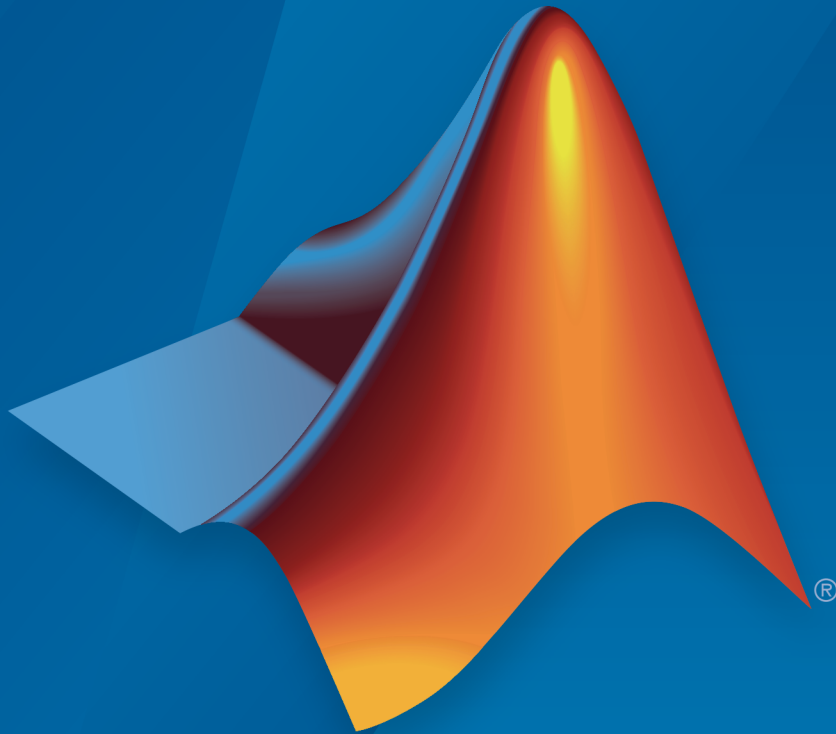


Robust Control Toolbox™ Release Notes



MATLAB®

How to Contact MathWorks



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



Phone: 508-647-7000



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

Robust Control Toolbox™ Release Notes

© COPYRIGHT 2005–2016 by The MathWorks, Inc.

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

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

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

R2016b

Improved Robustness Analysis Workflow: Calculate robustness margins using the new <code>robstab</code> and <code>robgain</code> functions	1-2
Improved Robustness Guarantees: Compute the structured singular value μ without frequency gridding	1-2
Improved Worst-Case Gain Computations	1-3
Functionality Being Removed or Changed	1-4

R2016a

Control system tuning tools moved to Control System Toolbox	2-2
Functionality being removed or changed	2-2

R2015b

Robust Tuning with <code>systune</code> Command or Control System Tuner App: Automatically tune controllers to maximize performance over a range of parameter values	3-2
---	-----

Gain Scheduling with <code>sysstune</code> and <code>sITuner</code>: Automatically tune the Lookup Table and Interpolation blocks used to model gain-scheduled controllers in Simulink	3-3
<code>tunableSurface</code> Object: Parameterize and tune gain-scheduled controllers using improved workflow	3-3
<code>getNominal</code> command for extracting nominal value of uncertain model	3-4
<code>usample</code> samples uncertain blocks and preserves other control design blocks	3-4
New property for limiting maximum frequency in random samples of <code>ultidyn</code>	3-5
Functionality being removed or changed	3-5

R2015a

Robust tuning of controller parameters against a set of plant models specified through parameter variations in Control System Tuner app	4-2
Open Control System Tuner app with saved session from command line	4-2

R2014b

Quick Loop Tuning option in Control System Tuner app for tuning control systems to target loop bandwidth and stability margins	5-2
Tuning goals for automated tuning to meet transient response and disturbance rejection requirements	5-2

MATLAB code generation from Control System Tuner app for automatically scripting control system tuning tasks	5-3
Enhanced constraints on controller dynamics for control system tuning	5-3
New syntax in <code>TuningGoal.Poles</code> for directly specifying constraints on dynamics	5-4
<code>TuningGoal.StepResp</code> renamed to <code>TuningGoal.StepTracking</code>	5-4
<code>DisturbanceInput</code> property of <code>TuningGoal.Rejection</code> renamed to <code>Location</code>	5-4
Functionality being removed or changed	5-5

R2014a

Control System Tuner app for automated tuning of control systems	6-2
Step response and LQG requirements for control system tuning with <code>systune</code> and <code>looptune</code> commands	6-2
Improvements to <code>TuningGoal</code> requirements for control system tuning	6-3
Tuning Goals for constraining dynamics impose implicit stability constraints	6-3
Option to limit dynamics constraint to poles in a particular feedback loop	6-3
<code>TuningGoal.Tracking</code> allows specification of peak error	6-4
Specification of signal scaling in MIMO closed-loop Tuning Goals	6-4
Option to remove stability constraint from loop-shape and gain-limiting Tuning Goals	6-5
<code>ScalingOrder</code> property added to <code>TuningGoal.Margins</code>	6-5

Improved control system tuning of Simulink models with sys tune or looptune functions using sITuner interface (with Simulink Control Design)	6-6
---	-----

R2013b

Automatic tuning of gain-scheduled control systems with sys tune and looptune commands	7-2
Automatic tuning of discrete-time control systems with sys tune and looptune commands	7-2
Sensitivity, overshoot, minimum and maximum loop gain requirements for control system tuning with looptune and sys tune	7-3
looptuneSetup command for switching from looptune to sys tune to use additional sys tune functionality	7-3
hinfnorm command for computing H_∞ norm	7-4
Some properties of TuningGoal requirements renamed	7-4
Power iteration method option for structured singular value computation with mussv	7-5
Option to specify feedback sign for stability margin calculation with ncfmargin	7-5

R2013a

Minimum damping requirement for closed-loop poles in TuningGoal.Poles object	8-2
---	-----

TuningGoal.Rejection object for specifying disturbance rejection requirement	8-2
looptune returns detailed results from multiple random starts	8-2
Additional automated tuning examples	8-3

R2012b

systemtune command for multiobjective tuning with soft and hard constraints	9-2
H_2 performance, stability margin, pole location, and disturbance rejection requirements	9-2
Robust tuning of one controller against a set of plant models	9-3
Option to constrain tuned parameter values and to restrict some tuning requirements to a frequency band	9-3
ltiblock.pid2 and loopswitch objects for tuning two-degree-of-freedom PID controllers and marking loop opening sites for open-loop requirements	9-4
TuningGoal.MaxGain and GainLimit property renamed ...	9-4
Options in hinfstructOptions and looptuneOptions renamed or removed	9-5

R2012a

Parallel Computing Support for looptune and hinfstruct ..	10-2
---	------

Faster and More Accurate H-infinity Norm Computation Using SLICOT Algorithms	10-2
---	-------------

R2011b

looptune Tunes Fixed-Structure Control Systems	11-2
Control System Tuning for Simulink Models with looptune or hinfstruct Using sITunable Interface	11-2
wcgainplot for Visualizing Worst-Case Gains	11-3
Functionality Being Removed or Changed	11-3

R2011a

Enhanced Workflow for H-Infinity Synthesis of Fixed- Structure Control Systems	12-2
---	-------------

R2010b

New Commands for H-Infinity Synthesis of Fixed-Structure Control Systems	13-2
---	-------------

R2010a

Bug Fixes

R2009b

New Option to Improve Robust Performance by Accounting for Real Uncertain Parameters	15-2
New Command to Linearize Simulink Models with Uncertainty	15-2
New Interface for Simulating Effects of Uncertainty in Simulink Models	15-2
New Command to Model Multiple LTI Responses as One Uncertain System	15-3
New and Updated Demos	15-3
Functions, Properties and Blocks Being Removed	15-3

R2009a

Bug Fixes	
------------------	--

R2008b

Bug Fixes	
------------------	--

R2008a

Ability to Use LOOPMARGIN with Simulink	18-2
--	-------------

R2007b

No New Features or Changes

R2007a

New Simulink Blocks 20-2

R2006b

New Function ltiarray2uss 21-2

R2006a

No New Features or Changes

R14SP3

No New Features or Changes

mussvunwrap Is Renamed	24-2
New Functions <code>actual2normalized</code> and <code>normalized2actual</code>	24-2

R2016b

Version: 6.2

New Features

Bug Fixes

Compatibility Considerations

Improved Robustness Analysis Workflow: Calculate robustness margins using the new `robstab` and `robgain` functions

New functions improve the workflow for computing robust stability margins and robust performance margins of uncertain systems. The new functions compute the worst-over-frequency margins, and can also return the margins as a function of frequency.

