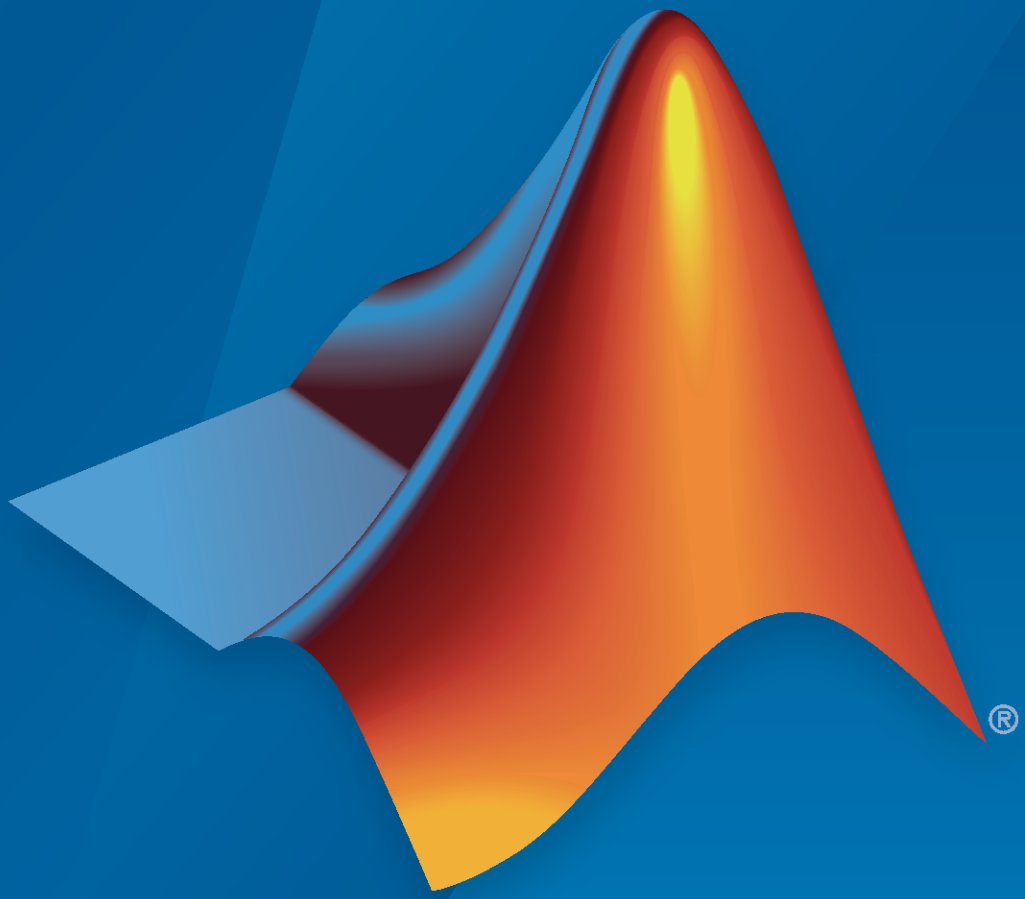
Simulink® Control Design™ Release Notes



MATLAB® & SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

Simulink® Control Design™ Release Notes

Contents

R2023a

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2019a

R2018b

R2018a

R2017b

R2017a

R2016b

R2016a

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010b

R2010a

R2009b

R2009a

R2008b

R2008a

R2007b

R2007a

R2006b

R2006a

R14SP3

R14SP2

R2023a

Version: 7.0

New Features

Bug Fixes

Frequency Response Estimator Block: Estimate frequency responses using PRBS input signals

The Frequency Response Estimator block now supports estimation using pseudorandom binary sequence (PRBS) input signals. A PRBS signal is a periodic, deterministic signal with white-noise-like properties that shifts between two values. PRBS signals reduce total estimation time compared to using sinestream or superposition input signals, while producing comparable estimation results. PRBS signals are useful for estimating frequency responses for communications and power electronics applications. For more information, see “PRBS Input Signals”.

For more information, see Frequency Response Estimator. For an example, see “Online Estimation of Frequency Responses of a Nonlinear Plant”.

Passivity Enforcement Block: Modify control actions to satisfy passivity constraints and action bounds

You can use the new Passivity Enforcement block to compute control actions that are closest to specified control actions while satisfying action bounds and passivity constraints. This block requires Optimization Toolbox™ software.

The block uses a quadratic programming (QP) solver to find the control action u that minimizes the function $|u - u_0|^2$ in real time. Here, u_0 is the unmodified control action from the controller.

The solver applies the following constraints to the optimization problem.

$$\rho y_p{}^T y_p + y_p{}^T f_p + y_p{}^T g_p u \le 0$$

$$u_{\min} \le u \le u_{\max}$$

Here:

- ρ is the passivity index.
- y_p is the passivity output function, defined as $y_p = h_p(x)$.
- f_p and g_p are the functions defined by the passivity input function $u_p = f_p(x) + g_p(x)u$.
- $u_{\min}$ is a lower bound for the control action.
- $u_{\max}$ is an upper bound for the control action.

For more information, see “Passivity Enforcement for Control Design”.

Model Reference Adaptive Control: Model disturbances and uncertainty using neural network with single hidden layer

While tracking a reference model using the Model Reference Adaptive Control block, you can now model disturbances and uncertainty in your system using a single hidden layer (SHL) neural network. An SHL neural network is a universal function approximator, which can be useful for modeling unstructured uncertainty in your system.

To learn the disturbance model, the MRAC controller adapts the weights of the hidden layer and output layer of the neural network. You can configure the SHL neural network model by specifying the number of neurons in the hidden layer and the learning rate.

R2022b

Version: 6.2

New Features

Bug Fixes

Active Disturbance Rejection Control: Design disturbance rejection control for plants with unknown dynamics and internal and external disturbances

Active disturbance rejection control (ADRC) is a model-free control algorithm that is useful for designing controllers for plants with unknown dynamics and internal and external disturbances. You can implement ADRC using the Active Disturbance Rejection Control block.

For more information on ADRC, see Active Disturbance Rejection Control.

For examples that show how to use the block, see:

- Design Active Disturbance Rejection Control for Water-Tank System
- Design Active Disturbance Rejection Control for Boost Converter
- Design Active Disturbance Rejection Control for BLDC Speed Control Using PWM

New Example: Cascade PID control tuning for buck converter

The new example Multiloop Control Design for Buck Converter shows how to tune inner-loop current control and outer-loop voltage control at the same time using `systune`.

R2022a

Version: 6.1

New Features

Bug Fixes

Indirect Model Reference Adaptive Control: Track reference plant model by estimating parameters of uncertain controlled system

The Model Reference Adaptive Control block now supports indirect model reference adaptive control (MRAC). Previously, the block supported only direct MRAC.

Direct MRAC computes control actions to make an uncertain controlled system track the behavior of a given reference plant model. However, indirect MRAC estimates the parameters of the uncertain system and derives the control signal based on the estimated parameters.

For more information, see Model Reference Adaptive Control.

Barrier Certificate Enforcement Block: Modify control actions to satisfy barrier certificate constraints and action bounds

You can use the new Barrier Certificate Enforcement block to compute control actions that are closest to specified control actions while satisfying action bounds and barrier certificate constraints. This block requires Optimization Toolbox software.

A barrier certificate defines a safety set $h(x) \geq 0$, which is a function of the system state x. When the initial system states start within the safety set, the Barrier Certificate Enforcement block adjusts the control actions to keep the states within the safety set. To do so, the block uses a quadratic programming (QP) solver to find the control action u that minimizes the function $|u - u_0|^2$ in real time. Here, u_0 is the unmodified control action from the controller.

The solver applies the following constraints to the optimization problem.

$$q_x f_x + q_x g_x u + \gamma h_x^{\beta} \geq 0$$

$$u_{\min} \leq u \leq u_{\max}$$

Here:

- f_x and g_x are coefficients of the plant dynamics equation $\dot{x} = f_x + g_x u$.
- h_x is the value of the barrier certificate.
- q_x is the partial derivative of h_x with respect to x.
- γ and β are parameters for controlling the effect of the barrier certificate.
- $u_{\min}$ is a lower bound for the control action.
- $u_{\max}$ is an upper bound for the control action.

For more information, see Barrier Certificate Enforcement for Control Design.

New Examples: Power electronics control design

This release includes the following new reference examples.

- Frequency Response Estimation to Measure Input Admittance and Output Impedance of Boost Converter — Identify frequency-response data models to measure input admittance and output impedance of a boost converter modeled using Simscape™ Electrical™ components. This example also compares the estimated frequency response models with the analytical transfer functions created using component parameters.

- Tune Gain-Scheduled Controller for PMSM Model Using Closed-Loop PID Autotuner Block — Automatically tune gain-scheduled PID controller for a PMSM model in just one simulation using the Closed-Loop PID Autotuner block.
- Islanded Operation of Remote Microgrid Using Droop Controllers — Implement droop control for islanded operation of a remote microgrid.

R2021b

Model Reference Adaptive Control: Track reference model output by adapting controller parameters in response to model uncertainty and external disturbances

Model reference adaptive control (MRAC) is a model-based, real-time adaptive control algorithm that computes control actions to make an uncertain controlled system track the behavior of a given reference plant model. Using Simulink Control Design software, you can implement MRAC using the Model Reference Adaptive Control block.

For more information on model reference adaptive control, see Model Reference Adaptive Control.

Discrete Extremum Seeking Control: Design hardware-deployable extremum seeking controller with a specified sample time

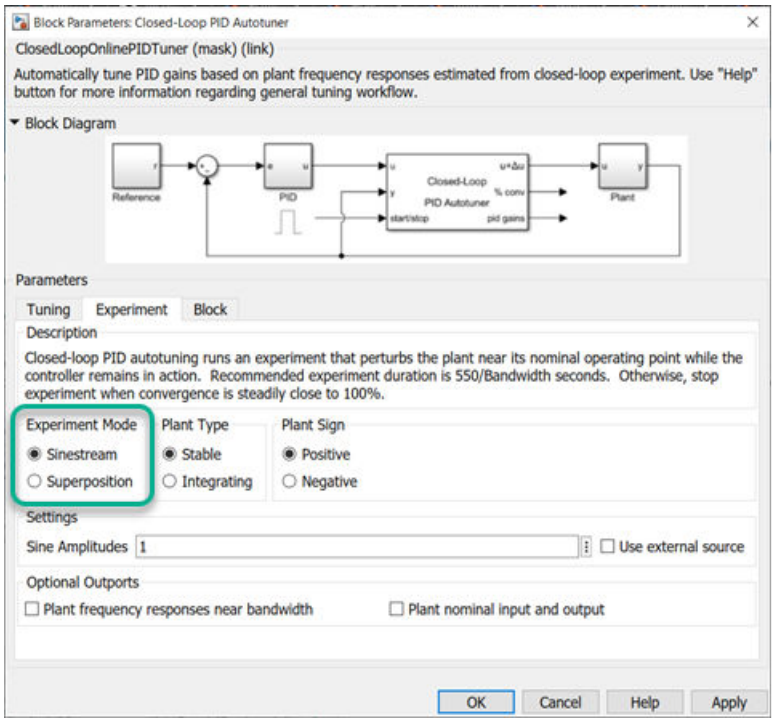
Using the Extremum Seeking Control block, you can now design an extremum seeking controller with discrete-time high-pass and low-pass filters and a discrete-time integrator. You can then generate code for this controller and deploy it to a hardware target.

For more information on extremum seeking control, see Extremum Seeking Control.

Closed-Loop PID Autotuner Block: Reduce execution time on hardware by performing frequency response estimation experiment using sinestream signals

The Closed-Loop PID Autotuner block tunes PID controllers based on estimated plant frequency responses. You can now estimate frequency responses of the plant using sinestream input signals. A sinestream signal consists of a series of sinusoidal perturbations applied one after another. When you perform a frequency response estimation experiment using sinestream input signals, the block applies perturbation at each frequency separately. For more information, see Sinestream Input Signals.

Sinestream signals reduce the execution time compared to superposition input signals, but also take longer to estimate the frequency response. Frequency response estimation using sinestream signals is useful when you have limited processing power and you want to reduce the execution time. To do so, on the **Experiment** tab of block parameters, set **Experiment Mode** to **Sinestream**.



For more information about deploying the Closed-Loop PID Autotuner block, see PID Autotuning in Real Time.

Model Linearizer: Linearize Simulink model to sparse state-space model

Using **Model Linearizer**, you can now obtain a sparse linear model from a Simulink model that contains a Sparse Second Order or Descriptor State-Space block.

For more information, see Linearize Sparse Models.

Operating Points: Obtain state, input, and output indices from operating point search reports

When defining a mapping function for customized trimming of Simulink models, you can now obtain operating point state, input, and output indices from operating point search reports. For more information, see `getStateIndex`, `getInputIndex`, and `getOutputIndex`.

New Examples: Tune controllers for automotive, UAV, and gain-scheduling applications

This release includes the following new reference examples.

- Design Controller for Vehicle Platooning — Tune a PID controller for spacing control in trailing vehicles in a platoon.
- UAV Inflight Failure Recovery — Tune PID controllers for a UAV quadcopter under nominal flight conditions and a fault condition where one of the rotor fails.

- PID Autotuning for UAV Quadcopter — Automatically tune attitude and position PID controllers for a UAV quadcopter using the Closed-Loop PID Autotuner block.
- Tune Gain-Scheduled Controller Using Closed-Loop PID Autotuner Block — Automatically tune gain-scheduled PID controller for a water-tank model using the Closed-Loop PID Autotuner block.

Functionality being removed or changed

PortWidth property of operating point inputs and outputs will be removed
Still runs

The input and output `PortWidth` properties of operating points and operating point specifications will be removed in a future release. Use the new `Nu` and `Ny` properties instead.

To update your code, change instances of `PortWidth` to either `Nu` or `Ny` as shown in the following table.

Not Recommended	Recommended
`op = operpoint('scdplane');` `numIn = op.Inputs(1).PortWidth;`	`op = operpoint('scdplane');` `numIn = op.Inputs(1).Nu;`

`op = operspec('scdplane');`
`numOut = op.Outputs(1).PortWidth;`
`numIn = op.Inputs(1).PortWidth;`
	`op = operspec('scdplane');`
	`numOut = op.Outputs(1).Ny;`
	`numIn = op.Inputs(1).Nu;`

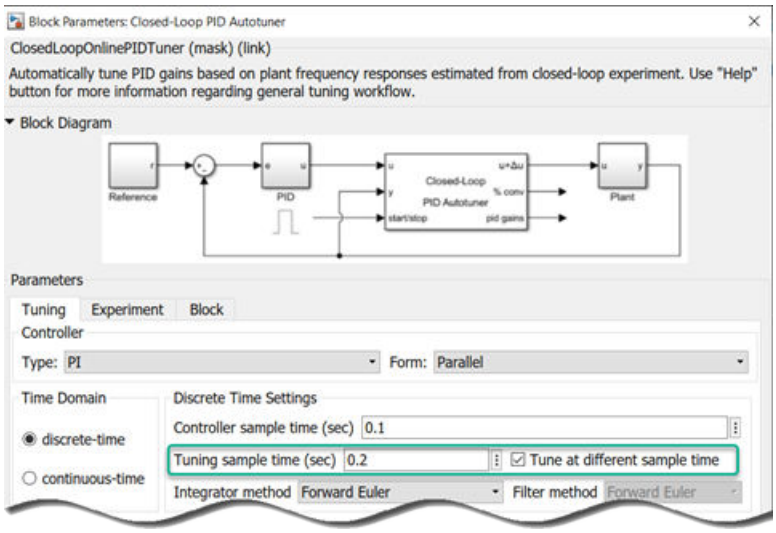
`[op,report] = findop('scdplane',10);`
`numOut = op.Outputs(1).PortWidth;`
`numIn = report.Inputs(1).PortWidth;`
	`[op,report] = findop('scdplane',10);`
	`numOut = op.Outputs(1).Ny;`
	`numIn = report.Inputs(1).Nu;`

R2021a

Closed-Loop PID Autotuner Block: Reduce target hardware throughput requirements by running autotuning at a slower sample rate than the PID controller

The Closed-Loop PID Autotuner block now lets you specify a different sample time for PID gain tuning. Previously, the block inherited the tuning sample time from the sample time specified in the **Controller sample time** parameter.

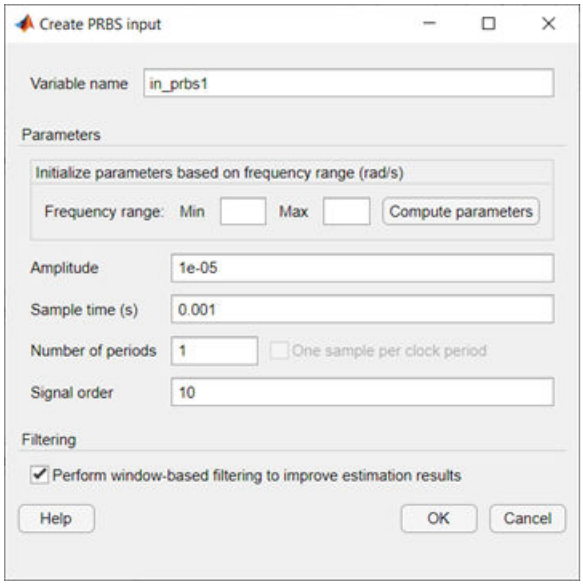
When you have a PID controller with a fast sample time and you run the tuning process at the same rate, some hardware might not complete PID gain calculation in a single time step. Therefore, when you have limited processing power and you want to tune controllers with fast sample times, enable the **Tune at different sample time** parameter and specify a sample time for tuning in the **Tuning sample time** parameter.



