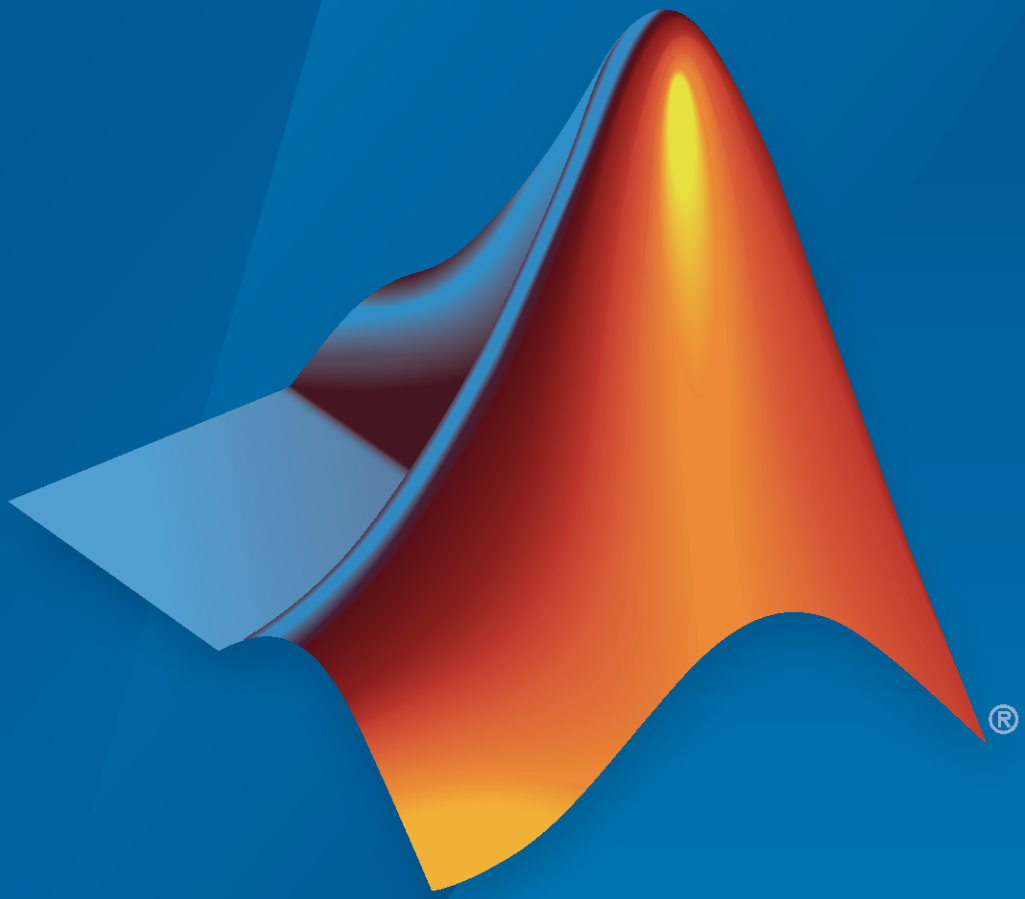
Control System Toolbox™ Release Notes



MATLAB®

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

Control System Toolbox™ Release Notes

Contents

R2026a

R2025b

R2025a

R2023b

R2023a

R2022b

R2022a

R2021b

R2021a

R2026a

Improved Workflow for Physical Assembly: Use new functions to define interfaces and create physical couplings

You can now use the new functions `addInterface` and `assemble` to create physical assembly of components. These functions replace the `interface` function and can handle more general situations when working in non-nodal (general) coordinates and allow you to create physical assemblies for `ss` and `sparss` models. The notion of coupling for `mechss` models remains the same as with `interface`, as you can still define the coupling interface using the index of interface nodes using the `addInterface` function.

For examples, see “Assemble Parts of System Using Coupling Interfaces”, and the `addInterface` and `assemble` function reference pages.

Compatibility Considerations

As a result of this change, `interface` is not recommended.

Model Reducer App: Support for Frequency Response Fitting and Zero-Pole Truncation methods

You can now obtain reduced-order models using the **Frequency Response Fitting** and **Zero-Pole Truncation** methods in the Model Reducer app.

- **Frequency Response Fitting** — Fit low-order models to obtain low-order approximations of sparse LTI models in the frequency band of interest. The frequency response fitting method is applicable to all types of sparse models with fewer limitations than Balanced Truncation or POD. In particular, it is effective at reducing models with unstable or undamped poles. Frequency response fitting is also applicable to non-sparse models but is usually less effective than Balanced Truncation for such models. Additionally, this method uses the AAA interpolation algorithm rather than a least-squared fitting algorithm. Accordingly, you must not use this method to fit models to noisy data as the algorithm will tend to interpolate noise and produce poor results.
- **Zero-Pole Truncation** — Obtain low-order zero-pole-gain approximations of sparse state-space models. This method computes a subset of the zeros and poles of sparse models, typically in a specific low-frequency band [0 f_{max}]. It can yield better low-frequency approximations than modal truncation at the expense of more computation. This method also provides direct control over the roll-off slope past the frequency f_{max}. Because this method calculates zeros for each input-output pair, it is most suitable for models with small input-output sizes. Additionally, this method is applicable only to models with a valid `sparss` representation.

Reduce Model Order Live Task: Support for multiple frequency and time intervals of focus for balanced truncation

You can now specify multiple time and frequency intervals to limit the analysis of state contributions to those intervals when using the Balanced Truncation method. Previously, Reduce Model Order Live Editor task supported only a single frequency interval and no time intervals.

modalsep and modalreal: Support for sparse models

You can now use `modalsep` and `modalreal` to compute a truncated modal form of sparse state-space models. This is helpful when you want to quickly obtain a truncated modal form of sparse models. For

more flexibility, it is recommended you use `reducespec` to first obtain a reduced-order model, then apply `modalsep` and `modalreal` to the reduced model.

⚠lyapchol and dlyapchol: Support for LR-ADI algorithm for solving equations with sparse matrices

`lyapchol` and `dlyapchol` now use the low-rank alternating directions implicit (LR-ADI) algorithm to compute a low-rank Cholesky factorization when both `A` and `E` are sparse matrices.

⚠Compatibility Considerations

Previously, the function converted *A* and *E* to full matrices before computing the factorization.

lyap and dlyap: New option to disable automatic scaling

`lyap` and `dlyap` now support disabling automatic scaling of matrices. Use the new `Scaling` input argument to enable or disable scaling.

```
lyap(__,Scaling="off")
dlyap(__,Scaling="off")
```

augdelay: Append internal delay signals to outputs of state-space model

Use the `augdelay` function to append the internal delay signals as extra outputs of the model. This is helpful when you want to monitor the contribution of these internal signals.

⚠★Linear Analysis Plots: Specify response properties using name-value arguments

When creating linear analysis plots, you can now specify response properties using name-value arguments. Any properties that you specify using name-value arguments override other arguments that set the same property. For example, the following command sets the plot color to red.

```
stepplot(sys,Color=[1 0 0])
```

⚠★Compatibility Considerations

As a result of this change, response properties that were once subproperties of `Responses.SourceData` are now subproperties of `Responses`. For more information, see “Chart object subproperties of Responses.SourceData property are now subproperties of Responses property” on page 1-8.

Linear Analysis Plots: Control automatic styling of responses

For linear analysis plots, you can now control the automatic color, line style, and marker style of responses in the plot.

This table shows the color, line style, and marker style properties for response chart objects, such as `stepplot` and `bodeplot`.

New Chart Object Property	Description
`ColorOrder`	Color order for multiple response plots (not supported for HSV plots)
`LineStyleOrder`	Line style order for multiple response plots (not supported for `pzplot`, `iopzplot`, or HSV plots)
`MarkerStyleOrder`	Marker style order for multiple response plots (not supported for `pzplot`, `iopzplot`, or HSV plots)
`ColorOrderMode`	Indicates whether the property value of `ColorOrder`, `LineStyleOrder`, or `MarkerStyleOrder` is user-specified (`"manual"`) or automatically specified by the software (`"auto"`).
`LineStyleOrderMode`	
`MarkerStyleOrderMode`	
`NextSeriesIndex`	Series index for the next response that you add to the chart. Once the software assigns this series index to a response, it increments `NextSeriesIndex` by one.

The `Responses` property of each chart object contains one response object for each response in a chart. This table shows the new style properties for response objects.

New Response Object Property	Description
`ColorMode`	Indicates whether the property value of `Color`, `LineStyle`, or `MarkerStyle` is user-specified (`"manual"`) or automatically specified by the software (`"auto"`).
`LineStyleMode`	
`MarkerStyleMode`	
`SeriesIndex`	Series index, which determines the color, line style, and marker style to select from the corresponding order property of the chart object. When you add a response to a chart, the software sets its series index to the value of the `NextSeriesIndex` property of the chart object.
`SeriesIndexMode`	Indicates whether the property value of `SeriesIndex` is user-specified (`"manual"`) or automatically specified by the software (`"auto"`).

Linear Analysis Plots: Time and frequency unit mode properties

Linear analysis chart objects now have `TimeUnitMode` and `FrequencyUnitMode` properties that indicate the specification modes of the `TimeUnit` and `FrequencyUnit` properties.

If you do not specify a time or frequency unit, the corresponding `TimeUnitMode` or `FrequencyUnitMode` property value is `"auto"`. In this case, the plot uses the unit of the first response being plotted.

If you specify a time or frequency unit, the corresponding `TimeUnitMode` or `FrequencyUnitMode` property value is `"manual"`.

⚠Linear Analysis Plots: Configure legends using Legend property

For all linear analysis plots, configure legends using the `Legend` property of the chart object. Also, you can now configure legend properties using name-value arguments with the `legend` function.

You can now configure the following legend properties:

- `Axes`
- `Location`
- `AxesMode`
- `Orientation`
- `Visible`
- `Interpreter`
- `FontSize`
- `FontWeight`
- `FontAngle`
- `FontName`

⚠Compatibility Considerations

The following chart object properties are now subproperties of the `Legend` property and their names have changed. The previous property names still work, but using them is not recommended.

Previous Property	New Property
`LegendAxes`	`Legend.Axes`
`LegendAxesMode`	`Legend.AxesMode`
`LegendVisible`	`Legend.Visible`
`LegendLocation`	`Legend.Location`
`LegendOrientation`	`Legend.Orientation`

Linear Analysis Plots: Customize plot settings using Property Inspector

For all linear analysis plots, you can now configure the plot settings using the Property Inspector.

Control System Toolbox settings moved to MATLAB Settings dialog box

Control System Toolbox settings have been moved to the MATLAB® Settings dialog box.

The `ctrlpref` function now opens the **Control System Toolbox** section of the MATLAB Settings dialog box.

⚠ Linear Analysis Plots with Custom Grids: Specify grid styling with new and updated properties

For linear analysis plots with custom grids, you can now specify grid styling using the following new or updated chart object properties.

Chart Objects	Property	Description of Change
• PZPlot • IOPZPlot • RLocusPlot	`AxesStyle.GridType`	• Includes the new `"cartesian"` grid type. • No longer includes the `"default"` grid type.
	`AxesStyle.GridTypeMode`	New property that indicates whether the `AxesStyle.GridType` property is user-specified (`"manual"`) or automatically specified by the software (`"auto"`).
• NicholsPlot • NyquistPlot	`AxesStyle.GridType`	New property for setting the grid type of Nichols and Nyquist plots to one of these values: • `"n-grid"` — Use a grid with contours for constant gain and phase values. • `"cartesian"` — Use a Cartesian grid.
	`AxesStyle.GridTypeMode`	New property that indicates whether the `AxesStyle.GridType` property is user-specified (`"manual"`) or automatically specified by the software (`"auto"`).
	`AxesStyle.GridMagnitudeSpec`	New property for specifying gain values of grid contours when `AxesStyle.GridType` is `"n-grid"`
	`AxesStyle.GridPhaseSpec`	New property for specifying phase values of grid contours when `AxesStyle.GridType` is `"n-grid"`

Impulse Response: Support for sparse index-2 DAEs

You can now use `impulse` and `impulseplot` to compute the impulse response of sparse index-2 DAEs.