- `robstab` — Calculate the robust stability margin. This margin is a measure of how far the uncertain elements of a system can deviate from their nominal values before the system becomes unstable.
- `robgain` — Calculate the robust performance margin. This margin is a measure of how far the uncertain elements of a system can deviate from their nominal values before the peak gain of the system exceeds some specified value.

For uncertain state-space (`uss` and `genss`) models, both functions use a new algorithm that always finds the smallest margin across all frequencies, as described in “Improved Robustness Guarantees: Compute the structured singular value μ without frequency gridding” on page 1-2. The new functions replace and augment the functionality previously provided by `robuststab` and `robustperf`. For more information about using the new functions, see the reference pages for `robstab` and `robgain`.

Compatibility Considerations

Using `robuststab` and `robustperf` is not recommended. Instead:

- Replace instances of `robuststab` or `robustperf` with `robstab` and `robgain`, respectively.
- Replace instances of `robuststabOptions` or `robustperfOptions` with `robOptions`.

Improved Robustness Guarantees: Compute the structured singular value μ without frequency gridding

The new `robstab` and `robgain` functions base their analysis on the structured singular value, μ . For uncertain state-space (`uss` or `genss`) models, these functions use a new algorithm that is guaranteed to detect critical peaks of μ , and always produces correct guarantees of robustness.

The new algorithm adaptively selects frequencies for computing μ . The returned upper bounds on μ are guaranteed to hold over each interval between frequencies. In previous

releases, μ analysis used a grid-based computation that could miss important peaks in μ and produce over-optimistic guarantees of robustness. For `ufrd` and `genfrd` models, the computation is still performed pointwise at the frequencies specified in the model.

Improved Worst-Case Gain Computations

The `wcgain` function computes bounds on the worst-case gain of an uncertain system, `uses`. This command now uses the new structured-singular-value algorithm described in “Improved Robustness Guarantees: Compute the structured singular value μ without frequency gridding” on page 1-2 . Therefore this function is guaranteed to return accurate bounds on the worst-case gains for `uss` or `genss` models. Previously, `wcgain` used a grid-based approach that could miss important peaks and produce over-optimistic worst-case gains.

R2016b also includes a new function for visualizing worst-case gain as a function of frequency, `wcsigma`. Like `wcgain`, this function is guaranteed to produce correct worst-case gains for `uss` or `genss` models. `wcsigma` replaces and improves the functionality previously provided by `wcgainplot`.

Specify options for `wcgain` and `wcsigma` using the new options command `wcOptions`. You can also use `wcOptions` for `wcmargin`, `wcsens`, and `wcnorm`.

Compatibility Considerations

Using `wcgainplot`, `wcgainOptions`, and `wcmarginOptions` is not recommended. Instead:

- Replace instances of `wcgainplot` with `wcsigma`.
- Replace instances of `wcgainOptions` or `wcmarginOptions` with `wcOptions`.

Additionally, there are some changes to the default behavior and supported options for `wcgain`:

- The `MaxOverFrequency` option of `wcgainOptions` is now the `VaryFrequency` option of `wcOptions`. To compute the worst-case gain as a function of frequency, use `wcOptions('VaryFrequency','on')`.
- In the `info` output of `wcgain`, when `VaryFrequency` is `'off'`, the field `Info.Frequency` now contains the frequency at which the worst-case peak gain occurs. Previously, `Info.Frequency` contained a vector of all frequencies used for analysis, even when `MaxOverFrequency` was `'on'`.

- In `wcOptions`, the option `'VaryFrequency' = 'on'` is not available for arrays of uncertain models.
- By default, the `Sensitivity` option of `wcOptions` is `'off'`. Previously, the default value of the `Sensitivity` option of `wcgainOptions` was `'on'`. Therefore, to compute the sensitivity of the worst-case gain to each uncertain element, use `wcOptions('Sensitivity','on')`.
- The `MaxOverArray` option of `wcgainOptions` no longer exists in `wcOptions`. Instead, when you provide an array of uncertain models, `wcgain` always returns the worst case gain over the entire array. To compute the worst-case gain individually for each model in an array, use a `for` loop to step through each array entry. For example, suppose that `uarray` is an array of N uncertain models. The following code computes the worst-case gain for each entry in `uarray`.

```
for k = 1:N
    [wgc(k),wcu(k)] = wcgain(uarray(:,:,k));
end
```

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
<ul style="list-style-type: none"> • <code>robuststab</code> and <code>robuststabOptions</code> • <code>robustperf</code> and <code>robustperfOptions</code> 	Still runs	<code>robstab</code> , <code>robgain</code> , and <code>robOptions</code>	Replace instances of <code>robuststab</code> or <code>robustperf</code> with <code>robstab</code> and <code>robgain</code> , respectively. See “Improved Robustness Analysis Workflow: Calculate robustness margins using the new <code>robstab</code> and <code>robgain</code> functions” on page 1-2.
<code>wcgainplot</code>	Still runs	<code>wcsigma</code>	Replace instances of <code>wcgainplot</code> with <code>wcsigma</code> .

Functionality	Result	Use Instead	Compatibility Considerations
			See “Improved Worst-Case Gain Computations” on page 1-3.
<ul style="list-style-type: none"> • <code>wcgainOptions</code> and <code>wcmarginOptions</code> 	Still runs	<code>wcOptions</code>	Replace instances of <code>wcgainOptions</code> or <code>wcmarginOptions</code> with <code>wcOptions</code> . For more details on the differences between the old options commands and <code>wcOptions</code> , see “Improved Worst-Case Gain Computations” on page 1-3.

R2016a

Version: 6.1

New Features

Bug Fixes

Compatibility Considerations

Control system tuning tools moved to Control System Toolbox

A Robust Control Toolbox™ license is no longer required to use the `systeme` or `looptune` commands or to use Control System Tuner. You can now:

- Tune control systems modeled in MATLAB® (tunable `genss` models) with a Control System Toolbox™ license.
- Tune control systems modeled in Simulink® with a Simulink Control Design™ license.

The following still requires a Robust Control Toolbox license:

- `hinfstruct` command
- Robust tuning of control systems with parameter uncertainty using `systeme`, `looptune`, or Control System Tuner

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
a, b, c, d, and e properties of <code>uss</code> models.	Still works	A, B, C, D, and E respectively.	If your code uses any of these properties, consider modifying your code to use the new property names.
<code>cpmargin</code>	Still works	<code>ncfmargin</code>	If your code uses <code>cpmargin</code> , modify it to use <code>ncfmargin</code> instead.

R2015b

Version: 6.0

New Features

Bug Fixes

Compatibility Considerations

Robust Tuning with `systemtune` Command or Control System Tuner App: Automatically tune controllers to maximize performance over a range of parameter values

Control System Tuner and the `systemtune` command now tune control systems for robustness against real parameter uncertainty in the plant. You represent parameter uncertainty in your control system model using uncertain real parameters `ureal` or `uss`. The software automatically finds the worst combinations of parameter values and tunes the controller to maximize performance over the parameter uncertainty range.

In MATLAB, build a generalized state-space (`genss`) model of your control system using `ureal` or `uss` blocks to represent real parameter uncertainty in the plant. You can tune the model with `systemtune` or in Control System Tuner exactly as you would for a tunable control system model without uncertainty. For a detailed example, see [Robust Tuning of Positioning System](#).

In Simulink, use linearization with block substitution to replace one more blocks in the model with uncertain values represented by `ureal` or `uss` objects. (Requires Simulink Control Design software.) See [Robust Tuning of Mass-Spring-Damper System](#).

In both cases, when you tune the model, the software automatically adjusts the tunable components to achieve the specified performance as well as possible throughout the uncertainty range. Analysis plots automatically display random samples of the uncertain system to give you a visual sense of the performance variation.

For more information about robust tuning generally, see [Robust Tuning Approaches](#).