For more information about deploying the Closed-Loop PID Autotuner block, see PID Autotuning in Real Time.

PRBS Input Signals: Automatically determine signal parameters and perform filtering to improve quality of frequency estimation results

When creating PRBS input signals for estimating frequency responses in **Model Linearizer**, you can now automatically determine the signal parameters **Number of periods** and **Signal order** based on a frequency range. Automatic parameter determination helps create an input signal that leads to accurate frequency response over a specified frequency range. For more information, see PRBS Input Signals.



Additionally, you can now:

- Use the **One sample per clock period** parameter to specify whether the signal remains constant for one sample per clock period or multiple samples per clock period. Use this parameter if you have **Number of periods** > 1. By default, this option is enabled and the generated signal is constant over one sample. When you disable this option, the generated signal is constant for the specified number of samples.

  At the command line, set the `OneSamplePerClockPeriod` property of the `frest.PRBS` object to `'on'` or `'off'` to generate signals that remain constant over one sample or multiple samples, respectively.
- Use the **Perform window-based filtering to improve estimation results** parameter to apply filtering, which produces a smoother frequency response estimation result.

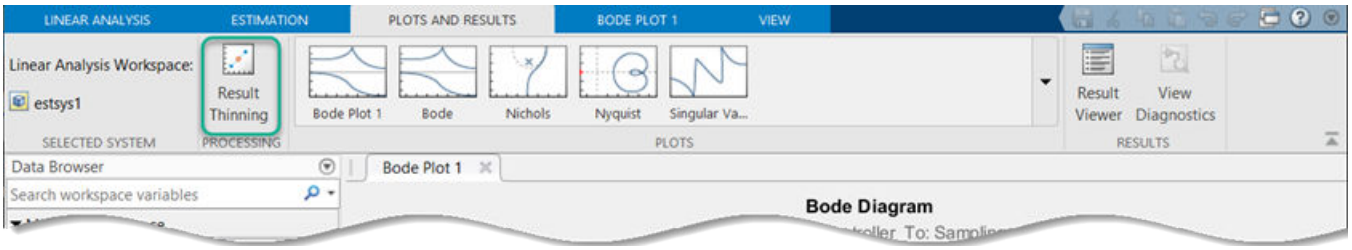
  At the command line, set the `UseWindow` property of the `frest.PRBS` object to `'on'` or `'off'` to enable or disable filtering, respectively.

Compatibility Considerations

If you import a session that was saved in a release from before R2021a, the frequency response estimation results might differ.

Frequency Response Estimation Result Thinning: Extract interpolated results of a frequency response estimation model across a range of frequencies

When you have an estimated frequency response result with a large number of frequency points, you can use **Result Thinning** in **Model Linearizer** to extract interpolated frequency response data from an estimated frequency response model across a specified frequency range and number of frequency points. For more information, see Result Thinning



Constraint Enforcement Block: Modify control actions to satisfy constraints and action bounds

You can use the new Constraint Enforcement block to compute control actions that are closest to specified control actions while satisfying action bounds and other constraints. The block uses a quadratic programming (QP) solver to find bounded control actions that satisfy additional constraints of the following form.

$$f_x + g_x u \le c$$

Here, f_x and g_x are constraint coefficients that you provide to the block at run time, c is the constraint bound, and u is the control action.

The Constraint Enforcement block requires Optimization Toolbox software.

For more information, see Constraint Enforcement for Control Design.

Extremum Seeking Control Block: Automatically adapt control system parameters to maximize an objective function using model-free real-time optimization

You can use the new Extremum Seeking Control block to tune controller parameters to maximize an objective function. Extremum seeking controllers are model-free adaptive controllers that are useful for adapting to unknown system dynamics and unknown mappings from control parameters to an objective function.

The block searches for optimal control parameters by modulating (perturbing) the parameters with sinusoidal signals and demodulating the resulting perturbed objective function.

For more information, see Extremum Seeking Control.

New Examples: Frequency response estimation with PRBS input signals

This release includes the following new reference examples.

- Frequency Response Estimation for Permanent Magnet Synchronous Motor Model — Identify a frequency-response data model of a field-oriented controlled permanent magnet synchronous motor. This example requires Motor Control Blockset™ software.
- Frequency Response Estimation in Model Linearizer Using Pseudorandom Binary Sequence — Identify a frequency-response data model of a boost converter modeled using Simscape Electrical

components. This example also shows how to apply result thinning to the frequency response estimation data.

- Frequency Response Estimation to Measure Input Admittance of Boost Converter — Identify a frequency-response data model to measure input admittance (1/impedance) of a boost converter modeled using Simscape Electrical components. This example also compares the estimated frequency response model with the analytical transfer function created using component parameters.

R2020b

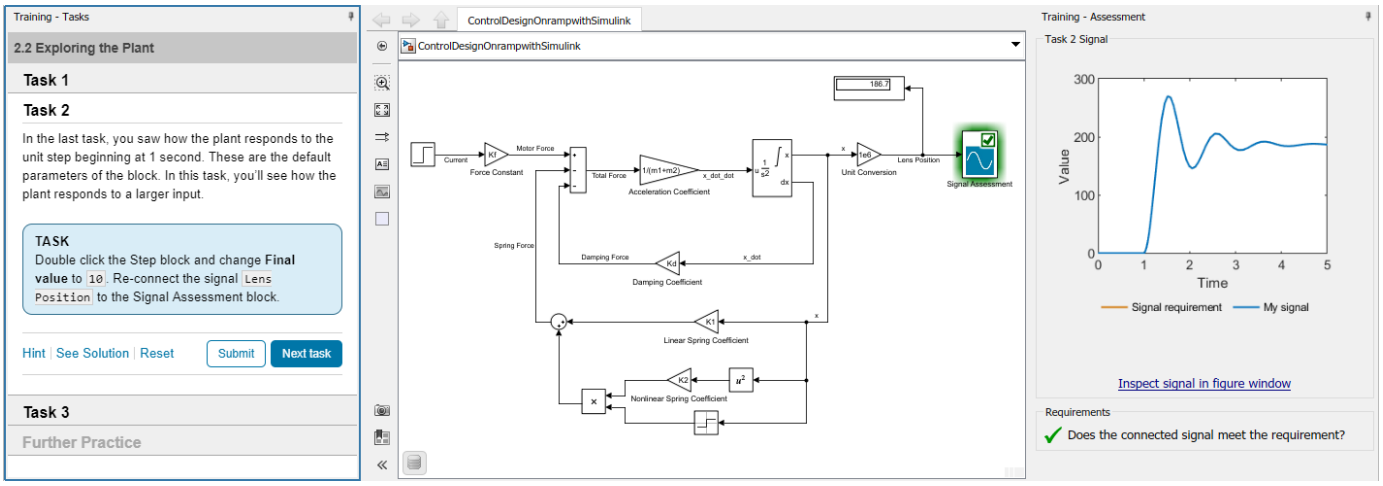
Version: 5.6

New Features

Bug Fixes

Control Design Onramp with Simulink: Self-paced, interactive tutorial for getting started with control design

To help you get started quickly with control design, Control Design Onramp with Simulink provides a self-paced, interactive tutorial. To teach concepts incrementally, it uses hands-on exercises. You receive automated assessments and feedback after submitting tasks.



Your progress is saved if you exit the training, so you can complete the training in multiple sessions.

To open Control Design Onramp with Simulink, use one of these options:

- On the Simulink Start Page, click **Control Design Onramp with Simulink**.
- In the MATLAB® Command Window, enter `learning.simulink.launchOnramp('controls')`.

Control Design Onramp with Simulink:

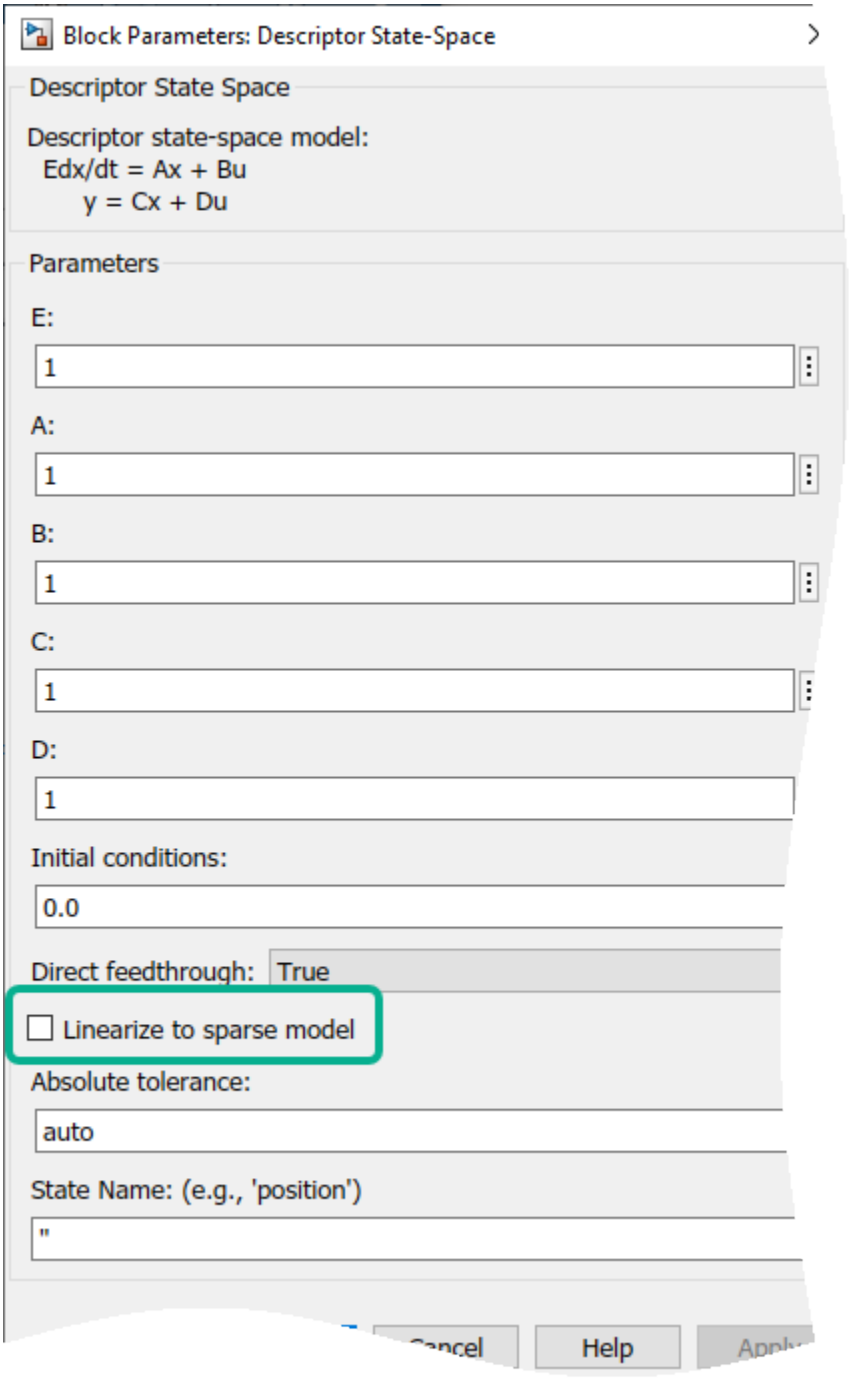
- Introduces control workflows in Simulink.
- Teaches and reinforces classical control theories using Simulink Control Design and Control System Toolbox™.
- Goes through basic control design with easy-to-follow examples.
- Walks you through linearizing a control system plant with **Model Linearizer** and tuning a PID controller with **PID Tuner**.
- Helps you practice what you learn with a walking robot project.

linearize and slLinearizer: Linearize a Simulink model to a sparse state-space model

You can now linearize and obtain a sparse model from a Simulink model when a Sparse Second Order or Descriptor State-Space block is present.

You obtain a:

- `mechss` model when you use a Sparse Second Order in your Simulink model.
- `sparss` model when you use a Descriptor State-Space block and check the **Linearize to sparse model** checkbox.



For more information, see Sparse Model Basics (Control System Toolbox). For an example, see Linearize Simulink Model to a Sparse Second-Order Model Object (Control System Toolbox).

R2020a

Version: 5.5

New Features

Bug Fixes

Linearization App: Linear Analysis Tool is now Model Linearizer

The **Linear Analysis Tool** is now the **Model Linearizer** app. The features of the app remain the same, only the name has changed. For more information on using the app, see Model Linearizer.

PRBS Input Signals: Estimate frequency response using pseudorandom binary sequences as input signals

You can now estimate the frequency response of your Simulink model using pseudorandom binary sequence (PRBS) input signals. A PRBS signal is a periodic, deterministic signal with white-noise-like properties that shifts between two values.

PRBS signals reduce total estimation time compared to using sinestream input signals, while producing comparable estimation results. PRBS signals are useful for estimating frequency responses for communications and power electronics applications. For more information, see PRBS Input Signals.

You can use PRBS input signals when estimating frequency responses at the command line and using the **Model Linearizer** app.

At the command line, create a PRBS input signal using an `frest.PRBS` object.

For an example that demonstrates frequency response estimation using various input signals, see Frequency Response Estimation Using Simulation-Based Techniques. For an example that estimates the frequency response of a power electronics system using a PRBS input signal, see Frequency Response Estimation for Power Electronics Model Using Pseudorandom Binary Signal.

Linearization Using Numerical Perturbation: Linearize blocks with single-precision input signals or states and no predefined analytic Jacobians

Simulink blocks that have single-precision input signals or states and do not have predefined analytic Jacobians no longer linearize to zero. Instead, Simulink Control Design software now linearizes such blocks using numerical perturbation. For information on changing the perturbation level for single-precision input signals and states, see Change Perturbation Level of Blocks Perturbed During Linearization.

R2019b

Version: 5.4

New Features

Bug Fixes

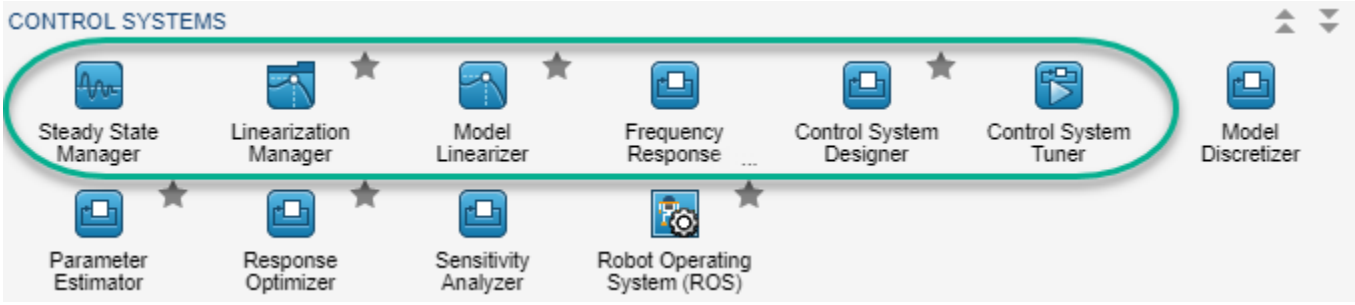
Simulink Toolstrip: Open Simulink Control Design apps and manage linearization

In R2019b, the Simulink Toolstrip replaces menus and toolbars in the Simulink Desktop. For more information, see Simulink Toolstrip: Access and discover Simulink capabilities when you need them.

The following Simulink Control Design tasks have changed in R2019b.

- Opening applications
- Managing analysis points
- Linearizing blocks

You now open Simulink Control Design applications on the **Apps** tab, in the gallery, under **Control Systems**.



For each of the following apps, the gallery entry matches the app name:

- **Steady State Manager**
- **Control System Designer**
- **Control System Tuner**

To open the **Linear Analysis Tool** with the **Linear Analysis** tab open, click **Model Linearizer**. To open the **Linear Analysis Tool** with the **Estimation** tab open, click **Frequency Response Estimator**.

To manage analysis points and block linearizations, you now use the **Linearization** tab. To open this tab, on the **Apps** tab, click **Linearization Manager**.

Then, in your model, select the signal or block you want to manage, and on the **Linearization** tab, perform the corresponding task.

Task	In the Model	On the Linearization Tab
Specify analysis points on signals	Select signal	In the gallery, select or clear the type of analysis point
Specify analysis points on bus elements	Select bus signal	Click **Select Bus Element**
Linearize block	Select block	Click **Linearize Block**
Specify block linearization	Select block	Click **Specify Block Linearization**

New Example: Design field-oriented controller using Closed-Loop PID Autotuner block

The new example Tune Field-Oriented Controllers for a PMSM Using Closed-Loop PID Autotuner Block shows how to tune a field-oriented controller for a PMSM-based electrical-traction drive using the Closed-Loop PID Autotuner block.

