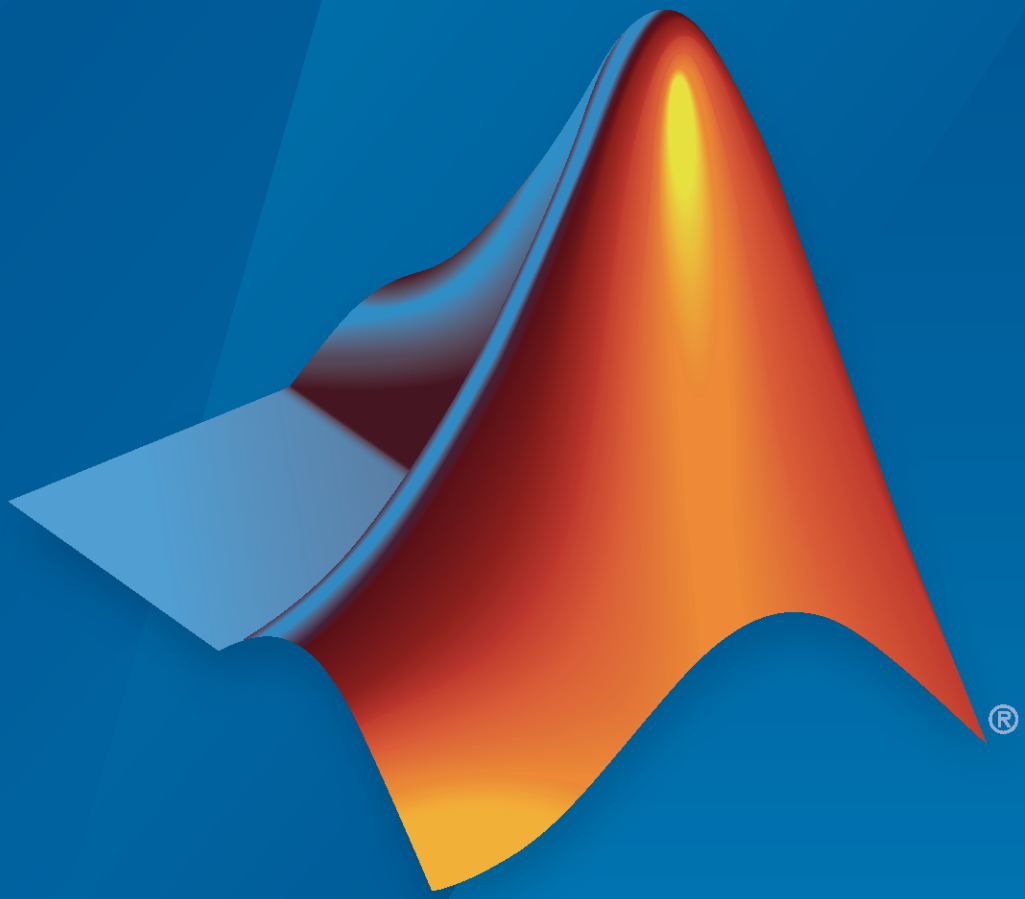
System Identification Toolbox™ Release Notes



MATLAB® & SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

System Identification Toolbox™ Release Notes

Contents

R2023a

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2019a

R2018b

R2018a

R2017b

R2017a

R2016b

R2016a

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

Bug Fixes

R2012b

R2012a

R2009b

R2009a

R2008b

R2008a

R2007b

R2007a

R2006b

R2006a

R14SP3

No New Features or Changes

R2023a

Estimation Data Types: Auxiliary functions expanded to accept estimation data in timetable, numeric matrix, and iddata formats

You can now apply legacy system identification toolbox auxiliary functionality, such as plotting and model information retrieval, using estimation data represented by timetables, numeric matrices, and legacy `iddata` objects. This change expands and completes the set of system identification functions, first introduced in R2022b, that accept estimation data in timetable and matrix formats.

In addition, a new function, `iddata2timetable`, has been added to allow you to convert easily from `iddata`-based data to `timetable`-based data. You can also convert timetable data `tt` to an `iddata` object `z` by using the command `z = iddata(tt)`.

Compatibility Considerations

Commands that were methods of `iddata` prior to R2023a are now standalone MATLAB® functions. Some command names have changed to disambiguate them from other existing MATLAB commands, and in one case, the system identification function has been superseded by the MATLAB version. The commands with the original names continue to work with `iddata` objects. For information on these commands, see “Use of previous iddata-only commands and utilities for these commands is not recommended” on page 1-3. For commands that have been converted from methods to standalone functions without name change, see “Methods of iddata converted to standalone functions” on page 1-4.

Idmodel Simulink Block: Specify initial conditions for all supported linear models using initialCondition object

In the Idmodel block, you can now specify initial conditions for any linear model that the block supports, such as the polynomial (`idpoly`), transfer function (`idtf`), and process (`idproc`) models. Previously, you could specify only initial states, which are applicable to the state-space (`idss`) and linear grey box (`idgrey`) models only. To support specifying initial conditions for these additional models, the block now accepts `initialCondition` objects in addition to numeric state vectors.

To specify initial conditions as an `initialCondition` object:

1. In MATLAB, estimate the initial conditions for your linear model and return the conditions as an `initialCondition` object. For details, see “Estimate Initial Conditions for Simulating Identified Models”.
2. In the Idmodel block, specify the workspace variable name of the `initialCondition` object in the **Initial conditions** parameter.

To denote zero initial conditions, specify the **Initial conditions** parameter as `0` or `[]`.

You can continue to specify initial states as a numeric vector for state-space models (`idss` and `idgrey` models).

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.

Nonlinear Models: Perform data normalization and regressor sparsification in the System Identification app

You can now control data normalization when you estimate nonlinear ARX and Hammerstein-Wiener models in the app. This capability was introduced at the command line for R2022a.

You can also apply regressor sparsification to search for the optimal set of regressors in nonlinear ARX models. This capability was introduced at the command line for R2022b.

To specify normalization options, in the **Estimation Options** tab of either the **Estimate Nonlinear ARX Models** or **Estimate Hammerstein Wiener** dialog box, in **Normalization Options**, select **Normalize** and specify the parameters. For more information about the normalization parameters, see the `Normalize` and `NormalizationOptions` options in `nlarxOptions` and `nlhwOptions`.

To specify sparsification options for nonlinear ARX model estimation, in the **Estimation Options** tab, select **Sparsify Regressors** and specify parameters. For more information about the sparsification parameters, see the `SparsifyRegressors` and `SparsificationOptions` argument descriptions in `nlarxOptions`.

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 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 1-4.

Functionality being removed or changed

Use of previous iddata-only commands and utilities for these commands is not recommended

Still runs

The following commands have been renamed for R2023a. Those commands that use estimation data directly have expanded to accept timetable-based and matrix-based estimation data. Other commands that were previously methods or utilities related to estimation functions are now standalone functions.

Old Name	New Name	Purpose
`plot`	`idplot`	Plot input/output data.
`idnlarx/plot`	`nlarxPlot`	Plot nonlinearity of nlarx model.
`idnlhw/plot`	`nlhwPlot`	Plot input and output nonlinearity and linear responses of Hammerstein-Wiener model.
`iddataPlotOptions`	`dataPlotOptions`	Plot options.
`feedback`	`checkFeedback`	Identify possible feedback data.

The previous commands continue to work with `iddata` objects.

Methods of iddata converted to standalone functions

The following commands have been converted from methods of `iddata` to standalone functions with the same name, and expanded to accept time-domain data in the form of timetables and matrices, in addition to `iddata` objects. Frequency-domain data continues to be packaged in data objects.

Name	Purpose
`advice`	Analysis and recommendations for data or estimated linear models.
`findstates`	Estimate initial states of model.
`data2state`	Map past data to states of state-space and nonlinear ARX models.
`getreg`	Regressor expressions and numerical values in nonlinear ARX model.
`idfilt`	Filter data using user-defined passbands, general filters, or Butterworth filters.
`idresamp`	Resample time-domain data by decimation or interpolation.
`misdata`	Reconstruct missing input and output data.
`nkshift`	Shift data sequences.
`pexcit`	Test level of excitation of input signals.

resample is not recommended
Still runs

The `resample` command is not recommended. Use `idresamp` instead to resample estimation data in the form of timetables, numeric matrices, and `iddata` objects, with or without Signal Processing Toolbox™.

`resample` continues to work with `iddata` objects. There are no plans to remove `resample` at this time.

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.

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: 10.0

New Features

Bug Fixes

Nonlinear System Identification: Create nonlinear state-space models using deep networks from Deep Learning Toolbox

You can now identify nonlinear dynamical systems, in their state-space form, using neural networks. The resulting nonlinear system can then be used for simulation, reduced order modeling, and control applications.

To identify a system from experiment data, first create a neural state-space object with a given number of states, inputs, and outputs using `idNeuralStateSpace`. Next, create neural networks to approximate the state and output functions using `createMLPNetwork`. Then, train the networks using `nlssest`, supplying time-domain experiment data, the neural state-space object, and training options as input arguments.

After the training is complete, you can validate the resulting state-space system by simulating it against real data, and then use the system to generate deployment code or to design a control system. For example, to facilitate Nonlinear MPC (Model Predictive Control Toolbox) design, generate the state transition and Jacobian functions using `generateMATLABFunction`.

Identifying neural state-space systems requires the Deep Learning Toolbox™. For more information on nonlinear system identification, see About Identified Nonlinear Models.

Estimation Data Formats: Estimate time-domain models using data from timetable objects and numeric matrices

You can now estimate time-domain models using data from `timetable` objects and numeric matrices. Previously, for most estimation functions, you had to first construct an `iddata` object and then use that object for estimation. You can still use `iddata` objects for both time-domain and frequency-domain data. However, most system identification functions that accept `iddata` objects also accept `timetable` objects and matrices. The syntaxes are similar. For `timetable` objects, use a single object that contains both input and output data. For numeric matrices, use separate matrices for input data and output data.

For more information about typical estimation syntax and data specification, see `tfest` and `ssest`. For more information about using data objects for system identification in general, see Data Types in System Identification Toolbox.

Linear Model Estimation: Estimate state-space models from impulse response data using era

You can now identify state-space models from impulse response data using the `era` command, which implements an eigensystem realization algorithm (ERA). Previously, you needed paired input and output signal data to estimate state-space models. The ERA algorithm lets you estimate dynamic models from source data, such as structural vibration data, and use time series data to fit models for prediction.

For more information and syntax options, see `era`. For an example that uses `era`, see System Identification Using Eigensystem Realization Algorithm (ERA).

Parametric Models: Improvements in frequency-domain data handling

You can now fit parametric models to power spectrum measurements or other magnitude-only spectrum measurements. Previously, you could not accurately fit models to magnitude-only spectrum

data. The software can now reconstruct the phase by making minimum-phase assumptions about the systems represented by the magnitude-only spectra. You can use this capability in the following ways:

- Fit a transfer function to power spectrum measurements using the `spectrumest` function. This function adds the reconstructed phase before estimating the model.
- Estimate linear models using `ssest` and `tfest` with magnitude-only `idfrd` or `frd` (Control System Toolbox) objects. These functions now add phase information to the magnitude-only frequency response data, and also issue a warning so that you are aware of the addition.

Also, you can add phase information directly to your frequency response data. Use `addMinPhase` to generate phase information for any complex vector $H(w)$ for which the measured magnitude $|H|$ is available but the phase is not.

You can also estimate `armax` and `bj` time series models using frequency-domain time series data. Previously, you could estimate these models only by using time-domain data.

Nonlinear ARX Models: Search for the optimal subset of regressors

You can now search for the optimal subset of regressors to use in your nonlinear ARX model by using sparsification estimation options. With these options, the `nlarx` function uses a sparsification algorithm to exclude sparse regressors from the nonlinear mapping function that estimates the model.

To use the optimal subset of regressors in your model, set these estimation options in `nlarxOptions`:

1. Set the `SparsifyRegressors` option to `true`.
2. Use the option `SparsificationOptions` to configure the algorithm parameters, such as the choice of sparseness measure and the sparsification penalty.

R2022a

Nonlinear ARX Models: Specify periodic regressors more easily with new regressor object

You can now specify periodic regressors using the new `periodicRegressor` object when you create a regressor set for a nonlinear ARX Model. Previously, you had to use the `customRegressor` object and specify a function handle to create a periodic regressor.

Periodic regressors are sine and cosine functions of delayed input and output variables. For example, $\sin(y(t–1))$ and $\cos(y(t–1))$ are both periodic regressors with delays of one sample. A `periodicRegressor` object encapsulates a set of periodic regressors. Use `periodicRegressor` objects when you create nonlinear ARX models using `idnlarx` or `nlarx`. You can specify a combination of `periodicRegressor` objects along with `linearRegressor`, `polynomialRegressor`, and `customRegressor` objects as regressors for nonlinear ARX models.

Nonlinear ARX Models: Create mapping objects that use machine learning SVM regression models

You can now use `idSupportVectorMachine` to create support vector machine (SVM) regression models as mapping objects for nonlinear ARX models. SVM models are nonparametric supervised learning models that are used in classification and regression problems. To use this model, you must have Statistics and Machine Learning Toolbox™ software.

SVM models, like Gaussian Process (GP) models, use a kernel-based formulation to predict the output as a function of the input and output regressors. However, unlike GP models, SVM models subject the fitting process to maximum error constraints. This approach results in models that use less memory than GP models, since the SVM models need to store only a subset of their training data—the data that corresponds to the support vectors—and are more robust to outliers.

You can create and configure this object to use the desired options. Then, you can specify the object as an input argument to the `nlarx` command or as the value of the `OutputFcn` property of the `idnlarx` object. You can also specify this mapping object when you perform nonlinear ARX estimation in the **System Identification** app.

For more information, see `idSupportVectorMachine`.

Nonlinear ARX Models: Use parallel processing for regression tree ensemble mapping objects

You can now use parallel processing when you specify `idTreeEnsemble` as a mapping object in a nonlinear ARX model. Because regression tree ensemble computations are independent for each regression tree, parallel processing can significantly reduce computation time for large ensembles. To specify parallel processing for an `idTreeEnsemble` object `E`, set the property `E.EstimationOptions.UseParallel` to `true`. The default value is `false`. To use the parallel processing capability, you must have Parallel Computing Toolbox™ software.

Nonlinear ARX Models: Configure additional estimation options for regression tree ensemble mapping objects

You can now configure additional options for estimating the individual regression trees, or “weak learners”, that make up the ensemble in an `idTreeEnsemble` mapping object. Previously, the object

used default options when estimating decision trees. The new `EstimationOptions.Learners` property includes options such as the maximum number of splits in the tree, pruning optimization, and minimum number of observations per leaf. The `estimationOptions` property also includes other new option properties, such as `learnRate`, the learning rate when training an ensemble for shrinkage.

For more information, see `idTreeEnsemble`.

Hammerstein-Wiener Models: Create output nonlinearities that use Gaussian process mapping objects

You can now use `idGaussianProcess` objects as mapping objects for the output nonlinearities of Hammerstein-Wiener models. Previously, you could specify `idGaussianProcess` objects only for nonlinear ARX models. To use this object, you must have Statistics and Machine Learning Toolbox software.

You can create and configure this object to use the options you desire. Then, you can specify this object as an input argument to the `nlhw` command or as the value of the `OutputNonlinearity` property of the `idnlhw` object. You can also specify this mapping object when you perform nonlinear Hammerstein-Wiener estimation in the **System Identification** app.

For more information, see `idGaussianProcess`.

Nonlinear ARX and Hammerstein-Wiener Models: Normalize estimation data using expanded options

You can now control data normalization when you identify nonlinear ARX (`idnlarx`) and Hammerstein-Wiener (`idnlhw`) models. Normalization can help improve the numerical conditioning of the estimation algorithm and improve the reliability and generalizability of the model. Previously, the software automatically performed zero-mean normalization and allowed no user control.

You specify normalization choices by setting the new `Normalize` and `NormalizationMethod` options in `nlarxOptions` and `nlhwOptions`.

When you estimate the model, the normalization process computes the data centering and scaling vectors and stores them in the `Normalization` property of the model. You can also directly modify these values in the model if you want to use custom scaling or centering values.

For more information on how model performs normalization, see the `Normalization` property of `idnlarx` and `idnlhw`. For more information on the normalization options, see `nlarxOptions` and `nlhwOptions`.

Compatibility Considerations

The R2022a normalization approach moves data normalization from the mapping objects to the nonlinear model itself and eliminates the need for normalization-related properties in the mapping objects for nonlinear ARX models. For more information, see “Previous idnlarx mapping object normalization information moved to idnlarx Normalization property” on page 3-5.