As a result of this change, proper orthogonal decomposition (`ProperOrthogonalDecomposition`) now supports impulse-based excitation of sparse index-2 DAEs.

stepinfo: Support for arrays of LTI systems

Using the `stepinfo` function, you can now compute step-response characteristics for an array of LTI systems.

ProperOrthogonalDecompositionOptions: Support for parallel execution with UseParallel option

You can now use the `UseParallel` option when you want to use a parallel pool for parallel execution when using `getrom (pod)`.

This table shows how to specify the setting depending on your goal.

Goal	Setting
Write code that runs on the MATLAB client and uses built-in multithreading to make best use of the local resources.	`R.Options.UseParallel="off"` (default)
Write portable code that runs on a parallel pool and, if a pool is not available, runs on the MATLAB client.	`R.Options.UseParallel="auto"`
Write code that runs on a parallel pool and errors if a pool is not available.	`R.Options.UseParallel="on"`

⚠Enhanced control over parallel execution

You now have more control over when to use a parallel pool to run the functions that support parallel execution.

- `zpk`
- `tf`
- `systune`
- `looptune`
- `getrom (zpk)`

The `UseParallel` argument of these functions (or their corresponding option set) now accepts new `"off"`, `"auto"` or `"on"` values. Specify `UseParallel` as `"auto"` to automatically use a parallel pool if one is available or as `"on"` to always use a parallel pool.

⚠Compatibility Considerations

Starting in R2026a, specifying the `UseParallel` as `true` or `false` is not recommended. For more information, see “Logical values for UseParallel argument not recommended” on page 1-10.

Models with Internal Delays: Simulation now based on approximate c2d discretization

Simulating state-space models with internal delays is now based on approximate `c2d` discretization.

In general, when you simulate models with internal delays:

- Accuracy increases as the time step size decreases. The accuracy may deteriorate for large time step sizes.
- Simulations with zero-order hold (ZOH) and first-order hold (FOH) approximate internal signals $w(t)$ by piecewise-constant or piecewise-linear signals.

- To gauge whether the selected time step size is small enough, you can compare `sys` and `c2d(sys,t(2)-t(1),method)` in frequency domain.

For an example, see “Validate Simulation Results for Models with Internal Delays”.

c2d: Returns matching initial condition for models with internal delays

The output argument `G` of `c2d` now has extra columns corresponding to $w[0]$ initial conditions for models with internal delays.

Additionally, the mapping of the discrete states is given by:

$$x_d[k] = G\begin{bmatrix} x_c(kT_s) \\ u((k-N_u)T_s) \\ w(kT_s) \end{bmatrix}.$$

The matrix `G` is useful to map continuous-time initial conditions to equivalent discrete-time initial conditions. Note that this expression depends on the input u delayed by the discrete input delays N_u.

Sparse Balanced Truncation: Support for automatic scaling

Use the new `Scaling` option of `SparseBalancedTruncationOptions` to enable or disable automatic scaling. Scaling can improve conditioning and accuracy of LR-ADI iterations by compressing the numerical range. This option uses `equilibrate` for scaling the sparse matrices.

Functionality being removed or changed

Specifying axes object as parent of linear analysis plot not recommended
Still runs

Specifying an `Axes` object or a `UIAxes` object as the parent container of a linear analysis plot is not recommended. Instead, specify the parent container as one of these objects:

- `Figure`
- `TiledChartLayout`
- `UIFigure`
- `UIGridLayout`
- `UIPanel`
- `UITab`

Chart object subproperties of Responses.SourceData property are now subproperties of Responses property
Behavior change

For all linear analysis plot chart objects, subproperties under the `Responses.SourceData` property are now direct subproperties of the `Responses` property. The `Responses.SourceData` property has been removed.

Peak response characteristic properties of frequency-domain plots renamed
Behavior change

For frequency-domain linear analysis plots, the peak response characteristic property names have changed. The behavior of the properties remains the same.

Chart Object	Old Property Name	New Property Name
BodePlot	`Characteristics.FrequencyPeakResponse`	`Characteristics.PeakResponse`
NyquistPlot		
NicholsPlot		
SigmaPlot	`Characteristics.SigmaPeakResponse`	

Specify multiline titles and labels in linear analysis plots using string arrays
Behavior change

When creating linear analysis plots, you can now specify multiline titles and axis labels using string arrays or cell arrays of character vectors.

```
stepplot(sys)
title(["first line";"second line"]);
```

Before R2026a, specify multiline titles and labels using a `newline` character.

```
stepplot(sys)
title("first line" + newline + "second line");
```

interface not recommended
Still runs

The `interface` function is not recommended. To perform assembly of components, use the new commands `addInterface` and `assemble`. These new functions handle more general situations when working with non-nodal (general) coordinates. The notion of physical coupling remains the same as for `interface`.

lyapchol and dlyapchol no longer converts sparse inputs to full
Behavior change

`lyapchol` and `dlyapchol` now support sparse inputs A and E, and use LR-ADI algorithm to compute a low-rank factorization. Previously, the function converted A and E to full matrices before computing the factorization. To get the old behavior, you can call `lyapchol` and `dlyapchol` with `full(A)` instead of A as input.

Linear Simulation Tool no longer supports importing input signals from files
Behavior change

The Linear Simulation Tool no longer supports importing signal values from the following files.

- Import from a MAT file
- Microsoft® Excel® spreadsheet.
- Comma-separated variable file
- Text file

Starting in R2026a, to import data from files, first import the data to the MATLAB workspace. You can then import the data into Linear Simulation Tool.

For more information on importing data from files, see “Supported File Formats for Import and Export”.

Logical values for UseParallel argument not recommended
Still runs

Starting in R2026a, for the following functions, specifying the `UseParallel` argument as `true` or `false` is not recommended. Use `"off"`, `"auto"`, or `"on"` values instead.

- `zpk`
- `tf`
- `systuneOptions`
- `looptuneOptions`
- `SparseZeroPoleTruncationOptions`

This table shows how to update your code depending on your goal.

Goal	Not recommended	Recommended
Write code that runs on the MATLAB client and uses built-in multithreading to make best use of the local resources.	`UseParallel=false`	`UseParallel="off"` (default)
Write portable code that runs on a parallel pool and, if a pool is not available, runs on the MATLAB client.	`UseParallel=true`	`UseParallel="auto"`
Write code that runs on a parallel pool and errors if a pool is not available.	N/A	`UseParallel="on"`

R2025b

Version: 25.2

Bug Fixes

Quality and stability improvements

R2025b delivers quality and stability improvements, building on the new features introduced in R2025a.

R2025a

★Model Order Reduction: Fit low-order models to frequency-response data of large sparse models using reducespec

You can now use `reducespec` to fit low-order models to obtain low-order approximations of sparse LTI models in the frequency band of interest. The frequency response fitting method is applicable to all types of sparse models with fewer limitations than Balanced Truncation or POD. In particular, it is effective at reducing models with unstable or undamped poles. Frequency response fitting is also applicable to non-sparse models but is usually less effective than Balanced Truncation for such models. Additionally, this method uses the AAA interpolation algorithm rather than a least-squared fitting algorithm. Accordingly, you must not use this method to fit models to noisy data as the algorithm will tend to interpolate noise and produce poor results.

To perform model order reduction using frequency-response fitting, use the model reduction workflow (see Task-Based Model Order Reduction Workflow). For examples, see `FrequencyResponseFitting` and `getrom (frfit)`.

Model Order Reduction: Obtain truncated zero-pole-gain approximation of large sparse models using reducespec

You can now use `reducespec` to obtain low-order zero-pole-gain approximations of sparse state-space models. This method computes a subset of the zeros and poles of sparse models, typically in a specific low-frequency band [0 f_{max}]. It can yield better low-frequency approximations than modal truncation at the expense of more computation. This method also provides direct control over the roll-off slope past the frequency f_{max}. Because this method calculates zeros for each input-output pair, it is most suitable for models with small input-output sizes. Additionally, this method is applicable only to models with a valid `sparss` representation.

To perform model order reduction using zero-pole truncation, use the model reduction workflow (see Task-Based Model Order Reduction Workflow). For examples, see `SparseZeroPoleTruncation` and `getrom (zpk)`.

Sparse Model Conversion: Obtain truncated zpk and tf representations of sparse models

You can now use `tf` and `zpk` to convert sparse state-space models and obtain a truncated transfer function or zero-pole-gain approximation. `zpk` and `tf` compute a subset of the zeros and poles of sparse models, typically in a specific low-frequency band [0 f_{max}]. This can yield better low-frequency approximations than modal truncation model-order reduction at the expense of more computation. `zpk` and `tf` also provide direct control over the roll-off slope past the frequency f_{max}. Use the new syntax `zsys = zpk(sparseSys,Name=Value)` and `tsys = tf(sparseSys,Name=Value)` to obtain an approximation based on specified options, such as frequency range of focus. For all available options, see `tf` and `zpk`. For examples, see Compute Truncated ZPK Approximation of Sparse Model and Compute Truncated Transfer Function Approximation of Sparse Model.

Sparse Model Analysis: Compute subset of poles in frequency range of interest

You can now use `pole` to compute a subset of poles for sparse state-space models in a specified frequency range of interest. By default, the function computes first 1000 poles with the smallest magnitude. To avoid computing a large number of poles, you can use the new syntax `p =`

`pole(sparseSys,Name=Value)` to specify options such as frequency range of interest or maximum number of poles to compute. Additionally, you can use `damp` to compute a subset poles and their characteristics and `isstable` to determine stability based on the subset of computed poles. For all available options and examples, see the function reference pages.

Model Reducer App: Interactively obtain reduced order models using Proper Orthogonal Decomposition method

You can now reduce model order using the proper orthogonal decomposition (POD) method in the Model Reducer app. POD is a simulation-based technique to extract dominant state directions and perform an approximate balanced truncation. This method takes snapshots of the state vector during simulation and uses principal component analysis (PCA) to obtain principal directions. For an example, see Proper Orthogonal Decomposition Using Model Reducer.

★Open-Loop Editor: Perform interactive graphical loop shaping

You can now create a graphical open-loop editor using the `openloopeditor` function. You can:

- Use the editor to interactively design a compensator for a specified plant. For an example, see Design Compensator Using Open-Loop Editor.
- Add the editor to your custom App Designer app. For an example, see Create Open-Loop Control Design App.

★Time-Domain Responses: Plot responses for LTI systems with complex coefficients

You can now plot time-domain responses of systems with complex coefficients using the following functions:

- `stepplot` — Step response
- `impulseplot` — Impulse response
- `initialplot` — Initial condition response
- `lsimplot` — Response to arbitrary input

You can plot the response for a complex system as:

- Real and imaginary responses on a single axis
- Magnitude and phase responses on a single axis
- A single response on a complex plane

You can also obtain complex response data using `step`, `impulse`, `initial`, and `lsim`.

stepinfo: Compute characteristics for complex responses

Using `stepinfo`, you can now obtain step-response characteristics of LTI systems with complex responses by specifying one of the following:

- Dynamic system model with complex coefficients

- Complex values for arguments `y`, `yfinal`, and `yinit`.

lsiminfo: Compute characteristics for complex responses

Using `lsiminfo`, you can now compute characteristics for complex responses by specifying complex values for arguments `y`, `yfinal`, and `yinit`.

lsiminfo: Compute peak response value

`lsiminfo` now provides two new characteristics, `Peak` and `PeakTime`, that return the peak value of $|y(t) - y_{init}|$ and the time at which the peak value occurs, respectively.

Linear Analysis Plots: Customize grid line appearance using new style properties

You can now customize grid line appearance for linear analysis plots using new style properties. To specify a style property, set the corresponding `AxesStyle` subproperty of the chart object. For example, to set the major grid line transparency, use the `GridAlpha` subproperty.

```
sp = stepplot(sys);
sp.AxesStyle.GridAlpha = 0.3;
```

This table lists the new style properties and their subproperties and supported plots.