R2019a

Version: 5.3

New Features

Bug Fixes

Frequency Response Estimator Block: Estimate frequency response in simulation or real-time environment

Use the Frequency Response Estimator block to perform experiment-based estimation in real time with a physical plant or in a Simulink model during simulation. The new block is in the **Simulink Control Design** block library.

To obtain an estimated frequency response, the block injects test sinusoidal signals into the plant at the nominal operating point and collects input-output data. The block then computes the estimated frequency response. You can specify the frequencies at which to measure the response. The Frequency Response Estimator block can perform estimation in either an open-loop or closed-loop configuration. You trigger the estimation process via a start/stop signal, so you can perform estimation at any time during simulation or real-time plant operation.

You can generate code and deploy the Frequency Response Estimator block on hardware to perform the estimation in real time. (Deployment requires a code-generation product such as Simulink Coder™.)

For more information, see Online Frequency Response Estimation Basics.

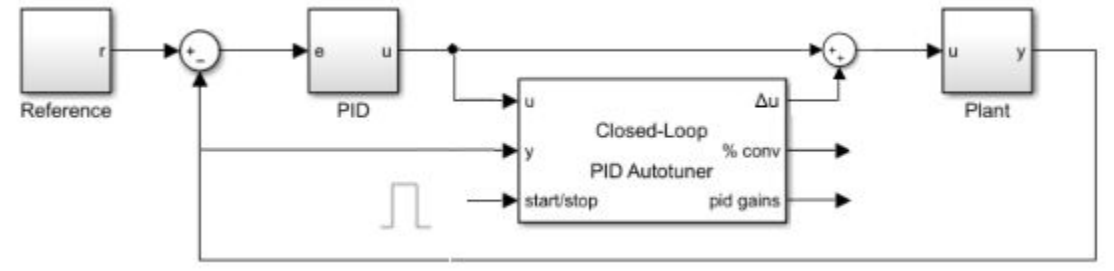
fastRestartForLinearAnalysis Command: Minimize number of model compilations for linearization and trimming workflows

Use the new `fastRestartForLinearAnalysis` command to prevent multiple model compilations when you use compiling functions like `linearize` or `findop`, as part of your linear analysis workflow. `fastRestartForLinearAnalysis` also supports single model compilation workflows that use an `slLinearizer` or `slTuner` object.

For more information, see `fastRestartForLinearAnalysis` and Improve Linear Analysis Performance.

Closed-Loop PID Autotuner Block: Inject perturbation signal without inserting block into loop

By default, Closed-Loop PID Autotuner takes the control action as its input, and provides the sum of the control action and the experiment signal to the plant input. This configuration requires you to insert the block between the controller and the plant in your model. You can now configure the block to provide the perturbation signal only, which lets you use the block with your control loop intact. Inject the perturbation signal into the plant input as a load disturbance using a Sum block, as shown in the following diagram.



To use the new configuration, in the block parameters, in the **Block** tab, under **Output Signal Configuration**, select **perturbation only**.

For more information about PID autotuning, see Closed-Loop PID Autotuner.

PID Tuning Example: Design PID Controllers for Power Factor Corrector

A new example, Design PID Controllers for a Power Factor Corrector using Frequency Response Estimation Method, shows how to tune controllers for a Power Factor Corrector model that does not linearize. The example uses the **Linear Analysis Tool** to obtain an estimated frequency response model of the plant. It uses the **PID Tuner** to tune PI controllers using the estimated response.

R2018b

Version: 5.2

New Features

Bug Fixes

Steady State Manager: Interactively specify and compute model operating points, and validate trimming results

You can now interactively compute steady-state operating points for Simulink models using the new **Steady State Manager**.

- Interactively obtain operating points from state, input, and output specifications
- Validate operating points against specifications
- Interactively obtain operating points from simulation snapshots
- Generate MATLAB code for computing operating points

If you are computing operating points for linearization applications, you can still trim your model using the **Linear Analysis Tool**.

For more information, see **Steady State Manager** and Compute Operating Points from Specifications Using Steady State Manager.

Linear Analysis Tool: Trim Simulink models using custom constraints and objective functions

You can now specify custom constraints and objective functions when trimming Simulink models using the **Linear Analysis Tool**. The software applies these custom constraints and cost functions in addition to the standard input, output, and state specifications. For more information on trimming models using custom constraints and objective functions, see Compute Operating Points Using Custom Constraints and Objective Functions.

Motor Control Examples: Tune gains of field-oriented controllers

Three new examples show how to tune PI controllers for field-oriented control using Simulink Control Design software. The first two examples design PI controllers for field-oriented control using the Closed-Loop PID Autotuner block. These examples sequentially tune the gains of multiple controllers in cascaded feedback loops.

- Tune Field-Oriented Controllers Using Closed-Loop PID Autotuner Block
- Tune Field-Oriented Controllers for an Asynchronous Machine Using Closed-Loop PID Autotuner Block

A third example shows how to programmatically tune field-oriented controllers for a permanent magnet synchronous machine using the `systune` function. This example simultaneously tunes the gains of multiple controllers in cascaded feedback loops.

- Tune Field-Oriented Controllers Using SYSTUNE

R2018a

Closed-Loop PID Autotuning: Deploy algorithm that performs PID autotuning without opening the feedback loop

Use the new Closed-Loop PID Autotuner block to tune a PID controller in real time against a physical plant in a closed-loop configuration. Unlike the Open-Loop PID Autotuner block introduced in R2017b (formerly called Online PID Tuner), the Closed-Loop PID Autotuner performs an estimation experiment without opening the feedback loop.

When you use the Closed-Loop PID Autotuner block, the existing controller remains in the loop to:

- Reject unknown plant disturbances to maintain safe operation of the plant during the estimation experiment.
- Reduce the risk that the perturbations used for the experiment drive the plant away from the desired operating point.

To achieve model-free tuning, the Closed-Loop PID Autotuner block:

1. Injects a test signal into the plant at the nominal operating point to collect plant input-output data and estimate frequency response in real time.
2. Tunes PID controller parameters based on the estimated plant frequency response.
3. Updates a PID Controller block or a custom PID controller with the tuned parameters, allowing you to validate closed-loop performance in real time.

You can generate code with the block and deploy it on hardware, letting you tune with or without Simulink in the loop. (Doing so requires a code-generation product such as Simulink Coder.) The Closed-Loop PID Autotuner block works with any asymptotically stable or integrating SISO plant, with or without time delay, and with or without direct feedthrough. It can tune any type of PID controller, provided that the closed-loop system is already stable with the existing controller. The block lets you trigger the tuning process via a start/stop signal, so you can tune and retune your controller at any time. The new block is in the **Simulink Control Design** block library.

For an example that uses the block, see Tune PID Controller in Real Time Using Closed-Loop PID Autotuner Block. For more information about closed-loop PID autotuning, see:

- Closed-Loop PID Autotuner block reference page
- When to Use PID Autotuning

Open-Loop PID Autotuning: Online PID Tuner block name change

The Online PID Tuner block name has changed to Open-Loop PID Autotuner. Models containing this block that you created with R2017b still work.

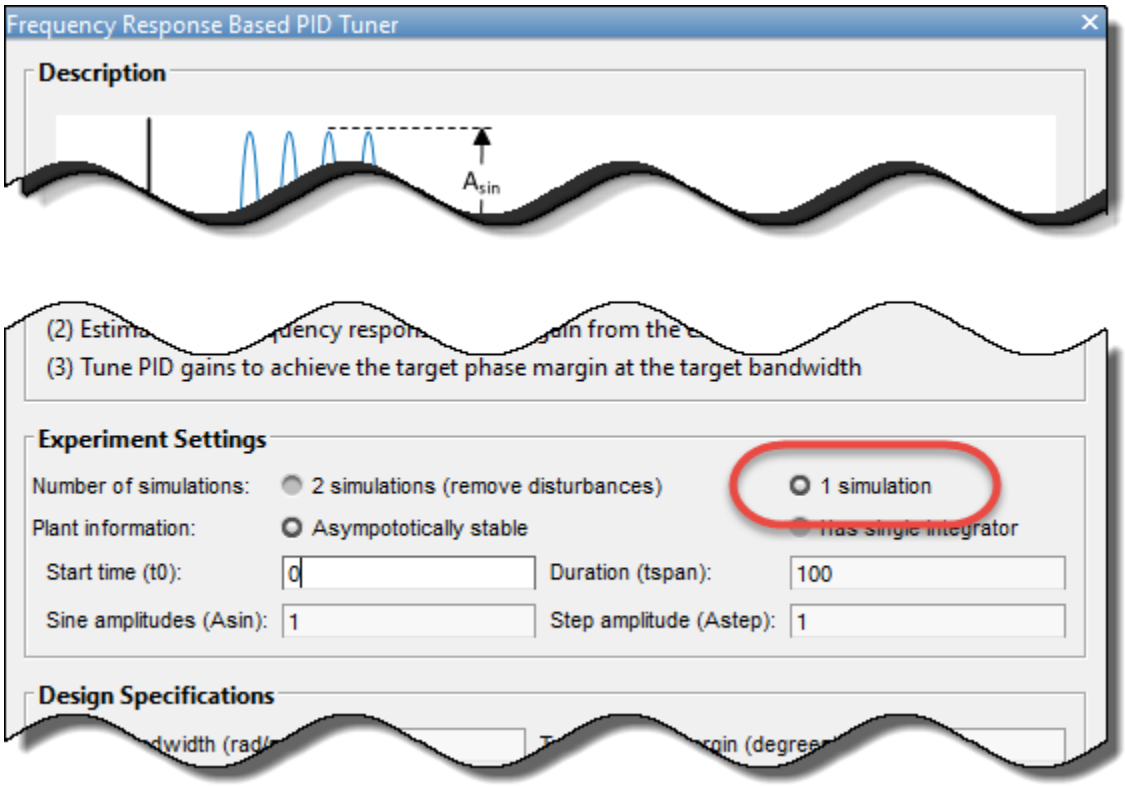
Frequency Response Based PID Tuner: Use single simulation to estimate plant frequency response for tuning PID gains

The Frequency Response Based PID Tuner introduced in R2017b can now tune using a single simulation. This option halves the overall PID tuning time, which can be significant when simulating your model takes a long time.

By default, the Frequency Response Based PID Tuner performs two simulations: one for a baseline plant response, and a second with perturbation signals applied at the controller output. The tuner

then uses the difference between the two simulations to estimate plant response, removing the effect of any disturbances in the model. The new option lets you skip the baseline simulation, and estimate plant frequency response using only the perturbed simulation. This option is useful when there are no disturbances in your model that have a large enough effect on the plant response to interfere with the estimation experiment.

In such cases, you can perform PID tuning faster by simulating the plant only once. To use this option, in the Frequency Response Based PID Tuner, in the **Experiment settings** section, select **1 simulation**.



The Frequency Response Based PID Tuner is particularly useful when PID Tuner cannot linearize the plant at the operating point you want to use for tuning. For more information, see Frequency-Response Based Tuning.

Two-Way Numerical Perturbation: Linearize Simulink models using full-model central-difference numerical perturbation

When linearizing a Simulink model using full-model numerical perturbation, you can now use two-way central-difference perturbation. This linearization method perturbs the root-level inputs and states of your model in both positive and negative directions. Previously, you could use only single-direction forward difference perturbation, which perturbs the inputs and states in a positive direction only.

Central difference numerical perturbation is typically more accurate than forward-difference perturbation. However, this algorithm can be slower due to additional model evaluations.

To use central-difference perturbation:

- At the command line, create a `linearizeOptions` option set, and specify the `LinearizationAlgorithm` option as `numericalpert2`.

  ```
  options = linearizeOptions('LinearizationAlgorithm','numericalpert2');
  ```

- In the Linear Analysis Tool, on the **Linear Analysis** tab, click **More Options**. Then, in the Options for exact linearization dialog box, in the **Choose algorithm** drop-down list, select `Numerical perturbation (central difference)`.

Linearization Advisor: Simscape states and inputs combined into single block diagnostic per Solver Configuration block

When troubleshooting linearization issues using the Linearization Advisor, diagnostic information for Simscape states and inputs is now combined into a single block diagnostic object. Previously, states and inputs were returned in separate diagnostic objects.

For more information on troubleshooting Simscape network linearization, see Linearize Simscape Networks.

Compatibility Considerations

To view diagnostic information for Simscape networks at the command line, you first query the Linearization Advisor object, `advisor`, for blocks of type `'simscape'`.

```
qSS = linqueryIsBlockType('simscape');
advSS = find(advisor,qSS);
```

To view state or input information in the previous release, you searched the block paths of the diagnostics in `advSS` for the text `EVAL_KEY/STATE` or `EVAL_KEY/INPUT`, respectively. For example, to find diagnostics with state information, you used:

```
paths = getBlockPaths(advisor);
index = contains(paths,'EVAL_KEY/STATE');
diag = getBlockInfo(advSS,index);
```

You then viewed the state or input information in the associated operating point object.

```
diag(i).OperatingPoint
```

Now, you access both the state and input information directly from the operating point of the Simscape block diagnostic object without searching the block paths.

```
advSS.BlockDiagnostics(i).OperatingPoint
```

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Separate `EVAL_KEY/STATE` and `EVAL_KEY/INPUT` diagnostics in Linearization Advisor for Simscape state and input information, respectively.	Not applicable	Single combined diagnostic for a Simscape network that contains both state and input information.	For more information, see “Linearization Advisor: Simscape states and inputs combined into single block diagnostic per Solver Configuration block” on page 11-4.

R2017b

PID Autotuning: Deploy PID autotuning algorithm to embedded software

Use the new Online PID Tuner block to tune a PID controller in real time against a physical plant or a Simulink model. This block lets you tune a PID controller to achieve a specified bandwidth and phase margin without a parametric plant model or an initial controller design.

To achieve model-free tuning, the Online PID Tuner block:

1. Injects a test signal into the plant at the nominal operating point to collect plant input-output data and estimate frequency response in real time. The test signal is a combination of sine and step perturbation signals added on top of the nominal plant input measured when the experiment starts. If the plant is part of a feedback loop, the block opens the loop during the experiment.
2. At the end of the experiment, tunes PID controller parameters based on estimated plant frequency responses near the open-loop bandwidth.
3. Updates a PID Controller block or a custom PID controller with the tuned parameters, allowing you to validate closed-loop performance in real time.

You can generate code and deploy it on hardware, letting you tune with or without Simulink in the loop. (Doing so requires a code-generation product such as Simulink Coder.)

Real-time scenarios for tuning include:

- Deploy the autotuning algorithm as a stand-alone embedded module on your hardware, removing Simulink from the loop.
- Running the Simulink model with the Online PID Tuner block in external mode, initiate, monitor, and analyze the autotuning process via Simulink. Then, apply the tuned parameters and deploy the PID controller.

The Online PID Tuner block works with any asymptotically stable SISO plant, whether low-order or high-order, with or without time delay, and with or without direct feedthrough. It can tune any type of PID controller. The block lets you trigger the tuning process via a start/stop signal, so you can tune and retune your controller at any time.

For more information, see PID Autotuning Basics and the Online PID Tuner block reference page.

PID Autotuning: Automatically tune PID controller gains for models with plants that do not linearize

The new Frequency Response Based PID Tuner lets you tune a PID controller using estimated plant frequency responses near the target open-loop bandwidth. The new tool simulates the model to collect plant input and output data. It then tunes PID controller parameters based on plant frequency responses estimated from the data. This tuner is a useful alternative when **PID Tuner** cannot linearize the plant at the operating point you want to use for tuning.

You can use the tuner for PID controller and PID controller (2DOF) blocks to achieve the bandwidth and phase margin you specify. You can use the tuner with any plant that is asymptotically stable or has a single integrator, even if disturbances are present in the plant model. It can tune any type of PID controller in either continuous or discrete time.

For more information, see Frequency Response Based Tuning Basics.

Simscape Model Trimming: Improve operating point calculation for Simscape models using new algorithms

You can now use new steady-state trimming optimizers when finding operating points for Simscape models. These optimizers enforce the consistency of the model initial condition at each evaluation of the objective function or nonlinear constraint function, which produces better trimming results for Simscape models. The new optimizers require Optimization Toolbox software.

For more information, see `findopOptions` and Find Steady-State Operating Points for Simscape Models.

Linearization Advisor: Troubleshoot linearization results using new interactive tool and commands

The new Linearization Advisor lets you debug incorrect or unexpected linearization results in the Linear Analysis Tool or at the command line. For example, you can determine why a model linearizes to zero. To debug linearization results, you can use the Linearization Advisor to:

- Find blocks in your model that are potentially problematic for linearization. For more information, see Identify and Fix Common Linearization Issues.
- Find blocks in your model that match specific criteria using custom queries. For more information, see Find Blocks in Linearization Results Matching Specific Criteria.
- Troubleshoot individual block linearizations. For more information, see Block Linearization Troubleshooting.