Compatibility Considerations

Previously, when you used `systemtune` to tune a model that had uncertainties, the software would set the uncertain blocks to their nominal values before tuning the system. Now, `systemtune` tunes the model for robustness against those uncertainties. To recover the old behavior, i.e., to tune a controller for the nominal system only, use `getNominal` to obtain the nominal value. For example:

```
[CL,fSoft,GHard,info] = systemtune(getNominal(CLO),SoftReqs,HardReqs);
```

In this example, `CLO` is a `genss` model containing uncertain blocks.

Gain Scheduling with `system` and `sITuner`: Automatically tune the Lookup Table and Interpolation blocks used to model gain-scheduled controllers in Simulink

You can now use the `sITuner` interface to automatically tune control systems modeled in Simulink in which plant dynamics change with operating conditions or time. (Requires Simulink Control Design software.)

In such gain-scheduled control systems, the controller gains vary as a function of one or more scheduling variables. In the Simulink model, use the Lookup Table or Interpolation blocks to implement the variable controller gains. You then use the new `tunableSurface` command to parameterize the dependency of these gains on the scheduling variables. The software automatically tunes the coefficients of that parameterization so that the control system meets the tuning requirements you specify over the entire grid of scheduling-variable values. The software also writes the tuned coefficients back to the Lookup Table or Interpolation blocks.

In previous releases, you could not parameterize Lookup Table or Interpolation blocks in terms of the functional form of its dependence on the scheduling variable. As a result, you could not automatically tune a gain-scheduled control element and write the tuned coefficients back to the Simulink model. Using `system` to tune and implement gain-scheduled controllers required a complex process of manually extracting coefficient values and inserting them in the blocks.

For more details, see [Set Up Simulink Models for Gain Scheduling](#).

For examples showing how to use `tunableSurface` to tune gain-scheduled controllers implemented with Lookup Table blocks, see:

- [Gain-Scheduled Control of a Chemical Reactor](#)
- [Tuning of Gain-Scheduled Three-Loop Autopilot](#)

`tunableSurface` Object: Parameterize and tune gain-scheduled controllers using improved workflow

The new `tunableSurface` object lets you express gain in terms of tunable parameters for tuning gain-scheduled controllers with `system`. In such gain-scheduled control systems, the controller gains vary as a function of one or more scheduling variables. You parameterize the dependency of controller gains on the scheduling variables. The software automatically tunes the coefficients of that parameterization so that the control

system meets the tuning requirements you specify over the entire grid of scheduling-variable values. `tunableSurface` replaces the `gainsurf` command.

In previous releases, you could use the `gainsurf` command to represent tunable surfaces for control system tuning. With that command, you had to explicitly supply the values of the gain surface calculated over the grid of design points. `tunableSurface` simplifies that workflow by allowing you to specify the gain surface in terms of functions of the scheduling variables, such as the basis functions of a polynomial expansion.

For more details about creating tunable gain surfaces, see:

- Parametric Gain Surfaces
- `tunableSurface` reference page

Compatibility Considerations

`tunableSurface` replaces `gainsurf`, which was used in previous releases to parameterize controller gains as functions of scheduling variables. `gainsurf` still works, but might be removed in a future release. If you have scripts or functions that use `gainsurf`, consider updating them to use `tunableSurface` instead.

`getNominal` command for extracting nominal value of uncertain model

Use `getNominal` to replace the uncertain elements of a generalized model with their nominal values. All other control design blocks in the generalized model are unchanged. For example, suppose that `M` is a generalized state-space (`genss`) model that has both uncertain blocks and tunable blocks. The command `getNominal(M)` returns a `genss` model having the same tunable blocks as `M`.

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

`usample` samples uncertain blocks and preserves other control design blocks

The `usample` command now preserves any non-uncertain control design blocks when you use it to sample the uncertain elements of a generalized model. For example, suppose that `M` is a generalized state-space (`genss`) model that has both uncertain blocks and tunable blocks. The command `usample(M,N)` samples the uncertain blocks, and returns an array of `genss` models having the same tunable blocks as `M`.

Compatibility Considerations

Previously, when applied to models having tunable control design blocks, `usample` used the current (nominal) value of those blocks, and returned an array of numeric models. To recover the previous behavior, use `getValue`. For example, the following command randomly samples the uncertain blocks of `M`, replaces the tunable blocks of `M` with their current values, and returns an array of numeric state-space models.

```
Msamp = getValue(usample(M,N));
```

New property for limiting maximum frequency in random samples of `ultidyn`

Use the `SampleMaxFrequency` property of `ultidyn` to limit the natural frequency of dynamics when you take random samples of `ultidyn` blocks. For example, the following command creates SISO uncertain dynamics.

```
dH = ultidyn('dH',[1 1],'SampleMaxFrequency',1);
```

When you take random samples of `dH`, such as with `usample`, the dynamics of the samples are no faster than 1 rad/s. The default value of `SampleMaxFrequency` is `Inf` (no limit).

Also, the `SampleStateDim` property of `ultidyn` is changed to `SampleStateDimension`.

Compatibility Considerations

The property name `SampleStateDim` still works, but might be removed in a later release. If you have scripts or functions that use `SampleStateDim`, consider updating them to use `SampleStateDimension` instead.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
<code>system(CLO,...)</code> where <code>CLO</code> contains uncertain blocks	Tunes robustly against real parameter uncertainty in <code>CLO</code>	<code>system(getNominal(CLO))</code>	Previously, <code>system</code> used the nominal value of all uncertain blocks in the tuned

Functionality	Result	Use This Instead	Compatibility Considerations
			model. Now, use <code>getNominal</code> explicitly to tune for the nominal system only. See “Robust Tuning with <code>systemtune</code> Command or Control System Tuner App: Automatically tune controllers to maximize performance over a range of parameter values” on page 3-2.
<code>gainsurf</code>	Still works	<code>tunableSurface</code>	If you have scripts or functions that use <code>gainsurf</code> , consider updating them to use <code>tunableSurface</code> instead. See “ <code>tunableSurface</code> Object: Parameterize and tune gain-scheduled controllers using improved workflow” on page 3-3

Functionality	Result	Use This Instead	Compatibility Considerations
usample(M,N)	Samples uncertain control design blocks of M, and preserves other control design blocks	getValue(usample(Previously, usample used the current value of non-uncertain control design blocks. See “usample samples uncertain blocks and preserves other control design blocks” on page 3-4
SampleStateDim property of ultidyn	Still works	SampleStateDimens	Consider replacing SampleStateDim with SampleStateDimension.

R2015a

Version: 5.3

New Features

Bug Fixes

Robust tuning of controller parameters against a set of plant models specified through parameter variations in Control System Tuner app


When you use Control System Tuner to tune a Simulink model of a control system, you can now generate multiple plant models by varying model parameters. You can then tune the control system to satisfy your specified tuning goals for all the resulting models.

Tuning to multiple models is useful to help ensure that the tuned control system is robust against parameter variations or changes in operating conditions. For example, if a parameter in your Simulink model represents a process temperature, you can generate multiple models spanning the range of expected temperature variations, and tune your control system to meet your design requirements for all those models at once.

For more information about tuning control systems for multiple models in Control System Tuner, see [Robust Tuning Using Multiple Plant Models in Control System Tuner](#). For an example showing how to specify parameter variations for tuning with Control System Tuner, see [Tuning Control System with Multiple Valued Plant Parameters using Control System Tuner](#).

Open Control System Tuner app with saved session from command line

Use the new syntax `controlSystemTuner(sessionfile)` to open Control System Tuner and load data from a saved session. When you use Control System Tuner, you

can click  **Save Session** to save session data to disk such as tuning goals you have created, response I/Os you have defined, operating points, and stored designs. The string `sessionfile` is the name of a session data file saved in the current working directory or on the MATLAB path. The software also opens the Simulink model associated with the saved session.

R2014b

Version: 5.2

New Features

Bug Fixes

Compatibility Considerations

Quick Loop Tuning option in Control System Tuner app for tuning control systems to target loop bandwidth and stability margins

Quick Loop Tuning lets you use a loop-shaping approach to tune SISO or MIMO feedback loops in Control System Tuner. You can use Quick Loop Tuning to tune control systems modeled in MATLAB or Simulink. With Quick Loop Tuning you can tune your system to meet target gain crossover and margin requirements without explicitly creating tuning goals that capture these requirements. You specify feedback loops to tune by selecting the control signals and measurement signals in a block diagram of your control system. Control System Tuner adjusts the tunable parameters of your system such that the open-loop gain crossover falls within the desired frequency range with the gain and phase margins you specify.