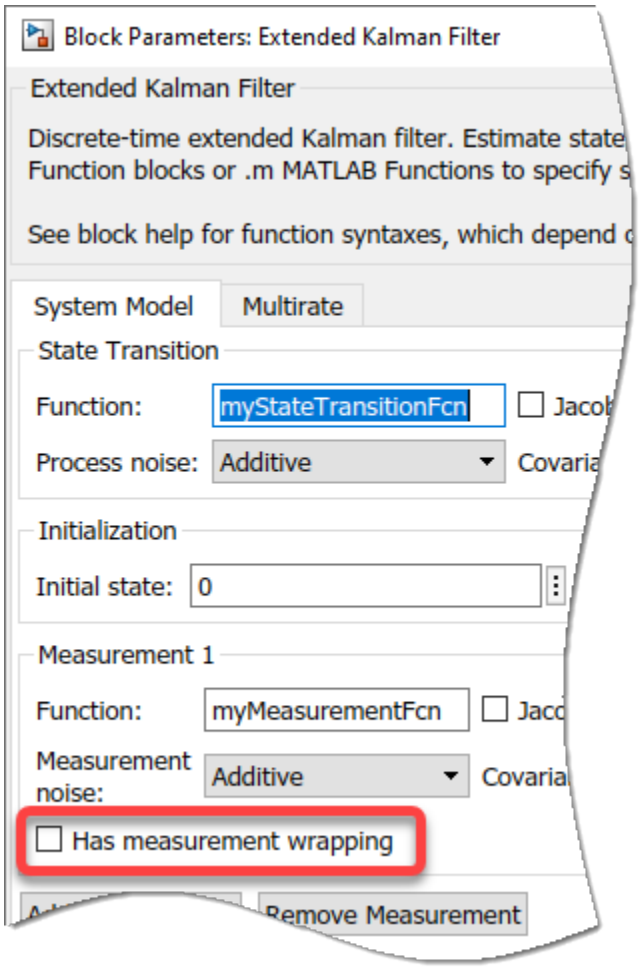
Extended and Unscented Kalman Filters: Support for measurements with circular wrapping

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

Use of previous idGaussianProcess NonLinearFcn property is not recommended
Still runs

The `NonLinearFcn` property of the `idGaussianProcess` object has been renamed to `Kernel`. The previous property name still works. There are no plans to exclude the previous name at this time.

This change has no impact on existing syntaxes for `idGaussianProcess`. If you have code that uses dot notation to directly set or view this property, consider changing your code to use the new name.

Previous idnlarx mapping object normalization information moved to idnlarx Normalization property
Behavior change

Because normalization has been moved from the mapping object level to the model level, related mapping object subproperties that contained parameters that were computed during estimation are now provided in the `Normalization` property of the `idnlarx` model.

As part of the normalization change, the regressor-selection process was also moved to the model level. The model now passes the actual regressor names rather than the selection indices to the mapping object, eliminating the need for an index property at the mapping object level.

The following table summarizes the mapping object subproperties that have been eliminated for R2022a. The information that these subproperties provided is available in the form of center and scale information in the subproperties of `Normalization`. For more information, see the `Normalization` property of `idnlarx`.

Main Properties / Subproperties	Input	Output	LinearMdl	Offset	NonlinearMdl
`Mean`	X	X			
`Range`	X	X			
`Minimum`			X	X	X
`Maximum`			X	X	X
`SelectedInputIndex`			X		X

This change applies for all mapping objects listed in Available Mapping Functions for Nonlinear ARX Models.

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

Creation Function	Old Name	New Name
`bodeoptions`	`BodePlotOptions` object	`BodeOptions` object
`pzoptions`	`PZMapOptions` object	`PZOptions` object
`nyquistoptions`	`NyquistPlotOptions` object	`NyquistOptions` object

R2021b

Nonlinear ARX Models: Create models that use regression functions based on machine learning algorithms

You can now create nonlinear ARX models using Gaussian process functions and regression tree ensembles. Gaussian process functions use a variety of kernels to implement a stochastic algorithm. Regression tree ensembles generate decision-tree structures by employing either bagging or boosting algorithms.

Two new nonlinear mapping objects implement these models. To use these objects, you must have Statistics and Machine Learning Toolbox software.

1. `idGaussianProcess` — Uses a nonparametric probabilistic technique that implements a zero-mean Gaussian process with a covariance that you specify by choosing a kernel. For more information on this technique, see Gaussian Process Regression Models (Statistics and Machine Learning Toolbox).
2. `idTreeEnsemble` — Uses a regression tree technique that implements an ensemble of several regression trees known as weak learners, each of them a decision tree with scalar-valued leaves that the algorithm derives using binary splits on regressors. For more information on this technique, see Regression Tree Ensembles (Statistics and Machine Learning Toolbox).

You can create and configure these objects to use the desired options. Then, you can specify these objects as input arguments to the `nlarx` command, or as the value of the `OutputFcn` property of the `idnlarx` object. You can also specify these mapping objects when you perform nonlinear ARX estimation in the **System Identification** app.

The following new examples estimate models using these regression functions.

- Machine-Learning-Based Identification of Two-Tank System
- Machine-Learning-Based Identification of Nonlinear Magneto-Rheological Fluid Damper
- Surrogate Modeling Using Gaussian Process-Based NLARX Model

Hammerstein-Wiener Models: Fix model linear component and nonlinearities to their known values

You can now fix parameters to their known values for the linear component and for the input and output nonlinearities in a Hammerstein-Wiener model (`idnlhw`). Previously, these parameters were always free, meaning that the software always estimated the values.

- Fix the linear component.

  The `idnlhw` object provides two new properties—`Bfree` and `Ffree`—that you can set to either fix the values of the `B` and `F` polynomial properties to their initial values or free the values so that the software estimates them. For more information, see `idnlhw`.
- Fix the input and output nonlinearities.

  All supported mapping functions provide a new property called `Free` that allows you to fix individual parameters, such as the values of `zeroInterval` in `idDeadZone`, to their initial values. For mapping functions that also support `idnlarx` objects, such as `idWaveletNetwork`, the `Free` property is available in the `LinearFcn`, `Offset`, and `NonlinearFcn` components.

For more information on these changes, see one of the `idnlhw` mapping objects, such as `idDeadZone`.

Live Editor Tasks: Interactively estimate spectral models and generate MATLAB code

Use the new **Estimate Spectral Model** Live Editor task to select input and output data, select a spectral estimation algorithm, specify the frequency vector, and plot the results. The task also automatically generates MATLAB code that becomes part of your live script. You can choose among the following algorithms:

- SPA — Implements the Blackman-Tukey spectral-analysis algorithm
- SPAFDR — Implements a variant of the Blackman-Tukey algorithm that uses frequency-dependent resolution
- Smoothed Fourier transform — Computes the ratio of the Fourier transform of the output to the Fourier transform of the input

For more information about this live task, see **Estimate Spectral Model**.

Renaming of Nonlinear Model Mapping Objects

The mapping objects used in the nonlinear components of the `idnlarx` and `idnlhw` objects have been renamed.

Compatibility Considerations

Scripts with the old names still run normally, but issue a warning. Consider using the new names for continuing compatibility with newly developed features and algorithms.

Old Name	R2021b Name
`wavenet`	`idWaveletNetwork`
`sigmoidnet`	`idSigmoidNetwork`
`treepartition`	`idTreePartition`
`customnet`	`idCustomNetwork`
`saturation`	`idSaturation`
`deadzone`	`idDeadZone`
`pwlinear`	`idPiecewiseLinear`
`poly1d`	`idPolynomial1D`
`unitgain`	`idUnitGain`
`linear`	`idLinear`
`neuralnet`	`idFeedforwardNetwork`

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`).

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 4-5.

Frequency-Domain Analysis: Support for models with complex coefficients

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

For `bode`, 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.

Functionality being removed or changed

Use of previous idnlarx and idnlhw mapping object names is not recommended.
Still runs

The mapping objects used in the `idnlarx` and `idnlhw` have been renamed. There are no plans to exclude the previous names at this time.

To update your code, replace the old name with the new name. The syntaxes are equivalent

For more information, see “Renaming of Nonlinear Model Mapping Objects” on page 4-3.

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 $\|y(t) - y_{final}\| \le SettlingTimeThreshold \times e_{max}$ for $t \ge T$, where e_{max} is the maximum error $\|y(t) - y_{final}\|$ 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 $\|y(t) - y_{final}\| \le SettlingTimeThreshold \times \|y_{final} - y_{init}\|$ for $t \ge T$. By default, `SettlingTime` measures the time it takes for the error to stay below 2% of $\|y_{final} - y_{init}\|$.
`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 $\|y(t) - y_{final}\| \le SettlingTimeThreshold \times e_{max}$ for $t \ge T$, where e_{max} is the maximum error $\|y(t) - y_{final}\|$ 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 $\|y(t) - y_{final}\| \le SettlingTimeThreshold \times \|y_{final} - y_{init}\|$ for $t \ge T$. By default, `SettlingTime` measures the time it takes for the error to stay below 2% of $\|y_{final} - y_{init}\|$.

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` (Control System Toolbox) 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

Nonlinear ARX Models: Customize your model with a more flexible model architecture that also provides improved accuracy and computation speed

The `idnlarx` model for R2021a includes architecture changes that allow you to specify the linear and nonlinear portions of your model with more ease and flexibility. New capabilities include the following:

- Choose regressors for linear and nonlinear components of the model independently. Previously, the object constrained the nonlinear component regressors to a subset of the linear regressors.
- Create linear regressors, polynomial regressors, and regressors that are based on custom formulas more easily using the new `linearRegressor`, `polynomialRegressor`, and `customRegressor` objects. These new regressor specification objects allow you to:
  - Directly specify a set of polynomial regressors by constructing a `polynomialRegressor` object. Previously, you had to create polynomial regressors by using `polyreg` and `addreg` to create an array of `customreg` objects.
  - Specify regressors with nonconsecutive lags.
  - Use the absolute values of variables in regressor formulas.
- Specify nonlinearities using an array of modernized mapping objects (for example, `sigmoidnet` and `wavenet`). These objects now allow you to do the following:
  - Retrieve information about regressor inputs and model outputs in new `Input` and `Output` properties.
  - Fix parameters during estimation.
  - Individually initialize and constrain the linear, nonlinear, and offset components of the mapping function. The object contains a separate property for each of these components.

The R2021a architecture includes reorganization of the regressor and nonlinear properties, as shown in the following table.

R2020b Properties	R2021a Properties	Description	Functional Changes
`na`, `nb`, `nk`, `CustomRegressors`	`Regressors`	Regressor specifications	New `Regressors` property stores new linear, polynomial, and custom regressor objects as a single array.
`Nonlinearity`	`OutputFcn`	Array of objects that contain a regressor-to-output mapping, such as `wavenet`	Existing property is renamed. Underlying modernized mapping functions replace previous nonlinearity estimators.
`NonlinearRegressors`	`RegressorUsage`	Specification of which regressors serve as inputs to which linear and nonlinear components of the output function	New table allows independent assignment of linear and nonlinear regressors.

For more information about these changes, see `idnlarx`.

Compatibility Considerations

The changes to `idnlarx` include changes to `idnlarx` properties or property names, mapping objects (previously known as nonlinearity estimators), and functions for creating and using regressors. Pre-R2021a objects and functions still run, but with some constraints and behavior changes. For more information, see “Functionality being removed or changed” on page 5-3.

Nonlinear Models: Use new nonlinear ARX capabilities and generate MATLAB code in the System Identification app

Two new dialog boxes replace the previous Nonlinear Models dialog box in the app. The Estimate Nonlinear ARX Models dialog box incorporates the R2021a nonlinear command-line upgrades described previously in these Release Notes.

- Estimate Nonlinear ARX Models — This dialog box, which replaces the previous Nonlinear ARX Models portion of the Nonlinear Models dialog box, allows you to specify and assign linear, polynomial, and custom regressor sets, and to configure the regressor-to-output mapping functions.
- Estimate Nonlinear Hammerstein-Wiener Models — This dialog box replaces the previous Nonlinear Hammerstein-Wiener Models portion of the Nonlinear Models dialog box while retaining the previous functionality.

For more information about these changes, see Identify Nonlinear Black-Box Models Using System Identification App.

In addition, the app now supports MATLAB code generation for nonlinear models. This capability allows you to reproduce your estimation results at the command line. To view the code after estimating a model, open the Data/model Info dialog box by double-clicking the model icon in the model board of the main **System Identification** app window. The **Diary and Notes** pane of the dialog box shows the MATLAB code that reproduces the model estimation.

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.

Functionality being removed or changed

Use of previous idnlarx properties is not recommended.
Still runs

Several properties of `idnlarx` have been modified. These changes affect the syntaxes in both `idnlarx` and `nlarx`. The use of the properties in the following table is discouraged. However, the software still accepts calling syntaxes that include these properties. There are no plans to exclude

these syntaxes at this time. The command syntax that uses ARX model orders continues be a recommended syntax.

Pre-R2021a Property	R2021a Property	Usage
ARX model orders `na,nb,nk`	`Regressors`, which can include `linearRegressor`, `polynomialRegressor`, and `customRegressor` objects.	`na,nb,nk` remains a valid `idnlarx` and `nlarx` input argument that the software converts to a `linearRegressor` object. You can no longer change order values in an existing `idnlarx` model by dot assignment or by using the `set` function. Create a new model object instead.
`customRegressors`	`Regressors`	Use `polynomialRegressor` or `customRegressor` to create regressor objects and add the objects to the `Regressors` array.
`NonlinearRegressors`	`RegressorUsage`	`RegressorUsage` is a table that contains regressor assignments to linear and nonlinear output components. Change assignments by modifying the corresponding `RegressorUsage`table entries.
`Nonlinearity`	`OutputFcn`	Change is in name only. This property remains an object or an array of objects that map regressor inputs to an output.

Use of previous nonlinearity estimator properties is not recommended
Still runs

The properties of the mapping objects, previously known as nonlinearity estimators, have been reorganized. These objects are `wavenet` (W), `sigmoidnet` (S), `treepartition` (T), `customnet` (C), and `linear` (L). The property changes do not apply to `neuralnet`. The use of the pre-R2021a properties in the following table is discouraged. However, the software still accepts commands that set these properties. There are no plans to exclude such commands at this time.

Pre-R2021a Property	R2021a Property	Applicable Mapping Objects
`NumberOfUnits`	`NonlinearFcn.NumberOfUnits`	W,S,T,C
`LinearTerm`	`LinearFcn.Use`, `Offset.Use`	W,S,C

Pre-R2021a Property	R2021a Property	Applicable Mapping Objects
`Parameters`	Split into three pieces: • `LinearFcn.Value` • `Offset.Value` • `NonlinearFcn.Parameters`	W,S,T,C,L `linear` (L) excludes `NonlinearFcn.Parameters`.
`Options`	`NonlinearFcn.Structure`	W,T

Nonlinear ARX Model Simulation and Estimation Algorithms: Numerical Changes
Behavior change

The nonlinear ARX model simulation and estimation algorithms have been updated to improve computation speed and numerical accuracy. Using the corresponding functions might produce results that are different from the results you obtained using previous versions.

getreg changes
Behavior change

`getreg` now returns the data in a `timetable` of regressor values instead of a matrix or a cell array of values. The calling syntax has also been modified, although the previous syntax still runs.

addreg, polyreg, and customreg are not recommended
Still runs

The `addreg`, `polyreg`, and `customreg` regressor commands are not recommended. Use the commands described in the following paragraphs.

- `addreg` — Add regressor objects `linearRegressor`, `polynomialRegressor`, and `customRegressor` directly to the model `Regressor` property by using the syntax `model.Regressors(end+1) = new_regressor_object`.
- `polyreg` — Use the `polynomialRegressor` command instead to construct polynomial regressors. Doing so improves the computation speed and the accuracy of results, reduces the memory footprint of the `idnlarx` object, and improves code generation in Simulink®.
- `customreg` — For polynomial regressors, use `polynomialRegressor` instead. For other custom regressors, use `customRegressor`.

There are no plans to remove `addreg`, `polyreg`, or `customreg` at this time.

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.

R2020b

Transfer Function and Polynomial Models: Estimate and apply initial conditions

You can now estimate initial conditions (ICs) for all estimated linear models, including transfer function and polynomial models, and then apply those ICs when simulating the models. Previously, you had to convert transfer function and polynomial models into state-space models first if you wanted to account for initial conditions consistent with the measurement data. For more information, see `initialCondition`.