Linear Analysis Tool: Constrain derivatives of model states that are not at steady state

When trimming a model using the Linear Analysis Tool, you can now constrain the derivatives of model states that are not at steady state during trimming. Previously you could only constrain these derivatives when trimming models at the command line. Using such constraints, you can trim the state derivatives to known nonzero values or specify derivative tolerances for states that cannot reach steady state.

For more information, see Compute Steady-State Operating Point from State Specifications.

New Example: Design PID controller using estimated frequency response of Simscape Electrical model

To design a controller for a Simscape Electrical plant model, you must first obtain a linear model of the plant. However, such models typically linearize poorly due to high-frequency switching components, such as pulse-width modulation (PWM) signal generators.

The new example Control Design of a Boost Converter Using Frequency Response Data shows how to estimate the frequency response of a Simscape Electrical model and design a PID controller using the estimated response.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`'graddescent_elim'` value for the `Optimizer` parameter of `findopOptions`	Still works	`'graddescent-elim'`	Consider replacing instances of `'graddescent_elim'` with `'graddescent-elim'`.

R2017a

Version: 4.5

New Features

Bug Fixes

Custom Constraints and Objective Functions for Trimming Simulink Models: Calculate operating points with increased flexibility

You can now add custom constraints and objective functions when trimming Simulink models. You can specify either custom constraints, a custom objective function, or both. During trimming, the `findop` command applies the custom constraints and objective function in addition to the standard constraints and objective function.

For complex models, you can simplify your custom constraints and custom objective function by specifying a custom mapping. This mapping defines the subsets of model states, inputs, and outputs required for computing the constraints and objective function. Creating a mapping function involves finding the indices of the required states, inputs, and outputs in the operating point specification for a model. You can find these indices using the new `getStateIndex`, `getInputIndex`, and `getOutputIndex` commands.

For more information, see Compute Operating Points Using Custom Constraints and Objective Functions.

Bounds on State Derivatives During Trimming: Constrain derivatives of model states that are not at steady state

When trimming a model at the command line, you can now constrain the derivatives of model states that are not at steady state during trimming. Using such constraints, you can trim the state derivatives to known nonzero values or specify derivative tolerances for states that cannot reach steady state.

For more information, see `operspec` and Compute Operating Points from State Specifications at Command Line.

addoutputspec Command: Add output specification to multiple operating point specification objects in an array

You can now simultaneously add an output specification for a Simulink model to multiple operating point specification objects in an array with one call to `addoutputspec`. To do so, all of the specification objects must have the same `Model` property. Previously, you had to add the output specification to each operating point specification individually, which required compiling the model multiple times.

For more information, see the `addoutputspec` reference page.

New Syntax For setBlockParam: Specify multiple block parameterizations at once

You can now specify custom parameterizations for multiple blocks in an `slTuner` interface with one call to `setBlockParam`. The following syntax assigns the parameterizations `tunable_mdl1,tunable_mdl2,...,tunable_mdlN` to blocks `blk1,blk2,...,blkN`, respectively. The blocks are designated as tuned blocks in an `slTuner` interface `st`.

```
setBlockParam(st,blk1,tunable_mdl1,blk2,tunable_mdl2,...,blkN,tunable_mdlN)
```

For more information, see the `setBlockParam` reference page.

R2016b

Version: 4.4

New Features

Bug Fixes

Batch Trimming for Parameter Variation: Vary model parameters, and compute multiple operating points using a single model compilation

You can now efficiently batch trim a model for variations in model parameters using the `findop` command. If the varying parameters are tunable, the software compiles the model only once, making batch trimming faster, especially for models that are expensive to compile.

To batch trim your model for a single varying parameter, specify the new `param` input argument as a structure with the following fields:

- `Name` — Parameter name, specified as a character vector or MATLAB expression
- `Value` — Parameter sample values, specified as a double array

To vary the value of multiple parameters, specify an array of such structures.

For more information, see Batch Compute Steady-State Operating Points for Parameter Variation.

Improved LPV System Construction: Compute operating point offsets for model inputs, outputs, states, and state derivatives during linearization

You can now compute operating point offsets for model inputs, outputs, states, and state derivatives when linearizing Simulink models.

To construct a linear parameter-varying (LPV) system, you linearize your model for a grid of operating points in the scheduling parameter space. Each grid point contains offset information about model inputs, outputs, states, and state derivatives (continuous-time) or updates (discrete-time). Starting in R2016b, you no longer manually construct the offset structure for each grid point. Instead, linearization commands can now return this offset information.

To obtain operating point offsets, first create a `linearizeOptions` option set, and set the new `StoreOffsets` option to `true`.

You can then linearize your model and obtain the operating point offsets using the:

- `linearize` command.
- `getIOTransfer`, `getLoopTransfer`, `getSensitivity`, and `getCompSensitivity` linearization commands for an `slLinearizer` interface.
- Linearization commands for an `slTuner` interface. In this case, use an `sltunerOptions` option set.

For each of these linearization functions, the new `info` output argument is a structure with an `Offsets` field. The new `getOffsetsForLPV` command extracts the linearization offsets from the `info` and converts them into the required format for the LPV System block.

For an example of LPV construction using operating point offsets, see Approximating Nonlinear Behavior Using an Array of LTI Systems. For more information on LPV systems, see Linear Parameter-Varying Models.

operspec Command: Create array of operating point specification objects

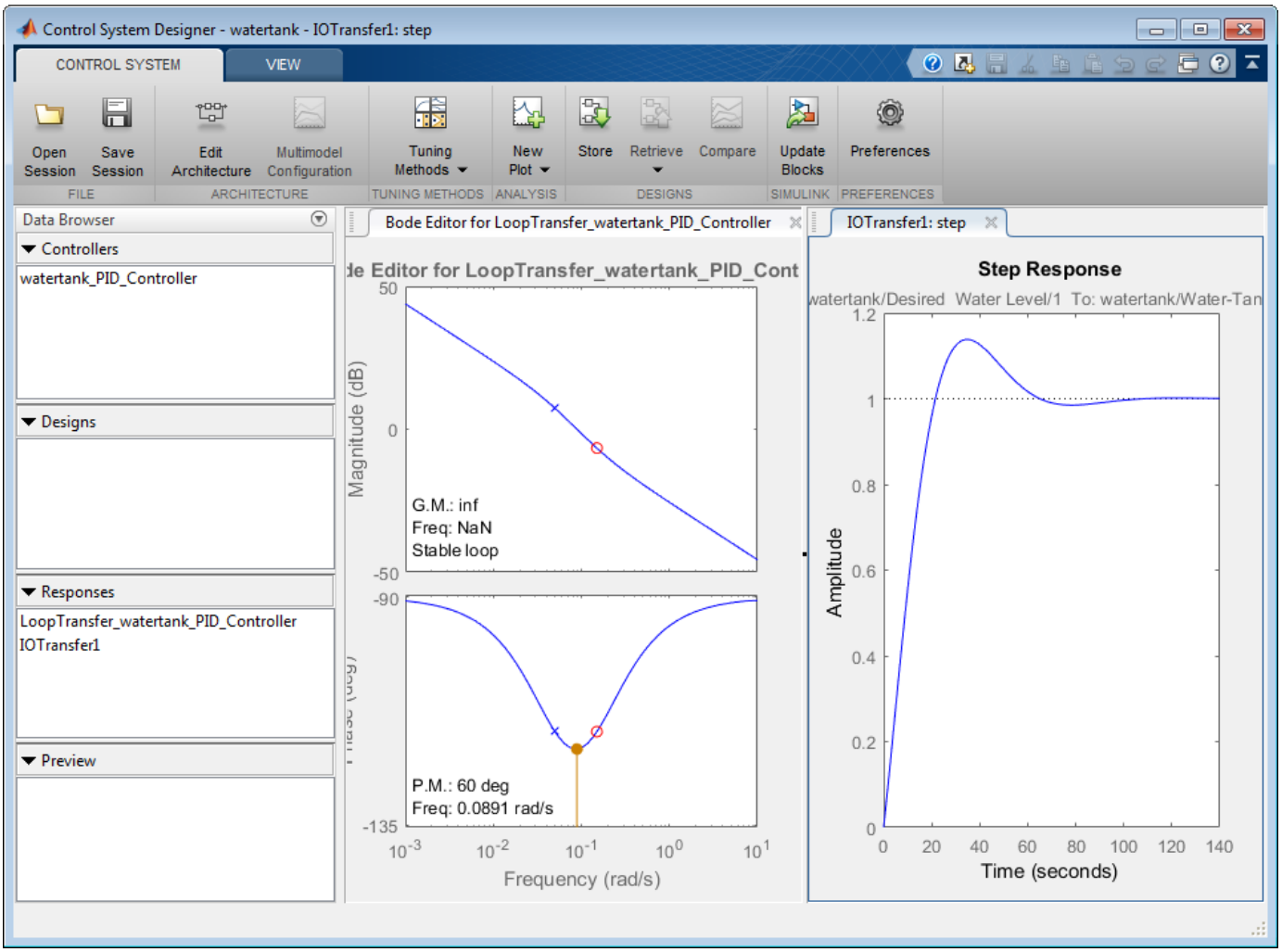
You can now create an array of operating point specification objects using the `operspec` command. You can modify the individual specification objects based on your trimming requirements and batch-compute operating points using a single model compilation.

For an example that uses an array of operating point specification objects, see Designing a Family of PID Controllers for Multiple Operating Points.

R2016a

Redesigned Control System Designer: Design SISO controllers for feedback systems in Simulink using improved interactive workflows

The redesigned **Control System Designer** app streamlines workflows for designing controllers for SISO feedback control systems in Simulink using graphical and automated tuning methods.



For more information on using **Control System Designer** in Simulink, see:

- Control System Designer
- Control System Designer Tuning Methods
- Design Compensator Using Automated PID Tuning and Graphical Bode Design
- Analyze Designs Using Response Plots

Control System Tuner App and systune Command: Automatically tune single-loop and multiloop control systems in Simulink to meet design requirements

A Robust Control Toolbox™ license is no longer required to use the `systune` or `looptune` commands or to use Control System Tuner. The Control System Tuner app and the `systune` command automatically tune control systems from high-level design goals you specify, such as reference tracking, disturbance rejection, and stability margins. You can now use these tools to tune control systems modeled in Simulink with a Simulink Control Design license.

To tune a control system, you specify the tunable elements of your control system. You then capture your design requirements using the library of tuning goals. The software jointly tunes all the free parameters of your control system regardless of control system architecture, the number of feedback loops it contains, or whether it is modeled in MATLAB or Simulink.

For information about using these tools, see:

- Tuning with Control System Tuner
- Programmatic Tuning

You can also use the `systune` command to tune gain-scheduled controllers for control systems in which plant dynamics change with operating conditions or time. For more information, see Gain Scheduling.

TimeUnit property added to slLinearizer and slTuner

You can now specify time units for `slTuner` and `slLinearizer` interfaces using the new `TimeUnit` property. This property specifies the time units for linearized models returned by `getIOTransfer`, `getLoopTransfer`, `getSensitivity`, and `getCompSensitivity`.

Starting in R2016a, for both `slLinearizer` and `slTuner` interfaces, specify time units using dot notation. For example:

```
sllin = slLinearizer('scdcascade',{'r','u1','u2','y1','y2','y1m','y2m'});
sllin.TimeUnit = 'minutes';
```

When loading `slLinearizer` and `slTuner` interfaces that were saved using a previous release, the `TimeUnit` property is set to `'seconds'` by default.

R2015b

Automatic Tuning of 2-DOF PID Controllers with Fixed Setpoint Weights

When you use PID Tuner to tune a PID Controller (2DOF) block, new options let you select controller types with fixed setpoint weights on the proportional and derivative terms, b and c. These new options include commonly-used fixed-weight controller configurations such as I-PD ($b = 0$, $c = 0$) and PI-D ($b = 1$, $c = 0$).

In R2015a, PID Tuner treated all PID coefficients, including setpoint weights, as free parameters when you tuned a PID Controller (2DOF) block. In previous releases, you could only tune 2-DOF controllers using the setpoint-weight values that you set manually in the block.

For more information about tuning 2-DOF PID controllers, see Design Two-Degree-of-Freedom PID Controllers. For more information about using PID Tuner, see Introduction to Automatic PID Tuning.

getTunedValue and setTunedValue commands for accessing tuned variables within slTuner interface

To access tuned variables within an `slTuner` interface, use the new `getTunedValue` and `setTunedValue` commands. Tuned variables are any Control Design Blocks involved in the parameterization of a tuned Simulink block, either directly or through a generalized parametric model.

For Simulink blocks parameterized by a generalized model or a tunable surface:

- `getBlockValue` now provides access only to the overall value of the block parameterization. To access the values of the tuned variables within the block parameterization, use `getTunedValue`.
- `setBlockValue` can no longer be used to modify the block value. Use `setTunedValue` to modify the values of tuned variables within the block parameterization.

For Simulink blocks parameterized by a Control Design Block, the block itself is the tuned variable. To modify the block value, you can use either `setBlockValue` or `setTunedValue`. Similarly, you can retrieve the block value using either `getBlockValue` or `getTunedValue`.

Also, the syntax `setBlockValue(st,M)`, where `M` is a generalized model, has been replaced. To set the values of tunable parameters within a custom parameterization using a generalized model, use the new `setTunedValue(st,M)` syntax. The `setBlockValue(st,M)` syntax will continue to work in future releases.

Compatibility Considerations

- If your code uses `getBlockValue` or `setBlockValue` to access the values of tunable elements within blocks parameterized by a generalized model, modify your code to use the new `getTunedValue` and `setTunedValue` commands. For example, when accessing or modifying the tunable parameter`'Ki'` in the `slTuner` interface `st`, make the following code substitutions:

Old Syntax	Result	New Syntax
`KiTuned = getBlockValue(st,'Ki')`	Error	`KiTuned = getTunedValue(st,'Ki')`

Old Syntax	Result	New Syntax
`setBlockValue(st,'Ki',10)`	Error	`setTunedValue(st,'Ki',10)`

- If your code uses `setBlockValue(st,M)` to set the values of tunable parameters within an `slTuner` interface, consider modifying your code to use the new `setTunedValue(st,M)` syntax.

getBlockParam and getBlockValue return parameterizations and values in a structure

New syntax of the `getBlockParam` command returns a structure that contains all block parameterizations of an `slTuner` interface. The syntax returns a structure, `S`, whose field names are the names of the tunable blocks in the `sltuner` interface, `st`.

```
S = getBlockParam(st)
```

For more information, see the `getBlockParam` reference page.

Similarly, a new syntax of the `getBlockValue` command returns a structure that contains the current values of all block parameterizations of an `slTuner` interface. The syntax returns a structure, `S`, whose field names are the names of the tunable blocks in the `sltuner` interface, `st`.

```
S = getBlockValue(st)
```

You can use this syntax to transfer the tuned values from one `slTuner` interface to another `slTuner` interface with the same tuned block parameterizations:

```
S = getBlockValue(st1);
setBlockValue(st2,S);
```

For more information, see the `getBlockValue` reference page.

Compatibility Considerations

Previously, the syntax `getBlockParam(st)` returned the current block parameterizations of `st` as a vector.

```
[Param1,Param2,...] = getBlockParam(st)
```

Now, using this syntax causes an error. You can still obtain block parameterizations in a list by specifying the block names as input arguments, as follows:

```
[Param1,Param2,...] = getBlockParam(st,Blkname1,Blkname2,...)
```

Similarly, the syntax `getBlockValue(st)` previously returned the current values of the block parameterizations of `st` as a vector.

```
[Val1,Val2,...] = getBlockValue(st)
```

Now, using this syntax causes an error. You can still obtain block parameterization values in a list by specifying the block names as input arguments, as follows:

```
[Val1,Val2,...] = getBlockValue(st,Blkname1,Blkname2,...)
```

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`val = getBlockValue(st,blk)` with `blk` representing a tunable element in `slTuner` interface `st`.	Error	`val = getTunedValue(st,blk)`	See “getTunedValue and setTunedValue commands for accessing tuned variables within slTuner interface” on page 16-2 for more information.
`setBlockValue(st,blk,val)` with `blk` representing a tunable element in `slTuner` interface `st`.	Error	`setTunedValue(st,blk,val)`	See “getTunedValue and setTunedValue commands for accessing tuned variables within slTuner interface” on page 16-2 for more information.
`setBlockValue(st,M)`	Still Works	`setTunedValue(st,M)`	See “getTunedValue and setTunedValue commands for accessing tuned variables within slTuner interface” on page 16-2 for more information.
`[Param1,Param2,...] = getBlockParam(st)`	Error	`S = getBlockParam(st)`	`getBlockParam(st)` now returns a structure that contains the current parameterizations of all tunable blocks in `st`. Update scripts and functions that use `getBlockValue(st)` to use an output structure.
`[Val1,Val2,...] = getBlockValue(st)`	Error	`S = getBlockValue(st)`	`getBlockValue(st)` now returns a structure that contains the current values of the parameterizations of all tunable blocks in `st`. Update scripts and functions that use `getBlockValue(st)` to use an output structure.