Style Property	AxesStyle Subproperty	Supported Linear Analysis Plots
Major grid line style	`GridLineStyle`	All
Major grid transparency	`GridAlpha`	• `impulseplot` • `initialplot` • `stepplot` • `lsimplot` • `bodeplot` • `nicholsplot` (MIMO systems) • `nyquistplot` (MIMO systems) • `sigmaplot` • Pole-zero plots (`iopzplot`, `pzplot`, `rlocusplot`) with a default grid type and a mix of continuous-time and discrete-time responses
Minor grid visibility	`MinorGridVisible`	
Minor grid color	`MinorGridColor`	
Minor grid line width	`MinorGridLineWidth`	
Minor grid line style	`MinorGridLineStyle`	
Minor grid transparency	`MinorGridAlpha`	
Custom grid type	`GridType`	• `iopzplot` • `pzplot` • `rlocusplot`
Custom grid damping ratios	`GridDampingSpec`	Pole-zero plots (`iopzplot`, `pzplot`, `rlocusplot`) with a

Style Property	AxesStyle Subproperty	Supported Linear Analysis Plots
Custom grid natural frequencies	`GridFrequencySpec`	nondefault grid type or with all responses in the same time domain
Custom grid sample time	`GridSampleTime`	

Linear Analysis Plots: Specify background color and box line width for axes

You can now specify these style properties for linear analysis plots:

- Background color — Use the `AxesStyle.BackgroundColor` property of the chart object.
- Box outline width — Use the `AxesStyle.BoxLineWidth` property of the chart object.

Linear Analysis Plots: Specify active data axes for hold-on workflows

When using hold-on workflows, you can now programmatically specify the active data axes of a MIMO linear analysis plot or a SISO Bode plot. To specify the active data axes, set the `DataAxes` property of the corresponding chart object.

Nichols Plot: Specify magnitude units as absolute

You can now configure a Nichols plot to use absolute units for magnitude by setting the `MagnitudeUnit` property to `"abs"`. When using absolute units, you can set the magnitude scale to either linear or logarithmic units by setting the `MagnitudeScale` property to `"linear"` or `"log"`, respectively.

For more information, see `NicholsPlot Properties`.

Linear Analysis Plots with Custom Grids: Toggle visibility of grid labels

You can now control the visibility of grid labels for these linear analysis plots that use custom grid lines:

- `pzplot`
- `iopzplot`
- `rlocusplot`
- `nicholsplot` (SISO systems)
- `nyquistplot` (SISO systems)

To toggle grid label visibility in a plot, use the `AxesStyle.GridLabelsVisible` property of the corresponding chart object. For example, display the grid for a pole-zero plot and hide the grid labels.

```
pzp = pzplot(sys);
pzp.AxesStyle.GridVisible = "on";
pzp.AxesStyle.GridLabelsVisible = "off";
```

spy: Specify parent graphics container other than axes

You can now use the `spy` function to plot the sparsity pattern of a sparse model in a parent container other than an axes. To plot the sparsity pattern, specify the `parent` argument as a graphics container, such as a `Figure` or `TiledChartLayout` object. You can also still specify an `Axes` or `UIAxes` object.

You can specify the parent container when you want to create a plot in a specified open figure or when creating apps in App Designer.

⚠view: Specify parent graphics container

You can now plot graphical information in a parent graphics container using these `view` functions with model order reduction objects:

- `view (balanced)`
- `view (ncf)`
- `view (modal)`
- `view (pod)`
- `view (frfit)`
- `view (zpk)`

To plot graphical information, specify the `parent` argument as a graphics container, such as a `Figure` or `TiledChartLayout` object.

You can specify the parent container when you want to create a plot in a specified open figure or when creating apps in App Designer.

⚠Compatibility Considerations

The `Parent=parent` argument replaces the `Axes=AX` argument. For more information, see “Axes input argument of view function not recommended” on page 3-7.

Classical Controller Design Techniques: Self-paced, interactive course available as part of Online Training Suite subscription

Classical Controller Design Techniques is a new course that teaches you to:

- Design classical controllers for real-world applications to meet design requirements using standard tools like the root locus and the Bode diagram.
- Test your controller in a simulated physical environment.

For more information, see Classical Controller Design Techniques.

PID Control Techniques: Self-paced, interactive course available as part of Online Training Suite subscription

PID Control Techniques is a new course that teaches you to:

- Design for real-world applications that use PID controllers.
- Automatically tune a PID controller using the PID Tuner.

For more information, see PID Control Techniques.

Control System Analysis Techniques: Self-paced, interactive course available as part of Online Training Suite subscription

Control System Analysis Techniques is a new course that teaches you to:

- Perform time domain and frequency domain analysis with your control systems.
- Explore critical system properties such as stability and stability margins.

For more information, see Control System Analysis Techniques.

Control System Modeling Essentials: Self-paced, interactive course available as part of Online Training Suite subscription

Control System Modeling Essentials is a new course that teaches you to:

- Create control systems in MATLAB and Simulink® from mathematical models, such as transfer functions or state-space equations, and by directly using physical modeling.
- Simulate the behaviors of systems.
- Choose the appropriate modeling method for your control system modeling problems.

For more information, see Control System Modeling Essentials.

⚠Functionality being removed or changed

Axes input argument of view function not recommended

Specifying the axes for plotting using `Axes=AX` is not recommended. Specify a parent graphics container using `Parent=parent` instead. The `Axes=AX` argument continues to work.

Specifying `parent` as an `Axes` or `UIAxes` object sets the plot parent to the parent of the specified axes object.

Update your code to specify a parent graphics container. For example, this code plots the response in an axes object obtained from a figure.

```
f = figure;
ax = gca;
view(R,Axes=ax)
```

To update the code, set the parent of the plot to the figure instead.

```
f = figure;
view(R,Parent=f)
```

ngrid not recommended

Still runs

`ngrid` is not recommended. To add grid lines to a Nichols plot, enable the `AxesStyle.GridVisible` property of the corresponding chart object instead.

In previous releases, you create a Nichols plot using `nichols` or `nicholsplot` and add grid lines using `ngrid`.

```
nichols(sys)
ngrid
```

Starting in R2025a, you create a `nicholsplot` object and enable the `AxesStyle.GridVisible` property.

```
np = nicholsplot(sys);
np.AxesStyle.GridVisible = "on";
```

You can also enable the grid using the `grid` command.

```
nichols(sys)
grid
```

sgrid not recommended
Still runs

`sgrid` is not recommended. To add an *s*-plane grid of constant damping factors and natural frequencies to a pole-zero plot of a continuous-time system, enable the `AxesStyle.GridVisible` property of the corresponding chart object instead. To configure the damping ratios and natural frequencies for the grid, use the `AxesStyle.GridDampingSpec` and `AxesStyle.GridFrequencySpec` properties, respectively.

You can also enable the default grid using the `grid` command.

This table shows some typical usages of `sgrid` and how to update your code to use chart object properties instead.

Discouraged Usage	Recommended Replacement
`rlocus(sys);` `sgrid`	`rlocus(sys);` `grid`
`rlocusplot(sys);` `sgrid`	`rp = rlocusplot(sys);` `rp.AxesStyle.GridVisible = "on";`

`pzp = pzplot(sys);`
`zeta = [0.2 0.4 0.6 0.8];`
`wn = [2 4];`
`sgrid(zeta,wn)`
	`pzp = pzplot(sys);`
	`zeta = [0.2 0.4 0.6 0.8];`
	`wn = [2 4];`
	`pzp.AxesStyle.GridVisible = "on";`
	`pzp.AxesStyle.GridDampingSpec = zeta;`
	`pzp.AxesStyle.GridFrequencySpec = wn;`

zgrid not recommended
Still runs

`zgrid` is not recommended. To add a *z*-plane grid of constant damping factors and natural frequencies to a pole-zero plot of a discrete-time system, enable the `AxesStyle.GridVisible` property of the corresponding chart object instead. To configure the damping ratios and natural frequencies for the grid, use the `AxesStyle.GridDampingSpec` and `AxesStyle.GridFrequencySpec` properties, respectively. To specify the grid sample time, use the `AxesStyle.GridSampleTime` property.

You can also enable the default grid using the `grid` command.

This table shows some typical usages of `zgrid` and how to update your code to use chart object properties instead.

Discouraged Usage	Recommended Replacement
`rlocus(sys);` `zgrid`	`rlocus(sys);` `grid`
`rlocusplot(sys);` `zgrid`	`rp = rlocusplot(sys);` `rp.AxesStyle.GridVisible = "on";`

`pzp = pzplot(sys);`
`zeta = [0.2 0.4 0.6 0.8];`
`wn = [2 4];`
`zgrid(zeta,wn)`
	`pzp = pzplot(sys);`
	`zeta = [0.2 0.4 0.6 0.8];`
	`wn = [2 4];`
	`pzp.AxesStyle.GridVisible = "on";`
	`pzp.AxesStyle.GridDampingSpec = zeta;`
	`pzp.AxesStyle.GridFrequencySpec = wn;`

`rp = rlocus(sys);`
`Ts = 0.1;`
`zgrid(Ts)`
	`pzp = rlocusplot(sys);`
	`Ts = 0.1;`
	`pzp.AxesStyle.GridVisible = "on";`
	`pzp.AxesStyle.GridSampleTime = Ts;`

MIMO IOPZPlot charts use s-plane and z-plane grids
Behavior change

`IOPZPlot` objects now use *s*-plane and *z*-plane grids in plots of MIMO systems. For such systems, you can no longer use Cartesian grid lines.

Model Reducer automatically selects the analysis plot based on reduction criteria
Behavior change

Model Reducer now automatically selects the analysis plot type based on the model reduction criteria for balanced truncation and proper orthogonal decomposition methods. For example, when you set **Reduction criteria** to `Minimum energy` the app automatically shows the plot for normalized state energies.

xperm no longer reorders equations for descriptor state-space models
Behavior change

Starting in R2025a, `xperm` reorders only the states and not the equations for descriptor state-space models. This means that A, E are transformed to $A(:,p)$, $E(:,p)$.

R2024b

⚠Linear Analysis Plots: Improved customization workflows and integration with MATLAB plotting tools

Linear analysis plotting functions that previously returned plot handles now return chart objects. The new chart objects improve workflows for customizing the plot appearance and improve integration with MATLAB plotting tools.

The new chart objects allow you to customize your plot using dot notation. For more information, see Customize Linear Analysis Plots at Command Line.

The new plotting objects improve integration with MATLAB plotting tools. For example:

- You can now add linear analysis plots to tiled chart layouts.
- You can programmatically add linear analysis plots to containers in App Designer. For an example, see Create App with Linear Analysis Response Plots.
- Legends are now added on a per-chart basis rather than to axes within a chart.
- You can add a response to an existing chart using the new `addResponse` function.
- You can programmatically change the parent for a chart object.
- For the new chart objects, saving and opening the parent figure now maintains the full interactivity of the plot; that is, you can modify the plot properties after loading. Previously, opening such a saved figure maintained only the plot appearance.
- Chart objects can dynamically add or remove axes when responses are updated, added, or removed. Therefore, the system with the largest I/O size no longer has to be the first system plotted during `hold on` workflows.

⚠Compatibility Considerations

The new charting objects introduce compatibility considerations that might require updates to your code. For more information, see:

- “Linear analysis plot functions return chart objects rather than handle objects” on page 4-5
- “updateSystem not recommended” on page 4-6
- “getoptions not recommended” on page 4-6
- “setoptions not recommended” on page 4-7

★Model Order Reduction: Compute low-order models using Proper Orthogonal Decomposition method

You can now compute reduced-order models using the proper orthogonal decomposition (POD) method with `reducespec`. POD is a simulation-based technique to extract dominant state directions and perform an approximate balanced truncation. This method takes snapshots of the state vector during simulation and uses principal component analysis (PCA) to obtain principal directions.

To perform simulations, extract state-snapshot data, and approximate the controllability and observability Gramians, the software uses built-in excitation signals (impulse, chirp, and PRBS). By default, the software uses an impulse excitation signal. Alternatively, you can bypass the inbuilt simulations and provide:

- State-snapshot data obtained from Simulink or a third-party software.
- Custom POD data generated from a simulation with `lsim`.

To perform model order reduction using POD, use the Task-Based Model Order Reduction Workflow.