Functionality being removed or changed

Extended and Unscented Kalman Filter Algorithms: Numerical changes

Behavior change

Numerical improvements in the algorithms used by the Extended Kalman Filter and Unscented Kalman Filter blocks and the `extendedKalmanFilter` and `unscentedKalmanFilter` functions might produce results that are different from the results you obtained using previous versions.

R2020a

Estimate Process Model Live Editor Task: Apply weighting prefilter to loss function

You can now apply a weighting prefilter to the loss function that the task minimizes when you estimate a process model. This filter accommodates a set of passbands, a SISO LTI model, or a frequency-weighting vector.

For more information, see the `Weighting Prefilter` parameter description in **Estimate Process Model**.

Estimate State-Space Model Live Editor Task: Apply weighting prefilter to loss function

You can now apply a weighting prefilter to the loss function that the task minimizes when you estimate a state-space model. This filter accommodates a set of passbands, an LTI model, and a frequency-weighting vector. If you have non-MIMO frequency-response data, you can specify filters that take the inverse or the inverse square root of the magnitude of the frequency response.

For more information, see the `Weighting Prefilter` parameter description in **Estimate State-Space Model**.

Functionality being removed or changed

goodnessOfFit: Fit result represents the error norm for all three cost functions, with a value of zero indicating a perfect fit
Behavior change

`goodnessOfFit` now returns the error norm E as the fit value for all three cost functions (MSE, NRMSE, and NMSE). Previously, `goodnessOfFit` returned the one's complement of the error norm, 1-E, for fit values that used the NRMSE or NMSE cost functions. This change allows consistent fit-value interpretation across the three cost functions, with the ideal fit value of zero representing a perfect fit.

Previously computed NRMSE and NMSE fit values are the one's complements of the fit values computed with the current software. Similarly, the NRMSE fit value is now the one's complement of the fit value that `compare` uses when reporting the percentage fit value. For example, if the previous `goodnessOfFit` fit value was 0.8, the current fit value is 0.2. A `goodnessOfFit` fit value of 0.2 is equivalent to a `compare` fit percentage of 80%.

R2019b

Version: 9.11

New Features

Bug Fixes

Live Editor Tasks: Interactively perform state-space and process model identification tasks and generate MATLAB code in a live script

Use new Live Editor tasks to select input and output data, configure model structure, specify optional parameters, and estimate a model, all without writing code. The tasks generate plots that let you interactively explore the effects of changing parameter values and options. The tasks also automatically generate code that becomes part of your live script.

In R2019b, System Identification Toolbox includes two tasks:

- **Estimate State-Space Model** — Estimate state-space models with specified or best-value plant order
- **Estimate Process Model** — Estimate process models with selectable model structure

To use tasks in the Live Editor, on the **Live Editor** tab, in the **Task** menu, select a task.

Residuals for Extended and Unscented Kalman Filters: Calculate residuals and residual covariances of filter predictions

The new function `residual` computes residuals for extended Kalman filter (EKF) and unscented Kalman filter (UKF) objects. These residuals represent the error between the filter predictions and the measurements. Use `residual` to help validate filter performance.

For more information, see `residual`.

For more information on extended and unscented Kalman filter objects, see `extendedKalmanFilter` and `unscentedKalmanFilter`.

Nyquist Plots: Programmatically Zoom on Critical Point

You can now use the `zoomcp` command to zoom programmatically on the critical point of a Nyquist plot that you create with `nyquistplot`. Previously, you could zoom the plot to the critical point by right-clicking and selecting **Zoom on (-1,0)**. Now, for a plot with handle `h`, entering the command `zoomcp(h)` achieves the same result as the right-click option. For an example, see the `nyquistplot` reference page.

R2019a

Version: 9.10

New Features

Bug Fixes

Online Parameter Estimation: Use finite-history estimation for recursive output error estimation

You can now perform recursive output error (OE) estimation using the finite-history algorithm option. Finite-history estimation is also known as sliding-window estimation, and it was introduced in R2018b for recursive AR and recursive ARX model structures. Recursive OE estimation using finite history is available in the `recursiveOE` function and the Recursive Polynomial Estimator block. For more information, see Recursive Algorithms for Online Parameter Estimation.

R2018b

Version: 9.9

New Features

Bug Fixes

Online Parameter Estimation: Use recursive model blocks in Simulink with batch data

You can now use batch data with the Simulink Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks. You can also use batch data with all the command-line recursive-estimator commands such as `recursiveLS` and `recursiveARX`. This update allows you to input data frames containing multiple samples directly to the estimators. Previously, these estimator blocks and models accepted only individual samples. So if your hardware provided data in frames, you needed to unpack the data samples yourself before passing the signal to the estimators.

You can also now perform finite-history estimation, otherwise known as sliding-window estimation. Previously, the available estimation methods all used infinite-history estimation, using data knowledge going back to the start of the simulation. Finite-history estimation is especially useful when you have rapidly changing parameters. Finite-history estimation is available for the Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks, and for the `recursiveLS`, `recursiveAR`, and `recursiveARX` commands. For more information, see Recursive Algorithms for Online Parameter Estimation.

R2018a

Particle Filter Simulink Block: Estimate states of nonlinear systems for online tracking and control system design

Perform state estimation for arbitrary nonlinear models using the new Particle Filter block in Simulink. Particle filters are flexible in comparison to Kalman filters, that is, they can also perform state estimation for nonlinear systems with non-Gaussian distributions.

Particle Filter block uses particles and sensor data to estimate the posterior distribution of the current state. The filter predicts the states using the nonlinear state transition function. Then, it corrects the estimate based on sensor data and measurement likelihood model. You can specify a fixed number of particles to use, a fixed number of state variables to estimate, and your state estimation method.

You can find the Particle Filter block in the **System Identification Toolbox > Estimators** block library in Simulink.

You can use Simulink Coder™ to deploy particle filters with multiple measurement models and fixed-size arrays for your application.

For more information on the Particle Filter block, see Particle Filter. For more information on the detailed workflow, see Parameter and State Estimation in Simulink Using Particle Filter Block.

c2d Function: Convert models to discrete-time using least-squares optimization

You can now convert continuous-time dynamic system models to discrete time using a new least-squares optimization method. This algorithm minimizes the error between the frequency responses of the continuous-time and discrete-time systems up to the Nyquist frequency. This method is particularly useful when you want to capture fast system dynamics but must use a larger sample time, for example, when computational resources are limited.

To convert a model using this approach, specify the discretization method as `'least-squares'`.

```
discreteModel = c2d(contModel,Ts,'least-squares');
```

Alternatively, you can create a `c2dOptions` option set, and set the `Method` property to `'least-squares'`. You can then use this option set with the `c2d` function.

```
options = c2dOptions('Method','least-squares');
discreteModel = c2d(contModel,Ts,options);
```

This conversion method supports only SISO models.

For more information, see `c2d` and `c2dOptions`.

ssest Function Enhancements: Identify state-space models from frequency-domain data with new estimation algorithm and additional weighting options

`ssest` has a new default algorithm for performing state-space model estimation from frequency-domain data. You are likely to see an accuracy improvement, particularly for data with dynamics over a large range of frequencies and amplitudes. This algorithm was first deployed for `tfest`.

For state-space estimation using frequency-response data in an `idfrd` object, you can now specify the estimation weighting filter as `'inv'` or `'invsqrt'`. Use the `WeightingFilter` option of `ssestOptions`. These new options enable you to specify common weighting schemes that are useful for capturing relatively low amplitude dynamics in data. For more information, see `ssestOptions`.

The new algorithm is not available for MIMO systems. In these cases, `n4sid` remains the default algorithm.

Compatibility Considerations

When estimating state-space models from frequency-domain data, the estimation results may not match results from previous releases. To perform estimation using the previous algorithm (n4sid), set the `InitializeMethod` option of `ssestOptions` to `'n4sid'`.

Renaming of Estimation and Analysis Options

Estimation and Analysis options were renamed.

Compatibility Considerations

Scripts with the old names still run normally. Consider using the new names for continuing compatibility with newly developed features and algorithms.

Option-Name Updates

Old Name	New Name	Option Sets
SearchOption	SearchOptions	armaxOptions, bjOptions, findopOptions, findstatesOptions, greyestOptions, nlarxOptions, nlgreyestOptions, nlhwOptions, oeOptions, polyestOptions, procestOptions, ssestOptions, tfestOptions
GnPinvConst	GnPinvConstant	
InitGnaTol	InitialGnaTolerance	
MaxFunEvals	MaxFunctionEvaluations	
MinParChange	MinParameterChange	
RelImprovement	RelativeImprovement	
MaxIter	MaxIterations	
TolX	StepTolerance	
TolFun	FunctionTolerance	
Old Name	**New Name**	**Option Sets**
EstCovar	EstimateCovariance	armaxOptions, arxOptions, bjOptions, greyestOptions, iv4Options, n4sidOptions, nlgreyestOptions, oeOptions, polyestOptions, procestOptions, ssestOptions, ssregestOptions, tfestOptions
Old Name	**New Name**	**Option Sets**
InitMethod	InitializeMethod	tfestOptions
InitOption	InitializeOptions	
Old Name	**New Name**	**Option Sets**
DiffScheme	DifferenceScheme	nlgreyestOptions
DiffMinChange	MinDifference	
DiffMaxChange	MaxDifference	
GradientType	Type	
Old Name	**New Name**	**Option Sets**
IterWavenet	IterativeWavenet	nlarxOptions
RegulKernel	RegularizationKernel	ssregestOptions, impulseestOptions, arxRegulOptions

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Estimation of state-space models from frequency-domain data, using the `ssest` command.	Still runs	Not applicable	`ssest` has a new default algorithm, so the estimation results may not match results from previous releases. See "ssest Function Enhancements: Identify state-space models from frequency-domain data with new estimation algorithm and additional weighting options" on page 11-2
The following estimation options have new names: • `SearchOption` • `GnPinvConst` • `InitGnaTol` • `MaxFunEvals` • `MinParChange` • `RelImprovement` • `MaxIter` • Tolx • TolFun	Still runs	• `SearchOptions` • `GnPinvConstant` • `InitialGnaTolerance` • `MaxFunctionEvaluations` • `MinParameterChange` • `RelativeImprovement` • `MaxIterations` • `StepTolerance` • `FunctionTolerance`	Consider updating scripts and functions to use the new names. See "Renaming of Estimation and Analysis Options" on page 11-3.
The following estimation options have new names: • `EstCovar`	Still runs	• `EstimateCovariance`	Consider updating scripts and functions to use the new names. See "Renaming of Estimation and Analysis Options" on page 11-3.
The following estimation options have new names: • `InitMethod` • `InitOption`	Still runs	• `InitializeMethod` • `InitializeOptions`	Consider updating scripts and functions to use the new names. See "Renaming of Estimation and Analysis Options" on page 11-3.
The following estimation options have new names: • `DiffScheme` • `DiffMinChange` • `DiffMaxChange` • `GradientType`	Still runs	• `DifferenceScheme` • `MinDifference` • `MaxDifference` • `Type`	Consider updating scripts and functions to use the new names. See "Renaming of Estimation and Analysis Options" on page 11-3.

Functionality	Result	Use This Instead	Compatibility Considerations
The following estimation options have new names: • `IterWavenet` • `RegulKernel`		• `IterativeWavenet` • `RegularizationKernel`	Consider updating scripts and functions to use the new names. See “Renaming of Estimation and Analysis Options” on page 11-3.

R2017b

Version: 9.7

New Features

Bug Fixes

Particle Filters: Estimate states of nonlinear systems for online tracking and control system design

Perform state estimation for arbitrary nonlinear system models using the new `particleFilter` command. Particle filters are flexible, that is, they can also perform state estimation for nonlinear systems with non-Gaussian distributions. Previously, you could perform state estimation only for systems with unimodal distributions using extended or unscented Kalman filters.

`particleFilter` uses particles and sensor data to estimate the posterior distribution of the current state. The filter predicts the states using the nonlinear state transition function. Then, it corrects the estimate based on sensor data and measurement likelihood model. You can specify a fixed number of particles to use, a fixed number of state variables to estimate, and your state estimation method based on the particle weights.

To use a particle filter for state estimation:

1. Create a particle filter, and set the transition and measurement likelihood functions.
2. Initialize the particle filter by specifying the number of particles to be used and your initial state guess. Also specify state bounds or covariance of the initial particle distribution.
3. Specify the state estimation and resampling method.
4. Perform state estimation.

You can use MATLAB Compiler™ or MATLAB Coder software to deploy the particle filter for your application.

For more information and examples, see the `particleFilter` reference page.

New Example on Identification Techniques for Modal Analysis

The new example Modal Analysis of a Flexible Flying Wing Aircraft shows the use of system identification techniques for modal analysis. The example shows computation of bending modes of a flexible wing aircraft. The vibration response of the wing is collected at multiple points along its span. The data is used to identify a dynamic model of the system. The modal parameters are extracted from the identified model. The modal parameter data is combined with the sensor position information to visualize the various bending modes of the wing.

Dynamic system models store Notes property as string or character vector

The `Notes` property of a dynamic system model stores any text that you want to associate with the model. This property now accepts either character-vector or `string` values, and stores whichever type you provide. For instance, if `sys1` and `sys2` are dynamic system models, you can set their `Notes` properties as follows:

```
sys1.Notes = "sys1 has a string.";
sys2.Notes = 'sys2 has a character vector.';
sys1.Notes
sys2.Notes
```

```
ans =

    "sys1 has a string."
```

```
ans =

  1×1 cell array

    {'sys2 has a character vector.'}
```

When you create a new model, the default value of `Notes` is now `[0×1 string]`. Previously, you could only specify the `Notes` property as a character vector or cell array of character vectors, and the default value was `{}`.

Some other dynamic system model properties accept strings as inputs, but store the values as character vectors or a cell array of character vectors.

R2017a

Extended and Unscented Kalman Filter Simulink Blocks: Estimate states of nonlinear systems for online tracking and control system design

You can now use the Extended Kalman Filter and Unscented Kalman Filter blocks to estimate the states of a discrete-time nonlinear system in Simulink. The blocks use first-order extended and unscented Kalman filter algorithms to estimate states as new data becomes available during the operation of the system. Previously, nonlinear state estimation using these algorithms was available at the command line only. You can use the state estimates for applications such as condition monitoring and fault detection. You can also generate C/C++ code for these blocks using Simulink Coder software.

For information about how to use these blocks, see the Extended Kalman Filter and Unscented Kalman Filter block reference pages. For examples, see Estimate States of Nonlinear System with Multiple, Multirate Sensors and Nonlinear State Estimation of a Degrading Battery System.

fmincon Solver: Use constrained minimization methods for model estimation

You can now use the sequential quadratic programming (SQP) and trust-region-reflective algorithms of the `fmincon` solver for linear and nonlinear model estimation. If you have Optimization Toolbox™ software, you can also use the interior-point and active-set algorithms of the `fmincon` solver. In the **System Identification** app, you can use only the SQP and interior-point algorithms.

These algorithms might result in improved estimation results in the following scenarios:

- Constrained minimization problems when there are bounds imposed on the model parameters.
- Model structures where the loss function is a nonlinear or non-smooth function of the parameters.
- Multi-output model estimation
- Estimating regularization parameters when using `ssregest`, `arxRegul`, or `impulseest` — You no longer need Optimization Toolbox software for using the default search method (`'fmincon'`).

The estimation option sets for which the new search method is available are: `oeOptions`, `bjOptions`, `armaxOptions`, `polyestOptions`, `procestOptions`, `greyestOptions`, `tfestOptions`, `ssestOptions`, `nlhwOptions`, `nlarxOptions`, `nlgreyestOptions`, `findopOptions`, and `findstatesOptions`.

Compatibility Considerations

If you do not have Optimization Toolbox software, the output of `ssregest`, `arxRegul`, or `impulseest` may not match previous releases. The output is now more accurate because the software uses the trust-region-reflective `fmincon` algorithm instead of Quasi-Newton line search for estimating regularization parameters.

State estimation using historical data

The computation of states of a model using past historical data has changed for the following commands:

- `data2state` — The command now computes the states by performing 1-step prediction error minimization. Previously, the command performed simulation error minimization.
- `sim`, `simsd`, `predict`, `pe`, `compare`, and `resid` — These commands compute initial states from past data by performing 1-step prediction error minimization. Previously, these commands performed simulation error minimization for simulated output, and `K`-step prediction error minimization for prediction horizon `K`.

Compatibility Considerations

For models with a nontrivial noise component, the results of the following commands do not match previous releases:

- The estimated states `x` and state covariance `xCov` using `data2state`.

  ```
  [x,xCov] = data2state( ___ )
  ```

- The output of `sim`, `simsd`, `predict`, `pe`, `compare`, and `resid` if you specify initial conditions using historical data — When you specify the `InitialCondition` option of `simOptions`, `simsdOptions`, `predictOptions`, `peOptions`, `compareOptions`, and `residOptions` as a structure with input-output data, there is a change in the output of these commands.