R2015a

Version: 4.2

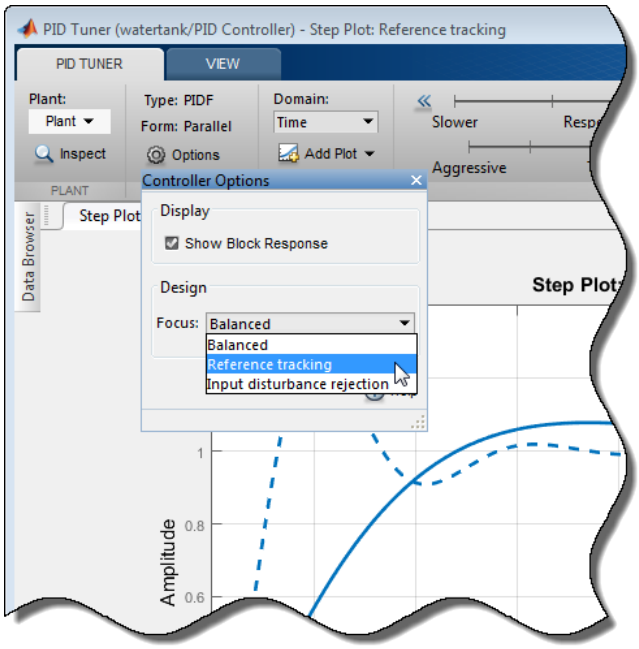
New Features

Bug Fixes

Improved input disturbance rejection with the PID tuning algorithm

Controller tuning with the PID Tuner app now yields better disturbance rejection by default. For a given target phase margin, the tuning algorithm selects PID coefficients that achieve a balance between reference tracking and input disturbance rejection. If you require more disturbance rejection or better reference tracking than the default algorithm provides, PID Tuner has a new Design Focus option. Use this option to alter the balance that the tuning algorithm sets between reference tracking and input disturbance rejection. For instance, setting the design focus to reference tracking improves the reference tracking performance of the tuned controller, with some cost to disturbance rejection. Similarly, setting the design focus to input disturbance rejection improves rejection with some cost to reference tracking. Changing design focus is most effective when tuning PID controllers, rather than controllers with fewer free parameters, such as PI.

To use the Design Focus option in PID Tuner, click **Options** and select a design focus from the **Focus** menu.



You can still use the **Response Time** and **Transient Behavior** sliders to further adjust the balance between reference tracking and input disturbance rejection.

For more information about using the design focus option, see Tune PID Controller to Favor Reference Tracking or Disturbance Rejection.

For more information about using PID Tuner, see Introduction to Automatic PID Tuning.

Automatic tuning of setpoint weight coefficients in 2-DOF PID Controller block for improved disturbance rejection

You can now use PID Tuner to tune all parameters of the PID Controller (2DOF) block, including the setpoint weights *b* and *c*. When you open the PID Tuner app from a PID Controller (2DOF) block in your Simulink model, the software automatically tunes all parameters of the block to achieve a balance between performance and robustness. When you use the Response Time and Transient

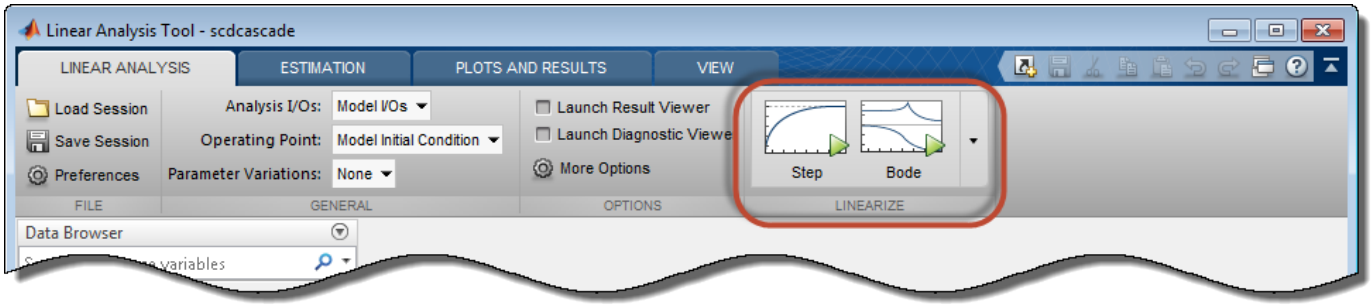
Behavior sliders to adjust that balance, PID Tuner adjusts all parameters, including *b* and *c* if necessary. Previously, the software tuned only the PID gains and filter coefficients, and you had to adjust the setpoint weights manually.

For more information about 2-DOF PID controllers, see Design Two-Degree-of-Freedom PID Controllers. For more information about using PID Tuner, see Introduction to Automatic PID Tuning.

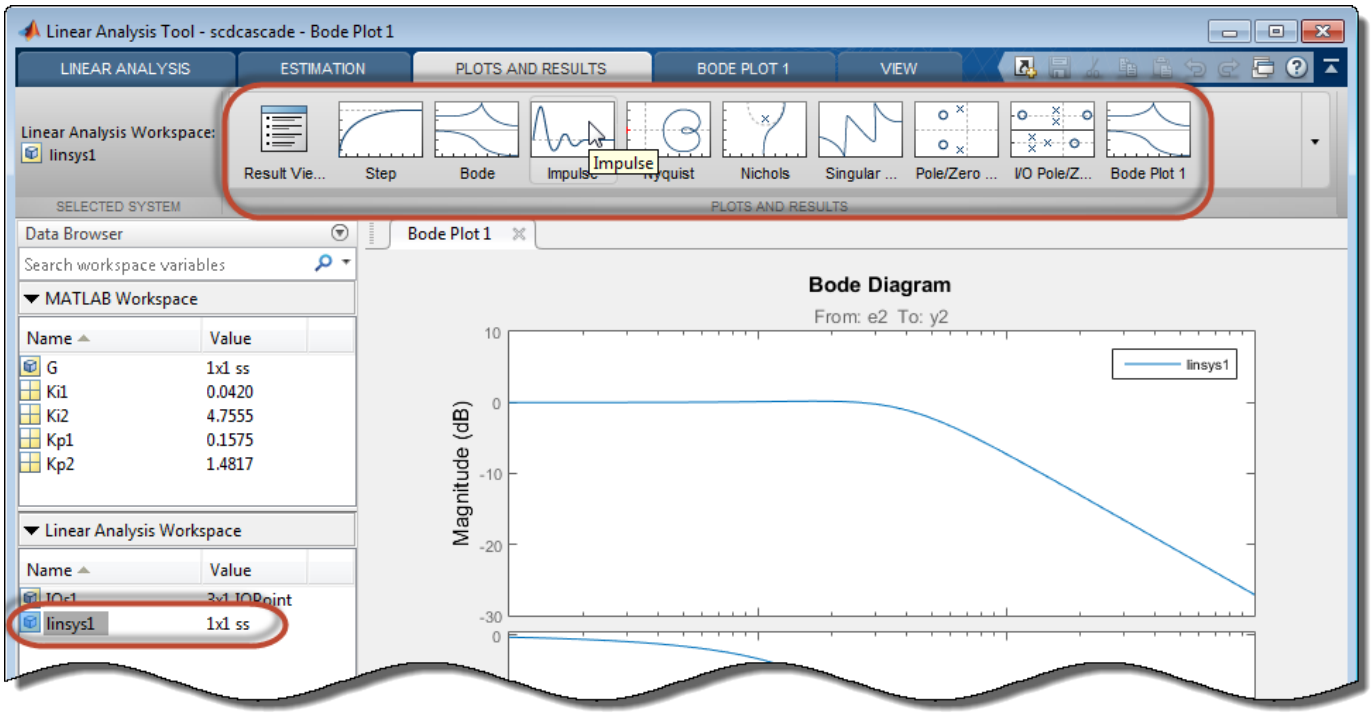
Linear Analysis Tool enhancements for improved linear analysis workflows

The redesigned Linear Analysis Tool streamlines workflows for linear analysis tasks such as finding steady-state operating points (trimming), linearization, and frequency-response estimation. Improvements include:

- The **Linear Analysis** and **Estimation** tabs are enhanced with a plot gallery for quick, visual selection of plot types for viewing linearized or estimated system responses.



- The new **Plots and Results** tab also provides a gallery of plot types for creating additional plots of system responses. In the **Plots and Results** tab, select a dynamic system model from the MATLAB Workspace or the Linear Analysis Workspace. The plot gallery updates to reflect the available plots for the type of system you have selected. If the system you select was created in the Linear Analysis Tool session, the result viewer is also available in the plot gallery.



To access the Linear Analysis Tool from a Simulink model, in the Simulink Editor, select **Analysis > Control Design > Linear Analysis**. See the following topics for more information about using Linear Analysis Tool for:

- Computing operating points — Steady-State Operating Points (Trimming) from Specifications
- Exact linearization — Linearize Simulink Model at Model Operating Point
- Frequency Response Estimation — Estimate Frequency Response Using Linear Analysis Tool

Simplified and faster linear analysis of Simulink models across different model parameter values in Linear Analysis Tool

You can now use the Linear Analysis Tool to efficiently batch linearize a model at varying plant or controller parameter values. Batch linearization refers to extracting multiple linearizations from a model for various combinations of model parameters. Using the new parameter-variation capability of Linear Analysis Tool, you can obtain multiple linearizations for varying values of any parameter in your Simulink model. For example, you can linearize your system at multiple values of plant coefficients, controller gains, or controller sample times.

Use Linear Analysis Tool response plots to examine time-domain and frequency-domain responses of the resulting linear models. For instance, you can compute and plot linearizations for the plant model, overall closed-loop transfer function, and plant disturbance rejection for varying plant and controller parameter values.

In cases where the varying parameters are all tunable, Linear Analysis Tool requires only one model compilation to compute multiple open-loop and closed-loop transfer functions for varying parameter values. This efficiency is especially advantageous for models that are expensive to compile repeatedly.

This new functionality brings the batch linearization capability of the `slLinearizer` interface and the `linearize` command to the Linear Analysis Tool. Any parameter variations you specify in Linear Analysis Tool are included in MATLAB code you generate from the session.

For more information about performing varying parameters for linearization with the Linear Analysis Tool, see Batch Linearize Model for Parameter Value Variations Using Linear Analysis Tool.

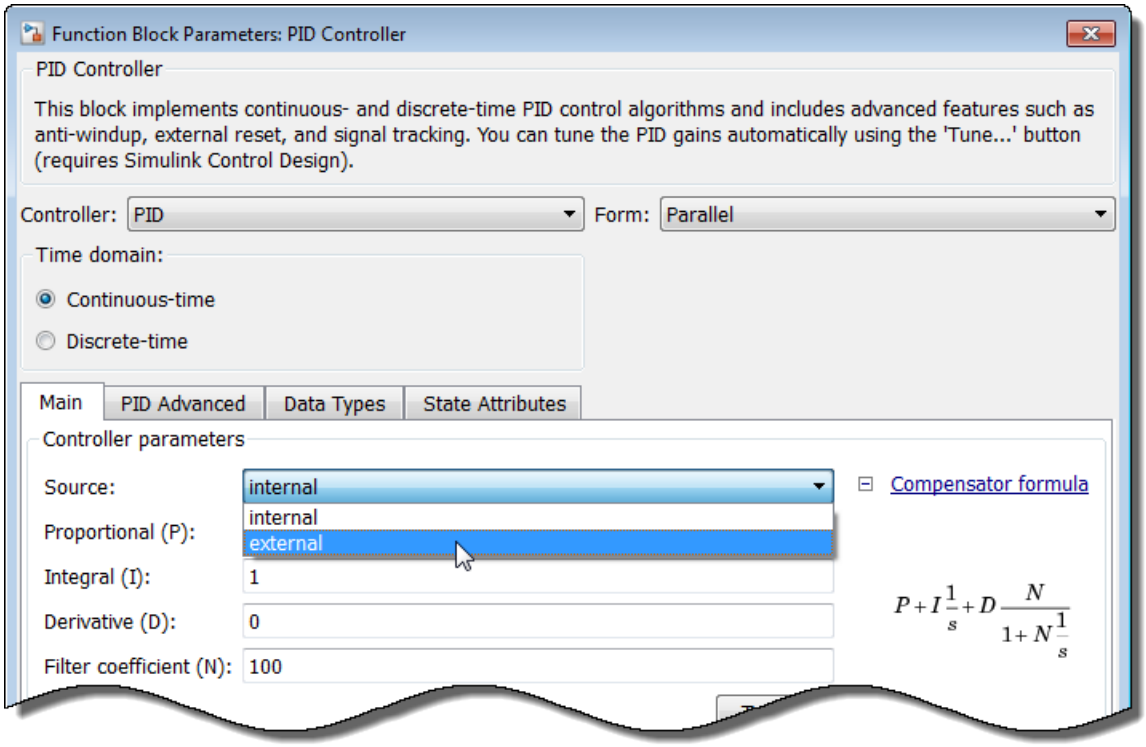
For more information about batch linearization, see What Is Batch Linearization?.

Option to provide PID gains as external inputs to PID Controller and PID Controller (2DOF) blocks

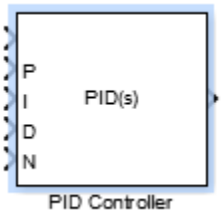
A new option in the PID Controller and PID Controller (2DOF) blocks adds signal inputs for the PID gains and filter coefficients. Previously the PID parameters had to be entered in the block dialog box as numerical values or MATLAB expressions. Enabling external inputs for the parameters allows you to compute PID gains and filter coefficients externally to the block and provide them to the block as signal inputs.

External gain input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use external gain input to implement gain-scheduled PID control, in which controller gains are determined by logic or other calculation in the Simulink model and passed to the block.

To enable external inputs for the PID coefficients, in the block dialog box, in the **Controller parameters** section, in the **Source** menu, select `external`.



When you click **OK** or **Apply**, the new inputs appear on the block in the Simulink model.



For an example illustrating the use of external gain inputs for gain scheduling, see Implement Gain-Scheduled PID Controllers. For more information about using the PID controller blocks, see the PID Controller and PID Controller (2DOF) block reference pages.

R2014b

Version: 4.1

New Features

Bug Fixes

Unfiltered-derivative option in discrete-time PID Controller blocks

You can now specify an unfiltered derivative term in the discrete-time PID Controller and PID Controller (2DOF) blocks. Previously, these blocks required a finite derivative filter constant on the derivative term.

To specify an unfiltered derivative, in the **Main** pane of the block dialog box, uncheck **Use filtered derivative**. Unchecking this option replaces the derivative filter with a discrete differentiator. The option is checked by default for compatibility with previous versions.

For more information about the blocks, see PID Controller and PID Controller (2DOF).

FOH and matched methods for automatic rate conversion in slTuner interface

When you use `systune` to tune a Simulink model, block tuning is performed at the rate specified in the `slTuner` interface to the model (see the `Ts` property of `slTuner`). When you write the tuned values back to Simulink, if the sample time of the Simulink block differs from the sample time used for tuning, a rate conversion takes place automatically . You can use the `setBlockRateConversion` command to specify the rate conversion method.

In addition to the ZOH (zero-order hold) and Tustin methods, `setBlockRateConversion` now supports FOH (first-order hold) and matched methods. The matched method is only available for SISO blocks.

For more information, see the `setBlockRateConversion` reference page.

Improved support for genss block parameterization in slTuner interface

When you assign a custom parameterization of a tunable block in an `slTuner` interface to a Simulink model, you can now use the `setBlockValue` command to set the value of a particular tunable element within the custom parameterization of the block. Similarly, you can now use `getBlockValue` to query the value of a tunable element within the block.

Previously, `setBlockValue` and `getBlockValue` could only set or query the current value of the entire parameterization of a Simulink block. There was no way to set or query the current value of a single tunable element of a custom parameterization.

For more information, see the reference pages for `setBlockValue` and `getBlockValue`.

Support for additional multiplication modes in slTuner parameterization of Gain block

When you create an `slTuner` interface to a Simulink model that contains a Gain block with vector inputs and outputs, a parameterization is assigned to the block for any of the following multiplication modes of the block:

- `Element-wise (K.*u)`
- `Matrix (K*u)`

- `Matrix (u*K)`

(The multiplication mode is specified in the **Main** tab of the block dialog box.) Previously, the `slTuner` interface could only assign a parametrization for `Element-wise (K.*u)` and `Matrix (K*u)` modes.

You can now set parameter values for the Gain block in the `slTuner` interface using `setBlockValue` for all these multiplication modes. Previously, `setBlockValue(ST,blockID,value)` worked only for `Matrix (K*u)` mode.

