	×
	Monotonic Varial	
	Specify a requirement that a vector, matrix	
Name: Monotoni		
Variable	bpTd 👻	<1-dimensional> Show in Model
Direction		
Dimension 1:	Strictly increasing 🗸	
Create Plot		OK Cancel Help

In the **Response Optimizer**, click **Optimize**. This time the ship heading angle meets the required step response envelope, as can be seen in the step response plot in **Response Optimizer** app and in the Optimization Progress dialog, where the value at the last iteration is negative, which indicates the requirement is satisfied. The requirement for safe acceleration is also satisfied, as seen in the

Optimization Progress dialog, where the value at the last iteration is also negative. Similarly, the lookup table breakpoints satisfy the monotonic requirements.



teration	F-count	StepRespEnvelope (<=0)	SafeAccel (<=0)	Monotonic_bpK (<0)	Monotonic_bpTd (<0)	
0	29	0.0307	0.3079	-2	-2	
1	61	-0.0072	0.1432	-0.5999	-0.6000	
2	94	0.0363	-0.1189	-0.3059	-0.3060	
3	123	0.0191	-0.2200	-0.3303	-0.3086	
4	153	0.0078	-0.1399	-0.3873	-0.3295	
5	183	-0.0046	-0.0978	-0.4292	-0.3482	Ξ
						Ŧ
Optimizat The optim	ion converg	02-Jan-2017 17:44:05 ged, 02-Jan-2017 17:44 ntered 22 errors during Design Optimization wo	the optimization. De	tails of the errors have	e been written to	•

In this example, the ship plant varied with the ship speed, so the controller gains also needed to vary. Gain scheduling was implemented using lookup tables. By tuning the gains and breakpoint values in the controller, the ship was able to follow the reference heading angle, while also constraining total acceleration to ensure a safe and comfortable ride for passengers.

Related Examples

To learn how to optimize the lookup tables in the gain scheduled controller using the sdo.optimize command, see "Design Optimization Using Lookup Table Requirements for Gain Scheduling (Code)" on page 6-44.

```
% Close the model and restore state of warnings.
bdclose('sdoShipSteering')
warning(warnState); % restore state of warning
```

See Also

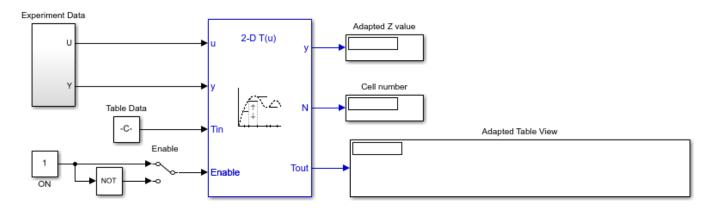
Related Examples

- "Design Optimization Using Lookup Table Requirements for Gain Scheduling (Code)" on page 6-44
- "Estimate Lookup Table Values from Data" on page 6-17

- "Specify Time-Domain Design Requirements in the App" on page 3-16
- "Specify Variable Requirements in the App" on page 3-31

2-D Adaptive Lookup Table Generation

This example shows how to create a 2-D lookup table from experimental data.



Copyright 2014 The MathWorks, Inc.

Click here to open the Adaptive Lookup Tables library.

See Also

Adaptive Lookup Table (2D Stair-Fit)

Related Examples

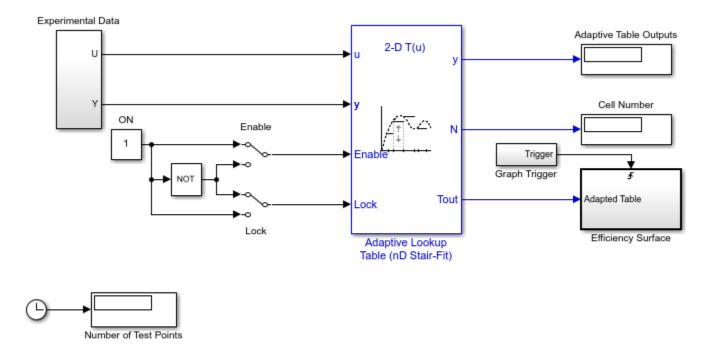
• "Model Engine Using n-D Adaptive Lookup Table" on page 6-33

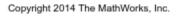
More About

• "What are Adaptive Lookup Tables?" on page 6-2

Engine Volumetric Efficiency Surface Matching

This example shows how to generate a lookup table to approximate an engine volumetric efficiency surface using experimental data.





Click here to open the Adaptive Lookup Tables library.

See Also

Adaptive Lookup Table (2D Stair-Fit)

Related Examples

• "Model Engine Using n-D Adaptive Lookup Table" on page 6-33

More About

• "What are Adaptive Lookup Tables?" on page 6-2