Computation of standard deviation of simulated and forecasted outputs and states

The `sim` and `forecast` commands now account for additive disturbances in your identified model when computing standard deviations of simulated or forecasted outputs or state estimates. Previously, these commands only accounted for model parameter covariance and initial state covariance during uncertainty calculations.

Compatibility Considerations

The following computed standard deviations do not match the results from previous releases:

- Standard deviations of forecasted output `yf_sd` and estimated states `x_sd` using the following syntax of `forecast`.

  ```
  [yf,x0,sysf,yf_sd,x,x_sd] = forecast( ___ )
  ```

- Standard deviations of simulated output `y_sd` and estimated states `x_sd` using the following syntaxes of `sim`.

  ```
  [y,y_sd] = sim( ___ )
  ```

  ```
  [y,y_sd,x] = sim( ___ )
  ```

  ```
  [y,y_sd,x,x_sd] = sim( ___ )
  ```

- Confidence intervals plotted for the simulated model output in the **System Identification** app. For information about how the plot is generated, see Simulation and Prediction in the App.

Initial condition handling of nonlinear grey-box models

There is a change in the default handling of initial conditions of nonlinear grey-box models by the `sim`, `predict`, `pe`, `compare`, `resid`, `forecast`, and `findstates` commands. The changes ensure

that the software honors the initial state specification in the `InitialStates` property of the `idnlgrey` model. You specify the initial condition handling in the following option sets:

- `predictOptions`, `peOptions`, `compareOptions`, `residOptions`, `forecastOptions` — When `InitialCondition` is set to the default value `'e'`, only those initial states that are designated as free in the `InitialStates` property of the `idnlgrey` model are now estimated. Previously, all the states of the model were estimated.
- `findstatesOptions` — When `InitialState` is set to the default value `'e'`, only those initial states that are designated as free in the `InitialStates` property of the `idnlgrey` model are now estimated. Previously, all the states of the model were estimated.
- `simOptions` — The default `InitialCondition` value is now `[]`. For the default value, the software specifies the initial states of a nonlinear grey-box model `sys` as `getinit(sys,'Value')`. There is no compatibility issue with this change.

Compatibility Considerations

For `predictOptions`, `peOptions`, `compareOptions`, `residOptions`, `forecastOptions`, and `findstatesOptions` option sets, if you specify the `InitialCondition` or `InitialState` option as:

- `'e'` — The software now estimates only those states that are specified as free in the `idnlgrey` model. Previously, all the states of the model were estimated. To estimate all the states, specify all states of the model `sys` as free by setting the `Fixed` field of `sys.InitialStates` as `false`. Alternatively, use `setinit`. For more information, see the reference page of the relevant option set.
- `'fixed'` — This option might be removed in a future release. Use alternate code as described here.

  When `InitialCondition` is `'fixed'`, `sys.InitialStates` sets the values of the initial states, but all the states are considered fixed for state estimation. To reproduce this behavior, first specify all the states as fixed by setting the `Fixed` field of `sys.InitialStates` as `true`. Then specify the `InitialCondition` option of your option set as `'e'`. For more information, see the reference page of the relevant option set.
- `'model'` — This option might be removed in a future release. Use alternate code as described here.

  When `InitialCondition` is `'model'`, `sys.InitialStates` sets the values of the initial states, which states to estimate, and their minimum and maximum values. To reproduce this behavior, specify the `InitialCondition` option of your option set as `'e'`. For more information, see the reference page of the relevant option set.

`'fixed'` and `'model'` are not available for the `forecastOptions` and `findstatesOptions` option sets.

Specifying constant historical data for computing initial states

There is a change in the specification of constant input or output historical data for the computation of initial conditions by the following commands:

- `simOptions`, `simsdOptions`, `predictOptions`, `peOptions`, `compareOptions`, and `residOptions` — Previously, to specify constant past data, you specified the `InitialCondition`

option of these options sets as a structure with `Input` and `Output` fields that are row vectors. For example, suppose that for a two-output time series model with no inputs, the past outputs have constant values 5 and 10. To compute the initial conditions when predicting the model response, you could previously specify the constant output values as the row vector `[5 10]`.

```
IO.Input = [];
IO.Output = [5 10];
opt = predictOptions('InitialCondition',IO);
```

The software no longer interprets a row vector as constant values for the `Input` and `Output` fields.

- `data2state` — Previously, to specify constant past data, you specified the `PastData` input argument as a structure with `Input` and `Output` fields that are row vectors. The software no longer interprets a row vector as constant values for the `Input` and `Output` fields.

Compatibility Considerations

If you specify past input or output data as row vectors, you see a change in output of the `sim`, `simsd`, `predict`, `pe`, `compare`, `resid`, and `data2state` commands. To recover the results from previous releases, update your scripts as follows:

- For constant past output data, specify `Output` as an *m*-by-*Ny* matrix of the constant values, where *m* is the order of the model and *Ny* is the number of outputs. For example, for a third-order two-output time series model, where the past outputs have constant values 5 and 10, specify `Output` as a 3-by-2 matrix of the constant values.

  ```
  IO.Output = [5 10;5 10;5 10];
  ```

- For constant past input data, specify the `Input` field as an *m*-by-*Nu* matrix of constant values, where *Nu* is the number of inputs.

Estimating continuous-time models using band-limited time-domain data

If you specify time-domain estimation data as band-limited, the software no longer converts the data to frequency-domain before performing continuous-time model estimation. Previously, if you specified the `InterSample` property of your `iddata` object as `'bl'`, the time-domain data was converted to frequency-domain data, and the sample time of the data was set to zero.

Compatibility Considerations

If you are using band-limited time-domain data for estimating continuous-time models, the estimation results might not match previous releases.

To recover the results from previous releases, convert the time-domain `iddata` object to frequency-domain using `fft`, and set the sample time to zero. Use the frequency-domain data for model estimation.

```
dataFD = fft(data);
dataFD.Ts = 0;
```

Increased flexibility in defining model transformation functions in translatecov command

When you use the `translatecov` command to translate parameter covariance across model transformation operations, you specify the transformation operations using a transformation function, `fcn`. Now, `fcn` can take inputs of any data type, provided one of the inputs is a linear identified model with parameter covariance information. Previously, all the inputs to `fcn` had to be linear identified models. For an example, see Translate Parameter Covariance to Closed-Loop Model.

New example showing application of system identification tools for diagnostics and prognostics

The Condition Monitoring and Prognostics Using Vibration Signals example shows how to extract features from vibration signals measured from a ball bearing, and use the features for health monitoring and prognostics.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`ssregest`, `arxRegul`, and `impulseest`	Still runs	Not applicable	If you do not have Optimization Toolbox software, the output of `ssregest`, `arxRegul`, and `impulseest` may not match previous releases. For more information, see “fmincon Solver: Use constrained minimization methods for model estimation” on page 13-2.
`[x,xCov] = data2state(___)`	Still runs	Not applicable	Estimated states `x` and state covariance `xCov` do not match the values from previous releases for models with nontrivial noise components. For more information, see “State estimation using historical data” on page 13-2.
Specifying initial conditions using historical data in the `simOptions`, `simsdOptions`, `predictOptions`, `peOptions`, `compareOptions`, and `residOptions` commands.	Still runs	Not applicable	The results do not match the values from previous releases for models with nontrivial noise component. For more information, see “State estimation using historical data” on page 13-2.

Functionality	Result	Use This Instead	Compatibility Considerations
`[yf,x0,sysf,yf_sd,x,x_sd] = forecast(___)`	Still runs	Not applicable	Standard deviations of forecasted output `yf_sd` and estimated states `x_sd` do not match previous releases. For more information, see “Computation of standard deviation of simulated and forecasted outputs and states” on page 13-3.

`[y,y_sd] = sim(___)`
`[y,y_sd,x] = sim(___)`
`[y,y_sd,x,x_sd] = sim(___)`
	Still runs	Not applicable	Standard deviations of simulated output `y_sd` and estimated states `x_sd` do not match previous releases. For more information, see “Computation of standard deviation of simulated and forecasted outputs and states” on page 13-3.

Confidence intervals plotted for the simulated model output in the **System Identification** app.	Still runs	Not applicable	The computed confidence interval does not match previous releases. For more information, see “Computation of standard deviation of simulated and forecasted outputs and states” on page 13-3.
For `idnlgrey` models, specifying `InitialCondition` or `InitialState` option of the following option sets as `'e'`, `'fixed'`, or `'model'`: `predictOptions`, `peOptions`, `compareOptions`, `residOptions`, `forecastOptions`, and `findstatesOptions`.	Still runs	Specify `InitialCondition` and `InitialState` as the default option `'e'`.	The behavior of the default value `'e'` of the `InitialCondition` and `InitialState` options has changed. To recover the old default behavior, update your code as described in “Initial condition handling of nonlinear grey-box models” on page 13-3.
Specifying constant input or output historical data as row vectors in `data2state`, `simOptions`, `simsdOptionspredictOptions`, `peOptions`, `compareOptions`, and `residOptions`.	Still runs	Do not use row vectors	The software no longer interprets a row vector as constant values. To recover the results from previous releases, update your scripts as described in “Specifying constant historical data for computing initial states” on page 13-4.
Using band-limited time-domain data for estimating continuous-time models.	Still runs	Not applicable	The estimation results might not match previous releases. To recover the results from previous releases, update your scripts as described in “Estimating continuous-time models using band-limited time-domain data” on page 13-5.

Functionality	Result	Use This Instead	Compatibility Considerations
`V = arxstruc(ze,zv,NN,MaxSize);`	Still runs	`V = arxstruc(ze,zv,NN);`	The `MaxSize` input is now ignored during computation of the loss function for an ARX model using `arxstruc`.
Using `NoiseData` option of `simsdOptions` to add specified noise to simulated data.	Still runs	Not applicable	The `NoiseData` option may be removed in a future release.
`OutputWeight` option of `forecastOptions`	Still runs	Not applicable	The `OutputWeight` option may be removed in a future release.

R2016b

Standalone Applications for System Identification: Deploy data preparation and model estimation code using MATLAB Compiler

You can create standalone applications to deploy code that includes model estimation commands such as `arx`, `armax`, `tfest`, `ssest`, `greyest`, `nlarx`, `nlhw`, and `nlgreyest`. Previously, you could not compile offline estimation commands using MATLAB Compiler software. Now you can compile all command-line functionality, except `advice`.

For a complete list of estimation commands, see Commands for Model Estimation.

Extended and Unscented Kalman Filters: Estimate states of nonlinear systems for online tracking and control system design

You can now estimate the states of discrete-time nonlinear systems at the command line using first-order extended Kalman filter algorithms and unscented Kalman filter algorithms. The new state estimation commands `extendedKalmanFilter` and `unscentedKalmanFilter` are useful for online estimation of states when new data is available during the operation of the system. You can use the state estimates for applications such as condition monitoring and fault detection.

You can use MATLAB Compiler or MATLAB Coder software to deploy the estimators in your application.

For examples of online state estimation, see Nonlinear State Estimation Using Unscented Kalman Filter and Fault Detection Using an Extended Kalman Filter.

Frequency-Domain Identification Improvements: Identify transfer function models faster and more accurately from frequency-response data

A new algorithm is now used for performing transfer function estimation from frequency-domain data using `tfest`. You are likely to see faster and more accurate results, particularly for data with dynamics over a large range of frequencies and amplitudes.

In addition, for transfer function estimation using frequency-response data, you can now specify the estimation weighting filter as `'inv'` or `'invsqrt'` using the `WeightingFilter` option of `tfestOptions`. These new options enable you to specify common weighting schemes that are useful for capturing relatively low amplitude dynamics in data, or for fitting data with high modal density. The options also make it easier to specify channel-dependent weighting filters for MIMO frequency-response data.

Compatibility Considerations

When estimating transfer functions from frequency-domain data, the estimation results may not match results from previous releases. To perform estimation using the previous algorithm, append `'-R2016a'` to the syntax:

```
sys = tfest( ___ ,'-R2016a')
```

Reorganization of Focus Estimation Option: Increased flexibility for configuring linear model estimation

The `Focus` estimation option of linear model estimation commands has been split into three options. Previously, you specified `Focus` as one of `'simulation'`, `'prediction'`, `'stability'`, or a custom filter. The restructuring of the options provides greater flexibility in configuring model estimation. For example, you can now choose to minimize the prediction error *and* specify a custom filter for prefiltering.

The new options are:

- `Focus` — Choose minimization of prediction or simulation error during estimation. `Focus` is no longer available in the `tfestOptions` and `oeOptions` option sets.
- `EnforceStability` — Ensure that the estimated model is stable. This option is not available in the `procestOptions` and `ssregestOptions` option sets.
- `WeightingFilter` — Specify a custom weighting prefilter for estimation.

The new options are available for the following option sets — `arxOptions`, `armaxOptions`, `bjOptions`, `oeOptions`, `iv4Options`, `polyestOptions`, `procestOptions`, `tfestOptions`, `ssestOptions`, `ssregestOptions`, `n4sidOptions`, and `greyestOptions`.

Compatibility Considerations

- If you specify `Focus` as `'stability'`, then `EnforceStability` is now set to `1`, and the `Focus` option is `'prediction'`. Consider using the `EnforceStability` option instead.
- If you specify `Focus` as a filter `prefilter`, then `WeightingFilter` property is now set to `prefilter` and `Focus` is `'simulation'`. Consider using the `WeightingFilter` option instead.
- If you are performing frequency-domain estimation of transfer functions, the stability constraint is no longer enforced by default. To enforce model stability, use the `EnforceStability` option.

compare Plot Updates: Plot model response error and confidence regions

The `compare` command has the following changes to the output plot:

- You can now plot the error between model response and validation data. Previously, only the model response was plotted.
- You can view the confidence region for simulated model response.

To display the error plot and confidence region, right-click the plot, and use the context menu.

Updates to predict and pe commands: Plot predicted response and prediction error for multiple models

The `predict` and `pe` commands have the following changes to syntax and output plot:

- You can now plot the predicted model response and the prediction error for multiple identified models at the same time. Previously, you could only plot the predicted response and prediction error of a single model. You can also specify the line specifications for the plots, including line style, line color, and marker type.

- You can now toggle between predicted model response plot and prediction error plot using the context menu of the plot. To access the menu, right-click the plot. You can also use the menu to configure the plots. For example, you can select the models to view.

Updates to forecast command: Plot forecasted model response

You can now plot the forecasted output of a model using the `forecast` command. You can specify the line specifications for the plot, including line style, line color, and marker type. You can also plot the forecasted output of multiple models at the same time.

To configure the plot, you can use the context menu of the plot. For example, you can use the menu to select the models to view, and to plot the confidence region of the forecasted outputs. To access the menu, right-click the plot.

Handling of delays during linear model estimation using time-domain data

The handling of delays during linear model estimation using time-domain data has changed. If you specify a delay in the model, the regressors generated during model estimation are now identical, regardless of how the delays are specified.

For example, the coefficients for the ARX polynomial models estimated by the following two syntaxes are now the same.

```
m1 = arx(data,[1 1 5])

m2 = arx(data,[1 1 1],'InputDelay',4)
```

The net delay in both models is the same but is represented differently. The `B` polynomial for `m1` has four additional leading zeros, while the `InputDelay` property of `m2` is four.

Some ways of specifying delay, depending upon the model structure and input-output size are: `InputDelay` (for all models), `IODelay` (for polynomial models and transfer functions), `Td` (for process models), and `nk` order (in the estimators for polynomial models such as `arx`). Note that only `nk` values greater than 1 are treated as delays in polynomial models, because `nk=1` denotes lack of feedthrough.

In addition, if you specify the `InputDelay` property of the model, the input signal is now prepended with `n` zero values to pre-compensate for the `n` input delays. Previously, the output signal in estimation data was precompensated for those delays by discarding the first `n` output samples. This change in the handling of input delays improves the estimation of initial conditions, resulting in better fits to the data.

Compatibility Considerations

If you are estimating linear models with known delay `n` using time-domain data, the estimation results might be different from previous releases. You can recover the results from previous releases for:

- Single-output data with model delay specified using `InputDelay`, `IODelay`, or `nk` order properties
- Multi-output data with model delay specified using the `InputDelay` property

To recover the estimation results from previous releases, discard the first `n` output samples by shifting your output data by `n`. For example, to estimate an ARX model with orders `[na nb nk]`, shift the output by delay `n = nk-1`, where `nk > 1`. Use this shifted data for model estimation, and do not specify the delay during estimation. Specify the delay in the `InputDelay` property of the estimated model.

```
n = nk-1;
data2 = nkshift(data,n);
m = arx(data2,[na nb 1]);
m.InputDelay = n;
```

Phase-Wrap Branch Option: Specify cutoff point for wrapping phase in response plots

By default, response plots that show phase response, such as Bode and Nichols plots, display the exact phase. You can make these plots wrap the phase into the interval [–180º,180º) by checking **Wrap Phase** in the plot Property Editor or the Toolbox Preferences Editor.