R2014a

slTuner interface for improved control system tuning of Simulink models with systune or looptune functions, including tuning of gain-scheduled controllers (with Robust Control Toolbox)

Use the new `slTuner` interface for tuning control systems in Simulink models. This interface replaces `slTunable`. The `slTuner` interface allows you to:

- Tune model blocks and subsystems to meet tuning goals using the `systune` and `looptune` functions. (Both functions require a Robust Control Toolbox license.)
- Perform robust tuning of a controller against a set of plant models using `systune`. You can configure an `slTuner` interface to vary model parameter values and operating points. When you call `systune` for the interface, the software returns a controller that satisfies the tuning goals for all the specified model variations.
- Validate the controller design by examining the transfer function for relevant I/O sets using the `getIOTransfer`, `getLoopTransfer`, `getSensitivity`, and `getCompSensitivity` functions.

`slTuner`, similar in design to `slLinearizer`, simplifies I/O management in the controller tuning and validation workflow. You specify signals of interest as analysis points. You can use these analysis points to configure design requirements and specify linearization inputs/outputs when you extract transfer functions.

For more information on command-line tuning of Simulink models with `slTuner`, see Command-Line Control System Tuning.

Compatibility Considerations

The `slTunable` interface will continue to work for backward compatibility. However, only the `slTuner` interface will be supported and enhanced in future releases. Therefore, adoption of the `slTuner` interface is strongly recommended.

For documentation of the `slTunable` interface, see `slTunable` in the R2013b documentation.

Redesigned PID Tuner for improved PID tuning workflow

The redesigned PID Tuner streamlines workflows for automatic and interactive tuning of PID controllers. Improvements include:

- Additional options to import a plant into the PID Tuner. These options are especially useful if your Simulink plant model linearizes to zero. You can:
  - Relinearize the model using a simulation snapshot operating point that you find based on the controller error signal.
  - Identify an LTI plant model from simulated input-output data. This option requires a System Identification Toolbox™ license.

    For more information about this option, see “PID controller tuning using system identification to model the plant from simulation input-output data in the PID Tuner” on page 19-3.
- Ability to compare the response of multiple plant models to the same controller. For example you can test the robustness of a controller to plant model uncertainty. Suppose you have tuned a controller for one plant model. You can import variations of the plant model into the PID Tuner.

Then, you can plot the controller response for all the plant models in a single figure to compare the controller performance.

- Ability to display multiple response plots simultaneously. For example, you can tune a controller while simultaneously monitoring the reference tracking and output disturbance rejection plots.

To access the PID Tuner, in the PID block dialog box, click **Tune**.

For more information about using the PID Tuner, see PID Controller Tuning.

PID controller tuning using system identification to model the plant from simulation input-output data in the PID Tuner

If you have System Identification Toolbox software, you can use the PID Tuner to identify a linear plant model from data obtained by simulating the Simulink model. You use the identified model to tune your PID Controller block. For example, suppose you want to tune the PID controller block in a model that contains a Triggered Subsystem block. The analytical block-by-block linearization algorithm does not support event-based subsystems and therefore the model linearizes to zero. Now, you can simulate the plant response for a specified input and use this simulated data to identify the plant model. The PID Tuner automatically tunes the PID controller for the identified model. You can then interactively adjust the performance of the tuned control system, and save the identified plant and tuned controller.

For an example showing how to use system identification to model a plant for PID tuning, see Interactively Estimate Plant from Measured or Simulated Response Data.

Option to specify multiple substitute linearizations of a Simulink block for batch linearization

The method for specifying substitute linearizations for blocks and model subsystems is simplified. Also, you can specify multiple substitute linearizations for a block and obtain a linearization for each substitution (batch linearization). Use this functionality, for example, to study the effects of varying the linearization of a Saturation block on the model dynamics.

You can specify a substitute linearization as an input to the `linearize` command and as the `BlockSubstitutions` property of the `slLinearizer` and `slTuner` interfaces. Use a structure with the following fields:

- `Name` — Full blockpath, specified as a string. For example, `'scdenginectrlpidblock/valve timing'`.
- `Value` — Substitute linearization, specified as one of the following:
  - Double, for example `1`. Use for SISO models only. For models having either multiple inputs or multiple outputs, or both, use an array of doubles. For example, `[0 1]`. Each array entry specifies a linearization for the corresponding I/O combination.
  - LTI model, with I/Os that match the block specified by `Name`. For example, `zpk([],[-10 -20],1)`.
  - Array of LTI models. For example, `[zpk([],[-10 -20],1); zpk([],[-10 -50],1)]`.

    If you vary model parameter values, then the LTI model array size must match the grid size.
  - Structure with fields: `Specification`, `Type`, and `ParameterNames`, `ParameterValues`. For an example, see Specifying Linearization for Model Components Using System Identification.

Previously, the software supported only this method of specifying a substitute linearization.

To specify substitute linearizations for multiple blocks, create an array of the described structure.

This improvement is also applicable in the Simulink editor when you right-click a block and select **Linear Analysis > Specify Selected Block Linearization**. Select the **Specify block linearization using one of the following** check box. When you select `MATLAB Expression` in the list, you can now specify a double, an array of doubles, an LTI model, or an array of LTI models in the text box.

R2013b

Enhanced linearize command, providing faster batch linearization for model parameter variations

`linearize` now allows you to efficiently batch linearize a model for variations in model parameter values. Use this feature to study the effects of varying a model parameter value. For example, you can vary plant parameter values and analyze the controller robustness to plant model uncertainty. The software uses only one model compilation, making batch linearization faster, especially for models that are expensive to compile repeatedly.

For more information, see Batch Linearize Model for Parameter Value Variations Using linearize and Specify Parameter Samples.

slLinearizer interface, providing faster batch linearization for multiple I/O sets

Use the `slLinearizer` interface to efficiently batch linearize a model. The interface includes linearization commands that you use to extract any open-loop or closed-loop transfer function for varying operating points and parameter values. For instance, you can extract linearizations for the plant model, overall closed-loop transfer function, and plant disturbance rejection, at multiple operating points, for varying plant/controller parameter values.

`slLinearizer` extends the functionality of `linearize` and uses only one model compilation to compute multiple open-loop and closed-loop transfer functions for varying operating points and parameter values. This efficiency is especially advantageous for models that are expensive to compile repeatedly.

For examples that illustrate how to use `slLinearizer`, see:

- Vary Parameter Values and Obtain Multiple Transfer Functions Using slLinearizer
- Vary Operating Points and Obtain Multiple Transfer Functions Using slLinearizer.

linearizeOptions and findopOptions for specifying options for linearization and operating point search

The new options commands `linearizeOptions` and `findopOptions` replace the command `linoptions`. This change simplifies the use of options for linearization and operating point search by separating the available options for each task. See the reference pages for these options commands for more information about how to use them.

Compatibility Considerations

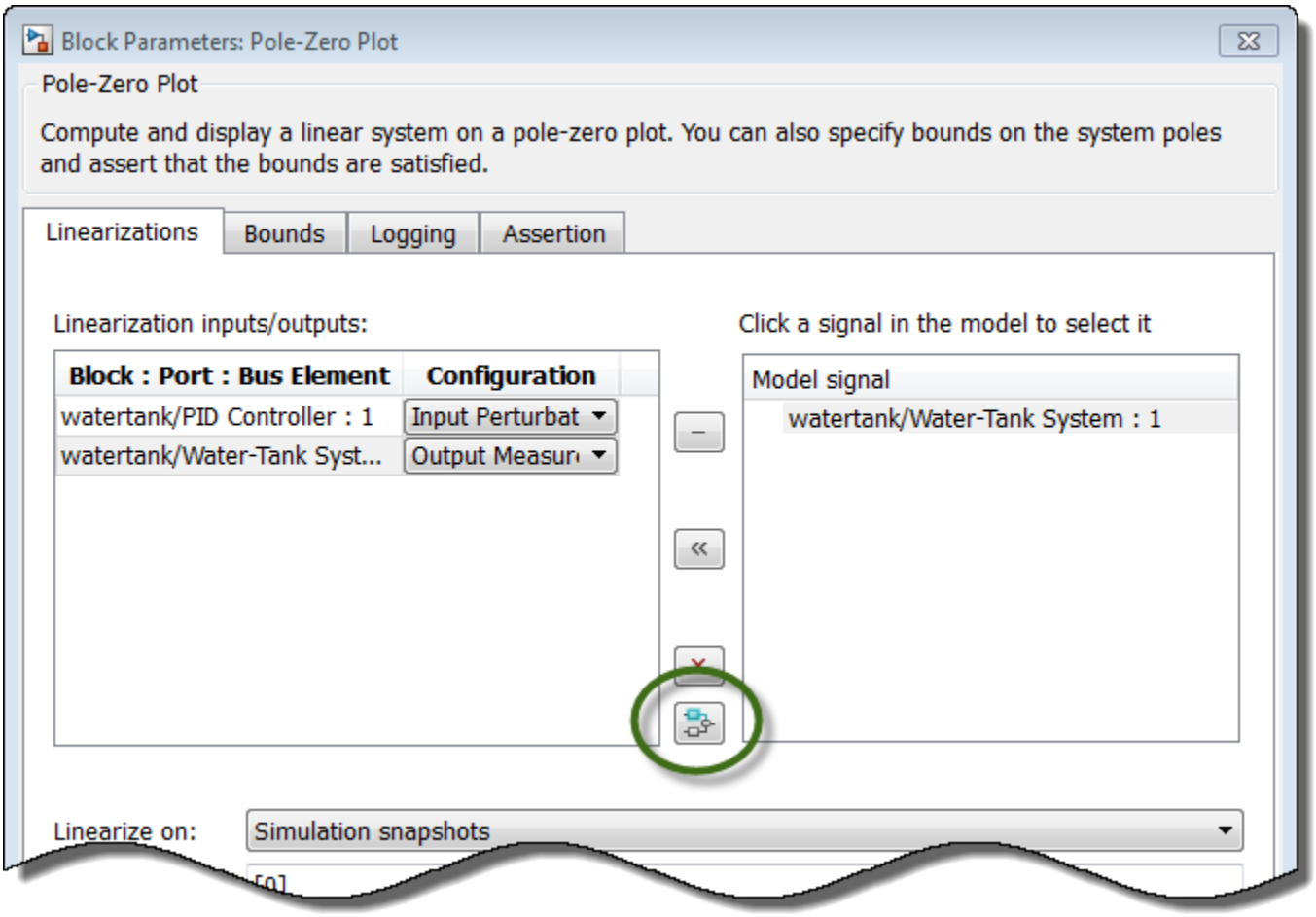
The `linoptions` command will be removed in a future release, and issues a warning as of this release. If you have scripts or functions that use `linoptions`, consider updating them to instead use `linearizeOptions` for linearization and `findopOptions` for operating point search.

Highlight linear analysis points in Linear Analysis Plot Blocks and Model Verification Blocks

The Linear Analysis Plot Blocks and Model Verification Blocks now include a button that highlights signals in your Simulink model which you have selected as linearization inputs or outputs. This

highlighting makes it easier to identify linear analysis signals in your model when you are working in the block dialog box.

In the Linearizations tab of the block, select a signal from the **Linearization inputs/outputs** table. Then click the highlight button:



The block that originates the selected signal is highlighted in the Simulink model.

writeBlockValue command can update Simulink model with tuned parameter values from generalized LTI model

The `writeBlockValue` method of `slTunable` has new syntaxes that allow you to update a Simulink model with tunable parameter values from a generalized LTI model.

`writeBlockValue(ST,M)` updates the current values of the tunable parameters in an `slTunable` interface, `ST`, to match their values in a generalized model, `M`. The command also updates the corresponding parameters in the Simulink model associated with `ST`.
`writeBlockValue(ST,M,BlockID)` updates only the parameters in the specified blocks.

These syntaxes are useful when you use Robust Control Toolbox tuning commands such as `systune`, `looptune`, or `hinfstruct` to tune system responses that you extract from the `slTunable` interface.

For example, suppose you extract a particular closed-loop response from an `slTunable` interface using `getIOTransfer`. You then use `hinfstruct` to tune this closed-loop response. To validate the tuned parameters in the full non-linear Simulink model, use `writeBlockValue` to write the tuned parameter values back into the model.

For more information, see the `slTunable.writeBlockValue` reference page.

Format of BlockData structure identical for snapshot and operating point linearization

When you specify the linearization of a block as a function, the input argument to that function is the structure `BlockData`, whose fields are given in the `linearize` reference page. The field `BlockData.Inputs` is now a data structure array, regardless of whether you are linearizing the model at a snapshot time or at a specified operating point. The data structure `BlockData.Inputs` has the following fields:

- `BlockName` — Contains the name of the block whose output connects to the input of the block whose linearization you specifying. For example, if you are specifying the linearization of a block called `Dynamics`, and the second input of `Dynamics` is driven by a signal from a block called `Torque`, then `BlockData.Inputs(2).BlockName` is the full block path name of `Torque`.
- `PortIndex` — Identifies which output port of `BlockName` corresponds to the input of the block whose linearization you are specifying. For example, if the third output from `Torque` drives the second input of `Dynamics`, then `BlockData.Inputs(2).PortIndex = 3`.
- `Values` — The value of the signal specified by `BlockName` and `PortIndex`. If this signal is a vector-valued signal, `Values` is a vector of corresponding dimension.

Previously, `BlockData.Inputs` had this format only for snapshot linearization. For linearization at a specified operating point, `BlockData.Inputs` was a numeric array of input values.

Compatibility Considerations

If you have scripts or functions that assume `BlockData.Inputs` is a numeric array, update your code to reflect the new structure of `BlockData.Inputs`.

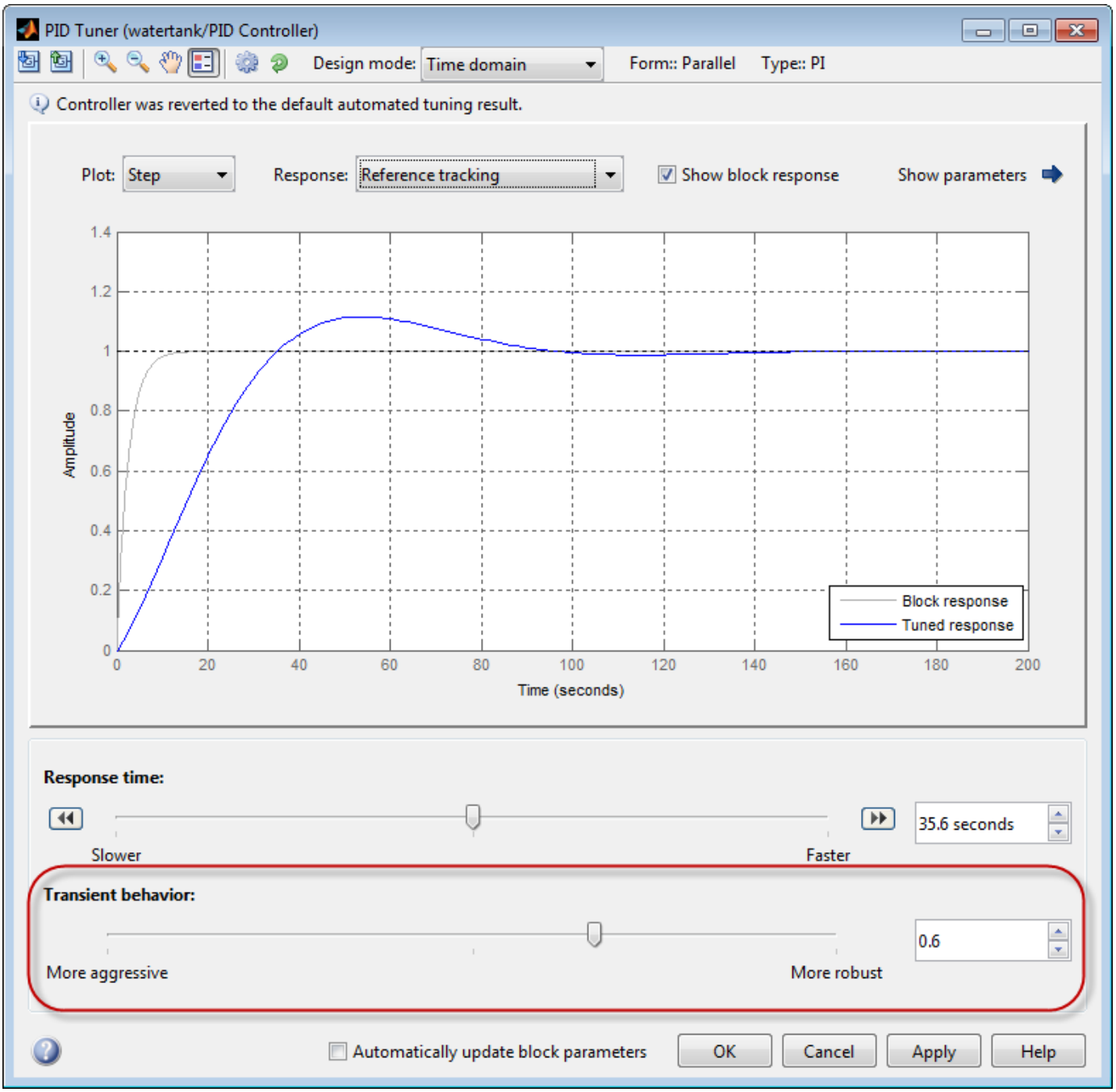
Linear Analysis Blocks and Model Verification Blocks save operating points with computed linear systems

The data logging capability of the Linear Analysis Blocks or Model Verification Blocks now includes an option to save the operating points corresponding to the computed linear systems. When you check **Save data to workspace** and **Save operating points for each linearization** in the Logging tab of the block dialog box, a field named `operatingPoints` is added to the data structure containing the logged data. This field stores the operating point corresponding to each logged linear system in the data structure.