For more information about using Quick Loop Tuning, see [Quick Loop Tuning of Feedback Loops in Control System Tuner](#).

Tuning goals for automated tuning to meet transient response and disturbance rejection requirements

New tuning goals let you explicitly specify a target transient response or a minimum disturbance rejection in a tuned control system. These tuning goals are available both in Control System Tuner and at the command line when tuning with `systune`.

The transient response goal lets you shape how the closed-loop system responds to a specific input signal. You specify the desired transient response as a reference model. The target transient response is the response of the reference model to an impulse, step, ramp, or custom input signal. To use the transient response goal:

- In Control System Tuner, in the **Tuning** tab, in the **New Goal** menu, select **Transient Response Matching**.
- At the command line, specify the design requirement using `TuningGoal.Transient`.

The step rejection goal lets you specify a minimum standard for rejecting disturbances. You specify characteristics such as the maximum amplitude and settling time of the response at some point in your control system to a step disturbance injected at another point in the system. Alternatively, specify a reference system whose response to step input is the target response. To use the step rejection goal:

- In Control System Tuner, in the **Tuning** tab, in the **New Goal** menu, select **Rejection of Step Disturbances**.

-
- At the command line, specify the design requirement using `TuningGoal.StepRejection`.

MATLAB code generation from Control System Tuner app for automatically scripting control system tuning tasks

You can now generate a MATLAB script for control system tuning from Control System Tuner. Generated MATLAB scripts are useful when you want to programmatically reproduce a result you obtained interactively. You can also use generated code to perform multiple tuning operations with systematic variations in tuning configurations such as model operating point or tuning goals.

For more information, see [Generate MATLAB Code from Control System Tuner for Command-Line Tuning](#).

Enhanced constraints on controller dynamics for control system tuning

New functionality gives you more flexibility when specifying constraints on controller dynamics for control system tuning. The following new features are available in both Control System Tuner using **Controller Poles Goal** and when tuning at the command line using `TuningGoal.ControllerPoles` (formerly `TuningGoal.StableController`).

- You can now specify a minimum damping constant for the poles of a tunable block. Previously, the damping constant of controller poles could take any value between zero and 1.
- You can now specify a negative value for the minimum decay rate of controller poles, allowing for unstable controllers. Previously, the minimum decay rate had to be positive, and therefore always enforced the stability of the constrained block.
- Fixed integrators in the constrained tunable block are no longer considered when evaluating the constraint. In other words, the tuning goal now constrains locations of all poles in the block except fixed integrators, such as the I term in a PID controller.

For more information about these features, see:

- [Controller Poles Goal](#), for tuning in Control System Tuner.
- [The `TuningGoal.ControllerPoles` reference page](#), for tuning at the command line.

Compatibility Considerations

`TuningGoal.StableController` has been renamed to `TuningGoal.ControllerPoles`. Scripts and functions that use `TuningGoal.StableController` do not generate errors. However, `TuningGoal.StableController` will not be maintained in future releases. You should replace instances of `TuningGoal.StableController` in your code with `TuningGoal.ControllerPoles`.

New syntax in `TuningGoal.Poles` for directly specifying constraints on dynamics

When you use `TuningGoal.Poles` to constrain the dynamics of a tuned control system, you can now directly specify the minimum decay rate, minimum damping, and maximum natural frequency when you create the tuning goal. To do so, use the following syntaxes:

```
R = TuningGoal.Poles(MinDecay,MinDamping,MaxFreq);  
R = TuningGoal.Poles(Location,MinDecay,MinDamping,MaxFreq);
```

Previously, to specify such constraints on controller dynamics, you had to first create the tuning goal, and then modify its `MinDecay`, `MinDamping`, and `MaxFrequency` properties.

For more information, enter see the `TuningGoal.Poles` reference page.

`TuningGoal.StepResp` renamed to `TuningGoal.StepTracking`

The tuning requirement `TuningGoal.StepResp` is now called `TuningGoal.StepTracking`.

Compatibility Considerations

Scripts and functions that use `TuningGoal.StepResp` do not generate errors. However, `TuningGoal.StepResp` will not be maintained in future releases. You should replace instances of `TuningGoal.StepResp` in your code with `TuningGoal.StepTracking`.

`DisturbanceInput` property of `TuningGoal.Rejection` renamed to `Location`

The `DisturbanceInput` property of the tuning requirement `TuningGoal.Rejection` is now called `Location`, to unify the names of similar properties of several tuning

requirements. If `Req` is a `TuningGoal.Rejection` requirement, you can access this property using `Req.Location`.

Compatibility Considerations

Scripts and functions that use the `DisturbanceInput` property do not generate errors. However, the `DisturbanceInput` property will not be maintained in future releases. You should replace instances of `DisturbanceInput` in your code with `Location`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>TuningGoal.Stable</code>	Still works	<code>TuningGoal.ControllerPoles</code>	Consider replacing <code>TuningGoal.StableControllerPoles</code> with <code>TuningGoal.ControllerPoles</code> .
<code>TuningGoal.StepResponse</code>	Still works	<code>TuningGoal.StepTracking</code>	Consider replacing <code>TuningGoal.StepResponse</code> with <code>TuningGoal.StepTracking</code> .
<code>DisturbanceInput</code> property of <code>TuningGoal.Rejection</code>	Still works	<code>Location</code> property	Consider replacing <code>DisturbanceInput</code> with <code>Location</code> .

R2014a

Version: 5.1

New Features

Bug Fixes

Compatibility Considerations

Control System Tuner app for automated tuning of control systems

The new Control System Tuner lets you interactively tune SISO or MIMO control systems modeled in MATLAB or Simulink. Control System Tuner tunes the control system parameters to meet design requirements you specify, such as reference tracking, disturbance rejection, stability margins, loops shapes, and sensitivity. You can examine multiple system responses in both the time and frequency domains to evaluate performance of the tuned control system.

If you have Simulink Control Design software, you can tune a control system represented by a Simulink model. Control System Tuner can tune most blocks used to create a control system in Simulink. These blocks include **Gain**, **PID Controller**, **Transfer Fcn**, **State-Space**, **Zero-Pole**, **Discrete Filter**, and the **LTI System** block. Any controller architecture created using these blocks can be tuned. To access Control System Tuner for tuning a Simulink model, select **Analysis > Control Design > Control System Tuner**.

Control System Tuner can also tune a control system represented by a tunable **genss** model. Any control architecture constructed with Control Design Blocks such as `ltiblock.pid`, `ltiblock.tf`, or `realp` blocks can be tuned. To open Control System Tuner for tuning a control system modeled in MATLAB, use the `controlSystemTuner` command.

For more information about using Control System Tuner, see:

- Automated Tuning Basics
- Tuning with Control System Tuner

Step response and LQG requirements for control system tuning with `systeme` and `looptune` commands

New `TuningGoal` requirement objects allow you to specify tuning objectives for automated tuning of control systems with `systeme` and `looptune`.

- `TuningGoal.StepResp` — Requires that the step response between specified locations in the control system match the step response of a specified reference system. For details about this requirement, see the `TuningGoal.StepResp` reference page.

- `TuningGoal.LQG` — Specifies a linear-quadratic-gaussian (LQG) goal for control system tuning. This requirement lets you quantify control performance as an LQG cost. For details about this requirement, see the `TuningGoal.LQG` reference page.

Improvements to `TuningGoal` requirements for control system tuning

This release introduces a variety of improvements to `TuningGoal` requirement objects for automated tuning of fixed-structure control systems with `systemtune` and `looptune`.

Tuning Goals for constraining dynamics impose implicit stability constraints

`TuningGoal.StableController` and `TuningGoal.Poles` now impose implicit stability constraints on controller or system dynamics. This allows you to require poles of the controller or the closed-loop control system to be stable, without necessarily limiting the minimum decay or maximum frequency of those poles. Previously, you had to specify finite values for minimum decay and maximum frequency when using these tuning goals.

Compatibility Considerations

The default values of the `MinDecay` and `MaxFrequency` properties of these requirements have changed. If you have scripts that use `TuningGoal.StableController` or `TuningGoal.Poles` requirements with default values, update those scripts to explicitly set the finite values you want.