In R2016b, checking **Wrap Phase** enables a new **Branch** field that lets you specify the value at which accumulated phase wraps in the response plot. For example, entering 0 causes the plot to wrap the phase into the interval [0º,360º).

At the command line, turn on phase wrapping by setting the `PhaseWrapping` option of `bodeoptions` or `iddataPlotOptions` to `'on'`. Specify the phase-wrap cutoff point using the new `PhaseWrappingBranch` option.

In R2015b and R2016a, phase-wrapped plots used the interval [0º,360º). Before R2015b, phase-wrapped plots used the interval [–180º,180º).

Functionality Being Removed or Changed

Functionality	Result	Use This Instead	Compatibility Considerations
Estimation of transfer functions from frequency-domain data, using the `tfest` command.	Still runs	Not applicable	The estimation results may not match results from previous releases because a new estimation algorithm is used. To perform estimation using the previous algorithm, append `'-R2016a'` to the syntax: `sys = tfest( ___ ,'-R2016a')` For more information, see “Frequency-Domain Identification Improvements: Identify transfer function models faster and more accurately from frequency-response data” on page 14-2.

Functionality	Result	Use This Instead	Compatibility Considerations

Linear model estimation option `Focus` specified as a filter, `prefilter`. For example:
`opt = arxOptions;`
`opt.Focus = prefilter;`
	Still runs	`opt = arxOptions;`
		`opt.WeightingFilter = prefilter;`
			The `WeightingFilter` property is now set to `prefilter` and `Focus` is `'simulation'`.
			For more information, see "Reorganization of Focus Estimation Option: Increased flexibility for configuring linear model estimation" on page 14-3.

Linear model estimation option `Focus` specified as `'stability'`. For example:
`opt = arxOptions;`
`opt.Focus = 'stability';`
	Still runs	`opt = arxOptions;`
		`opt.EnforceStability = 1;`
			The `EnforceStability` property is now set to `1` and `Focus` is `'prediction'`.
			For more information, see "Reorganization of Focus Estimation Option: Increased flexibility for configuring linear model estimation" on page 14-3.

Estimation of transfer functions from frequency-domain data, using the default `tfestOptions` option set.	Still runs	To enforce stability, use the `EnforceStability` option.	The stability constraint is no longer enforced by default during estimation. For more information, see "Reorganization of Focus Estimation Option: Increased flexibility for configuring linear model estimation" on page 14-3.
Estimation of linear models with delays using time-domain data.	Still runs	Not applicable	The estimation results might not match results from previous releases due to a change in the handling of delays. To recover the results from previous releases, update your scripts as described in "Handling of delays during linear model estimation using time-domain data" on page 14-4.

R2016a

Improved Time-Series Forecasting: Forecast linear and nonlinear model output

The `forecast` command predicts time-series data in the future using a linear or nonlinear identified model. The command now supports new features:

- You can forecast the output of nonlinear models (nonlinear ARX and nonlinear grey-box). Previously, `forecast` only supported linear models.
- `forecast` returns the standard deviation of the forecasted output for identified nonlinear grey-box models and all linear models.
- `forecast` also computes the state trajectory into the future and returns the standard deviation of the state trajectory.

Use `forecast` for prognostic applications. For an example, see Perform Multivariate Time Series Forecasting.

Updates to resid command syntax and output plot

The `resid` command has the following changes to syntax and output plot:

- You can use `residOptions` to specify the maximum lag for residual correlation and impulse response calculations.
- You can plot the residual for multiple identified systems (`sys1`, ... `sysN`) at the same time. Use the following syntax:

  ```
  resid(Data,sys1,...,sysN)
  ```

  You can also specify the line specifications for the plots, including line style, line color, and marker type. Use the following syntax:

  ```
  resid(Data,sys1,Linespec1,...,sysN,Linespecn)
  ```
- The residual correlation plots for all input-output combinations of multi-input multi-output systems now appear at the same time in the plot window. You no longer need to press a key to view different correlation plots. To choose the inputs and residuals to display, right-click the plot, and use the context menu.
- The Bode plot of the frequency response from the residuals to the inputs now plots the phase, in addition to the magnitude of the response.

Compatibility Considerations

If you specify the maximum lag using the syntax `resid(sys,Data,Type,MaxLag)`, then consider using `residOptions` to specify the maximum lag.

Compute state trajectory standard deviation using sim, and specify initial state covariance

The `sim` command now supports the following new features:

- You can compute the standard deviation, `x_sd`, of the estimated state trajectory, `x`, of state-space models. Use the following syntax:

```
[y,y_sd,x,x_sd] = sim(sys,udata)
```

Here, `y` and `y_sd` are the simulated response and standard deviation of an identified model, `sys`. The input data is `udata`.

- You can also specify the covariance of initial states when computing simulated response. To specify the covariance, use the `X0Covariance` option of `simOptions`.

findstates command returns covariance of estimated states

The covariance of estimated states is now returned in the `Report` output of `findstates`. To calculate the covariance, use the following syntax:

```
[x0,Report]= findstates( ___ )
```

Here, `x0` is the estimated states, and `Report.Covariance` returns the covariance of `x0`. Use `Report` with any of the input arguments of `findstates`.

data2state command estimates current states of all types of identified models

You can now use the `data2state` command to estimate the current states of all linear and nonlinear identified models. Previously, `data2state` was available only for nonlinear ARX models.

New examples showing application of system identification tools for diagnostics and prognostics

The new examples are:

- Perform Multivariate Time Series Forecasting — This example uses the `forecast` command for predicting future values of multivariate time series data.
- Detect Abrupt System Changes Using Identification Techniques — This example compares the use of `segment` command and online estimation techniques for detecting changes in a system.

The new examples add to the suite of existing examples in the Diagnostics and Prognostics category.

Functionality Being Removed or Changed

Functionality	Result	Use This Instead	Compatibility Considerations
`MaxLag = 30; resid(sys,Data,Type,MaxLag)`	Still runs	`opt = residOptions('MaxLag',30); resid(Data,sys,Type,opt)`	For more information, see “Updates to resid command syntax and output plot” on page 15-2.
`a`, `b`, `c`, `d`, and `k` properties of `idss`	Still runs	`A`, `B`, `C`, `D`, and `K`, respectively.	Replace all instances of `a`, `b`, `c`, `d`, and `k` with `A`, `B`, `C`, `D`, and `K`, respectively.

Functionality	Result	Use This Instead	Compatibility Considerations
`num`, `den`, and `ioDelay` properties of `idtf`	Still runs	`Numerator`, `Denominator`, and `IODelay`, respectively.	Replace all instances of `num`, `den`, and `ioDelay` with `Numerator`, `Denominator`, and `IODelay`, respectively.
`a`, `b`, `c`, `d`, `f`, and `ioDelay` properties of `idpoly`	Still runs	`A`, `B`, `C`, `D`, `F`, and `IODelay`, respectively.	Replace all instances of `a`, `b`, `c`, `d`, `f`, and `ioDelay` with `A`, `B`, `C`, `D`, `F`, and `IODelay`, respectively.
`b` and `f` properties of `idnlhw`	Still runs	`B` and `F`, respectively.	Replace all instances of `b` and `f` with `B` and `F`, respectively.
`a`, `b`, `c`, `d`, `k`, `Structure.FcnType`, and `Structure.ExtraArgs` properties of `idgrey`	Still runs	`A`, `B`, `C`, `D`, `K`, `Structure.FunctionType`, and `Structure.ExtraArguments`, respectively.	Replace all instances of `a`, `b`, `c`, `d`, `k`, `Structure.FcnType`, and `Structure.ExtraArgs` with `A`, `B`, `C`, `D`, `K`, `Structure.FunctionType`, and `Structure.ExtraArguments`, respectively.

R2015b

Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder

You can now estimate models online at the command-line using new online estimation commands: `recursiveAR`, `recursiveARMA`, `recursiveARX`, `recursiveARMAX`, `recursiveOE`, `recursiveBJ`, and `recursiveLS`. For more information, see Perform Online Parameter Estimation at the Command Line.

You can then deploy the generated code or standalone application in your target hardware using MATLAB Compiler or MATLAB Coder.

Compatibility Considerations

The old recursive estimators, `rarx`, `rarmax`, `roe`, and `rbj`, are not compatible with MATLAB Compiler or MATLAB Coder, and may be removed in a future release. Use the new online estimation commands instead.

Bayesian and Akaike Information Criteria (BIC and AIC) Metrics: Compare identified models and select orders

Bayesian Information Criteria (BIC) and Akaike Information Criteria (AIC) are now computed during model estimation. These metrics provide a measure of model quality that you can use to compare different models and pick the best one. The most accurate model has the smallest AIC and BIC values.

The software computes and stores the following new values in the `Report.Fit` property of the estimated model:

- Raw AIC (`AIC`)
- Small sample-size corrected AIC (`AICc`)
- Normalized AIC (`nAIC`)
- Bayesian Information Criteria (`BIC`)

For more information on these metrics, see Loss Function and Model Quality Metrics.

Alternatively, you can use the `value = aic( ___ ,measure)` syntax to return the various AIC values. For more information, see the `aic` reference page.

procest command returns estimated input offsets

You can now use the following syntax for returning the estimated value of the offset in input signal:

```
[sys,offset] = procest( ___ )
```

`procest` automatically estimates the input offset when the model contains an integrator, or when you set the `InputOffset` estimation option to `'estimate'` using `procestOptions`.

Unified sim command for simulating linear and nonlinear identified models

The syntaxes for simulating linear and nonlinear identified models have been unified into a single `sim` command. Starting in R2015b, use a `simOptions` option set to configure your simulation. The previous syntax for nonlinear model simulation will continue to work in future releases.

Compatibility Considerations

- If your code uses any of the following functionality when simulating nonlinear models, consider updating the code.

Nonlinear Model	Functionality	Use This Instead
`idnlarx`, `idnlhw` or `idnlgrey`	Simulate model with additive noise using `y = sim(model,u,'Noise')`	`opt = simOptions('AddNoise',true);` `y = sim(model,y,opt);`
`idnlarx`, `idnlhw` or `idnlgrey`	Simulate models with initial states specified using `y = sim(model,u,'InitialState',init)`	`opt = simOptions('InitialCondition',init);` `y = sim(model,y,opt);`
`idnlgrey`	Return simulation final states using `[y,y_sd,XFINAL] = sim(model,u)`	`[y,y_sd,x] = sim(model,u);` `XFINAL = x(end,:)';`

Option for setting orientation of input-output data plots

You can now specify the orientation of input-output data plots created using `plot`. Display options for input-output data include:

- All in one row
- All in one column
- All outputs in a column, and all inputs in a second column
- All outputs in a row, and all inputs in a second row

To do so in the plot window, right-click the plot, and choose **Orientation** option from the context menu.

At the command line, use the `Orientation` option of the `iddataPlotOptions` option set.

Updates to compare command plot interface

The plot generated using `compare` has the following changes:

- **Fit%**, the normalized root mean square measure of the goodness of the fit, now displays in the legend of the plot instead of in a separate panel to the right of the plot.
- The context menu now has the following new options:
  - **I/O Grouping** — Use this option to plot data I/O channels in their own separate axes (**None**), or group them together (**All**).

- **Characteristics > Mean Value** — Use this option to view the mean value of the data.
- **Data Experiment** — For multi-experiment data only. Use this option to toggle between data from different experiments. This option replaces the separate tabs that displayed multi-experiment data in the plot.

To access the context menu, right-click the plot.

Modified normalized gradient algorithm for online estimation

To prevent jumps in estimated parameters, the normalized gradient algorithm now includes a bias term in the scaling factor of the adaptation gain. For details about the algorithm, see Recursive Algorithms for Online Parameter Estimation. The default value of the bias is `eps`. Increase the bias when you see jumps in the estimated parameters.

- To change the bias in Simulink, in the Block Parameters dialog box of Recursive Polynomial Model Estimator and Recursive Least Squares Estimator blocks, in the **Algorithm and Block options** tab, use the **Normalization Bias** field.
- To change the bias at the command line, use the `NormalizationBias` property of the online estimation commands on page 16-2.

Change in output and initial estimate specification of Recursive Polynomial Model Estimator block

The dimensions of Recursive Polynomial Model Estimator block output and initial estimate specification have changed.

- The Parameters outport of the block now outputs the estimated parameters A, B, C, D, and F as row vectors. For MISO polynomial models, B is a matrix where the i-th row parameters correspond to the i-th input. Previously, the estimated parameters were output as column vectors.
- In the block dialog box **Model Parameters** tab, if **Initial Estimate** is set to `Internal`, specify the initial parameter guess (**Initial A(q)**, **Initial B(q)**, **Initial C(q)**, **Initial D(q)**, or **Initial F(q)**) as a row vector only. Previously, initial guesses could also be specified as column vectors. For MISO polynomial models, specify **Initial B(q)** as a matrix where i-th row parameters correspond to the i-th input.

Compatibility Considerations

- If your Simulink model requires the estimated parameters output from the block outport to be a column vector, transpose the block output as follows:
  - Split the Parameters outport bus signal into its individual parameter components (A, B, C, D, and/or F; depending on the model choice) by using the Bus Selector block from Simulink Signal Routing library.
  - Use the Permute Dimensions block from Simulink Math Operations library to convert each signal into a column vector.
  - Combine the parameters back into a bus signal, if necessary, using the Bus Creator block from Simulink Signal Routing library.
- If you specified the initial guess for any of the parameter values (**Initial A(q)**, **Initial B(q)**, **Initial C(q)**, **Initial D(q)**, or **Initial F(q)**) as column vectors, an error occurs during simulation.

Specify them as row vectors. For MISO polynomial models, transpose the **Initial B(q)** matrix so that the *i*-th row parameters correspond to the *i*-th input.

Change in input specification of Model Type Converter block

The Model Type Converter block inport now only accepts bus signal elements specified as row vectors. Previously, you specified the bus elements as column vectors. For MISO data, specify *B* polynomial coefficients as a matrix where the *i*-th row parameters correspond to the *i*-th input.

Compatibility Considerations

If you specified the inport bus signal elements as column vectors, an error occurs during simulation. Specify them as row vectors. For MISO polynomial models, transpose the *B* matrix so that the *i*-th row parameters correspond to the *i*-th input.

Functionality Being Removed or Changed

Functionality	Result	Use This Instead	Compatibility Considerations
`rarx`	Warns	`recursiveAR` or `recursiveARX`	See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder” on page 16-2 for more information.
`rarmax`	Warns	`recursiveARMA` or `recursiveARMAX`	See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder” on page 16-2 for more information.
`roe`	Warns	`recursiveOE`	See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder” on page 16-2 for more information.
`rbj`	Warns	`recursiveBJ`	See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder” on page 16-2 for more information.
`y = sim(model,u,'Noise')` for `idnlarx`, `idnlhw`, or `idnlgrey` models	Still works	`opt = simOptions('AddNoise',true); y = sim(model,y,opt);`	See “Unified sim command for simulating linear and nonlinear identified models” on page 16-3 for more information.
`y = sim(model,u,'InitialState',init)` for `idnlarx`, `idnlhw`, or `idnlgrey` models	Still works	`opt = simOptions('InitialCondition',init); y = sim(model,y,opt);`	See “Unified sim command for simulating linear and nonlinear identified models” on page 16-3 for more information.

Functionality	Result	Use This Instead	Compatibility Considerations
`[y,y_sd,XFINAL] = sim(model,u)` for `idnlgrey` models	Still works	`[y,y_sd,x] = sim(model,u); XFINAL = x(end,:)';`	See “Unified sim command for simulating linear and nonlinear identified models” on page 16-3 for more information.
Simulink model requiring output from Parameters outport of Recursive Polynomial Model Estimator block to be a column vector.	Error	Transpose the block output.	See “Change in output and initial estimate specification of Recursive Polynomial Model Estimator block” on page 16-4 for more information.
Initial parameter guess specified as column vector in Recursive Polynomial Model Estimator block.	Error	Specify as row vector. For MISO polynomial models, transpose the original **Initial B(q)** matrix.	See “Change in output and initial estimate specification of Recursive Polynomial Model Estimator block” on page 16-4 for more information.
Inport bus signal elements specified as column vectors in Model Type Converter block.	Error	Specify the bus elements as row vectors. For MISO polynomial models, transpose the original *B* matrix.	See, “Change in input specification of Model Type Converter block” on page 16-5 for more information.