For more information and examples, see:

- `ProperOrthogonalDecomposition`
- `ProperOrthogonalDecompositionOptions`
- `incrementalPOD`

Sparse Modal Truncation: Reduce nonsymmetric sparse models

You can now perform modal truncation of nonsymmetric sparse problems using `reducespec` and Model Reducer. Before R2024b, sparse modal truncation was limited to first-order models with $A = A^T$ and $E = E^T$ definite, or second-order models with $K = K^T$, $M = M^T$ definite, and Rayleigh-type damping.

Balanced Truncation: Reduce discrete-time sparse models

You can now perform balanced truncation of `sparss` and `mechss` models in discrete-time using `reducespec` and Model Reducer. Before R2024b, sparse balanced truncation was limited to continuous-time models.

Balanced Truncation: Select model order using new criteria based on total neglected state energy

When reducing model order using the balanced truncation method, you can now select model order based on the fraction of total energy you want to neglect. The `BalancedTruncation` and `SparseBalancedTruncation` objects now contain a new property `Loss` to store the energy loss as a fraction of total energy, and `getrom` provides a new name-value argument `MaxLoss` to pick the reduced order based on maximum energy loss as a fraction of total energy. For example, to obtain a reduced model that neglects 1% of the total energy, you can specify `rsys = getrom(R,MaxLoss=0.01)`.

Balanced Truncation: Set balancing algorithm using Algorithm property, renamed from Goal

The `Goal` property of `BalancedTruncationOptions` and `Goal` name-value argument of `balreal` are now called `Algorithm`. The behavior of this property remains the same. There are no plans to remove support for existing references to `Goal`.

lsim: Perform POD analysis of state snapshots using new syntax

Use the new `lsim` syntax to perform proper orthogonal decomposition (POD) of state snapshots during your custom simulation for a POD model order reduction application (see “Model Order Reduction: Compute low-order models using Proper Orthogonal Decomposition method” on page 4-2). To start a new POD analysis or add to previous POD results, use the syntax `[y,t,x,~,xPOD] = lsim(sys,__,xPOD)`, where `xPOD` is an `incrementalPOD` object.

lsim: Specify initial conditions using findop

You can now also specify initial conditions using an operating point object computed using `findop`. This allows you to start the simulation from a steady-state operating condition with nonzero past *u*, *w*, and *y* values for models with delays. For example, to start a simulation from nonzero *y* value, you can specify:

```
op = findop(sys,y=3);
y = lsim(sys,u,t,op)
```

⚠initial: Specify initial conditions using findop or RespConfig

You can now specify initial conditions for the `initial` function using the syntax `[y,t] = initial(sys,IC,__)`, where `IC` is one of the following:

- Initial state values, specified as a vector of length equal to the number of states.
- Response configuration, specified as a `RespConfig` object. Use this object to specify initial state and parameter values for LPV models.
- Operating condition, specified as an object created using `findop`.

⚠Compatibility Considerations

As a result of this change, the syntax `[y,tOut] = initial(sys,{xinit,pinit},t,p)` is not recommended. For more information, see “initial: Syntax not recommended” on page 4-5.

RespConfig: Set baseline input value using Bias property, renamed from InputOffset

The `InputOffset` property of `RespConfig` is now called `Bias`. The behavior of this property remains the same. There are no plans to remove support for existing references to `InputOffset`.

findop: Improvements for specifying internal delay specifications and additional properties in output objects

You can now use the `dw` name-value argument of `findop` to specify the internal signal mismatch $dw = z - w$ when computing an operating condition. To reflect this change, the output objects now contain new fields `dw` and `rw`, where `dw` is the mismatch value at the operating condition and `rw` is the relative residual for the delay equation $dw + w - z = 0$.

Additionally:

- The function now returns `rx` and `ry` as relative residuals for the solved equations. This makes it easier to determine if the software achieved the operating point specification.
- The output objects now contain new fields for specified time `t` and parameter `p` values for LTV and LPV models. For all other models, the function returns `t = 0` and `p = []`.

sminDAE: Eliminate algebraic states in sparse state-space models while preserving sparsity

Use the new function `sminDAE` to eliminate algebraic states in sparse state-space models while preserving overall sparsity.

Varying State-Space Blocks: Specify initial condition externally as block input

The LPV System, LTV System, Varying State Space, and Discrete Varying State Space blocks now allow you to specify initial state values at the block input port **IC**. To enable this port, set the **Initial condition source** parameter to `external`.

⚠Functionality being removed or changed

Balanced Truncation: Normalized state energy computation fixed

Behavior change

For balanced truncation model order reduction, the software now computes normalized state energy as $\frac{\sigma_k}{\sigma_1}$, where σ_k is the energy of kth principal state. Before R2024b, the normalized energy was computed as $\left(\frac{\sigma_k}{\sigma_1}\right)^2$. For more information, see the `Energy` property description on `BalancedTruncation` and `SparseBalancedTruncation`.

initial: Syntax not recommended

Still runs

You can now provide initial conditions for state and parameter values for linear-parameter varying (LPV) models using a `RespConfig` object or an operating point object computed using `findop`. As a result of this change, the syntax `[y,tOut] = initial(sys,{xinit,pinit},t,p)` is not recommended. To update your code, use the following recommended workflows:

`RespConfig`	`IC = RespConfig(InitialState=xinit,InitialParameter=p` `[y,tOut] = initial(sys,IC,t,p);`
`findop`	`IC = findop(sys,Ti,pinit,x=xinit);` `[y,tOut] = initial(sys,IC,t,p);`

Linear analysis plot functions return chart objects rather than handle objects

Behavior change

Starting in R2024b, linear analysis plotting functions return chart objects rather than `handle` objects. For more information, see “Linear Analysis Plots: Improved customization workflows and integration with MATLAB plotting tools” on page 4-2.

The following functionality changes might require updates to your code.

- The `gca` function now returns the linear analysis chart object rather than an axes within the plot.
- You can no longer access the graphics objects within a linear analysis plot using the `Children` property of its parent figure.

Data compatibility checks for plotting functions that previously generated warnings now generate errors

Errors

Some data compatibility checks that previously generated warnings for linear analysis plotting functions now generate errors.

For example, the following code generates an error when plotting a step response for a state-space model with complex matrix elements.

```
A = [-3.50,-1.25-0.25i;2,0];
B = [1;0];
C = [-0.75-0.5i,0.625-0.125i];
D = 0.5;
Gc = ss(A,B,C,D);
stepplot(Gc)
```

As another example, the following code generates an error when plotting a delayed step response for a model with internal delays.

```
sys = rss(2,2,2);
sys = setDelayModel(sys,2);
stepplot(sys,RespConfig(Delay=1));
```

updateSystem not recommended

Still runs

`updateSystem` is not recommended. Update the model in linear analysis plots by modifying the chart object properties using dot notation.

To change the model for a response plot with a single system, use the following command, where `rp` is the response plot object and `newSys` is the updated model.

```
rp.Responses.SourceData.Model = newSys;
```

To change the model for the second system in a response plot with multiple responses, use the following command.

```
rp.Responses(2).SourceData.Model = newSys;
```

getoptions not recommended

Still runs

`getoptions` is not recommended. Obtain plot options by accessing chart object properties using dot notation. `getoptions` continues to work.

This table shows some typical usages of `getoptions` and how to update your code to access chart object properties instead. Some chart object property names do not match the corresponding option name for `getoptions`.

If your code has this form:	Use this code instead:
`bp = bodeplot(sys);` `phaseUnits = getoptions(bp,"PhaseUnits");`	`bp = bodeplot(sys);` `phaseUnits = bp.PhaseUnits;`
`bp = bodeplot(sys);` `phaseUnits = getoptions(bp,"FreqScale");`	`bp = bodeplot(sys);` `phaseUnits = bp.FrequencyScale;`

setoptions not recommended
Still runs

`setoptions` is not recommended. Set plot options by accessing chart object properties using dot notation. `setoptions` continues to work.

This table shows some typical usages of `setoptions` and how to update your code to access chart object properties instead. Some chart object property names do not match the corresponding option name for `setoptions`.

If your code has this form:	Use this code instead:
`bp = bodeplot(sys);` `setoptions(bp,"PhaseUnits","deg");`	`bp = bodeplot(sys);` `bp.PhaseUnit = "deg";`
`bp = bodeplot(sys);` `setoptions(bp,"FreqScale","linear");`	`bp = bodeplot(sys);` `bp.FrequencyScale = "linear";`

Varying State-Space Blocks: Initial state parameter x0 renamed to InitialCondition

In the LPV System, LTV System, Varying State Space, and Discrete Varying State Space blocks, the Initial state parameter `x0` for programmatic use is now called `InitialCondition`. If your code sets this parameter programmatically using, for example `set_param(blockPath,"x0","[0 0.1]")`, update your code to `set_param(blockPath,"InitialCondition","[0 0.1]")`.

R2024a

★Sparse Model Order Reduction: Interactively obtain reduced order models using Model Reducer app

You can now compute reduced-order models of sparse state-space models interactively using the Model Reducer app. The software supports sparse model order reduction using these methods:

- Balanced truncation — Obtain low-order approximation by discarding states with low contribution.
- Modal truncation — Obtain low-order approximation by discarding undesired modes.

⚠Model Reducer App: Obtain reduced models using modal truncation

You can reduce model order using the modal truncation method in the Model Reducer app. Using modal truncation, you can obtain reduced-order models by discarding undesired modes based on their frequency range, damping range, or minimum DC contribution.

⚠Compatibility Considerations

Modal truncation replaces the mode selection method. The modal truncation method provides better flexibility for choosing the criteria for discarding modes.

reducespec: Support for sparse modal truncation of discrete-time models

You can now perform modal truncation of `sparss` and `mechss` models in discrete-time. Sparse modal truncation in discrete-time is only applicable if $A+E$ is definite.

★Create ss and sparss state-space models with offsets

Use the new `Offsets` property of the `ss` and `sparss` objects to store model offsets. Offsets usually arise when linearizing nonlinear dynamics at some operating conditions. This property helps you manage linearization offsets and use them in operations such as response simulation, model interconnections, and model transformations.

For an example, see Use Linearization Offsets to Help Compare Nonlinear and Linearized Responses.

augoffset: Map state-space model offsets to extra input channel

Use `augoffset` to obtain an offset-free state-space model by mapping the model offsets as an extra input channel. For more information, see `augoffset`.

RespConfig and fixInput: Support for state-space models with offsets

`RespConfig` now lets you configure a model response relative to the input offset u_0 with initial condition x_0 for `ss` and `sparss` models with offsets. Additionally, `fixInput` now supports fixing an input to a value specified using input offset u_0.

LTI System Block: Support for state-space models with offsets and complex coefficients

You can now simulate a state-space model with offsets using the LTI System block. Use the **LTI system variable** parameter to specify a state-space variable containing model offsets in the `Offsets` property. Offsets usually arise when linearizing nonlinear dynamics at some operating conditions. For more information about how to store offsets, see `ss`.

Additionally, you can now simulate LTI models with complex coefficients using the LTI System block.

⚠LPV and LTV Model Sampling: Return offsets directly in state-space model array property

Use the `psample` function to store the model offsets directly in the `Offsets` property of a `ss` model array when you sample the dynamics of a linear time-varying (LTV) or linear parameter-varying (LPV) model.

⚠Compatibility Considerations

`psample` replaces `sample`, which returns model offsets as an additional output argument.

ssInterpolant: Use new syntax to create gridded models when offsets are stored in ss array

Use the new syntax `sys = ssInterpolant(ssArray)` to construct a gridded model when offsets are stored in the `Offsets` property of the state-space model array `ssArray`. For more information, see `ssInterpolant`.