Property	Previous Default Value	New Default Value
<code>TuningGoal.Poles.MinDecay</code>	1e-6	0
<code>TuningGoal.StableController.MinDecay</code>	1e-6	0
<code>TuningGoal.Poles.MaxFrequency</code>	1e6	Inf
<code>TuningGoal.StableController.MaxFrequency</code>	1e6	Inf
<code>TuningGoal.Poles.MinDamping</code>	1e-6	0

Option to limit dynamics constraint to poles in a particular feedback loop

A new syntax for creating the `TuningGoal.Poles` requirement allows you to constrain only the poles of the sensitivity function measured at a specified location. Use this syntax to narrow the scope of the requirement to a particular feedback loop.

For example, suppose you have a cascaded-loop control system in which the inner and outer loops contain loop-opening locations 'InnerLoop' and 'OuterLoop', respectively. The following command uses the new syntax to constrain the poles of the inner loop sensitivity function:

```
Req = TuningGoal.Poles('InnerLoop');  
Req.MinDamping = 0.5;  
Req.Openings = 'OuterLoop';
```

Req imposes a minimum damping on the poles of the inner loop sensitivity function measured with the outer loop open. The dynamics of blocks that do not participate to the inner loop are ignored.

For more information about using this constraint, see the `TuningGoal.Poles` reference page.

TuningGoal.Tracking allows specification of peak error

A new syntax for creating the `TuningGoal.Tracking` requirement allows you to specify a maximum tracking error for a particular input-output pair in terms of a response time, a relative DC error, and a peak relative error across all frequencies. These parameters are converted to the following expression for the maximum tracking error:

$$\text{MaxError} = \frac{(\text{PeakError})s + \omega_c (\text{DCError})}{s + \omega_c}.$$

For more information about how to specify tracking error requirements, see the `TuningGoal.Tracking` reference page.

Specification of signal scaling in MIMO closed-loop Tuning Goals

New properties in several closed-loop Tuning Goals allow you to specify the relative amplitudes of multiple input and output signals in the loops constrained by the requirements. Use these properties to reduce cross-coupling in tuned systems when the choice of units results in a mix of small and large signals.

- `TuningGoal.Tracking` and `TuningGoal.Overshoot` now have an `InputScaling` property. This information is used to scale the off-diagonal terms in the transfer function from reference to tracking error. This scaling ensures that cross-couplings are measured relative to the amplitude of each reference signal.

-
- `TuningGoal.Gain` and `TuningGoal.Variance` now have `InputScaling` and `OutputScaling` properties. The values you set for these properties are used to scale the closed-loop transfer function $T(s)$ on which you impose the tuning requirement. The requirement is evaluated for the scaled transfer function $D_o^{-1}T(s)D_i$. D_o and D_i are diagonal matrices formed from the `OutputScaling` and `InputScaling` property, respectively.

For more information on how to interpret and use these properties, see the reference pages for the Tuning Goals.

Option to remove stability constraint from loop-shape and gain-limiting Tuning Goals

The new `Stabilize` property of loop-shaping and gain-limiting Tuning Goals allows you turn off the implicit closed-loop stability constraint. If stability for the specified loop is not required or cannot be achieved, set `Stabilize` to `false` to relax the stability constraint.

This property is available for the following Tuning Goals:

- `TuningGoal.LoopShape`
- `TuningGoal.Gain`, `TuningGoal.WeightedGain`
- `TuningGoal.MinLoopGain`, `TuningGoal.MaxLoopGain`

For more information on how to use the `Stabilize` property, see the reference pages for the Tuning Goals.

ScalingOrder property added to TuningGoal.Margins

The `TuningGoal.Margins` tuning goal has a new property, `ScalingOrder`. This property controls the number of states in the diagonal scalings involved in computing MIMO stability margins. Increasing the order may improve results at the expense of increased computations.

Previously, this scaling order was set as a tuning option in `systuneOptions`.

Compatibility Considerations

If you have scripts that use the `ScalingOrder` option of `systuneOptions`, set the `ScalingOrder` property of `TuningGoal.Margins` instead.

Improved control system tuning of Simulink models with `systemtune` or `looptune` functions using `sITuner` interface (with Simulink Control Design)

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

- Tune model blocks and subsystems to meet tuning goals using the `systemtune` and `looptune` functions.
- Perform robust tuning of a controller against a set of plant models using `systemtune`. You can configure an `sITuner` interface to vary model parameter values and operating points. When you call `systemtune` for the interface, the software returns a control system 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.

`sITuner`, similar in design to `sLinearizer`, 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 `sITuner`, see:

- Programmatic Control System Tuning
Loop-Shaping Design

Compatibility Considerations

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

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

R2013b

Version: 5.0

New Features

Bug Fixes

Compatibility Considerations

Automatic tuning of gain-scheduled control systems with `systemtune` and `looptune` commands

You can now use `systemtune` and `looptune` to automatically tune control systems in which plant dynamics change with operating conditions or time. In such gain-scheduled control systems, the controller gains vary as a function of one or more scheduling variables. You parameterize the dependency of controller gains on the scheduling variables. The software automatically tunes the coefficients of that parameterization so that the control system meets the tuning requirements you specify over the entire range of plant operating conditions. The new `gainsurf` command helps you parameterize your controller gains as functions of scheduling variables.

Several new examples illustrating the workflow for gain-scheduled tuning, including:

- Tuning of Gain-Scheduled Three-Loop Autopilot
- Gain-Scheduled Control of a Chemical Reactor

For additional information about tuning gain-scheduled controllers, see [Gain-Scheduled Controllers](#).

Automatic tuning of discrete-time control systems with `systemtune` and `looptune` commands

You can now use `systemtune` and `looptune` for automatic tuning of discrete-time control systems. This capability includes both:

- Control systems represented by discrete-time generalized LTI models (`genss` models with `Ts` property not equal to zero).
- Control systems represented by an `s1Tunable` interface to a Simulink mode. Set the `Ts` property of the `s1Tunable` interface to the sampling time at which you want to linearize the model.

To tune a discrete-time control system, use the same procedure and command syntax and you use to tune a continuous-time control system. For examples of discrete-time tuning, see:

- Digital Control of Power Stage Voltage
- MIMO Control of Diesel Engine

Sensitivity, overshoot, minimum and maximum loop gain requirements for control system tuning with looptune and systune

New `TuningGoal` requirement objects allow you to specify a variety of tuning objectives for automated tuning of fixed-structure control systems with `systune` and `looptune`. New tuning requirements include:

- `TuningGoal.Sensitivity` — Constraint on sensitivity to disturbance
- `TuningGoal.Overshoot` — Constraint on overshoot in step response
- `TuningGoal.MinLoopGain` — Minimum loop gain constraint
- `TuningGoal.MaxLoopGain` — Maximum loop gain constraint

Additionally, `TuningGoal.LoopShape` has two new syntaxes. These syntaxes allow you to specify a target crossover frequency or range of crossover frequencies for an open-loop response in your control system.

For more information about these `TuningGoal` requirement objects see the reference pages for each requirement object, and:

- Using Design Requirement Objects
- Specifying Design Requirements for `systune`
- Performance and Robustness Specifications for `looptune`

looptuneSetup command for switching from looptune to systune to use additional systune functionality

The new `looptuneSetup` command provides a bridge between the tuning commands `looptune` and `systune`. `looptuneSetup` takes the argument list for `looptune` and constructs an equivalent argument list for `systune`. The `looptuneSetup` command is valid for systems represented in either MATLAB or Simulink.

You can use this command to switch from `looptune` to `systune` to take advantage of the additional flexibility and functionality of `systune`. For example, `looptune` requires that you tune all channels of a MIMO feedback loop to the same target bandwidth. Converting to `systune` allows you to specify different crossover frequencies and loop shapes for each loop in your control system. Also, `looptune` treats all tuning requirements as soft requirements, optimizing them but not requiring that any constraint be exactly met. Converting to `systune` allows you to enforce some tuning requirements as hard constraints, while treating others as soft requirements.

You can also use `looptuneSetup` to probe into the tuning requirements that `looptune` implicitly imposes. When you use `looptune`, you specify a target loop bandwidth and stability margins. `looptune` expresses these as hard and soft tuning constraints, specified as `TuningGoal` objects. You can use `looptuneSetup` to examine these constraints. After examining the constraints, you can then alter them and pass them to `systune` for further tuning.