R2013a

Transient behavior slider added to PID Tuner for increased control over reference tracking and disturbance rejection performance

The PID Tuner now has a **Transient behavior** slider for emphasizing either reference tracking or disturbance rejection. When you open the PID Tuner, the tool starts in the `Time domain` design mode, displaying a step plot of the reference tracking response. The new **Transient behavior** slider is beneath the **Response time** slider.



You can use the **Transient behavior** slider when:

- The tuned system's disturbance rejection response is too sluggish for your requirements. In this case, try moving the **Transient behavior** slider to the left to make the controller more aggressive at disturbance rejection.
- The tuned system's reference tracking response has too much overshoot for your requirements. In this case, try moving the **Transient behavior** slider to the right to increase controller robustness and reduce overshoot.

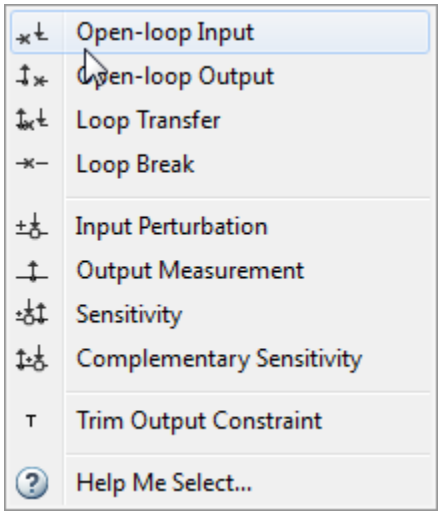
In `Frequency domain` design mode, the PID Tuner has **Bandwidth** and **Phase margin** sliders. These sliders are the frequency-domain equivalents of the **Response time** and **Transient behavior** sliders, respectively.

For an example illustrating the balance between reference tracking and disturbance rejection, see Tune PID Controller to Balance Tracking and Disturbance Rejection Performance.

Linear analysis points redesigned to clarify I/O types and loop openings

Linear analysis points (linearization inputs, outputs and loop-opening locations), that you use to specify the portion of the Simulink model to linearize, have been redesigned. The redesign clarifies what the I/O point means and helps you select the right type of point to compute the desired response.

The analysis points that you select in the **Linear Analysis Points** submenu or the Linear Analysis Tool have been renamed. The points also have updated icons that correspond to markers that appear on your model to indicate the linear analysis point type. The corresponding strings for the `type` argument in the `linio` command have also been renamed.



You can also specify inputs or outputs as open loop while specifying the I/O type. For example:

- In the **Linear Analysis Points** submenu or the Linear Analysis Tool, select **Open-loop Output**.
- At the command line, type `io(2)=linio('magball/Magnetic Ball Plant',1,'openoutput')`.

Thus, when performing tasks such as plant linearization or open-loop linearization, you do not need to specify the opening separately.

The following table maps the renamed analysis points to the point types in previous releases:

Command Line		Linear Analysis Points Submenu or Linear Analysis Tool	
Previous Releases	**R2013a**	**Previous Releases**	**R2013a**
`type = 'in'` and `openloop = 'on'`	`type = 'openinput'`	**Input Point** and **Open Loop**	**Open-loop Input**
`type = 'in'` and `openloop = 'off'`	`type = 'input'`	**Input Point**	**Input Perturbation**
`type = 'out'` and `openloop = 'on'`	`type = 'openoutput'`	**Output Point** and **Open Loop**	**Open-loop Output**
`type = 'out'` and `openloop = 'off'`	`type = 'output'`	**Output Point**	**Output Measurement**
`type = 'inout'` and `openloop = 'off'`	`type = 'sensitivity'`	**Input-Output**	**Sensitivity**
`type = 'outin'` and `openloop = 'on'`	`type = 'looptransfer'`	**Output-Input** and **Open Loop**	**Loop Transfer**
`type = 'outin'` and `openloop = 'off'`	`type = 'compsensitivity'`	**Output-Input**	**Complementary Sensitivity**
`type = 'none'` and `openloop = 'on'`	`type = 'loopbreak'`	**Open Loop**	**Loop Break**

Use the context-sensitive help, or see the `linio` reference page to assist you in selecting an analysis point. For examples on how to specify linear analysis points graphically, see Specify Portion of Model to Linearize in Simulink Model and Specify Portion of Model to Linearize in Linear Analysis Tool.

Linear Analysis Blocks and Model Verification Blocks save data in single object with Simulink model logging output

The data logging option in the Linear Analysis Blocks or Model Verification Blocks saves linear systems computed by the blocks. As of R2013a, when you configure your Simulink model to save simulation output in a single object, the logging output of these blocks is saved as a field in the simulation output object. Previously, these blocks saved logging output as a separate variable in the MATLAB workspace, regardless of the model configuration settings.

For more information about data logging in Simulink, see Export Simulation Data and the `Simulink.SimulationOutput` reference page.

Compatibility Considerations

If you have a Simulink model which you have configured to save simulation output in a single object, and you use data logging option in a Linear Analysis Block or Model Verification Block, the resulting logging output is no longer a separate variable in the MATLAB workspace. If you use the logging output in any scripts, update those scripts to use the `Simulink.SimulationOutput` object instead of a separate variable.

R2012b

MATLAB code generation from Linear Analysis Tool for batch estimation of model frequency responses

You can now generate MATLAB code for frequency response estimation from the Linear Analysis Tool. You can generate either a MATLAB script or a MATLAB function. Generated MATLAB scripts are useful when you want to programmatically reproduce a result you obtained interactively. A generated MATLAB function allows you to perform multiple estimations with systematic variations in estimation parameters such as operating point (batch estimation).

For more information, see Generate MATLAB Code for Repeated or Batch Frequency Response Estimation.

Operating point calculation (trimming) from multiple specifications with only one model compilation

The `findop` command now can find operating points for multiple operating point specifications with a single model compilation. This feature allows you to find multiple trimmed operating points without the overhead of compiling the model for each trimming computation. For an example, see Batch Compute Operating Points with Single Model Compilation.

For more information about operating point calculation and operating point specifications, see the `findop` and `operspec` reference pages.

Export and import operating point specifications in Linear Analysis Tool

When you modify an operating point specification in the Linear Analysis Tool, you can now export the specification to the MATLAB workspace. Exported specifications are saved as operating point specifications objects (see `operspec`). Exporting specifications can be useful when you expect to perform multiple trimming operations using the same or a very similar set of specifications. Additionally, you can export interactively-edited operating point specifications when you want to use the `findop` command to perform multiple trimming operations with a single compilation of the model (see “Operating point calculation (trimming) from multiple specifications with only one model compilation” on page 22-2).

You can also import saved operating point specifications to the Linear Analysis Tool and use them to interactively compute trimmed operating points. Importing a specification can be useful when you want to trim a model to a specification that is similar to one you previously saved. In that case, you can import the specification to the Linear Analysis Tool and interactively change it. You can then export the modified specification, or compute a trimmed operating from it.

For more information, see Import and Export Specifications For Operating Point Search.

For more information about operating point specifications, see the `operspec` and`findop` reference pages.

MATLAB script or function generation from Linear Analysis Tool for repeated or batch linearization

When you generate MATLAB code for linearization from the Linear Analysis Tool, you now have a choice of generating a script that uses the current linearization parameters, or a function that takes

parameter values as input. Previously, you could only generate a MATLAB function with no input parameters.

Generated MATLAB scripts are useful when you want to programmatically reproduce a result you obtained interactively. A generated MATLAB function allows you to perform multiple linearizations with systematic variations in linearization parameters such as operating point (batch linearization).

For more information, see Generate MATLAB Code for Repeated or Batch Linearization.

Print plots to MATLAB figure in Linear Analysis Tool

You can now export a plot from the Linear Analysis Tool to a MATLAB figure window.

For more information, see Print Plot to MATLAB Figure in Linear Analysis Tool.

Commands for setting and querying rate conversion methods in tunable blocks

New commands allow you to query and specify the rate conversion method that the `slTunable` interface uses for converting the sampling time of the parametrization of tunable blocks in a Simulink model.

- `slTunable.getBlockRateConversion` — Query rate conversion method of tunable block
- `slTunable.setBlockRateConversion` — Set rate conversion method of tunable block

These commands are useful, for example, when you use the `slTunable` interface to tune a mixed-rate Simulink model, such as a model having a discrete-time controller and a continuous-time plant. The `slTunable` interface automatically converts the sampling times of tunable blocks where necessary. Using `slTunable.setBlockRateConversion`, you can control the conversion method that the `slTunable` interface uses.

For more information, see:

- Tuning of a Digital Motion Control System
- `slTunable.getBlockRateConversion` and `slTunable.setBlockRateConversion` reference pages

“Ignore saturation when linearizing” checked by default in PID Controller and PID Controller (2DOF) blocks

The default linearization behavior of the PID Controller and PID Controller (2DOF) blocks now forces linearization commands to ignore block output limits. Ignoring output limits allows you to linearize a model around an operating point even if that operating point causes the PID Controller block to exceed the output limits.

To cause linearization commands not to ignore block output limits, clear the **Ignore saturation when linearizing** checkbox in the block dialog box, PID Advanced tab.

showBlockValue renamed to showTunable

The command `showBlockValue` is now called `showTunable`. Use `showTunable` to obtain the current value of block parametrizations from an `slTunable` interface.

Compatibility Considerations

Replace instances of `showBlockValue` in your code with `showTunable`.

R2012a

Version: 3.5

New Features

Bug Fixes

Create Linearization Input/Output Sets in the Linear Analysis Tool

You can now interactively create I/O sets for linearization or frequency response estimation in the Linear Analysis Tool, without adding linearization points to your model. Previously, you had to modify your model to interactively create I/O sets.

For more information, see Create Linearization I/O Sets In Linear Analysis Tool.

For more information about using the Linear Analysis Tool, see Linearize at Model Operating Point and Estimating Frequency Response.

Specify Feedback Sign for getLoopTransfer Without Specifying Loop Openings

An additional syntax for `slTunable.getLoopTransfer` allows you to specify the feedback sign without having to provide an `openings` argument. Use the syntax

```
L = getLoopTransfer(ST,location,sign)
```

to calculate the point-to-point open-loop transfer function at `location` of the Simulink model described by the `slTunable` interface `ST`. The software uses the sign convention specified by `sign` (`+1` or `−1`) to compute `L`.

Previously, to specify a feedback sign, you had to use the syntax

```
L = getLoopTransfer(ST,location,openings,sign)
```

This syntax required you to specify an `openings` argument to specify `sign`. The new syntax allows you to specify `sign` while using the default loop openings.

For more information, see the `slTunable` and `slTunable.getLoopTransfer` reference pages.

R2011b

Redesigned Graphical Tool for Improved Linear Analysis Workflows

The new interactive Linear Analysis Tool streamlines workflows for linear analysis tasks such as finding steady-state operating points (trimming) and linearization.

To access the Linear Analysis Tool:

1. Open a Simulink model.
2. In the Simulink model, select **Tools > Control Design > Linear Analysis**.

For more information about using the Linear Analysis Tool for operating-point and linearization workflows, see:

- Steady-State Operating Points
- Linearization

Interactive Frequency Response Estimation and Validation of Linearization Results

The new interactive Linear Analysis Tool provides a graphical interface for frequency response estimation.

To access the Linear Analysis Tool:

1. Open a Simulink model.
2. In the Simulink model, select **Tools > Control Design > Linear Analysis** to launch the Linear Analysis tool.
3. Click the Frequency Response Estimation tab to begin an estimation task.

For more information about using the Linear Analysis Tool for frequency response estimation, see Frequency Response Estimation.

Optimization of Model Parameters to Meet Design Requirements Specified by Model Verification Blocks

If you have Simulink Design Optimization™ software, you can optimize the Simulink model to meet frequency-domain requirements specified in Model Verification blocks. For example, you can optimize the linear system to meet Bode magnitude or gain and phase margin requirements. For more information, see Design Optimization To Meet Frequency-Domain Requirements (GUI).

You can also include time-domain requirements such as step response characteristics for optimization. For more information, see Design Optimization to Meet Time- and Frequency-Domain Requirements.

Automatic Tuning of PID Controller Blocks in a Referenced Model

You can now use the PID Tuner to tune a PID Controller block in a model that is referenced in one or more open models. When you launch the PID Tuner from a block within a model reference, the software prompts you to select which of the open models is the top-level model for analysis and tuning. Previously, the PID Tuner could only use the model containing the PID Controller block as the top-level model.

For more information, see Tuning a PID Controller Within a Model Reference.

Control System Tuning for Simulink Models with looptune or hinfstruct Using slTunable Interface

If you have Robust Control Toolbox software, you can use tuning commands, such as `slTunable.looptune` and `hinfstruct`, to tune control systems modeled in Simulink. The `slTunable` object provides an interface between your Simulink model and these commands.

Use `slTunable` to specify information about your control structure and parametrization. `slTunable` also automates tasks such as linearizing the Simulink model, parametrizing the tunable blocks of your system, and applying tuned parameter values to the model. After you create and configure an `slTunable` object for your control architecture, you can tune the control system using `slTunable.looptune` or `hinfstruct`.

For more information, see Tuning Fixed Control Architectures.

Change in Default Number of Samples in frest.Chirp

The default number of samples (`NumSamples`) in a `frest.Chirp` input signal for frequency response estimation is now given by the formula:

$$\frac{4\pi}{Ts*\min(FreqRange)}.$$

`Ts` is the sampling time of the chirp signal. `FreqRange` is the vector of signal frequencies of the chirp signal.

This formula returns twice the value returned in previous releases.

For example, if you create a default chirp input signal with the command

```
input = frest.Chirp
```

the value of `input.NumSamples` is 10000, instead of the previous value of 5000.

For more information, see the `frest.Chirp` reference page.

Compatibility Considerations

If you have scripts that rely on the default `NumSamples` formula, modify your scripts to account for the new value.

R2011a

Version: 3.3

New Features

Bug Fixes

Ability to Select Individual Bus Elements as Linearization Input and Output Points

Instead of selecting an entire bus, you can now select individual elements of the bus signal as linearization inputs or outputs (I/Os). By selecting individual bus elements, you can:

- Obtain the linearization only for the channels of interest.
- Specify multiple I/Os, possibly as different types, in the same bus.

Select individual bus elements as I/Os when you want to:

- Linearize a Simulink model using the GUI or programmatically.
- Visualize time- and frequency-domain response during simulation using the Linear Analysis Plots blocks.
- Verify frequency-domain characteristics using the Model Verification blocks.

For more information, see Select Individual Bus Elements as Linearization Points and the `linio` reference page.

Enhanced LINLFT Command Optionally Returns Linearization of Excluded Blocks

A new optional output argument to `linlft` returns the linearization of blocks you specify for exclusion from the linearized model.

For more information, see the `linlft` reference page.

Access to Current Linearization of a Simulink Block for Specifying Custom Linearization

When you use a specification function to specify a custom linearization of a Simulink block, you can now access the current linearization in your function. The current linearization is stored in the new `BlockLinearization` field of the `BlockData` structure. The software automatically creates the `BlockData` structure and passes it to your block linearization specification function.

For more information and an example showing how to use the `BlockLinearization` field of `BlockData`, see Augmenting the Linearization of a Block.

Enhanced PID Controller Blocks Display Compensator Formula in Block Dialog Box

The PID Controller and PID Controller (2DOF) blocks now display the current compensator formula in the block dialog box. This display reflects the current settings for controller type, controller form, and time domain.

R2010b

Version: 3.2

New Features

Bug Fixes

New Blocks for Plotting and Verifying Linear System Characteristics of Simulink Models

This version of Simulink Control Design software provides new blocks for:

- “Plotting Linear System Characteristics of Simulink Models” on page 26-2.
- “Verifying Linear System Characteristics of Simulink Models” on page 26-2.

Plotting Linear System Characteristics of Simulink Models

Simulink Control Design software provides six new blocks in the **Linear Analysis Plots** library. Use these blocks to plot the time- and frequency-domain characteristics of a linear system computed from a nonlinear Simulink model. The linear system is computed and plotted during simulation at:

- Simulation snapshot times. The default snapshot time is 0.
- Trigger-based simulation events.

You can also use these blocks to specify bounds on the linear system characteristics, and view the bounds on the plot.

Because these blocks are same as the Model Verification blocks except for the default settings of the bound parameters, you can optionally use the blocks to verify that the bounds are satisfied during simulation.

For more information, see:

- Visualize Bode Response of Simulink Model During Simulation
- Linear Analysis Plots block reference pages
- `Plotting Linear System Characteristics of a Chemical Reactor demo`

Verifying Linear System Characteristics of Simulink Models

Simulink Control Design software provides six new blocks in the **Model Verification** library. Use these blocks to verify that the time- and frequency-domain characteristics of a linear system, computed from a nonlinear Simulink model, satisfy specified bounds during simulation. For example, you can verify whether the linearized behavior of your model satisfies upper and lower magnitude bounds on a Bode plot or gain and phase margins.