Varying State Space Blocks: Specify varying offsets as block inputs

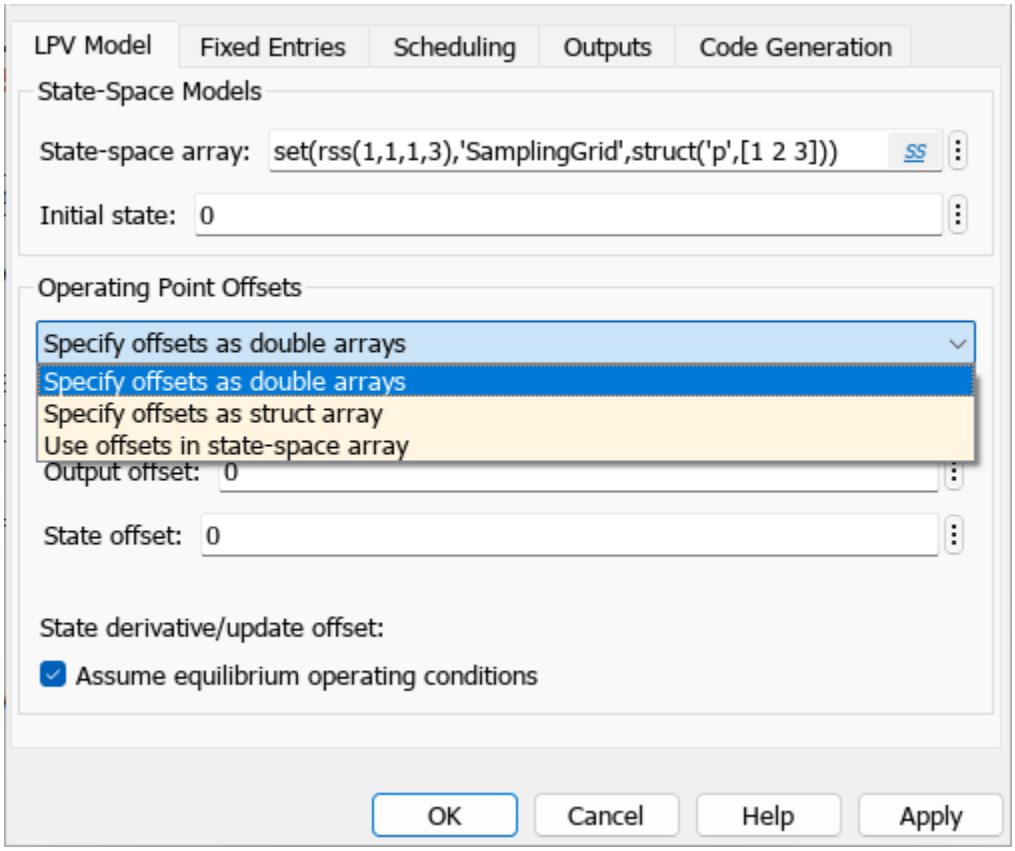
Use the **Offsets** tab of the Varying State Space and Discrete Varying State Space block parameters to enable the input ports for specifying instantaneous values of the varying offsets.

LPV System Block: Improved support for offsets and other enhancements

The LPV System block now supports specifying operating point offsets in two new formats:

- Specify offsets as struct array — Specify offsets as a structure array with fields `u`, `y`, `x`, and `dx` specifying the input, output, state, and state derivative offsets, respectively.
- Use offsets in state-space array — Use the offsets specified in the `Offsets` property of the state-space array.

Before R2024a, the block supported specifying offsets only as double arrays in the block parameters. Use the **Operating Point Offsets** list to specify the format of offsets.



Additionally, the block has the following new enhancements:

- The block architecture now uses a Varying State Space block underneath. This streamlines the support for offsets.
- The block now supports fast restart for simulation. In this mode, only the block inputs and the **Initial state** parameter are tunable. The rest of the block parameters are non tunable because changing them alters the model topology.

★LTV System Block: Model and simulate linear time-varying systems in Simulink

Use the new LTV System block in the **Linear Parameter Varying** library to simulate linear time-varying (LTV) systems in Simulink.

The LTV System block interpolates a collection of time-dependent state-space models with offsets. The block accepts an array of state-space models and corresponding operating point offsets to simulate an LTV model. The `SamplingGrid` property of the array defines how these models depend on time.

Varying Delay and Discrete Varying Delay Blocks: Model delays in discrete and continuous time

Use the new Varying Delay and Discrete Varying Delay blocks in the **Linear Parameter Varying** library to implement delays in discrete time or continuous time, respectively, for signals in your

Simulink models. These blocks are helpful when you want to model a mix of zero, fixed, and varying delays, which typically arise in LPV systems.

lpvss and ltvss: Support to specify input and output delays

The `ltvss` and `lpvss` objects now let you specify input and output delays for LTV and LPV models, respectively. Use the `Delays` argument of the data function to define fixed or varying delays. For more information, see Using LTV and LPV Models in MATLAB and Simulink.

Improvements to c2d and c2dOptions

You can now use `c2d` to discretize models with offsets such as state-space or gridded linear parameter-varying models.

Additionally, `c2dOptions` provides two new options

- `DelayModeling` — Specify whether to model extra delays as internal delays (default) or additional states.
- `Consistency` — Enforce state and delay consistency in state-space arrays. This option is helpful when discretizing gridded LPV or LTV models.

absorbDelay: Specify type of delay to absorb using new syntax

Use the new syntax `sysnd = absorbDelay(sysd,scope)` to specify which delay type to absorb using the `scope` argument. This helpful when you only want to eliminate specific types of delays and leave other delays unchanged. For information, see `absorbDelay`.

dss2ss: Convert descriptor state-space model to explicit form

Use `dss2ss` to convert a descriptor state-space model to explicit form. For more information, see `dss2ss`.

sminreal: New syntax

Use the new syntax `[asysr,xkeep] = sminreal(asys,"consistent")` to eliminate states and internal delays that do not contribute to the input/output map for all the models in the state-space array `asys`. The reduced states and delays are the same for all models. For more information, see `sminreal`.

margin and allmargin: Specify frequency range for stability analysis

You can now specify a frequency range when determining closed-loop stability with `margin` and `allmargin`. The analysis ignores stability issues outside the range you specify with the `Focus` option. For details, see `margin` and `allmargin`.

icare and idare: Improved singularity detection

The `icare` and `idare` functions now return `X = []`, `K = []`, `L = NaN(n,1)`, and `info.Report = 4` when pencil is singular, that is, `[B;S;R]` is rank deficient.

mechss: Support for continuous and discrete conversions

You can now convert between continuous-time and discrete-time, and resample `mechss` models using `c2d`, `d2c` and `d2d` with the `'tustin'` method. The `'tustin'` method computes the second-order form of the Tustin discretization. This is equivalent to applying Tustin to the first-order `sparss` equivalent of the `mechss` model. This form is more favorable to modal approximation since it preserves symmetry.

connect: Support for connecting multiple interconnected systems with analysis points

You can specify analysis points when modeling block diagrams with `connect`. When combining the resulting models with any interconnection function, the software automatically renames these analysis point blocks to avoid conflict. Before R2024a, such combinations were not possible.

★New Example Series: Thermal modeling and control design for CPU chip cooling system

The four-part Thermal Modeling and Control Design for CPU Chip Cooling System tutorial shows a workflow that you can use to take a high-fidelity physical model of a single component all the way through a fully integrated control system design process. This workflow includes the following steps:

1. Create Heat Sink Finite Element Model and Export Data for State-Space Simulation — Create a finite element model of heat sink and extract matrices for state-space model simulation.
2. Import Finite Element Model Data to Simulink — Import finite element model data into Simulink using the Descriptor State-Space block to simulate the model.
3. Create Low-Order LPV Model of CPU and Heat Sink Model — Batch linearize the CPU and heat sink model, create an LPV model from the linearized data, and obtain a reduced order model.
4. Tune PI Controller for Heat Sink Model — Tune PI controller to obtain the desired response from the control loop.

⚠Functionality being removed or changed

Model Reducer App: Mode Selection method is now Modal Truncation

Behavior change

The **Modal Truncation** method replaces the **Mode Selection** method in the Model Reducer app. Modal truncation method provides better flexibility for choosing the criteria for discarding modes.

c2d: Does not add extra states during conversion

Behavior change

The `c2d` command no longer adds extra states when modeling extra delays. By default, `c2d` now uses internal delays and returns the discretized model with the same number of states. This results in a more predictable behavior and simplifies the model interconnection mapping.

To revert to the old behavior before R2024a, create an option set using `c2dOptions` and set `DelayModeling` to `"state"`.

c2dOptions: FractDelayApproxOrder renamed to ThiranOrder

Behavior change

The `FractDelayApproxOrder` property of `c2dOptions` is renamed to `ThiranOrder`.

LPV System block: Ignores constant parameters in SamplingGrid
Behavior change

The LPV System block now ignores singleton dimension and constant parameter in the sampling grid of the input model array. Therefore, you must not feed the parameters, that is, the fields of the `SamplingGrid` structure with constant values to the **par** port of the block. Use only the varying fields with this port.

Before R2024a, the block returned an error when fed parameters with constant values.

sample not recommended

The `sample` command is not recommended. Use `psample` to sample the dynamics of LPV and LTV models instead.

findop: Output changed
Behavior change

The output `op` of `findop` has changed. The function now returns an object instead of a structure.

- For `mechss` models, `op` is a `OperatingPoint2` object.
- For all other models, `op` is a `OperatingPoint` object.

Additionally, when computing operating point condition for discrete-time models, the function now returns a different values for `dx` (non-`mechss` models) and `dq` and `d2q` (`mechss` models) fields.

Field	R2024a	Before R2024a
`dx`	`x[k+1]`	`x[k+1] –x[k]`
`dq`	`q[k+1]`	`q[k+1] –q[k]`
`d2q`	`q[k+2]`	`q[k+2] + q[k] – 2q[k+1]`

This change provides better consistency and alignment with how the software handles state-space models with offsets. If your existing code uses these fields to initialize an operating condition, consider checking your results in R2024a.

R2023b

⚠Improved Model Order Reduction Workflow: Use object-oriented framework to represent reduction specifications

The Control System Toolbox software now uses objects to perform model-order reduction for linear time-invariant (LTI) and sparse LTI models. These objects provide an improved workflow for configuration and analysis of reduced-order models (ROMs).

For more information about the new workflow, see `reducespec` and Task-Based Model Order Reduction Workflow. For all available functionality, see Model Order Reduction.

⚠Compatibility Considerations

The new workflow replaces the previous model-order reduction workflows from Control System Toolbox and Robust Control Toolbox™. For more information, see “balred and balredOptions not recommended” on page 6-4 and “Functionality being removed or changed” (Robust Control Toolbox).

Sparse Model Order Reduction: Obtain reduced order models for sparss and mechss models

You can now compute reduced-order models of sparse state-space models using the new model order reduction workflow. The software supports sparse model order reduction using these methods.

- Balanced truncation — Obtain low-order approximation by discarding states with low contribution.
- Modal truncation — Obtain low-order approximation by discarding undesired modes.

For more information about the new model order reduction workflow, see Task-Based Model Order Reduction Workflow.

LTI Model Order Reduction: Support for modal truncation method

You can now compute reduced-order models for linear time-invariant (LTI) models using the modal truncation method. Use this method to reduce model order by discarding modes based on locations or normalized DC contributions.

To perform modal truncation of LTI models, use the new model order reduction workflow. For more information, see Task-Based Model Order Reduction Workflow.

Sparse State-Space Models: Improved performance for frequency response computation

Sparse state-space models now use multi-threading by default for frequency response computation. As a result, the computation performance has improved.

⚠modalreal and compreal: Compute modal and companion state-space realizations

Use the new functions `modalreal` and `compreal` to compute the modal and companion state-space realizations, respectively.

Additionally, the software now supports computing modal realizations of descriptor state-space models.

Compatibility Considerations

`modalreal` replaces `modreal` and `canon(sys,'modal')` for computing modal state-space realizations.

`compreal` replaces `canon(sys,'companion')` for computing companion state-space realization.

modalsep and modalsum: Perform modal decomposition and summation of modal components

Use the new functions `modalsep` and `modalsum` to decompose an LTI system into its modal components and combine a subset of modal components into an LTI system, respectively.

Additionally, the software now supports modal decomposition of descriptor state-space models.

Compatibility Considerations

`modalsep` replaces `modsep` for computing modal decompositions.

xelim: Eliminate states from state-space models

Use the new function `xelim` to simplify a state-space model by eliminating specified states.

Compatibility Considerations

`xelim` replaces `modred` for simplifying models by state elimination.

ssequiv: Perform equivalence transformation for state-space models

Use the new function `ssequiv` to perform an equivalence transformation on a state-space model.