R2015a

nlgreyest command for nonlinear grey-box model estimation

You can use the `nlgreyest` estimator to estimate nonlinear grey-box models. Use the `nlgreyestOptions` option set to configure the model estimation objective and search method used by the estimator. For more information, see the corresponding reference pages.

Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators

You can use option sets for the nonlinear ARX, Hammerstein-Wiener and nonlinear grey-box model estimators to configure the model estimation objective and search method. Instead of using name-value pair input arguments in `nlarx` and `nlhw`, or the `Algorithm` property of the model, use the following commands:

- `nlarxOptions` — Option set for `nlarx`
- `nlhwOptions` — Option set for `nlhw`
- `nlgreyestOptions` — Option set for `nlgreyest`

To learn more about the estimation options, see the corresponding reference pages.

Compatibility Considerations

- The option sets replace the `Algorithm` property of nonlinear ARX (`idnlarx`), Hammerstein-Wiener (`idnlhw`), and nonlinear grey-box (`idnlgrey`) models.

  The following table shows the mapping of the fields of `Algorithm` to those of the estimation options set.

Algorithm Property Field	Option Set Field
`LimitError`	`Advanced.ErrorThreshold`
`Criterion/Weighting`	`OutputWeight` • If, `Algorithm.Criterion` was `'det'`, use `OutputWeight = 'noise'`. • If, `Algorithm.Criterion` was `'trace'`, set `OutputWeight` to the values in `Algorithm.Weighting`.
`MaxIter`	`SearchOption.MaxIter`
`Tolerance`	`SearchOption.Tolerance`
`MaxSize`	`Advanced.MaxSize`
`Advanced.Search`	`SearchMethod` and `SearchOptions`

- For nonlinear ARX models, the `Focus` property has been replaced by the `Focus` option in the `nlarxOptions` option set. See the reference page for more information.

Reorganization of nonlinear model estimation reports

A new property of nonlinear models, `Report`, provides information on the estimation. This read-only property replaces the `EstimationInfo` property and provides additional information regarding:

- Estimated parameters. For nonlinear grey-box models, it also contains the values of initial states, and parameter and initial state covariance matrices.
- The option set used for estimation.
- Information on data used for estimation, such as percentage fit to estimation data and the mean square error.

The `Report` fields are mostly uniform for the identified nonlinear models. However, certain fields of `Report` are model dependent.

To learn more about the estimation report, see Estimation Report, and the model and estimator reference pages.

Compatibility Considerations

- The `Report` property replaces the `EstimationInfo` property of nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box models.

  The following table shows the mapping of the fields of `EstimationInfo` to those of `Report`.

EstimationInfo Field	Report Field
`LossFcn`	`Fit.LossFcn`
`FPE`	`Fit.FPE`
`DataName`	`DataUsed.Name`
`DataLength`	`DataUsed.Length`
`DataTs`	`DataUsed.Ts`
`DataDomain`	`DataUsed.Type`
`DataInterSample`	`DataUsed.InterSample`
`WhyStop`	`Termination.WhyStop`
`UpdateNorm`	`Termination.UpdateNorm`
`LastImprovement`	`Termination.LastImprovement`
`Iterations`	`Termination.Iterations`
`InitialState`	No replacement
`Warning`	No replacement

- For nonlinear grey-box models, the `SimulationOptions` algorithm property is now a property of the `idnlgrey` model itself. See the model reference page for more information.

findopOptions command to create option set for operating point computation of nonlinear ARX or Hammerstein-Wiener models

You can use `findopOptions` to create an option set for computing the operating point of a nonlinear ARX (`idnlarx`) or Hammerstein-Wiener (`idnlhw`) model. Use the option set with `idnlarx/findop` and `idnlhw/findop` to specify the optimization search options.

Unified findstates command for nonlinear models

The `findstates` methods of nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box models have been replaced with a single `findstates` command which provides a more unified syntax. You

can also use `findstatesOptions` to create an option set for estimating initial states of the nonlinear models.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`Algorithm` property	Still works	`nlarxOptions` `nlhwOptions`	See “Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators” on page 17-2
`Focus` property of `idnlarx` models	Still works	`Focus` option in the `nlarxOptions` option set	See “Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators” on page 17-2
`EstimationInfo` property	Still works	`Report` property	See “Reorganization of nonlinear model estimation reports” on page 17-2

R2014b

AR, ARMA, Output-Error, and Box-Jenkins online model estimation with Recursive Polynomial Model Estimator block

The Recursive Polynomial Model Estimator block has been enhanced to estimate the coefficients of linear time-invariant and linear time-varying AR, ARMA, Output-Error (OE) or Box-Jenkins (BJ) models. The parameters are estimated as new data becomes available during the operation of the system. For more information, see Online Estimation.

You can also estimate a state-space model online from these models by using the Recursive Polynomial Model Estimator and Model Type Converter blocks together. Connect the outport of the Recursive Polynomial Model Estimator block to the inport of the Model Type Converter block to obtain online values of the state-space matrices. The conversion ignores the noise component of the models. In other words, the state-space matrices only capture the `y[k]/u[k]` relationship, which is `B(q)/F(q)` for OE and BJ models.

Kalman Filter block for estimating states of linear time-invariant and linear time-varying systems

Use the Kalman Filter block to estimate the states of linear time-invariant and linear time-varying systems online. The states are estimated as new data becomes available during the operation of the system. The system can be continuous-time or discrete-time. You can generate code for this block using code generation products such as Simulink Coder.

You can access this block from the Estimators sublibrary of System Identification Toolbox library. For an example of using this block, see State Estimation Using Time-Varying Kalman Filter.

Initial guesses for A(q) and C(q) polynomials in Recursive Polynomial Model Estimator block

The first element of the initial guesses for the `A(q)` and `C(q)` polynomials in the Recursive Polynomial Model Estimator block must be specified as 1. When the **Initial Estimate** option is `Internal`, you specify these values in the **Initial A(q)** and **Initial C(q)** parameters in the Block Parameters dialog box. When the **Initial Estimate** option is `External`, you specify these values using the **InitialParameters** inport of the block.

In previous releases, the software auto-scaled these values to 1.

Compatibility Considerations

If you specified the **Initial Estimate** parameter as `Internal`, an error occurs during simulation. If you specified this parameter as `External`, a warning occurs. Before you simulate the model, scale the initial guesses for the `A(q)` and `C(q)` polynomials by dividing both these vectors by their first elements.

ident command renamed to systemIdentification

The `ident` command to open the System Identification app has been renamed to `systemIdentification`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`AR Estimator`, `ARMAX Estimator`, `ARX Estimator`, `BJ Estimator`, and `OE Estimator` blocks	Still works	Recursive Polynomial Model Estimator	Consider replacing these blocks with the Recursive Polynomial Model Estimator block to perform recursive estimation.
`PEM Estimator` block	Still works	No replacement	Not applicable

R2014a

Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks for online model parameter estimation

Use the Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks to perform online model parameter estimation in Simulink. Online parameter estimation, also known as online estimation or online tuning, refers to estimating model parameters as new data becomes available during the operation of the model. You can generate code for these blocks using code generation products such as Simulink Coder. For example, you can estimate the coefficients of a time-varying plant from measured input-output data and feed them to an adaptive controller. After validating the online estimation in simulation, you can generate code for your Simulink model and deploy the same to an embedded target.

These blocks are in the Estimators library.

For examples of how to use these blocks, see Preprocess Online Estimation Data and Validate Online Estimation Results.

Compatibility Considerations

The following blocks will be removed in a future release: AR Estimator, ARMAX Estimator, ARX Estimator, BJ Estimator, OE Estimator, and PEM Estimator.

Interactive identification of single-input/single-output plants from measured data in PID Tuner app

As a part of the control design workflow, you can interactively identify a plant using measured data in the PID Tuner app in Control System Toolbox™. For example, to design a PID controller for a manufacturing process, you can start with response data from a bump test on your system. You can import this data instead of a plant model in the tuner. You can then interactively identify a linear plant model whose response fits the response data.

The PID Tuner automatically tunes a PID controller for the identified model. You can then interactively adjust the PID controller gains, and save the identified plant and tuned controller. For more information, see System Identification for PID Control.

To access the PID Tuner, enter `pidtool` at the MATLAB command line. For an example, see Interactively Estimate Plant Parameters from Response Data.

Interactive identification of single-input/single-output plants from simulation data when tuning PID Controller blocks using Simulink Control Design

You can obtain a linear representation of a Simulink model and tune the gains of a PID Controller block for the plant in the PID Tuner app. The identification-based approach serves as an alternative to the linearization-based approach and is useful where linearization fails to yield a good plant model. This functionality requires Simulink Control Design™ software.

The identification works by simulating the Simulink model and then using the simulated input-output data to obtain a plant model. You identify the plant using interactive graphical tools in the PID Tuner app. Next, you use the identified model to tune your PID Controller block. For example, suppose you

want to tune the PID Controller block in a model that contains a Triggered Subsystem block. The analytical block-by-block linearization algorithm does not support event-based subsystems, and therefore the model linearizes to zero. Now, you can simulate the Simulink model for a chosen input and use the simulated data to identify a plant model. The PID Tuner automatically tunes the PID controller for the identified model. You can then interactively adjust the performance of the tuned control system, and save the identified plant and tuned controller. For more information, see System Identification for PID Control.

To access the PID Tuner, in the PID Controller block dialog box, click **Tune**. For an example, see "Design a PID Controller Using Simulated I/O Data" in the Simulink Control Design documentation.

ssregest, a regularization-based state-space model estimator, for improved accuracy on short, noisy data sets

You can use `ssregest` to estimate state-space models. This estimator is known to perform better than `n4sid` for short, noisy data sets. For some problems, the quality of fit using `n4sid` is sensitive to options, such as `N4Horizon`, whose values can be difficult to determine. In comparison, the quality of fit with `ssregest` is less sensitive to its options, which makes `ssregest` simpler to use.

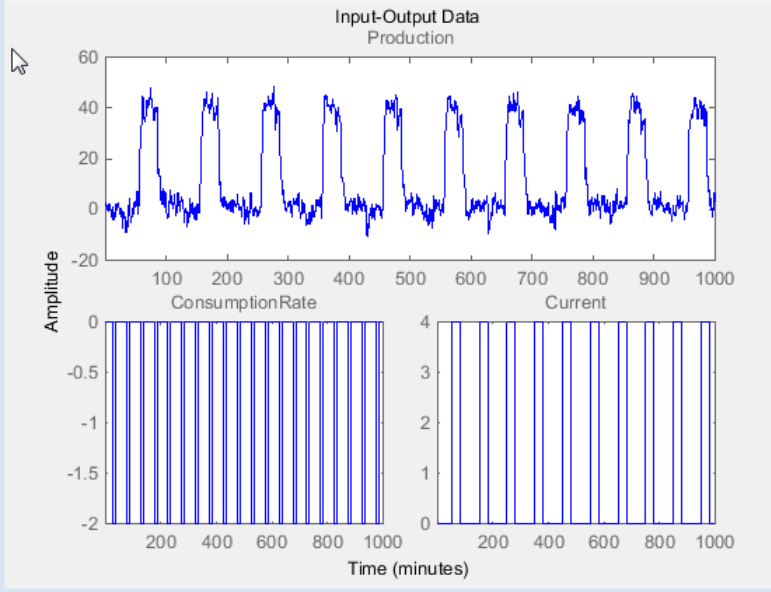
`ssregest` estimates a regularized ARX model and converts the ARX model to a state-space model. The software then uses balanced model reduction techniques to reduce the state-space model to the specified order. You can specify estimation options for `ssregest` using `ssregestOptions`.

You can also select this estimator in the System Identification Tool. In the State Space Models dialog box, expand **Estimation Options** and select `Regularized Reduction` from the **Estimation Method** drop-down list.

plot command for iddata object enhanced

The `plot` command for input-output data `iddata` has the following enhancements:

- Multiexperiment data or datasets with more than one input or output channels are plotted on a single plot
- Input and output channels can be grouped together



You can customize the plot, such as group and ungroup channels, and explore data characteristics, such as peak and mean value, using the right-click menu.

You can also customize the plot, such as specify axes labels, using `iddataPlotOptions`.

Options set and specification of input delay and noise source integrator for arxRegul command

You can now use `arxRegulOptions` to specify regularization options for `arxRegul`. Regularization options include the regularization kernel to use, such as `'TC'` and `'SE'`, and search method for estimating regularization constants.

You can also specify input delay and presence of a noise source integrator as `Name-Value` pair arguments in `arxRegul`.

Compatibility Considerations

Replace `[lambda,R] = arxRegul(data,orders,kernel)` and `[lambda,R] = arxRegul(data,orders,kernel,max_size)` syntaxes with `[lambda,R] = arxRegul(data,orders,options)` syntax. Specify `kernel` and `max_size` in the options set created using `arxRegulOptions`.

R2013b

Version: 8.3

New Features

Bug Fixes

Regularized estimation of linear and nonlinear models for obtaining parameter values with less variance

You can now obtain regularized estimates of parameters for linear and nonlinear models. Previously, you could specify this option for correlation model estimation only, using `impulseestOptions`.

Regularization reduces variance of estimated model parameters by trading variance for bias. Regularization is useful for:

- Identifying overparameterized models, such as nonlinear ARX models
- Imposing apriori knowledge of model parameters in structured models, such as grey-box models
- Incorporating knowledge of system behavior in ARX and FIR models

Using regularization adds a penalty proportional to the parameter dimension and values in the cost function that is minimized for estimation. Without regularization, the parameter estimates are obtained by minimizing a weighted quadratic norm of the prediction errors ε(t,θ):

$$V_N(\theta) = \frac{1}{N}\sum_{t=1}^{N} \varepsilon^2(t,\theta)$$

where t is the time variable, N is the number of data samples and $\varepsilon(t,\theta)$ is the predicted error computed as the difference between the observed output and the predicted output of the model.

A regularized estimation minimizes:

$$\widehat{V}_N(\theta) = \frac{1}{N}\sum_{t=1}^{N} \varepsilon^2(t,\theta) + \frac{1}{N}\lambda\theta^T R\theta,$$

where λ is a constant that trades off variance for bias in the estimated values of parameters θ. R is an associated weighting matrix.

For more information on regularization, see Regularized Estimates of Model Parameters.

You can specify the regularization constants `Lambda`, `R`, and `Nominal` at the command line or in the System Identification Tool:

- At the command line, use the `Regularization` option available in the estimation options set (`tfestOptions`, `ssestOptions`,...) for linear models.

  For nonlinear models, the option is available in the `Algorithm` property of `idnlarx`, `idnlhw`, and `idnlgrey` models.

  For ARX models, you can generate `Lambda` and `R` values automatically from a given regularization kernel using the `arxRegul` command.

  See the estimator reference pages and Regularized Identification of Dynamic Systems for examples.
- In the System Identification Tool, click **Regularization** in the linear model estimation dialog box or click **Estimation Options** in the Nonlinear Models dialog box.

  For an example, see Estimate Regularized ARX Model Using System Identification Tool.

ssarx subspace identification method for robust estimation of state-space models using closed-loop data

`N4Weight`, which represents the weighting scheme used for singular-value decomposition by the N4SID algorithm, now includes a `ssarx` option. This option is an ARX estimation-based algorithm to compute the weighting. Specifying this option allows the N4SID algorithm to compute unbiased estimates of the model parameters when using data that is collected in a closed-loop operation. For more information about the algorithm, see Jansson, M., "Subspace identification and ARX modeling", *13th IFAC Symposium on System Identification*, Rotterdam, The Netherlands, 2003.