For more information, see the following reference pages:

- `looptuneSetup`
- `slTunable.looptuneSetup`

hinfnorm command for computing H_∞ norm

The new `hinfnorm` command computes the H_∞ norm of SISO or MIMO systems. For SISO systems, the H_∞ norm is defined as the largest value of the frequency response magnitude. For MIMO systems, H_∞ norm is the largest singular value across frequencies.

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

Some properties of `TuningGoal` requirements renamed

The following properties of `TuningGoal` requirement objects are renamed to better reflect their purpose and uses:

Object	Previous Property Name	New Property Name
<code>TuningGoal.LoopShape</code>	<code>LoopTransfer</code>	<code>Location</code>
<code>TuningGoal.Margins</code>	<code>LoopTransfer</code>	<code>Location</code>
<code>TuningGoal.Tracking</code>	<code>ReferenceInput</code>	<code>Input</code>
<code>TuningGoal.Tracking</code>	<code>TrackingOutput</code>	<code>Output</code>

Compatibility Considerations

If you have scripts or functions that use any of these properties, consider updating your code to use the new property names instead. Using the previous property names does not generate an error in this release, but the names might be removed in a future release.

Power iteration method option for structured singular value computation with `mussv`

A new 'p' option to the `mussv` command allows you to specify a power iteration method for computing the lower bound on structured singular values (μ values). This method is recommended for cases of complex uncertainty. When at least one of the uncertain blocks specified in the block diagonal matrix structure is complex, `mussv` now uses the power iteration method by default.

For pure real uncertainty, `mussv` uses a gain-based lower bound algorithm by default.

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

Compatibility Considerations

Previously, `mussv` used a gain-based lower bound algorithm for both pure real and mixed uncertainty. Therefore, you might now obtain different results for the lower bounds with mixed uncertainty.

Option to specify feedback sign for stability margin calculation with `ncfmargin`

The `ncfmargin` command includes a new input argument that lets you specify the sign of the feedback interconnection assumed for the margin calculation. Use the syntax `[marg, freq] = ncfmargin(P,C,sign)` or `[marg, freq] = ncfmargin(P,C,sign,tol)` to specify a negative or positive feedback interconnection. For more information, see the `ncfmargin` reference page.

Compatibility Considerations

Previously, the relative accuracy `tol` was the third input argument to `ncfmargin`. If you have scripts or functions that use the syntax `[marg, freq] = ncfmargin(P,C,tol)`, update them to use `[marg, freq] = ncfmargin(P,C,-1,tol)` instead.

R2013a

Version: 4.3

New Features

Bug Fixes

Compatibility Considerations

Minimum damping requirement for closed-loop poles in `TuningGoal.Poles` object

You can now specify the minimum damping ratio of closed-loop poles for automated tuning of fixed-structure control systems with `systemtune` or `looptune`. To do so, create a `TuningGoal.Poles` object and set its `MinDamping` property to the minimum damping ratio you want to specify. Additionally, you can now use the `FOCUS` property to limit enforcement of the `TuningGoal.Poles` requirements to poles within a specified frequency range.

For more information about the `TuningGoal.Poles` requirement, see the `TuningGoal.Poles` reference page. For more information about using requirement objects to tune control systems, see [Using Design Requirement Objects](#).

`TuningGoal.Rejection` object for specifying disturbance rejection requirement

You can now specify a disturbance rejection requirement for automated tuning of fixed-structure control systems with `systemtune` or `looptune`. The new `TuningGoal.Rejection` object allows you to specify a frequency-dependent attenuation factor for a disturbance injected at a specified location in the control system.

For more information about the `TuningGoal.Rejection` requirement, see the `TuningGoal.Rejection` reference page. For an example, see [PID Tuning for Setpoint Tracking vs. Disturbance Rejection](#).

For more information about using requirement objects to tune control systems generally, see [Using Design Requirement Objects](#).

`looptune` returns detailed results from multiple random starts

The `info` output of `looptune` now includes detailed results from each optimization run. When you use the `RandomStart` option of `looptuneOptions` to perform multiple optimization runs, the field `info.Runs` of the `info` output now contains a `struct` array. Each entry in the `struct` array includes results from the corresponding optimization run such as minimum constraint values and tuned block values. You can optionally use this information to analyze independent optimization results.

See the `looptune` reference page for more information.

Compatibility Considerations

The `Extra` field of `info` is now renamed to `Runs`. If you use `info.Extra` in a script, update your code to use `info.Runs` instead.

Additional automated tuning examples

New examples in this release include:

- Multi-Loop Control of a Helicopter
- Fault-Tolerant Control of a Passenger Jet
- Multi-Loop PID Control of a Robot Arm

R2012b

Version: 4.2

New Features

Bug Fixes

Compatibility Considerations

systune command for multiobjective tuning with soft and hard constraints

The new `systune` command allows automated tuning of fixed-structure control systems to high-level tuning objectives.

To use `systune`, you specify tuning objectives such as reference tracking, disturbance rejection, or stability margins. You can specify both soft requirements (objectives) and hard requirements (constraints). `systune` automatically tunes the parameters of your control system to meet the requirements.

You can use `systune` to tune control systems modeled in either MATLAB or Simulink.

For more information, see:

- [Tuning Control Systems with SYSTUNE](#)
- [Tuning Control Systems in Simulink](#)
- [Automated Tuning](#)
- [The `systune` reference page](#)

H₂ performance, stability margin, pole location, and disturbance rejection requirements

New `TuningGoal` requirement objects allow you to specify a variety of tuning objectives for automated tuning of fixed-structure control systems with `systune` and `looptune`.

New tuning requirements include:

- `TuningGoal.Margins` — Tune to stability margin requirements by specifying minimum gain and phase margins for any feedback loop in your control system.
- `TuningGoal.Poles` — Constrain closed-loop dynamics of your control system.
- `TuningGoal.StableController` — Constrain dynamics or ensure stability of tunable elements.
- `TuningGoal.WeightedGain` — Limit on frequency-weighted gain from specified inputs to specified outputs in your control system.
- `TuningGoal.Variance` and `TuningGoal.WeightedVariance` — Tune to H₂ performance requirements by minimizing or constraining variance amplification. `TuningGoal.Variance` specifies the maximum output variance for a unit-

variance input signal from a specified input to a specified output in your control system. `TuningGoal.WeightedVariance` imposes a frequency-weighted variance amplification limit.

For more information about these `TuningGoal` requirement objects see the reference pages for each requirement object, and:

- Using Design Requirement Objects
- Specifying Design Requirements for `systune`
- Performance and Robustness Specifications for `looptune`

Robust tuning of one controller against a set of plant models

The new `systune` command can simultaneously tune the parameters of multiple models or control configurations. This feature allows you, for example, to tune a single controller against a range of plant models, to help ensure that the tuned control system is robust against parameter variations. As another example, you can tune for reliable control by simultaneously to multiple plant configurations that represent different failure modes of a system. In either case, `systune` finds values for tunable parameters that best satisfy the specified tuning objectives for all models.

For more information, see [Tune Controller Against Set of Plant Models](#).

Option to constrain tuned parameter values and to restrict some tuning requirements to a frequency band

You can now optionally impose lower and upper bounds on tunable parameters when tuning fixed-structure control systems using `systune`, `looptune`, or `hinfstruct`. For example, you can constrain a gain to always be positive, or impose a maximum value on a filter time constant.

To impose bounds on tunable parameters, set the `Maximum` and `Minimum` properties of the parameter in the corresponding Control Design Block. For example, create a scalar gain block and constrain the gain to be positive:

```
gainblock = ltiblock.gain('gainblock',1,1);  
gainblock.Gain.Minimum = 0;
```

Then, use `gainblock` as a component in a tunable `genss` model of the control system. When you tune the control system, the tuning command enforces the constraint.

Additionally, you can limit the range of frequencies in which almost any `TuningGoal` requirement is enforced for fixed-structure control system tuning with `systemtune` or `looptune`. The only exceptions are `TuningGoal.Variance` and `TuningGoal.WeightedVariance`.

For example, you can enforce a stability margin requirement in a frequency band extending for one decade on each side of the target gain crossover frequency.