Input Model	Transformed Model
$E\dot{x} = Ax + Bu$ $y = Cx + Du$	$T_L E T_R \dot{x} = T_L A T_R x + T_L B u$ $y = C T_R x + Du$

stabsep and freqsep: Use simplified syntax

For `stabsep` and `freqsep`, you can now provide options using name-value arguments.

Compatibility Considerations

As a result of this change, `stabsepOptions` and `freqsepOptions` are not recommended. For more information on how to update your code, see “stabsepOptions and freqsepOptions not recommended” on page 6-6.

ssInterpolant: Support for scattered parameter samples in interpolated LPV models

Starting in R2023b, `ssInterpolant` can interpolate state-space data specified at scattered points in 2-D or 3-D parameter space. The maximum of three parameters includes time values when the LPV model explicitly depends on time.

findop: Compute operating condition from specifications for LTI and LPV models

Use the new function `findop` to compute operating condition from specifications for linear time-invariant, linear time-varying, and linear parameter varying-models. Computing trim condition helps you correctly initialize simulations or determine initial output values for operations such as computing step response characteristics.

step, impulse, and initial: New syntax

Use the new syntax `[y,tOut] = step(sys,[t0,tFinal])` or similar to simulate the response from time `t0` to `tFinal`.

For more information, see `step`, `impulse`, and `initial`.

bdqz: Perform block-diagonal QZ decompositions

Use the new function `bdqz` to compute block-diagonal QZ decompositions.

New Example: Detect and Mitigate Attacks in Platooning

The new example Detect and Mitigate Attacks in Platooning shows how to detect and mitigate false data injection attacks in the vehicle-to-vehicle (V2V) communication channel of a platoon.

⚠Functionality being removed or changed

balred and balredOptions not recommended
Still runs

`balred` and `balredOptions` commands are not recommended. Use the new function `reducespec` with `"balanced"` method to perform balanced truncation of LTI models. For more information about the new workflow, see Task-Based Model Order Reduction Workflow.

balreal syntax changed
Behavior change

The syntax and input arguments for `balreal` have changed. The function now allows you to provide options using name-value arguments. For more information, see `balreal`.

Model Reducer and Reduce Model Order : Generated code uses new model order reduction workflow
Behavior change

The Model Reducer app and the Reduce Model Order task now generate code using the new model order reduction workflows. For example, this table describes the change in the model order reduction workflow in the generated code.

Method	Generated Code before R2023b	Generated Code in R2023b

Balanced Truncation	`%% Reduce LTI model order using balanced truncation`
	`System = G; % Define System to reduce`
	`Order = 14;`
	
	`% Create option set for balred command`
	`Options = balredOptions();`
	`% Offset for the stable/unstable boundary`
	`Options.Offset = 1e-05;`
	
	`% Compute reduced order approximation`
	`ReducedSystem = balred(System,Order,Options);`
	
	`% Create comparison plot`
	`bode(System,ReducedSystem);`
		`%% Reduce LTI model order using balanced truncation`
		`System = G; % Define System to reduce`
		
		`% Compute reduced order approximation`
		`R = reducespec(System,'balanced');`
		`% Set options for Balanced Truncation spec`
		`% Offset for the stable/unstable boundary`
		`R.Options.Offset = 1e-05;`
		`% Compute MOR data once`
		`R = process(R);`
		`% Get reduced order model`
		`ReducedSystem = getrom(R,Order=14);`
		
		`% Create comparison plot`
		`bode(System,ReducedSystem);`

Mode Selection	`%% Reduce LTI model order using mode selection`
	`System = G; % Define System to reduce`
	`UpperCutoffFrequency = 100;`
	`LowerCutoffFrequency = 10;`
	
	`% Create option set for freqsep command`
	`Options = freqsepOptions();`
	`% Accuracy loss factor for stable/unstable decomposition`
	`Options.SepTol = 100;`
	
	`% Select modes between lower and upper cut-off frequencies`
	`ReducedSystem = freqsep(System,...`
	`    [LowerCutoffFrequency UpperCutoffFrequency],Options);`
	
	`% Create comparison plot`
	`bode(System,ReducedSystem);`
		`%% Reduce LTI model order using mode selection`
		`System = G; % Define System to reduce`
		
		`% Select modes between lower and upper`
		`R = reducespec(System,'modal');`
		`% Set options for Modal Truncation spec`
		`% Accuracy loss factor for stable/unstable decomposition`
		`R.Options.SepTol = 1e-11;`
		`% Compute MOR data once`
		`R = process(R);`
		`% Get reduced order model`
		`ReducedSystem = getrom(R,Frequency=[10`
		`% Create comparison plot`
		`bode(System,ReducedSystem);`

For more information about the new workflow, see `reducespec` and Task-Based Model Order Reduction Workflow.

canon not recommended
Still runs

The `canon` command is not recommended. Use the functionality described in this table instead.

Realization	Old Command	New Command
Modal	`sysT = canon(sys,'modal')`	`sysT = modalreal(sys)`
Companion	`sysT = canon(sys,'companion')`	`sysT = compreal(sys)`

For more information about the new commands, see `modalreal` and `compreal`.

modsep not recommended
Still runs

The `modsep` command is not recommended. Use `modalsep` to perform modal decomposition.

modred renamed to xelim
Still runs

The `modred` command is not recommended. Use `xelim` to simplify models by eliminating states.

stabsepOptions and freqsepOptions not recommended
Still runs

As a result of changes to `stabsep` and `freqsep`, the `stabsepOptions` and `freqsepOptions` commands are not recommended. Use name-value arguments to specify options for `stabsep` and `freqsep`.

- `stabsepOptions` — Specify the `Offset` option as the second argument to `stabsep`, and `Focus` and `SepTol` as name-value arguments to `stabsep`. This table describes the change in the workflow.

Before R2023b	R2023b
`opt = stabsepOptions("Offset",0.01);` `[Gs,Gf] = stabsepOptions(G,opt);`	`Offset = 0.01;` `[Gs,Gf] = freqsep(G,Offset);`
`opt = stabsepOptions("Focus","unstable","SepTol",100);` `[Gs,Gf] = stabsepOptions(G,opt);`	`[Gs,Gf] = freqsep(G,Focus="unstable",SepTol=100);`

- `freqsepOptions` — Specify the `SepTol` option directly as a name-value argument to `freqsep`. This table describes the change in the workflow.

Before R2023b	R2023b
`opt = freqsepOptions("SepTol",5e10);` `[Gs,Gf] = freqsep(G,2,opt);`	`[Gs,Gf] = freqsep(G,2,SepTol=5e10);`

damp output behavior changed
Behavior change

The `damp` function output behavior has changed when the system has a pole at zero or infinity.

- Continuous time

Pole Location	Natural Frequency	Damping Ratio
`0`	`0`	`-1`
`Inf`	`Inf`	`-1`

- Discrete time

Pole Location	Natural Frequency	Damping Ratio
`0`	`Inf`	`+1`
`1`	`0`	`-1`
`Inf`	`Inf`	`-1`

Numerically, there is no difference between stable and unstable infinite poles. Therefore, the function considers all infinite poles as unstable.

Previously, the function returned `NaN` for these cases.

step and impulse: Response characteristics computation changes
Behavior change

The computation of some response characteristics has changed when using a nondefault configuration created using `RespConfig`.

- The **Peak Response** characteristic now shows the maximum deviation from the initial output value. That is, the peak value of $|y(t) - y_{init}|$ when $t \geq t_0 + t_d$, where y_{init} is the output value just before the step change or impulse.

  As a result, in the **Peak Response** data tip, **Peak amplitude** is renamed to **Peak deviation**.
- Settling time and transient time are now computed relative to the time when the step change or impulse occurs ($t = t_0 + t_d$). This ensures that these characteristics are unaffected by the delay.

In general, the software computes the response characteristics from the time of the step change or impulse, with the initial output value y_{init} as the value just before the step change or impulse.

R2023a

lpvss and ltvss: Support for linear parameter-varying and linear time-varying state-space models

Use the new `lpvss` and `ltvss` objects to represent linear parameter-varying (LPV) and linear time-varying (LTV) state-space models, respectively. These models can approximate nonlinear systems and allow you to efficiently apply linear design techniques to nonlinear models. For more information, see the `lpvss` and `ltvss` reference pages.

You can use `ssInterpolant` to build a gridded LTV or LPV model that interpolates local LTI behaviors into global LTV or LPV behavior from an array of state-space (`ss`) models sampled in time or parameter space. To obtain such an array of `ss` models, you can use the `linearize` (Simulink Control Design) command to linearize a nonlinear Simulink model over a grid of operating conditions. For an example, see Approximate Nonlinear Aircraft Pitch Dynamics Using LPV Model.

Additionally, you can:

- Use the `sample` command to sample the parameter-varying or time-varying dynamics at a single point or a grid.
- Use `setTestValue` and `getTestValue` to manage the test values used to validate the data function.
- Use the `step`, `impulse`, `lsim`, and `initial` commands to visualize and analyze the time-domain response of LTV and LPV models.
- Use signal-based connections to combine models and configure your control system. For more information, see Model Interconnection.
- Convert between continuous-time and discrete-time and resample `ltvss` and `lpvss` models using `c2d`, `d2c`, and `d2d`.

For more information on LTV and LPV models, see LTV and LPV Modeling and Using LTV and LPV Models in MATLAB and Simulink. For examples, see LPV and LTV Models.

⚠Time-Domain Response Commands: Support for LPV and LTV models

You can visualize and analyze the time-domain response of the new `lpvss` and `ltvss` objects using the `step`, `impulse`, `lsim`, and `initial` commands.

For `lpvss` models, you can specify the parameter trajectory `p` using the new syntax `step(lpvSys,t,p)` or similar. These functions also return the parameter trajectory information as an output argument for LPV models.

Additionally:

- `initial` lets you specify initial conditions for both states and parameters for LPV models.
- `step` and `impulse` allow you to specify a nonzero start time and initial conditions. For more information, see “RespConfig: Specify configuration for step or impulse responses” on page 7-3.

For more information and examples, see the `step`, `impulse`, `lsim`, and `initial` reference pages.

⚠Compatibility Considerations

As a result of these changes to `step` and `impulse`, they now support a nonzero start time for the input signal. For more information, see “step and impulse: Support for nonzero start time” on page 7-4.

⚠RespConfig: Specify configuration for step or impulse responses

You can now use the new `RespConfig` object to configure the responses created using `impulse` and `step` commands. Using this object, you can specify:

- Input signal offset and level change amplitude
- Nonzero start time for responses
- Initial state values for state-space models, including the new `ltvss` and `lpvss` models
- Initial parameter values for `lpvss` models

⚠Compatibility Considerations

`RespConfig` replaces `stepDataOptions` for creating a response configuration for the `step` command. For more information, see “stepDataOptions is not recommended” on page 7-4.

Extended Kalman Filter (EKF): Use automatic differentiation (autodiff) to generate Jacobian state transition and measurement functions

In the `extendedKalmanFilter` object, you can now use automatic differentiation (autodiff) techniques to generate the Jacobian functions of the state transition and measurement functions. Previously, to specify custom analytical Jacobian functions, you had to write the functions yourself.

To automatically generate state transition and measurement Jacobian functions for an `extendedKalmanFilter` object, see the `generateJacobianFcn` function.

Linear Analysis Plots: Modify time and frequency vectors for existing linear analysis plots

You can now modify the time and frequency vectors for existing linear analysis plots.

- For `step` and `impulse` plots, you can specify the time vector by right-clicking the plot area and selecting **Specify time**.
- For `bode`, `nyquist`, `nichols`, and `sigma` plots, you can specify the frequency vector by right-clicking the plot area and selecting **Specify frequency**.

For more information on plotting responses, see Plotting System Responses.

Varying Parameter Blocks: Disable feedthrough term for zero-feedthrough blocks

Clear the new **Enable feedthrough** checkbox to create zero-feedthrough state-space models and transfer functions with the following Simulink blocks:

- Varying Transfer Function
- Varying State Space
- Discrete Varying Transfer Function
- Discrete Varying State Space

Disabling the feedthrough port for zero-feedthrough models is numerically more reliable than feeding a zero constant into the feedthrough port.