You can perform the verification during simulation at:

- Simulation snapshot times. The default snapshot time is 0.
- Trigger-based simulation events

Because these blocks are same as the Linear Analysis Plots blocks except for the default settings of the bound parameters, you can use the blocks to view the bounds on time- and frequency-domain plots.

You can also use these blocks with the Simulink Model Verification library blocks to design complex logic for model verification.

If you have Simulink Verification and Validation™ software, you can construct simulation tests for your model using the Verification Manager.

For more information, see:

- Model Verification.
- Model Verification block reference pages
- `Verifying Frequency-Domain Characteristics of an Aircraft` demo

New Tools for Identifying Time-Varying Source Blocks for Frequency Response Estimation

When you are performing frequency response estimation, time-varying source blocks in the signal path can skew the results. Previously, to obtain accurate estimation, you manually identified source blocks and inserted open-loop points in your model to disable them.

The new `frest.findSources` command automatically detects time-varying source blocks in the signal path of the output linearization points in your Simulink model. Additionally, a new option to the `frestimate` command, `BlocksToHoldConstant`, allows you to disable these blocks during frequency response estimation.

For more information, see the `frest.findSources` and `frestimateOptions` reference pages.

There is also a new Model Advisor check for detecting time-varying source blocks in your model. For more information about using the Model Advisor, see Consulting the Model Advisor in the *Simulink User's Guide*.

Tuning Tools Update Workspace Variables That Define Parameters of Tuned Blocks

When you use Simulink Control Design tools to tune a block with parameters defined as workspace variables, the software now updates the values of the workspace variables. This update occurs when you write the compensator design to your Simulink model. The block remains parametrized by the variables. The software can update variable parameters that represent numerical values or `Simulink.Parameter` objects, whether they reside in the base workspace or the model workspace.

Previously, tuning a block parametrized by workspace variables overwrote the block parameters with numerical values. Thus, such tuning did not change the value of the workspace variables.

Enhanced PID Tuner Including New Response Plots

This release introduces several enhancements to the PID Tuner, including:

- New response plot options let you analyze system responses in either time domain (step plot) or frequency domain (Bode plot). Available responses are reference tracking, input and output disturbance rejection, controller effort, open loop, and plant.
- New options when the plant linearizes to zero at the operating point defined in the Simulink model.

  When the plant linearizes to zero on launch of the PID Tuner, the PID Tuner provides an option to linearize at a different operating point. The PID Tuner also lets you import an LTI model of your plant, such as a model obtained by frequency response estimation. Previously, the PID Tuner did not launch when the plant linearized to zero.

  For an example illustrating these options, see the new Simulink Control Design demo, `Designing PID Controller in Simulink with Estimated Frequency Response`.

For more information about using the PID Tuner, see Automatic PID Tuning in the *Simulink Control Design User's Guide*.

New Demo Illustrating Control Design for a Plant That Has Parameter Variations

The `Reference Tracking of a DC Motor with Parameter Variations` demo illustrates compensator design for a plant that has parameter variations. The demo shows the following techniques:

- Performing batch linearization to obtain multiple linear models from a single Simulink model
- Using the new SISO Design Tool ability to analyze a control design for multiple models simultaneously

R2010a

New Parallel Computing Support For Frequency Response Estimation

If you have the Parallel Computing Toolbox™ software installed, you can use parallel computing to speed up frequency response estimation.

For more information, see Speeding Up Estimation Using Parallel Computing.

New Commands Support Recomputing Frequency Response Estimation Results at Specific Frequencies

If you use the sinestream input signal for estimation, you can now recompute your frequency response estimation for only those specific frequencies that do not reach steady state. Previously, you had to redo the entire estimation if a few frequencies needed recomputing.

For more information, see Time Response Not at Steady State.

New frest.simcompare Output Argument Returns Simulation Output Data From Linear System

This release provides enhanced `frest.simCompare` functionality, which allows the return of simulated output data. If your linear model uses state-space representation, you can also return the state vector.

For more information, see the `frest.simCompare` reference page.

New Options in Simulink Results Viewer GUI for Viewing Frequency Response Estimation Results

The Simulation Results Viewer GUI, which you display using `frest.simView`, now provides additional options for analyzing your frequency response estimation:

- You can now enable phase unwrapping in the frequency response plot. To unwrap phase, right-click the Bode plot, and select **Unwrap Phase**.
- You can also import frequency response estimation results into an open Simulation Results Viewer, which replaces the existing results. Previously, you had to open a new Viewer each time you viewed estimation results. To import results, select **File** > **Import** in the Simulation Results Viewer.

New Option for Labeling Bus Signal I/O Names in the SISO Design Task

You can now configure the SISO Design Task to automatically label model I/Os using bus signal names. Model I/O labels describe available loops in the Graphical Tuning tab and label response plots in the Analysis Plots tab of the SISO Design Task GUI. Previously, labels for bus signals were based on the block and port path, or the Simulink signal name.

To use this option, select **Tools** > **Options** in the SISO Design Task GUI, and enable **Use bus signal names to label model I/O**.

Existing Simulink Blocks Now Have Analytic Jacobians

The following Simulink blocks now have Analytic Jacobians, which are optimized for memory consumption in large-scale models:

- Switch
- Multiport Switch
- Math Function (Transpose and Hermitian Transpose)

For more information on each block, see the block reference pages.

Change in Format of Time Series in frestimate Output

When you use the `simout` output of the `frestimate` command, you obtain a cell array of `Simulink.Timeseries` objects. Previously, the time dimension was always the first dimension of each `Simulink.Timeseries` object.

As of R2010a, the time dimension of each `Simulink.Timeseries` in `simout` is:

- The first dimension, if the time series is 2-D
- The last dimension, if the time series is 3-D or higher

Compatibility Considerations

If you have scripts that run `frestimate` on a model having an output point on a vector or matrix signal and perform operations on the resulting 3-D or higher-dimensional `Simulink.Timeseries` object in `simout`, modify your scripts to reflect the new placement of the time dimension in the time series.

For additional information, see:

- `frestimate` reference page
- `Simulink.Timeseries` reference page

R2009b

Version: 3.0

New Features

Bug Fixes

New GUI for Tuning New PID Controller Blocks

You can automatically tune the new Simulink PID Controller blocks in the PID Tuner GUI R2009b introduces. You launch the PID Tuner directly from the PID block dialogs boxes. These blocks are available in the Continuous and Discrete Simulink libraries.

For more information on tuning PID Controller using the PID Tuner, see Automatic PID Tuning.

New Automated PID Tuning Algorithm

You can now tune compensators using a new automated PID tuning algorithm called Robust Response Time, which is available in the SISO Design Task. You specify the open-loop bandwidth and phase margin, and the software computes PID parameters to robustly stabilize your system.

For information on tuning compensators using automated tuning methods, see Completing the Design.

Ability to Compute Frequency Response of Simulink Models

You can now compute the frequency response of a Simulink model using simulation. You use new commands to easily create input signals and simultaneously simulate and estimate the frequency response without changing your model. You can also use this capability to validate the accuracy of exact linearizations.

Ability to Specify the Linearization of Simulink Blocks

You can now specify the linearization of Simulink blocks, subsystems, and model references without having to replace any block in your model. You can specify the linearizations as LTI models or Robust Control Toolbox uncertain models.

Ability to Design Compensators for Plant Models With Time Delays

You can now design compensators for plants with exact time delay representations. Previously, you had to specify a Padé approximation before designing compensators.

For more information on designing compensators for plants with time delays, see Designing Compensators for Plants with Time Delays.

New Commands to More Efficiently Compute Multiple Linearizations

You can now more efficiently compute multiple linearizations for large models when only a few blocks or model references change per linearization. You linearize the fixed portion of the model once using `linlft` and linearize the varying portion multiple times. Then, you combine the results using `linlftfold` to obtain linearizations equal to those you would receive if you linearized the entire model multiple times.

For more information, see the `linlft` and `linlftfold` reference pages.

Ability to Set Default Plot Type for Linear Analysis Results from GUI

You can now set the default plot type for viewing linear analysis results computed in the Control and Estimation Tools Manager. This setting applies to all future Control and Estimation Tools Manager sessions.

To set this option in the Simulink Control Design Preferences, select **File > Preferences** in the Control and Estimation Tools Manager.

R2009a

Version: 2.5

New Features

Bug Fixes

Ability to Generate MATLAB Code from the GUI for Creating Operating Points and Linearizing Models

You can now generate MATLAB code reflecting the configuration in the GUI when creating operating points and linearizing models.

Ability to Tune Additional Blocks

You can now tune the following blocks:

- Blocks that you discretized using the Simulink Model Discretizer
- Blocks in the Simulink Extras library that specify initial states or outputs

For information about how to tune these blocks, see Selecting Blocks to Tune.

New Option for Labeling Bus Signal I/O Names in Linearization Results

You can now compute linear models that show the bus signal names for linearization I/O points located on buses. You can select this option using the following:

- Linearization Options GUI
- `linoptions` command

R2008b

Version: 2.4

New Features

Bug Fixes

New Upsampling Option for Rate Conversion When Linearizing Simulink Models

Version 2.4 includes an upsampling rate conversion method for linearization. This method upsamples discrete-time LTI systems at any sampling rate that is an integer-value-times faster than the sampling rate of the original system. You can select the upsampling rate conversion method in the following ways:

- Using the linearization options GUI
- From the command line using `linoptions`

For more information on the upsampling rate conversion method, see the `linoptions` reference page.

Ability to Specify State Order of Linearized Models from the Command Line

You can now specify the order of the states in your linearized model directly from the command line using the `linearize` command. Previously, you could only specify state order using the GUI.

For more information on specifying state order from the command line, see the `linearize` reference page.

Ability to Filter the Linearization Inspector to Show Blocks in the Linearization Path

You can now filter the list of blocks in the Linearization Inspector to show only the blocks in the linearization path. This filtering makes it easier for you to find blocks in the linearization path that you want to inspect.

Ability to Disable the Calculation of Linearization Diagnostics and Inspector Data in the GUI

You can now disable the calculation of the linearization inspector and diagnostics information when you linearize using the GUI. This capability allow you to choose when you want to calculate diagnostic information.

R2008a

Version: 2.3

New Features

Bug Fixes

New Diagnostic Messages Help You Troubleshoot Linearization Results

You can now view diagnostic messages for your linearized model that help you diagnose and troubleshoot linearization results. These messages identify blocks in your model that encounter the following block issues during linearization:

- Blocks that have been marked as not supported for linearization
- Blocks with linearization configuration warning messages
- Blocks without pre-programmed exact Jacobian that linearize using numerical approximation

Ability to Find Operating Points for Simscape Models

You can now find operating points for models that include Simscape and SimHydraulics® blocks using the Simulink Control Design `findop` command.

For more information on finding operating point for Simscape Models, see the Simscape documentation.

Updated Error and Warning Message System

The Simulink Control Design error and warning IDs and messages have been updated. If you use error and warning IDs in your code, you must update your code to reflect the new IDs.

R2007b

Version: 2.2

New Features

Bug Fixes

Ability to Linearize Models with Model-Reference Blocks by Any Linearization Method

You can now perform any type of linearization for models containing Model blocks that reference other Simulink models.

Previously, you could only perform numerical perturbation linearization for models containing Model blocks. Now, you can also perform block-by-block linearization for such models when you set the simulation mode of the Model blocks to Normal.

Ability to Design Compensators for Models Containing Model-Reference Blocks

You can now design compensators for tunable blocks inside external models referenced by your model. Your model references such external models by using Model blocks. You can also update the tunable blocks in the external model with the new compensator designs.

To view and select tunable blocks for compensator design from the referenced model, set the simulation mode of the Model block to Normal.

For more information about the types of blocks that you can tune, see Selecting Blocks to Tune.

Ability to Generate Linearized Models with Exact Time-Delay Representations

You can now use the Simulink Control Design software to compute linearized models with exact time-delay representations. Time delays in the original nonlinear model can result from any of the following blocks:

- Transport Delay
- Variable Time Delay
- Variable Transport Delay
- Unit Delay
- Integer Delay

Previously, you could only achieve approximate linearizations of models with continuous time delays using a Padé approximation.

Ability to Linearize Periodic Function-Call Subsystems

You can now linearize periodic function-call subsystems with a constant sample time as discrete subsystems. You must set the sample time of the function-call trigger block to equal the sample time of the function-call generator block.

For more information on function-call subsystems, see Function-Call Subsystems.

R2007a

Version: 2.1

New Features

Bug Fixes

Ability to Linearize Using an Operating Point Specified Directly in a Model

You can use the Simulink Control Design software to linearize around any operating point that you specify directly in a Simulink model. This capability allows you to make changes to a model and then perform a linearization around the newly specified operating point with the click of a single button. Previously, this same task required two steps: creating a new operating point and then linearizing around this operating point.

There are two ways to linearize around an operating point specified directly in the model:

- From the Control and Estimation Tools Manager
- Using `linearize`

Ability to Capture Linearization Snapshots in GUI

You can use the Control and Estimation Tools Manager to linearize at snapshots of your model operating point at the following simulation points:

- Specified simulation times, such as when the simulation reaches a steady state solution
- Events during a specified simulation interval

As in prior releases, you can use `linearize` to perform linearization at snapshots in your model operating point, as described in `linearize` in the reference pages.

Ability to Perform Control Design at Snapshots in GUI

You can use the Control and Estimation Tools Manager to perform control design at snapshots of your model operating point at the following simulation points:

- Specified simulation times, such as when the simulation reaches a steady state solution
- Events during a specified simulation interval

Ability to Retrieve Stored Compensator Designs

When you design a compensator using the SISO design tool, you can store the current design and then continue making changes to this design. A new button called **Retrieve Design** lets you retrieve the stored design at any time by undoing the design changes you made since you last stored the design. For more information on retrieving stored compensator designs, see Storing and Retrieving Designs.

R2006b

Version: 2.0.1

Bug Fixes

R2006a

Version: 2.0

New Features

Bug Fixes

Compensator Design in Simulink Is Now Supported

This release provides new tools to streamline the workflow for designing Single-Input Single-Output (SISO) control loops directly in Simulink. In previous releases, designing a compensator was a multistep process that involved several tools.

The new tools support any linearizable control architecture, such as single loops, multiple loops, and cascaded loops. With the new tools, you simply select the blocks you want to tune. Then, the Simulink Control Design software automatically analyzes your model to identify the relevant control loops and opens a preconfigured session of the SISO Design Tool (in the Control System Toolbox software). For more information, see “Designing Compensators” in the Simulink Control Design documentation.

Supported tunable SISO Simulink blocks include Gain, Transfer Function, Zero-Pole-Gain, State-Space, and PID blocks.

In the SISO Design Tool you can

- Graphically tune multiple SISO loops in a single GUI.
- Gain visual insight into loop interactions and coupling effects.
- Focus the analysis on a specific loop in a multiloop design by removing the effect of other feedback loops.
- Generate first-cut compensator designs using systematic design algorithms, such as Ziegler-Nichols PID tuning, IMC design, or LQG design.
- Optimize linear responses to meet time and frequency-based design constraints (requires the Simulink Response Optimization™ software).
- Directly tune Simulink block parameters, such as PID gains, zero-pole-gain representations, and masked blocks.
- Tune continuous or discrete control loops.
- Examine the closed-loop response of any portion of a model.

After a design is completed, you can write the tuned parameter values back to your model for verification with the full nonlinear system.

R14SP3

Version: 1.3

New Features

Bug Fixes

Control and Estimation Tools Manager Enhanced

You can copy and edit operating points within the Control and Estimation Tools Manager. .

You can initialize a model for simulation using operating points from within the Control and Estimation Tools Manager.

Support for Operating Point Search and Linearization of Models with Model Reference Blocks

You can linearize and compute operating points for models that reference other models using the Model block. Linearization of model reference models must use the numerical perturbation linearization algorithm. This algorithm accepts state and input perturbation values in the form of an operating point object.

R14SP2

Version: 1.2

New Features

Bug Fixes

Context-Sensitive Help Added

Access context-sensitive help for the Linearization, Operating Point Search, and Linearization State Ordering Options window of the Control and Estimation Tools Manager. To access help on a field within the options window, right-click the option's label and select **What's this?** from the context menu. Help for the option will appear in this window.

View Linearizations in the Control and Estimation Tools Manager

You can view state space, transfer function, and zero-pole gain representations of linearized models within the Control and Estimation Tools Manager without exporting to the workspace. These linearized models appear in the linearization summary pane of the Control and Estimation Tools Manager.

Discretization Methods Added

You can select from three different discretization methods for linearization of multirate and hybrid models.

List of Blocks with Preprogrammed Analytic Jacobians Added

You can view a complete list of blocks indicating which blocks have preprogrammed analytic Jacobians for use with the block-by-block analytic linearization algorithm.

Block Name Readability Improved

You can use either truncated block names or full block names in the state space matrices of a linearized model, and within the LTI Viewer, to improve readability.