To limit the range of frequencies in which a requirement is enforced, use the `Focus` property of the `TuningGoal` requirement object. For example, create a requirement that limits the gain from an input `du` to an output `u` to 10. Limit enforcement of the requirement to the frequency range 10–1000 rad/s.

```
Req = TuningGoal.Gain('du','u',10);  
Req.Focus = [10 1000];
```

ltiblock.pid2 and loopswitch objects for tuning two-degree-of-freedom PID controllers and marking loop opening sites for open-loop requirements

New Control Design Blocks in Control System Toolbox allow you to specify more control structures and more types of constraints for fixed-structure control system tuning in MATLAB:

- `ltiblock.pid2` — Tunable two-degree-of-freedom PID controller
- `loopswitch` — Control Design Block for specifying feedback loop opening locations in a tunable `genss` model of a control system

For more information, see the `ltiblock.pid2` and `loopswitch` reference pages.

TuningGoal.MaxGain and GainLimit property renamed

The tuning requirement `TuningGoal.MaxGain` is now called `TuningGoal.Gain`. Additionally, the `GainLimit` property of that tuning requirement is now called `MaxGain`.

For more information, see the `TuningGoal.Gain` reference page.

Compatibility Considerations

Replace instances of `TuningGoal.MaxGain` in your code with `TuningGoal.Gain`. Replace references to the `GainLimit` property with `MaxGain`.

Options in `hinfstructOptions` and `looptuneOptions` renamed or removed

The following options in `hinfstructOptions` and `looptuneOptions` are changed:

- `SpecRadius` is now called `MaxFrequency`. Additionally, NaN is no longer a supported value for this option. For an unconstrained `MaxFrequency` value, use `Inf`.
- `StableOffset` is now called `MinDecay`.
- `StableRadius` option has no effect.
- `StableExclude` option of `hinfstructOptions` has no effect. `hinfstruct` now automatically excludes from stability tests Control Design Blocks such as weighting functions or multipliers. These blocks do not affect the closed-loop stability of the actual control system to tune.

For more information about these options, see the `hinfstructOptions` and `looptuneOptions` reference pages.

Compatibility Considerations

If you use any of the affected options in your code, update your code to reflect the current names and supported values.

R2012a

Version: 4.1

New Features

Bug Fixes

Parallel Computing Support for `looptune` and `hinfstruct`

If you have Parallel Computing Toolbox™ software installed, you can use parallel computing to speed up tuning of fixed-structure control systems with the `looptune` or `hinfstruct` commands. When you run multiple randomized `looptune` or `hinfstruct` optimization starts, parallel computing speeds up tuning by distributing the optimization runs among MATLAB workers.

For more information about using parallel computing to speed up `looptune` or `hinfstruct` tuning, see:

- Speed Up Tuning with Parallel Computing Toolbox Software in the Robust Control Toolbox documentation.
- The Robust Control Toolbox demo Using Parallel Computing to Accelerate the Tuning Process.

For more information about tuning fixed-structure control systems with `looptune` or `hinfstruct`, see Tuning Fixed Control Architectures in the Robust Control Toolbox documentation.

Faster and More Accurate H-infinity Norm Computation Using SLICOT Algorithms

H_∞ norm calculations now use the SLICOT library of numerical algorithms. These algorithms improve the speed and accuracy of functions such as `hinfstruct` and `looptune`.

For more information about the SLICOT library, see <http://slicot.org>.

R2011b

Version: 4.0

New Features

Bug Fixes

Compatibility Considerations

looptune Tunes Fixed-Structure Control Systems

Use `looptune` to tune fixed-structure control systems to meet your requirements. To use `looptune`, specify design requirements such as loop bandwidth, stability margin, setpoint tracking, or target loop shape. `looptune` automatically tunes the parameters of your controller to meet the specified requirements.

The requirements objects `TuningGoal.MaxGain`, `TuningGoal.Tracking`, and `TuningGoal.LoopShape` let you express your design requirements directly. You do not have to first convert them to weighting functions or mathematical constraints on an optimization problem.

You can use `loopview` to validate the performance the performance of the tuned control structure against your specified design requirements.

For more information, see [Tuning Fixed Control Architectures](#) and the `looptune` and `loopview` reference pages.

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

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

Use `sITunable` to specify information about your control structure and parametrization. `sITunable` 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 `sITunable` object for your control architecture, you can tune the control system using `sITunable.looptune` or `hinfstruct`.

For more information, see [Tuning Fixed Control Architectures](#) and the following demos:

- [Tuning of a Digital Motion Control System](#)
- [Decoupling Controller for a Distillation Column](#)
- [Tuning of a Two-Loop Autopilot](#)
- [Tuning of Cascaded PID Loops](#)
- [Loop Shaping Design with HINFSTRUCT](#)
- [Fixed-Structure Autopilot for a Passenger Jet](#)

wcgainplot for Visualizing Worst-Case Gains

wcgainplot plots the nominal, sampled, and worst-case gains of uncertain systems as a function of frequency. Use wcgainplot for visual analysis of uncertain systems.

For more information, see the wcgainplot reference page.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
umat object can no longer contain ultidyn or udyn uncertainty.	<ul style="list-style-type: none">• Presence of ultidyn or udyn uncertain elements forces model type to uss or ufrd rather than umat.• Mixing ureal or ucomplex models with udyn or ultidyn objects produces uss instead of umat.	Expect a model type of uss or ufrd instead of umat when working with udyn or ultidyn uncertain elements.	Update code to work with uss or ufrd instead of umat when udyn or ultidyn elements are present.
uss(sys_frd), where sys_frd is a frd model object no longer converts sys_frd to ufrd.	Errors.	ufrd(sys_frd).	Replace uss(sys_frd) with ufrd(sys_frd).
ufrd(umat, freq, ...) no longer constructs an uncertain frd model from the umat object umat.	Converts umat to a ufrd object with frequencies freq.	Use frd(umat, freq, ...) to construct an uncertain frd model from the umat object umat.	Replace ufrd(umat, freq, ...) with frd(umat, freq, ...).
frd(sys_uss, w) where sys_uss is a uss model.	Warns; returns frd model containing data based on nominal response of sys_uss.	ufrd(sys_uss, w) to obtain a ufrd model.	Replace frd(sys_uss, w) with ufrd(sys_uss, w).

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Nominal value of <code>ultidyn</code> object.	Nominal value is <code>ss</code> model object.	None.	Update code to work with <code>ss</code> model objects when working nominal value of <code>ultidyn</code> .
<code>usubs</code> .	Applied to array of uncertain models, default substitution is <code>'-once'</code> .	Use <code>'-batch'</code> to perform batch substitution on uncertain model arrays.	Replace <code>usubs(...)</code> with <code>usubs(..., '-batch')</code> .
	<code>usubs(M, {a1;a2;...}, {v1;v2;...})</code> returns error.	<code>usubs(M, a1, v1, a2, v2, ...)</code> .	Replace <code>usubs(M, {a1;a2;...}, {v1;v2;...})</code> with <code>usubs(M, a1, v1, a2, v2, ...)</code> .
<code>usample(sys, 'a', na, 'b', nb)</code> where uncertain element <code>b</code> does not exist in <code>sys</code> .	Returns <code>na</code> -by- <code>nb</code> array with constant values across <code>nb</code> dimension, instead of <code>na</code> -by-1 array.	None.	Update code to reflect correct dimensionality.
<code>wcgopt</code> .	Still runs.	<code>wcgainOptions</code> or <code>wcmarginOptions</code> .	Replace <code>wcgopt</code> with <code>wcgainOptions</code> or <code>wcmarginOptions</code> .
<code>robuststab</code> and <code>robustperf</code> .	For <code>ufrd</code> models, <code>BadUncertainValues</code> field of <code>Info</code> output returns <code>Nf</code> -by-1 struct array, where <code>Nf</code> is the number of frequency points.	None.	Update code to work with <code>Nf</code> -by-1 struct array for <code>BadUncertainValues</code> instead of <code>Nf</code> -by-1 cell array.
	For nominally unstable models, performance margin is zero (instead of a negative value).	None.	Update code to reflect correct performance margin .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>robopt</code> .	Still runs.	<code>robuststabOptions</code> or <code>robustperfOptions</code> .	Replace <code>robopt</code> with <code>robuststabOptions</code> or <code>robustperfOptions</code> .
<code>actual2normalized</code> .	First output argument is normalized uncertain block value. The second output argument is normalized distance between block value and nominal value.	<code>[NV,ndist] = actual2normalized(BLK,AV)</code> .	Use second output argument <code>ndist</code> for normalized distance.
<code>reshape(unc_sys,S)</code> .	<code>S</code> does not include the I/O size of the models in the array <code>unc_sys</code> . For example, if <code>unc_sys</code> is a 6-by-1 array of 2-output, 4-input models, <code>reshape(unc_sys,[2 3])</code> converts <code>unc_sys</code> to a 2-by-3 array.	None.	Remove I/O size dimensions from <code>reshape</code> on uncertain model arrays.
<code>diag(uss_sys)</code> where <code>uss_sys</code> is a <code>uss</code> model.	Errors.	None.	Remove <code>diag(uss_sys)</code> .