New Example: Detect Attack in Cyber-Physical Systems Using Dynamic Watermarking

The new example Detect Attack in Cyber-Physical Systems Using Dynamic Watermarking shows how to use dynamic watermarking to detect attacks in a cyber-physical system.

⚠Functionality being removed or changed

Varying Transfer Function block formula changed
Behavior change

The Varying Transfer Function block formula has been changed to make the polynomial coefficient definitions consistent with the Discrete Varying Transfer Function block. Block diagrams to which the block was added in R2022b or earlier are unaffected by this change.

- A Varying Transfer Function block added to a Simulink model in R2023a or later has an instantaneous transfer function described by:

  $$H(s) = \frac{b_0 + b_1 s^{-1} + \cdots + b_N s^{-N}}{1 + a_1 s^{-1} + \cdots + a_N s^{-N}} = \frac{b_0 s^N + b_1 s^{N-1} + \cdots + b_N}{s^N + a_1 s^{N-1} + \cdots + a^N}.$$

- A block added in R2022b or before has an instantaneous transfer function described by:

  $$H(s) = \frac{b_0 + b_1 s + \cdots b_N s^N}{a_0 + a_1 s + \cdots a_{N-1} s^{N-1} + s^N}.$$

  Note that replacing a pre-R2023a block with the new block requires rewiring the coefficient inputs.

stepDataOptions is not recommended
Still runs

`stepDataOptions` command is not recommended. Use `RespConfig` to specify a response configuration for the `step` command instead.

If you create an options set using `stepDataOptions` to specify the `InputOffset` and `Amplitude` properties, the software now creates a `RespConfig` object instead, setting those properties. The remaining `RespConfig` properties are set to default values. For more information, see `RespConfig`.

step and impulse: Support for nonzero start time
Behavior change

You can now specify a nonzero start time for the `step` and `impulse` commands using a time vector input of the form `T0:dt:Tf`. Previously, the commands always applied the input at `t = 0`, regardless of `T0`.

R2022b

Version: 10.12

New Features

Bug Fixes

Create Plot Live Editor Task: Create linear analysis response plots interactively and generate code

You can now use the Create Plot Live Editor task to interactively create linear analysis response plots for your dynamic system model. The task also automatically generates code that becomes part of your live script.

For more information and an example, see the Create Plot Live Editor Task reference page.

interface: New option to obtain primal-assembly model

`interface` now includes a new optional argument to specify the method of physical coupling as either the dual-assembly method or primal-assembly method. The primal-assembly method uses a minimal number of degrees of freedom, but the system may suffer from fill-in. Use the syntax `interface(___,method)` to specify an assembly method of physical coupling, where `method` is either `'dual'` or `'primal'`. If you do not specify a method, the function uses the dual-assembly method by default. Previously, the function always returned a dual-assembly model.

For more information, see Algorithms.

New Examples: Power electronics control design

This release includes the following new reference examples.

- Feedback Amplifier Design for Voltage-Mode Boost Converter — This example illustrates tuning the components of a power supply controller to control the output voltage of a boost converter using loop-shape design and fixed-structure tuning methods. The workflow is demonstrated using a type-III controller. You need a Mixed-Signal Blockset™ license to run this example.
- Detect Replay Attacks in DC Microgrids Using Distributed Watermarking — This example shows how to use distributed watermarking to detect replay attacks in a cyber-physical system.

R2022a

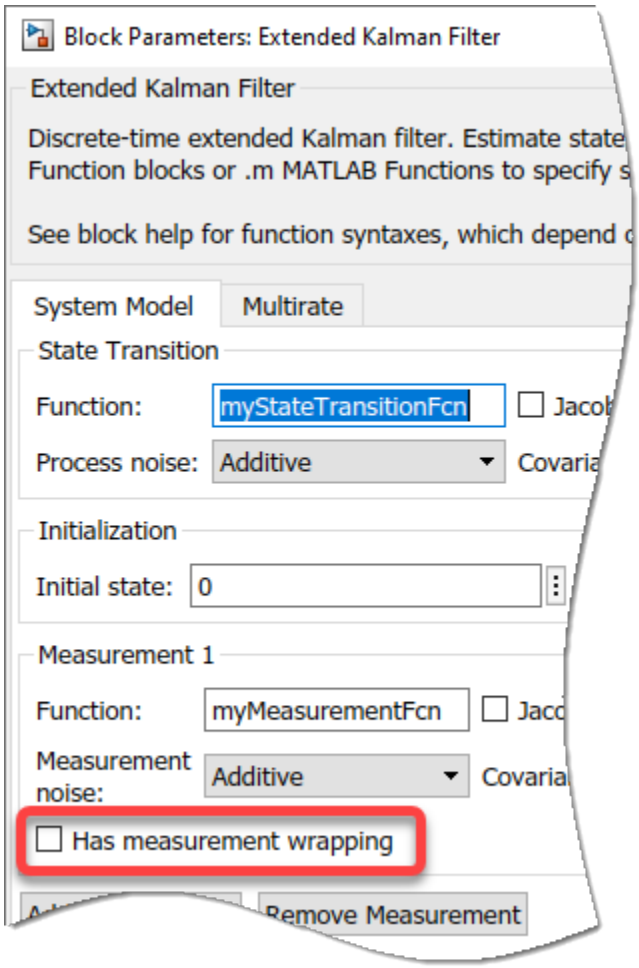
Extended and Unscented Kalman Filters: Support for measurements with circular wrapping

You can now use wrapped measurements with the `extendedKalmanFilter` and `unscentedKalmanFilter` objects by enabling the `HasMeasurementWrapping` property, and specifying a measurement function (through the `MeasurementFcn` property) with two outputs:

1. The measurement, specified as an N-element output measurement vector of the nonlinear system at time step k, given the state vector at time step k. N is the number of measurements of the system.
2. The measurement wrapping bounds, specified as an N-by-2 matrix where, the first column provides the minimum measurement bound and the second column provides the maximum measurement bound.

Enabling the `HasMeasurementWrapping` property wraps the measurement residuals in a defined bound, which helps to prevent the filter from divergence due to incorrect measurement residual values. This property is nontunable and can be set only during the object creation.

You can also enable measurement wrapping in the Extended Kalman Filter and Unscented Kalman Filter blocks using the **Has measurement wrapping** check box.



For an example, see State Estimation with Wrapped Measurements Using Extended Kalman Filter.

Functionality being removed or changed

Renamed plot options objects
Behavior change

The following plot options objects have been renamed for R2022a:

Creation Function	Old Name	New Name
`timeoptions`	`TimePlotOptions` object	`TimeOptions` object
`bodeoptions`	`BodePlotOptions` object	`BodeOptions` object
`pzoptions`	`PZMapOptions` object	`PZOptions` object
`nyquistoptions`	`NyquistPlotOptions` object	`NyquistOptions` object
`hsvoptions`	`HSVPlotOptions` object	`HSVOptions` object
`sigmaoptions`	`SigmaPlotOptions` object	`SigmaOptions` object
`nicholsoptions`	`NicholsPlotOptions` object	`NicholsOptions` object

Default value of bdschur input argument CONDMAX changed
Behavior change

The default value of `bdschur` input argument `CONDMAX` is now `1e4`. Previously, the default value was `1/sqrt(eps)`.

`CONDMAX` specifies an upper bound on the condition number of the transformation matrix *T*. Increasing `CONDMAX` reduces the size of the eigenvalue clusters in the transformed model but also decreases the accuracy of the transformation. The new default value balances the conditioning and transformation accuracy. If you have code that relies on the default value of `CONDMAX` being `1/sqrt(eps)`, update your code to explicitly set this input argument.

Default value of canon input argument condt changed
Behavior change

The default value of `canon` input argument `condt` is now `1e4`. Previously, the default value was `1e8`.

`condt` specifies an upper bound on the condition number of the transformation matrix *T*. Increasing `condt` reduces the size of the eigenvalue clusters in the transformed model but also decreases the accuracy of the transformation. The new default value balances the conditioning and transformation accuracy. If you have code that relies on the default value of `condt` being `1e8`, update your code to explicitly set this input argument.

R2021b

Frequency-Domain Analysis: Support for models with complex coefficients

You can now visualize and analyze frequency-domain responses of complex-coefficient systems using `bode`, `margin`, `nyquist`, `nichols`, `sigma`, `passiveplot`, and `sectorplot` functions. You can also now specify negative frequencies in the frequency vector `w` when using the syntax `bode(sys,w)` or similar.

For `bode`, `margin`, `sigma`, `passiveplot`, and `sectorplot`, when you plot complex-coefficient systems in:

- Log frequency scale, the plots show two branches, one for positive frequencies and one for negative frequencies. The arrows indicate the direction of increasing frequency values for each branch.
- Linear frequency scale, the plots show a single branch with a symmetric frequency range centered at a frequency value of zero. If you specify a frequency range of $[w_{min},w_{max}]$ for your plot, the frequency limits are set to $[–w_{max},w_{max}]$.

For an example, see Bode Plot of Model with Complex Coefficients .

The `nyquist` plot now shows a contour comprised of both positive and negative frequencies. The arrows indicate the direction of increasing frequency for each branch. For complex-coefficient systems, the two branches are not symmetric. For real-coefficient systems, the negative branch is obtained by symmetry. For an example, see Nyquist Plot of Model with Complex Coefficients

For complex-coefficient models, a `nichols` plot now shows a contour comprised of both positive and negative frequencies. For real-coefficient models, the plot shows only positive frequencies, even when complex-coefficient models are present. For an example, see Nichols Plot of Model with Complex Coefficients.

frd: Support for negative frequencies

You can now specify negative frequency points in a frequency-response data (`frd`) model object. `frd` models with negative frequencies are used to represent models with complex coefficients. Previously, the negative frequencies were not supported.

⚠stepinfo and lsiminfo: Support for nonzero initial value

The `stepinfo` and `lsiminfo` commands now let you specify an initial value for your response data. Previously, the commands always assumed a default initial value to be zero.

To compute step-response characteristics when you have a nonzero initial value, use the following syntax.

```
S = stepinfo(y,t,yfinal,yinit)
```

This command computes step-response characteristics from an array of step-response data `y`, corresponding time vector `t`, steady-state value `yfinal`, and initial value `yinit`. For more information, see `stepinfo`.

To compute linear response characteristics when you have a nonzero initial value, use the following syntax.

```
S = lsiminfo(y,t,yfinal,yinit)
```

This command computes response characteristics from an array of linear response data `y`, corresponding time vector `t`, steady-state value `yfinal`, and initial value `yinit`. For more information, see `lsiminfo`.

Additionally, the output structures of `stepinfo` and `lsiminfo` now contain a `TransientTime` field. Use this characteristic to measure how quickly the transients die off. This characteristic also applies to any response, including responses with zero final values (`impulse` and `initial`).

Compatibility Considerations

As a result of these changes to `stepinfo` and `lsiminfo`, the computation of some response characteristics has changed. For more information, see “stepinfo and lsiminfo: Response characteristics computation changes” on page 10-4.

mechss and sparss: Linearize structural and thermal models to sparse models

You can now obtain `mechss` and `sparss` model objects from structural and thermal models using the `linearize` (Partial Differential Equation Toolbox) function. This function also lets you extract finite element matrices used to obtain the `mechss` and `sparss` models. For examples, see Linear Analysis of Cantilever Beam and Linear Analysis of Tuning Fork.

Functionality being removed or changed

Support for opening SISO Design Tool sessions saved before release R2016a has been removed

Errors

Support for opening SISO Design Tool sessions saved before release R2016a will be removed in release R2021b.

If you have sessions saved before release R2016a, open and resave the session files using Control System Designer in any release from R2016a through R2021a.

ss2ss now returns different transformation results for descriptor state-space models

Behavior change

`ss2ss` performs the similarity transformation $\bar{x} = Tx$ on the state vector x and produces the equivalent state-space model `sysT`. For descriptor state-space models `ss2ss` now returns a different transformation result.