R2011a

Version: 3.6

New Features

Bug Fixes

Enhanced Workflow for H-Infinity Synthesis of Fixed-Structure Control Systems

New Generalized LTI models in Control System Toolbox allow you to model control systems with tunable parameters. Using these models simplifies controller tuning with `hinfstruct`. You can model a closed-loop transfer function, including tunable parameters, as a generalized state-space (`genss`) model and directly tune the parameters to minimize the closed-loop gain. The `hinfstruct` command can tune any fixed-structure SISO or MIMO control system using H_∞ synthesis techniques.

Additionally, new `realp` and `genmat` objects let you create parametric expressions. You can use such expressions to create custom tunable components. For example, you can define a low-pass filter parametrized by its cutoff frequency, or an observer-based controller parametrized by the state-feedback and observer gains.

For more information about creating tunable Generalized LTI models, see *Models with Tunable Coefficients* in the *Control System Toolbox User's Guide*.

For more information about H_∞ tuning with `hinfstruct`, see *Tuning Fixed Control Architectures* in the *Robust Control Toolbox Getting Started Guide*.

For examples of designing controllers for several different architectures using `hinfstruct`, see the following updated and new demos:

- Loop Shaping Design with HINFSTRUCT (updated)
- Tuning of a Two-Loop Autopilot (updated)
- Decoupling Controller for a Distillation Column (updated)
- Multi-Loop PID Control of a Robot Arm (updated)
- Fixed-Structure Autopilot for a Passenger Jet (new)

R2010b

Version: 3.5

New Features

Bug Fixes

New Commands for H-Infinity Synthesis of Fixed-Structure Control Systems

New commands in this release allow you to tune fixed-structure SISO and MIMO control systems using the techniques of H_∞ synthesis.

The new `hinfstruct` command lets you use the frequency-domain methods of H_∞ synthesis to tune control systems with a broad range of architectures and controller structures. For example, you can tune:

- Fixed-order, fixed-structure controllers, such as pure gains, PID controllers, or fixed-order transfer function or state-space models
- Single feedback-loop architectures with multiple tunable elements, such as a PID controller plus a filter
- Multiple feedback-loop architectures with multiple tunable elements

Specify the tunable elements of your system using the new parametrized Control Design blocks `ltiblock.gain`, `ltiblock.pid`, `ltiblock.tf`, and `ltiblock.ss`.

For examples of designing controllers for several different architectures using `hinfstruct`, see the following new demos:

- Loop Shaping Design with HINFSTRUCT
- Tuning of a Fixed-Structure Autopilot
- Decoupling Controller for a Distillation Column
- Multi-Loop PID Control of a Robot Arm

For more information, see Tuning Fixed Control Architectures in the *Robust Control Toolbox Getting Started Guide*.

R2010a

Version: 3.4.1

Bug Fixes

R2009b

Version: 3.4

New Features

Bug Fixes

Compatibility Considerations

New Option to Improve Robust Performance by Accounting for Real Uncertain Parameters

You can now improve robust performance by accounting for real uncertain parameters when designing controllers using μ -synthesis. The user-defined options you use in the `dksyn` command now includes a new option `MixedMU`. Set this option to 'on' to account for real uncertain parameters in your system. For more information, see the `dkitopt`, and `dksyn` reference pages.

New Command to Linearize Simulink Models with Uncertainty

If you have Simulink Control Design software installed, you can take model uncertainty into account when linearizing a Simulink model. You can then use the resulting uncertain linearized model (`uss` object) to perform linear analysis and robust control design.

If your model already contains Uncertain State Space blocks, use the new `ulinearize` command to obtain an `uss` model. If you want to account for uncertainty in your linear analysis without using Uncertain State Space blocks, you can specify individual Simulink blocks to linearize to an uncertain variable. For more information, see "Computing Uncertain State-Space Models from Simulink Models" in the *Robust Control Toolbox User's Guide*.

New Interface for Simulating Effects of Uncertainty in Simulink Models

This version of the product provides a new interface to simulate the effects of uncertainty in Simulink models. The interface includes the following:

- Uncertain State Space block to specify uncertain system in Simulink. You should replace USS System blocks in your existing models with the Uncertain State Space block. To do so, run the `slupdate` command on your models.
- `ufind` command to extract all uncertain variables from a Simulink model.
- `usample` command to generate random values of these uncertain variables.

For more information on simulating the effects of uncertainty using the new interface, see "Simulating Effects of Uncertainty" in the *Robust Control Toolbox User's Guide*.

New Command to Model Multiple LTI Responses as One Uncertain System

This version of the product includes a new `ucover` command that lets you model a family of LTI responses as one uncertain system. For more information, see the `ucover` reference page.

New and Updated Demos

The following new and updated demos illustrate use of the new features:

- "Control of Spring-Mass-Damper Using Mixed μ -Synthesis" shows use of the new `MixedMU` option and `dksyn` command for mixed- μ synthesis.
- "Linearization of Simulink Models with Uncertainty" shows how to compute uncertain state-space models using `ulinearize` and Simulink Control Design software.
- "Robustness Analysis in Simulink" uses the new interface for simulating effects of uncertainty in Simulink models.
- "Simultaneous Stabilization Using Robust Control" and "Modeling a Family of Responses as an Uncertain System" show use of the `ucover` command.
- "First-Cut Robust Design" shows use of the `usample`, `ucover` and `dksyn` commands.

To access the demos, type

```
demo('toolbox','robust control')
```

Functions, Properties and Blocks Being Removed

Function, Property or Block Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
<code>usiminfo</code>	Still runs	<code>ufind</code>	See "New Interface for Simulating Effects of Uncertainty in Simulink Models" on page 15-2.
<code>usimfill</code>	Still runs	<code>ufind</code>	See "New Interface for Simulating Effects of Uncertainty in Simulink Models" on page 15-2.

Function, Property or Block Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
usimsamp	Still runs	usample	See “New Interface for Simulating Effects of Uncertainty in Simulink Models” on page 15-2.
USS System block	Still runs	Uncertain State Space block	See “New Interface for Simulating Effects of Uncertainty in Simulink Models” on page 15-2.
ltiarray2uss	Still runs	ucover	See “New Command to Model Multiple LTI Responses as One Uncertain System” on page 15-3.

R2009a

Version: 3.3.3

Bug Fixes

R2008b

Version: 3.3.2

Bug Fixes

R2008a

Version: 3.3.1

New Features

Ability to Use LOOPMARGIN with Simulink

This version of Robust Control Toolbox software lets you analyze the robustness of nonlinear Simulink models using the LOOPMARGIN command.

If you have the Simulink Control Design product installed, you can perform stability margin analysis of a Simulink model by passing the model name and a point within that model to the LOOPMARGIN command.

R2007b

Version: 3.3

No New Features or Changes

R2007a

Version: 3.2

New Features

New Simulink Blocks

- **USS System** — This Robust Control Toolbox version introduces a new Simulink block, USS System. You can use this block to import uncertain systems into Simulink models.
- **Multiplot Graph** — Plot multiple signals in one figure.

R2006b

Version: 3.1.1

New Features

New Function `ltiarray2uss`

This Robust Control Toolbox version introduces a new function, `ltiarray2uss`. This function constructs an uncertain state-space model from an LTI array.

R2006a

Version: 3.1

No New Features or Changes

R14SP3

Version: 3.0.2

No New Features or Changes

R14SP2

Version: 3.0.1

New Features

mussvunwrap Is Renamed

mussvunwrap has been renamed. It is now called mussvextract.

New Functions `actual2normalized` and `normalized2actual`

This Robust Control Toolbox version introduced two new functions:

- `actual2normalized` — Calculate normalized distance between nominal value and given value for uncertain atom.
- `normalized2actual` — Convert value for atom in normalized coordinates to corresponding actual value.