For a descriptor state-space model

$$E\dot{x} = Ax + Bu$$
$$y = Cx + Du,$$

`ss2ss` now returns

$$ET^{-1}\dot{\bar{x}} = AT^{-1}\bar{x} + Bu$$
$$y = CT^{-1}\bar{x} + Du\,.$$

Previously, the function returned the following transformation.

$$TET^{-1}\dot{\bar{x}} = TAT^{-1}\bar{x} + TBu$$
$$y = CT^{-1}\bar{x} + Du$$

For more information, see `ss2ss`.

ss2ss: Similarity transformation is no longer supported for mechss models
Errors

`ss2ss` no longer supports sparse second-order (`mechss`) models. Performing similarity transformations on `mechss` models destroys symmetry and has no obvious general form.

stepinfo and lsiminfo: Response characteristics computation changes
Behavior change

Because of changes to `stepinfo` and `lsiminfo`, the computation of some response characteristics has changed. Additionally, the settling time calculation is now based on how quickly the response gets within a specified threshold of the final value.

The following table summarizes the changes to the fields of the structure returned by `stepinfo`.

Before R2021b	R2021b
`RiseTime` — Time it takes for the response to rise from 10% to 90% of the way from $y(1)$ to y_{final}.	`RiseTime` — Time it takes to go from 10% to 90% of the way from y_{init} to y_{final}.
`SettlingTime` — The first time T such that the error $\lvert y(t) - y_{final}\rvert \le SettlingTimeThreshold \times e_{max}$ for $t \ge T$, where e_{max} is the maximum error $\lvert y(t) - y_{final}\rvert$ for $t \ge 0$. By default, $SettlingTimeThreshold = 0.02$ (2% of the peak error). `SettlingTime` measures the time for the error to fall below 2% of the peak value of the error.	`SettlingTime` — The first time T such that the error $\lvert y(t) - y_{final}\rvert \le SettlingTimeThreshold \times \lvert y_{final} - y_{init}\rvert$ for $t \ge T$. By default, `SettlingTime` measures the time it takes for the error to stay below 2% of $\lvert y_{final} - y_{init}\rvert$.
`Peak` — Peak absolute value of $y(t)$.	`Peak` — Peak absolute value of $y(t) - y_{init}$.

The following table summarizes the changes to the fields of the structure returned by `lsiminfo`.

Before R2021b	R2021b
`SettlingTime` — The first time T such that the error $\lvert y(t) - y_{final}\rvert \le SettlingTimeThreshold \times e_{max}$ for $t \ge T$, where e_{max} is the maximum error $\lvert y(t) - y_{final}\rvert$ for $t \ge 0$. By default, $SettlingTimeThreshold = 0.02$ (2% of the peak error). `SettlingTime` measures the time for the error to fall below 2% of the peak value of the error.	`SettlingTime` — The first time T such that the error $\lvert y(t) - y_{final}\rvert \le SettlingTimeThreshold \times \lvert y_{final} - y_{init}\rvert$ for $t \ge T$. By default, `SettlingTime` measures the time it takes for the error to stay below 2% of $\lvert y_{final} - y_{init}\rvert$.

Additionally, the output structures of `stepinfo` and `lsiminfo` now contain a `TransientTime` field. This characteristic contains the peak-error-based settling time calculation used in releases before R2021b. Transient time is used to measure how quickly the transient dynamics die off.

Here:

- $y(t)$ is the system response.
- y_{init} is the initial value of $y(t)$ before the response occurs. By default, $y_{init} = 0$.
- y_{final} is the final value of $y(t)$. By default, y_{final} = last sample value of $y(t)$.
- *SettlingTimeThreshold* is the threshold for defining settling time. By default, *SettlingTimeThreshold* = 0.02.

These changes also apply to the characteristics of `step`, `impulse`, and `initial` plots. Additionally:

- For `step` plots, y_{init} is always assumed to be zero and y_{final} is the steady-state value.
- For the step response, transient time and settling time tend to differ for models with feedthrough, zeros at the origin, unstable zeros (undershoot), or large overshoot. They match for models with no undershoot or feedthrough, and with less than 100% overshoot.
- For the step response of models with feedthrough, the new `RiseTime` value can differ because $y(1)$ is nonzero whereas y_{init} is zero by default. Before R2021b, the rise time computed was the time it takes to go from 10% to 90% of the way from $y(1)$ to y_{final}, instead of y_{init} to y_{final} now.

R2021a

Model Reduction: Compute reduced-order models with small relative-error

You can now use the Balanced Stochastic Truncation method in `balred`. This method lets you control the relative error of the approximation and preserves roll-off characteristics of the original model. Use the new `ErrorBound` option in `balredOptions` to choose between absolute or relative error control. The `Regularization` option ensures a well-defined relative error at all frequencies.

- `ErrorBound` — You can choose between absolute and relative error approximation. Previously, you could only use absolute error approximation when performing model reduction. Relative error approximation is useful when your original model has a very high or low gain in the region that is important to your application. In such regions, absolute error approximations are not effective.
- `Regularization` — Use this option to choose the regularization level that ensures a well-defined relative error at all frequencies.

For more information, see `balredOptions`.

Relative-error approximation is also available in the Model Reducer app. To use it, in the **Balanced Truncation** tab, in the **Error Bound** drop-down list, select `relative`. You can also specify the regularization value in the **Regularization** field inside the **Options** menu in the **Balanced Truncation** tab.

For more information, see Balanced Truncation Model Reduction.

Compatibility Considerations

`balred` and `balredOptions` replaces the `hsvd` and `hsvdOptions` commands to compute and plot Hankel singular values and includes options that preserves roll-off characteristics from the original model. The `hsvd` and `hsvdOptions` commands are not recommended. For more information, see the *Functionality being removed or changed* section.

Model Reduction: Compute and plot Hankel singular values (HSV) using balred

You can now use `balred` to compute and plot the Hankel singular values (HSVs) and reduce the model with this information based on your desired fidelity. Previously, you had to use `hsvd` to compute the HSVs and then extract the reduced model-order approximation using `balred`. Additionally, `balred` can also return the error bounds, regularization level and the Cholesky factors of the gramians.

For more information, see `balred`.

Compatibility Considerations

`balred` replaces `hsvd` to compute and plot Hankel singular values. The `hsvd` command is not recommended. For more information, see the *Functionality being removed or changed* section.

⚠inv and imp2exp: Implicit form returned for state-space models by default

The `inv` and `imp2exp` functions now return the implicit form for `ss`, `genss` and `uss` (Robust Control Toolbox) model objects, by default. When the inverse is proper, you can use `inv(sys,'min')` and `B = imp2exp(A,yidx,uidx,'min')` to eliminate the extra states and obtain a model with as many states as `sys` or `A` respectively.

Use `isproper` or `ss(sys,'explicit')` to extract an explicit model if desired.

For more information, see `inv` and `imp2exp`.

⚠Compatibility Considerations

If your code uses the `inv` and `imp2exp` commands with state-space models, then it may produce results that are different from the results you obtained using previous versions.

Control System Designer: Option to disable automatic plot update during Response Optimization

You can now disable automatic plot updates during response optimization in the Control System Designer. Use the **Update plots during optimization** checkbox in the **Response Optimization** dialog under **Tuning Methods** to disable automatic plot updates during optimization.

You can also use the **Update plots in real-time** checkbox in the **Control System Designer Preferences** dialog to enable or disable synchronous updates of the response plots while using graphical tuning methods.

Disabling this option improves performance especially for multimodel control design.

State-Space Models: New Property to Support State Block Path Management in Linearization

The state-space model object, `ss`, now has a new `StatePath` property to facilitate state block path management in linearization.

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

c2dOptions: New Option to Specify Fit Order

You can now use the new `FitOrder` option to specify the fit order when using the `least-squares` method for continuous-discrete conversion. `FitOrder` specifies the order of the discrete-time model to be fitted to the continuous-time frequency response in the options set obtained using `c2dOptions`. Reducing the order helps with unstable poles or pole/zero cancellations at `z = -1`. Previously, some discretized models had pole-zero cancellations at `z = 1` and `z = -1` or unstable poles at `|z| > 1`.

For more information, see the `c2dOptions` reference page.

margin: Support for Sparse State-Space Models

You can now specify a frequency vector or frequency range to plot the Bode response of `sparss` and `mechss` model objects using `margin`. Previously, `margin` did not support sparse models.

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

⚠modred, stabsep and modsep: Support for Descriptor Models

`modred`, `stabsep` and `modsep` now supports descriptor models. Previously, the commands would first convert descriptor models to an explicit form before model simplification or decomposition, respectively.

⚠Compatibility Considerations

Numerical improvements to the algorithms used by `modred`, `stabsep` and `modsep` commands might produce results that are different from the results you obtained using previous versions for descriptor models.

⚠Functionality being removed or changed

hsvd and hsvdOptions are not recommended
Still runs

The `hsvd` and `hsvdOptions` commands are not recommended. Use `balred` to obtain the Hankel singular values (HSV) of dynamic system and also compute reduced-order model approximations using the same command, and `balredOptions` to create the required HSV option set. `balredOptions` also includes new options that preserve roll-off characteristics.

The following table shows some typical uses of `hsvd` and `hsvdOptions`, and how to update your code to use `balred` and `balredOptions` instead.

Not Recommended	Recommended
`hsv = hsvd(sys)`	`[~,info] = balred(sys)` computes the Hankel singular values and the error approximations of `sys`. For more information, see `balred`.
`hsv = hsvd(sys,opts)`	`[~,info] = balred(sys,opts)` computes the Hankel singular values with options specified in the option set `opts`. For more information, see `balred` and `balredOptions`.
`hsvd(sys)`	`balred(sys)` displays the Hankel singular values and approximation error on a plot. For more information, see `balred`.
`opts = hsvdOptions(Name,Value)`	`opts = balredOptions(Name,Value)` creates the option set with the specified options. For more information, see `balredOptions`.

balredOptions option StateElimMethod renamed to StateProjection
Still runs

The `StateElimMethod` option of the `balredOptions` command has been renamed to `StateProjection`. This option specifies the method of eliminating weakly coupled states for model reduction. For more information, see the 'StateProjection' option in the `balredOptions` reference page.

Compatibility Considerations

If your code uses this option, consider modifying it to use the new option name.

Options to set error tolerance has changed in the balredOptions, stabsepOptions and freqsepOptions commands
Still runs

The `AbsTol` and `RelTol` options for the `balredOptions`, `freqsepOptions` and `stabsepOptions` commands have been replaced by `SepTol`. Use this option to specify the accuracy loss factor and increasing `SepTol` helps separate modes straddling the stable/unstable or slow/fast boundary at the expense of accuracy. Previously, you had to use two options to achieve the same result. For more information, see the `balredOptions`, `freqsepOptions` and `stabsepOptions` function reference pages.

Compatibility Considerations

If your code uses the `AbsTol` and `RelTol` options, consider modifying it to use the new `SepTol` option.

The following table shows how to update your code to use the `SepTol` option instead.

Not Recommended	Recommended
`opts = balredOptions(...,'AbsTol',0.01,'RelTol',0.02)`	`opts = balredOptions(...,'SepTol',1e3)` creates the option set with the accuracy loss factor. For more information, see `balredOptions`.
`opts = stabsepOptions(...,'AbsTol',0.01,'RelTol',0.02)`	`opts = stabsepdOptions(...,'SepTol',1e3)` creates the option set with the accuracy loss factor. For more information, see `stabsepOptions`.
`opts = freqsepOptions(...,'AbsTol',0.01,'RelTol',0.02)`	`opts = freqsepdOptions(...,'SepTol',1e3)` creates the option set with the accuracy loss factor. For more information, see `freqsepOptions`.

Kalman Filter block: Numerical changes
Behavior change

Numerical improvements in the algorithms used by the Kalman Filter block might produce results that are different from the results you obtained using previous versions.

modred, stabsep and modsep: Numerical changes
Behavior change

Numerical improvements to the algorithms used by `modred`, `stabsep` and `modsep` commands might produce results that are different from the results you obtained using previous versions for descriptor models.

Support for opening SISO Design Tool sessions saved before release R2016a will be removed
Warns

Support for opening SISO Design Tool sessions saved before release R2016a will be removed in release R2021b.

If you have sessions saved before release R2016a, open and resave the session files using Control System Designer in any release from R2016a through R2021a.