To specify this option:

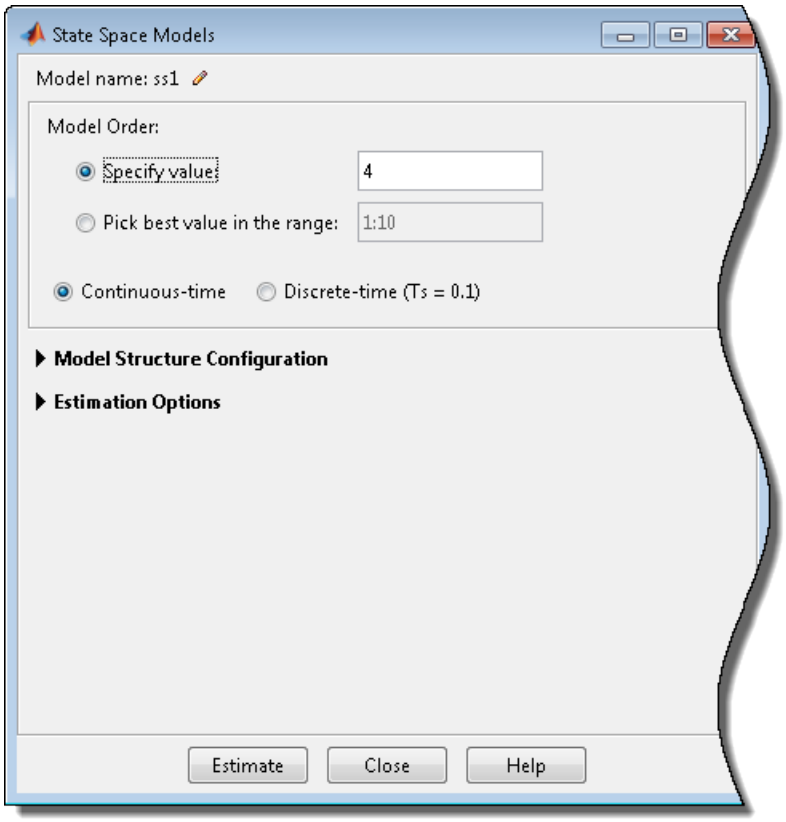
- At the command line, set the `N4Weight` option in `n4sidOptions` or `ssestOptions` to `'ssarx'`.
- In the System Identification Tool, in the State Space Models dialog box, expand **Estimation Options** and select `SSARX` from the **N4Weight** drop-down list.

For an example of using the subspace algorithm for closed-loop data, see the `n4sid` reference page.

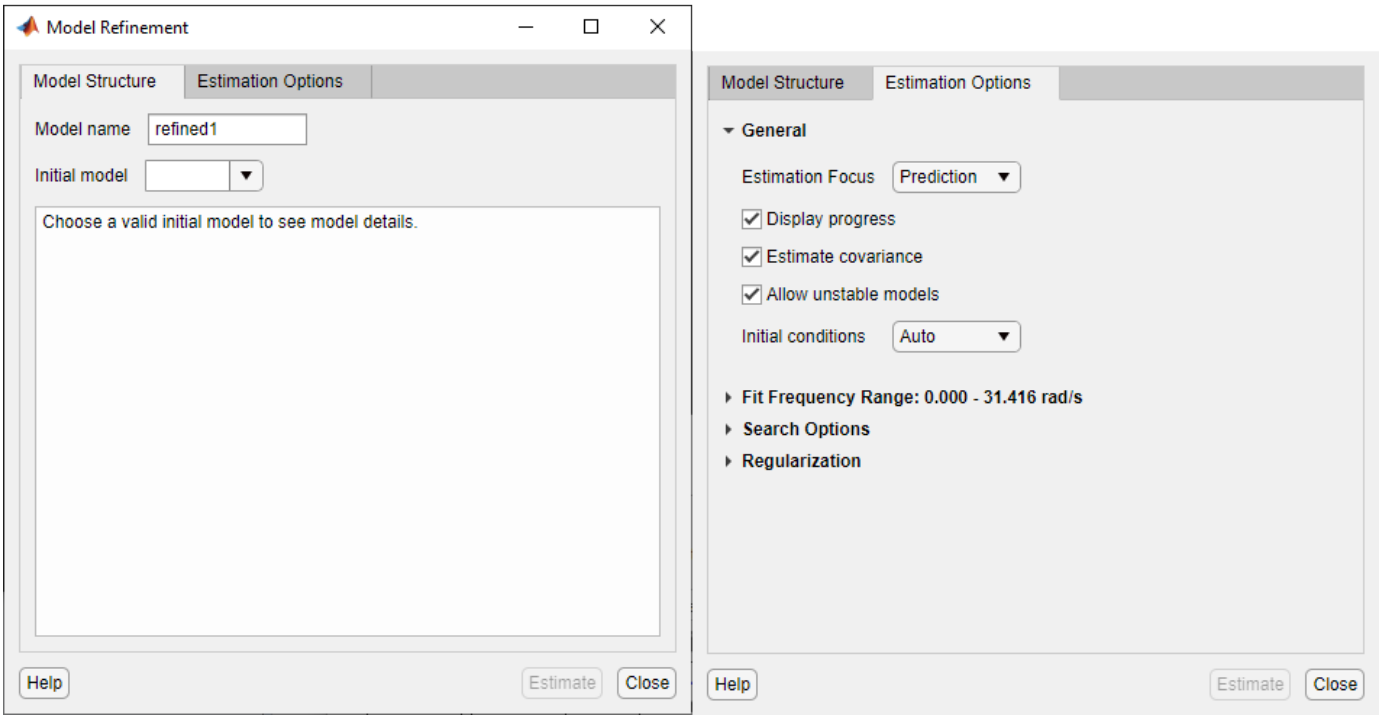
Redesigned state-space model and initial model refinement dialog boxes

The State Space Models and Linear Model Refinement dialog boxes have been redesigned to improve state-space model estimation and initial model refinement workflows.

To open the State Space Models dialog box, select **Estimate > State Space Models** in the System Identification Tool.



To access the redesigned Linear Model Refinement dialog box, in the System Identification Tool, select **Estimate > Refine Existing Models**.



The initial model must be in the Model Board of the System Identification Tool or a variable in the MATLAB workspace. This model can be a state-space, polynomial, process, or transfer function model.

For more information, click **Help** in the dialog boxes.

getpar and setpar commands to obtain and set parameter attributes of identified linear models

You can now use `getpar` with identified linear models to obtain parameter values, free or fixed status, minimum/maximum bounds, and labels. Identified linear models include process, input-output polynomial, state-space, transfer function, and grey-box models.

Similarly, use `setpar` to set these parameter attributes.

Unstable models option added to System Identification Tool

You can now estimate unstable models in the System Identification Tool. You can use this option to:

- Estimate transfer function models using frequency-domain data.
- Estimate state-space models using time- or frequency-domain data.
- Refine linear models using time- and frequency-domain data.

This functionality is the same as setting the estimation option `Focus` to `'prediction'` at the command line.

The option allows the estimation process to use parameter values that might lead to unstable models. An unstable model is delivered only if it produces a better fit to the data than other stable models computed during the estimation process. Such an unstable model might be useful, if, for example, you plan to design a controller for the model.

To set this option in the Transfer Function dialog box, expand **Estimation Options** and select the **Allow unstable models** check box. In the State Space Models and Linear Model Refinement dialog boxes, this option is selected by default.

SamplingGrid property for tracking dependence of array of sampled models on variable values

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, the new `SamplingGrid` property keeps track of the variable values associated with each model in the array. This information is shown when displaying or plotting the model array. The information is useful to trace results back to the independent variables.

Set this property to a structure whose fields are named after the sampling variables and contain the sample values associated with each model. All sampling variables should be numeric and scalar valued, and all arrays of sample values should be commensurate with the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct('rpm',nominal_engine_rpm)
```

where `sys` is an array containing three identified models obtained at rpms 1000, 5000, and 10000, respectively.

R2013a

Version: 8.2

Bug Fixes

R2012b

Regularized estimates of impulse response, specification of transport delays and estimation options using impulseest

You can obtain regularized estimates of impulse response using the regularization kernel (`RegulKernel`) estimation option. Regularization reduces variance of estimated model coefficients and produces a smoother response by trading variance for bias. You can also configure estimation options such as prefilter order and data offsets. You use `impulseestOptions` to specify the estimation options and pass them as an input to `impulseest`.

You can also specify filter orders and transport delays as inputs to `impulseest`.

Compatibility Considerations

- Using a time vector as an input to `impulseest` or specifying the `'noncausal'` flag warns and will be removed in a future version. Specify the order of the impulse response model instead.
- To compute the acausal part of the response up to a negative lag `L`, set the input delay input argument to `-L`.

translatecov command for translating model covariance across transformations

You can use `translatecov` to translate model covariance across model transformations such as continuous- and discrete-time conversions, concatenation and conversions to different model types. Previously, model covariance was lost when you performed such operations on a model directly. `translatecov` lets you perform these operations while also translating the covariance data. For example, transform an estimated continuous-time model `mc` to discrete-time:

```
md = c2d(mc,Ts);
md2 = translatecov(@(x)c2d(x,Ts),mc)
```

The first operation produces a discrete-time model, `md`, which does not contain parameter covariance data. The second operation produces the model, `md2`, which has the same structure and parameter values as `md`but contains parameter covariance data.

ssform command for quick configuration of state-space model structure

You can use `ssform` to configure model parameterization, feedthrough and disturbance dynamics. This command lets you quickly configure these properties when estimating state-space models in a structured way. You can use this command as a simpler alternative to explicitly modifying the `Structure` property of the `idss` model for some commonly applied changes. For example, typing `ssform(model,'Form','canonical','DisturbanceModel','estimate')` configures the model structure such that:

- Its `A`, `B`, and `C` matrices are in observability canonical form
- The `K` matrix entries are all treated as free parameters

Feedthrough specification for discrete-time transfer function model estimation

When estimating a discrete-time transfer function model, you can specify whether the model has feedthrough. Use the `Feedthrough` name-value pair in `tfest` or click **Feedthrough** in the graphical interface. For MIMO systems, you can specify feedthrough for individual channels or a common value across all channels.

R2012a

Summary

Important new features and changes in the System Identification Toolbox software for this release include:

- New functions that perform continuous-time estimation for state-space and transfer function models.
- Support for multi-output estimation for polynomial models (such as ARMAX, OE, and BJ) and process models.
- A new, uniform design for linear, parametric models. You can specify whether a coefficient should be estimated and now impose minimum/maximum bounds on estimated coefficients in a standardized manner.
- Consolidation of the functions dealing with linear time-invariant systems in the Control System Toolbox software. This unification of code allows for a streamlined workflow in estimating models and analyzing them and improves numerical accuracy and consistency.
- Many commands now have a more unified syntax, but, with few exceptions, old syntax continues to work in this release for backward compatibility. Incompatibilities introduced this release mainly involve configuration of estimation options, translation of parameter covariance, reordering of output arguments for some functions and the treatment of certain model properties.

Note Instances where the changes will break existing code or yield different results have been marked as "Backward incompatibility".

New Features in This Version

New features this release include:

Continuous-Time Transfer Function Identification for Time- and Frequency-Domain Data

A new function, `tfest`, lets you estimate a linear transfer function based on a system’s response. `tfest` can be used for time- and frequency-domain data.

The output of `tfest` is an `idtf` model, which is a new identified linear model. An `idtf` model stores the identified numerator, denominator, and any transport delays using its `num`, `den`, and `ioDelay` properties, respectively.

For information regarding estimating a continuous-time transfer function using time-domain data, see How to Estimate Transfer Function Models by Specifying Number of Poles.

For information regarding estimating a continuous-time transfer function using frequency-domain data, see How to Estimate Transfer Function Models with Transport Delay to Fit Given Frequency Response Data.

Time-Series Modeling and Forecasting, Including Generating ARIMA Models

Forecasting

A new function, `forecast`, lets you forecast the response of an identified linear model for a specified future time interval. You may also specify the future inputs for models that are not time-series models.

`forecast` complements the functionality of `predict`, which evaluates fixed-step ahead predictions on historic data.

Use `forecastOptions` to create an option set to specify forecasting options.

For more information, see `forecast` and `forecastOptions`.

Generating ARIMA Models

A new property for `idpoly` models, `IntegrateNoise`, designates if a model output contains an integrator in its noise source. Use the `IntegrateNoise` property to create, for example, ARI, ARIMA, ARIX, and ARIMAX models.

The `IntegrateNoise` property takes a logical vector of length `Ny`, where `Ny` is the number of outputs.

For more information, see Estimating ARIMA Models.

Estimation of Multi-Output Polynomial and Process Models

Multi-Output Polynomial Models

`idpoly` models can now represent multi-output polynomial models. Use `idpoly` to create a multi-output polynomial model. You can also use the various estimator functions (`ar`, `arx`, `bj`, `oe`, and `armax`) to estimate a multi-output `idpoly` model.

A new function, `polyest`, may also be used to estimate a multi-output polynomial model of arbitrary structure. For more information, see `polyest` and `polyestOptions`.

Compatibility Consideration:Backward incompatibility. See “idarx Models No Longer Returned in Multi-Output Model Estimation” on page 23-15.

Multi-Output Process Models

`idproc` models can now represent multi-output process models. Use `idproc` to create a multi-output process model. You can also use the new process model estimator function, `procest`, to estimate a multi-output `idproc` model.

For more information, see `procest` and `procestOptions`.

Interactive Response Plots with Better Look and Feel

Enhanced response plots for identified linear models allow you to interactively:

- Choose the system characteristics that are displayed. To view a system characteristic, right-click on the plot, select **Characteristics**, and then select the system characteristic of interest.
- Modify plot properties, such as whether the grid is on or off, axes labels and units, advanced plot options, etc. To modify the plot properties, right-click on the plot, and select **Properties**. The Property Editor dialog box opens. Modify the plot property of interest.

You can plot the confidence intervals associated with identified linear models. You can now plot the confidence interval interactively, by right-clicking on the plot and selecting **Characteristics > Confidence Region**. You can also use the new function, `showConfidence`, to display the confidence region on a plot via the command line.

Models Created with System Identification Toolbox Can Be Used Directly with Control System Toolbox Functions

Identified linear models that you create using System Identification Toolbox software can now be used directly with Control System Toolbox analysis and compensator design commands. In previous releases, doing so required conversion to Control System Toolbox model types.

Identified linear models include `idfrd`, `idss`, `idproc`, `idtf`, `idgrey`, and `idpoly` models.

Identified linear models can be used directly with:

- Any Control System Toolbox or Robust Control Toolbox™ functions that operate on dynamic systems, including:
  - Response plots — `nichols`, `margin`, and `rlocus`.
  - Model simplification — `pade`, `balred`, and `minreal`.
  - System interconnections — `series`, `parallel`, `feedback`, and `connect`

  For a complete list of these functions, type:

  ```
  methods('DynamicSystem')
  ```
- Analysis and design tools such as `ltiview`, `sisotool`, and `pidtool`.
- The LTI System block in Simulink models.

Improved Reliability of Numerical Computations

Algorithm sharing between the System Identification Toolbox and the Control System Toolbox products increase the accuracy and consistency of results for various operations. Operations affected include frequency-response and pole-zero computation, model conversion, settling-time deduction, and model discretization (`c2d` and `d2c`).

The handling of parameter covariance for over-parameterized systems has also improved. You can now fetch parameter covariance data in a factored form for over-parameterized systems, where the raw covariance matrix is ill-defined.

Estimating Functions and Estimation Option Sets

You can use the new estimating functions `tfest`, `ssest`, `procest`, `greyest`, `polyest`, and `impulseest` to estimate various model types. The new functions are based on the prediction error method, PEM.

Also, you can now configure model estimation objective functions and search schemes using dedicated option sets. To create and configure the option set for a model estimating function, use the corresponding option set function:

Model Estimating Function	Options Set Function	Estimated Linear Model Type
`ar`	`arOptions`	`idpoly` (AR structure polynomial)
`armax`	`armaxOptions`	`idpoly` (ARMAX structure polynomial)
`arx`	`arxOptions`	`idpoly` (ARX structure polynomial)
`bj`	`bjOptions`	`idpoly` (Box-Jenkins polynomial)
`greyest`	`greyestOptions`	`idgrey`
`iv4`	`iv4Options`	`idpoly`
`n4sid`	`n4sidOptions`	`idss`
`oe`	`oeOptions`	`idpoly` (Output-error polynomial)
`polyest`	`polyestOptions`	`idpoly`
`procest`	`procestOptions`	`idproc`
`ssest`	`ssestOptions`	`idss`
`tfest`	`tfestOptions`	`idtf`

For more information regarding these functions, enter `doc function_name` at the MATLAB command prompt.

Compatibility Considerations

The option sets replace the `Algorithm` model property.

The `Algorithm` property is no longer supported. The fields of `Algorithm` map to estimation options as follows:

Algorithm Property Field	Options Set Field
`LimitError`	`Advanced.ErrorThreshold`
`Advanced.Threshold.Zstability`	`Advanced.StabilityThreshold.z`
`Advanced.Threshold.Sstability`	`Advanced.StabilityThreshold.s`
`Advanced.Threshold.AutoInitThreshold`	`Advanced.AutoInitThreshold`
`Criterion/Weighting`	`OutputWeight` • If, `Algorithm.Criterion` was `'det'`, use `OutputWeight = 'noise'`. • If, `Algorithm.Criterion` was `'trace'`, use `OutputWeight = Algorithm.Weighting`.

Algorithm Property Field	Options Set Field
`FixedParameter`	No replacement. Use the `Structure` property of the identified linear model to designate its fixed parameters.
`MaxIter`	`SearchOption.MaxIter`
`Tolerance`	`SearchOption.Tolerance`
`MaxSize`	`Advanced.MaxSize`
`Advanced.Search`	`SearchMethod` and `SearchOptions`. These fields are available for only iterative estimation methods, such as `tfestOptions`.

Model Analysis and Validation Option Sets

You can now use option sets to configure the various attributes of model simulation and prediction commands. The option sets configure, among other things, how the initial conditions and data offsets are handled. They replace the property-value pairs used by the analysis commands as input arguments. To create and configure the option set for an analysis or validation function, use the corresponding option set creating function:

Analysis/Validation Function	Options Set Function
`predict`	`predictOptions`
`compare`	`compareOptions`
`sim`	`simOptions`
`simsd`	`simsdOptions`
`forecast`	`forecastOptions`
`findstates`	`findstatesOptions`
`pe`	`peOptions`

For more information regarding these functions, enter `doc function_name` at the MATLAB command prompt.

Compatibility Considerations

Specifying Initial Conditions and Noise Data To specify the initial conditions and noise specifications for `sim` or `simsd`, use the corresponding option set with the `InitialCondition`, `AddNoise`, and `NoiseData` options set appropriately. In previous releases, you could use name and value pair input arguments to specify these options.

Identified Linear Models

Support for Constraining and Fixing Parameters in All Identified Linear Models

You can now specify minimum/maximum bounds for, and fix or free for estimation, any parameter of an identified linear model. You use the new model property, `Structure`, to access a parameter and configure it.

Support for Model Arrays

You can now create arrays of identified linear models to analyze multiple models simultaneously. You can create an array using array subassignment. For example, `sys(:,:,k) = new sys;`.

You can also use the `stack` function to create an identified linear model array. For more information, see `stack`.

You can also use the new function, `rsample`, to create an array of models that sample an identified linear model within the uncertainty limits of its parameters. For more information see `rsample`.

Estimation Report

You can use the new `Report` property of identified linear models for information regarding the estimation performed to obtain the model.

For more information, see “Reorganization of Estimation Reports” on page 23-14.

Convert Time-Series Model to Input-Output Model for Analysis

Use the new function, `noise2meas`, to convert a time-series model, which has no measured inputs, to an input-output model for linear analysis. `noise2meas` complements the functionality of `noisecnv`, which converts an identified model with noise channels to a model with only measured inputs.

For more information, see `noise2meas`.

Specify Input/Output Pairs Using Subsystems

You can now specify subsystems as input/output models for all identified linear models, except `idgrey` models.

For example, `sys(i,j) = sys0;`

Group Inputs and Outputs

You can now group inputs and outputs for identified linear models using the `InputGroup` and `OutputGroup` properties, respectively.

For more information regarding specifying input groups, enter `help idlti.InputGroup` at the MATLAB command prompt.

For more information regarding specifying output groups, enter `help idlti.OutputGroup` at the MATLAB command prompt.

Model Parameter Interaction

New commands for interacting with the parameters of identified linear models include:

- `getpvec` — Fetch the model parameters.
- `setpvec` — Set the model parameters.
- `getcov` — Fetch the parameter covariance matrix.
- `setcov` — Set the parameter covariance matrix.
- `nparams` — Fetch number of model parameters.

For more information regarding these functions, enter `doc function_name` at the MATLAB command prompt.

Random Sampling

The new `rsample` function creates a set of perturbed systems corresponding to an identified linear model. Use this random sampling of an identified linear model for Monte-Carlo analysis.

For more information see `rsample`.

Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflows may not be supported in the future.

The following table lists affected model properties:

Property	Model Types Affected	What Happens in R2012a	Use This Instead
`ParameterVector`	`idss`, `idpoly`, `idgrey`, and `idproc`	Still available.	Use the new function `getpvec` to access model parameters. The list of parameters obtained from `ParameterVector` may differ from the list of parameters returned by `getpvec`.
`PName`	`idss`, `idpoly`, `idgrey`, and `idproc`	Still available.	Each identified linear model now has a `Structure` property, which consists of the parameters relevant to the model. Each of the parameters has an `Info` field, which may be used to store information regarding the parameter. To store the parameter name, use `Info.Label`.
`Algorithm`	`idss`, `idpoly`, `idgrey`, and `idproc`	Still available.	See “Estimating Functions and Estimation Option Sets” on page 23-4.

Property	Model Types Affected	What Happens in R2012a	Use This Instead
`CovarianceMatrix`	`idss`, `idpoly`, `idgrey`, and `idproc`	Still available.	Use the new functions, `getcov` and `setcov`, to interact with the covariance matrix of the model. Also, after a model, `sys`, is estimated, you may access the estimated covariance matrix using `sys.Report.Parameters`.
	All identified linear models.	**Backward incompatibility.** Parameter covariance is no longer translated for the following operations with identified linear models: • Model discretization • Model conversion • Model concatenation	N/A
`EstimationInfo`	`idss`, `idpoly`, `idgrey`, and `idproc`	Still available.	Replaced by the new model property, `Report`. For more information, see “Reorganization of Estimation Reports” on page 23-14.

Property	Model Types Affected	What Happens in R2012a	Use This Instead
`InputName`,`OutputName`	All identified linear models.	**Backward incompatibility.** By default, the input/output channel names are set to `''`. In previous releases, the default channel names were set to `{'u1',...}` and `{'y1',...}` for input and output channels, respectively. When an identified linear model is estimated using an `iddata` object, it will inherit the input/output channels names from the `iddata` object.	N/A
`TimeUnit`	All identified models.	You can now specify the `TimeUnit` as only one of the supported units. Supported units include: `'nanoseconds'`, `'microseconds'`, `'milliseconds'`, `'seconds'`, `'minutes'`, `'hours'`, `'days'`, `'weeks'`, `'months'`, and `'years'`.	N/A
`Ts`	`idss` and `idpoly`	**Backward incompatibility.** For discrete-time models, default is `Ts = -1`, which indicates an unspecified sample time. In previous releases, the default value of `Ts` was `1`.	N/A

Noise Channel Treatment When Converting Identified Linear Model to Numeric LTI Model

Backward incompatibility. You can convert an identified linear model to a numeric LTI model for use in Control System Toolbox. When you do so, the model returned contains only the measured components of the original model. In previous releases, the noise channels of the original model were also returned as extra inputs of the resulting model.

For example, consider the following polynomial model:

```
sys = idpoly([1 1],[1 2 3],[1 2])
```

In previous releases, executing `sys_tf = tf(sys)` returned a transfer function model with two inputs. The first input corresponded to the measured component, `B/A`. The second input corresponded to the noise component, `C/A`. `size(sys,2)` is `1` but `size(sys_tf,2)` is `2`. Thus, `sys` had one input, while `sys_tf` had two inputs.

In this release, `sys_tf = tf(sys)` returns a SISO transfer function with one input. This input corresponds to the measured component, `B/A`.`sys` and `sys_tf` both have the same number of inputs.

To obtain the noise input channels in addition to the measured inputs, as in previous releases, use the string `'augmented'` as an additional input.

```
sys_tf = tf(sys,'augmented');
```

The inputs of `sys_tf` are grouped in the `InputGroup` property. The inputs from the measured dynamics belong to the `Measured` input group, and the noise-related inputs belong to the `Noise` input group.

To obtain a model containing just the noise component of the original model, use the string `'noise'` as an additional input:

```
sys_tf = tf(sys,'noise');
```

Conversion to Identified Linear Model of Numeric LTI Models Ignores Input Groups

Backward incompatibility. In previous releases, when you converted a numeric LTI model that had an input group named `'noise'` into an identified linear model, the corresponding inputs were converted to noise channels in the resulting model. This behavior is no longer supported. You can use the `'split'` input argument when you convert a numeric LTI model to an identified model. Using the `'split'` input argument results in the last *Ny* inputs being treated as noise channels in the identified model. Here, *Ny* is the number of outputs.

For example, in previous releases:

```
sys = rss(2,2,5);
sys.InputGroup = struct('noise',4:5);
sys_idss = idss(sys);
```

resulted in `sys_idss` having the fourth and fifth inputs of `sys` being treated as noise channels.

In this release, use:

```
sys_idss = idss(sys,'split');
```

As `sys` has two outputs and five inputs, its last two input channels are converted to noise channels in `sys_idss`. `sys_idss` has three measured input channels.

Input Channel Referencing for Measured Components

You can configure an estimated model to be free of the influence of noise by setting the `NoiseVariance` property value to `0`. In previous releases, you achieved this result by subreferencing the inputs of the model using the `'measured'` string, as in `sys(:,'measured')`. This type of subreferencing is provided in this release for backward compatibility only and may not be supported in the future.

Input Channel Referencing for Noise Components

You can now extract only the noise components of an identified linear model using the syntax:

```
sys_noise_only = sys(:,[]);
```

Here, the `:` indexes all the outputs and `[]` specifies that none of the measured inputs are extracted. `sys_noise_only` has zero measured inputs and is consequently a noise model.

In previous releases, you achieved this result by subreferencing the inputs of the model using the `'noise'` string, as in `sys(:,'noise')`. This type of subreferencing is provided in this release for backward compatibility only and may not be supported in the future.

Model Precedence Rules

The precedence order among identified linear models is `idfrd > idss > idpoly > idtf > idproc` and `idss > idgrey`.

When you combine a numeric LTI model with an identified model, the resulting model is a numeric LTI model. Interconnecting and combining identified linear models using functions such as `series`, `parallel`, and `feedback`, and performing model addition results in a numeric LTI model. Input-output concatenation and model stacking of identified models returns an identified model object.

Simultaneous Model-Type Conversion and Property Value Setting

Model conversion functions will not support setting model property values in the future.

Replace calls such as:

```
sys_idfrd = idfrd(sys,w,'InputName','u1','InputDelay',3);
```

With:

```
sys_idfrd = idfrd(sys,w);
set(sys_idfrd,'InputName','u1','InputDelay',3);
```

Replace inpd2nk with absorbDelay

The `inpd2nk` is now obsolete. Use `absorbDelay` instead to absorb all time delays of a dynamic system model into the system dynamics or the frequency response data. In this release, calling `inpd2nk` results in the toolbox making an internal call to `absorbDelay`.

For more information, see `absorbDelay`.

System Identification Tool GUI

Transfer Function Models

You can now estimate transfer functions using the System Identification Tool GUI.

To open the Transfer Function dialog box:

1. Import a data set into the System Identification Tool GUI.
2. In the **Estimate** list, select **Transfer Function Models**.

For more information regarding transfer function estimation, open the Transfer Function dialog box, and click **Help**.

Process Models

You can now estimate multi-output process models using the System Identification Tool GUI.

To open the Process Models dialog box:

1. Import a data set into the System Identification Tool GUI.
2. In the **Estimate** list, select **Process Models**.

For more information regarding process model estimation, open the Process Model dialog box and click **Help**.

State-Space Models

You can now use the System Identification Tool GUI for these operations:

- Estimate continuous-time state-space models.
- Specify the parameterization form, such as canonical or modal.
- Specify feedthrough, which determines whether the D matrix of the state-space model is treated as free estimation parameter or fixed to zero.
- Specify input delay.

To open the Polynomial and State Space Models dialog box:

1. Import a data set into the System Identification Tool GUI.
2. In the **Estimate** list, select **State Space Models**.

For more information regarding state-space estimation, open the Polynomial and State Space Models dialog box and click **Help**.

Polynomial Models

You can now specify noise integration and input delays when estimating polynomial models using the System Identification Tool GUI.

You can also estimate multi-output polynomial models by specifying the appropriate model order.

To open the Polynomial and State Space Models dialog box:

1. Import a data set into the System Identification Tool GUI.
2. In the **Estimate** list, select **Polynomial Models**.

For more information regarding polynomial estimation, open the Polynomial and State Space Models dialog box and click **Help**.

Compatibility Consideration: You no longer select **Linear parameteric models** to open the Polynomial and State Space Models dialog box.

Changes Introduced in This Version

Changes introduced in this version:

- “Reorganization of Estimation Reports” on page 23-14

Reorganization of Estimation Reports

A new property of identified linear models, `Report`, provides information regarding the performed estimation. This property replaces the `EstimationInfo` property and provides additional information regarding:

- All estimated quantities — Parameter values and covariance, initiate state values for state-space models and values of input levels for process models.
- The option set used for estimation.
- Additional fit criteria — Percentage fit to estimation data and the mean square error.

The `Report` field is mostly uniform for the various identified linear models. However, certain fields of `Report` are model dependent.

To access the `Report` property of an identified linear model, `sys`, use `sys.Report`.

Compatibility Considerations

`Report` replaces the `EstimationInfo` property. The fields of `EstimationInfo` map to those of `Report` as:

EstimationInfo Field	Report Field
`LossFcn`	`Fit.LossFcn`
`FPE`	`Fit.FPE`
`DataName`	`DataUsed.Name`
`DataLength`	`DataUsed.Length`
`DataTs`	`DataUsed.Ts`
`DataDomain`	`DataUsed.Type`
`DataInterSample`	`DataUsed.InterSample`
`WhyStop`	`Termination.WhyStop` `Termination` information is not provided for models estimated using a noniterative estimation function, such as `arx` or `n4sid`.

EstimationInfo Field	Report Field
`UpdateNorm`	`Termination.UpdateNorm` `Termination` information is not provided for models estimated using a noniterative estimation function, such as `arx` or `n4sid`.
`LastImprovement`	`Termination.LastImprovement` `Termination` information is not provided for models estimated using a noniterative estimation function, such as `arx` or `n4sid`.
`Iterations`	`Termination.Iterations` `Termination` information is not provided for models estimated using a non-iterative estimation function, such as `arx` or `n4sid`.
`InitialState`	Either: • `InitialState` (state-space models) • `InitialCondition` (other identified linear models)
`Warning`	No replacement.

Polynomial Models

Polynomial Model Estimators

Use the new function, `polyest`, to estimate a polynomial model containing an arbitrary subset of A, B, C, D, and F polynomials.

For more information, see `polyest` and `polyestOptions`.

Also, the functions `ar`, `arx`, `bj`, `oe`, and `armax` now support multi-output polynomial estimation.

Integration on Noise Models (ARIMA models)

You can now introduce integrators in the dynamics of the disturbances added to the output of the model.

For more information, see “Generating ARIMA Models” on page 23-3.

idarx Models No Longer Returned in Multi-Output Model Estimation

`idarx` models are no longer returned when you use estimating functions for multi-output ARX models. Support for `idarx` models may not be provided in the future. Use `idpoly` models to estimate and represent multi-output ARX models instead.

Compatibility Consideration: Backward incompatibility. `arx`, `iv4`, and `ivx` now return `idpoly` models for multi-output estimation. In previous releases, they returned `idarx` models.

To convert an existing `idarx` model, `sys_idarx`, to an `idpoly` model, use `idpoly(sys_idarx)`.

Similarly, to convert an existing `idpoly` model, `sys_idpoly`, to an `idarx` model, use `idarx(sys_idpoly)`.

Specify Transport Delays

Use the new `idpoly` property, `ioDelay` to specify the transport delays for individual input/output pairs.

You can use `ioDelay` as an alternative to the `nk` order when estimating polynomial models. Using `ioDelay` reduces the complexity of the model by factoring out the leading zeros of the `B` polynomials, controlled by `nk`.

For example:

```
load iddata1 z1
load iddata2 z2
data = [z1 z2(1:300)];
na = [2 3; 1 2];
nb = [1 2; 2 2];
nk = [2 1; 7 0];
sys1 = arx(data,[na nb nk]);
sys2 = arx(data,[na nb zeros(2)],'ioDelay',nk);
```

In this case, `sys1` and `sys2` are equivalent, but `sys2.b` shows fewer terms in each `B` polynomial than `sys1.b`.

For more information, see `idpoly`.

Specify Display Variable

You can now specify the variable used to display model equations for `idpoly` models. Use the new model property, `Variable`. For continuous-time models, specify either `'s'` or `'p'` as the variable. For discrete-time models, use either `'z^-1'` or `'q^-1'` as the lag variable.

For more information, see `idpoly`.

Multi-Output Weighting Using arx

For estimating multi-output ARX models, use the `OutputWeight` estimation option to specify the output weighting. You create the option set for ARX model estimation using `arxOptions`. In previous releases, to do so you specified a `NoiseVariance` name-value pair input for `arx`.

`arx` uses the following syntaxes for assigning output weight:

Syntax	Output Weight Value
`arx(data,[na,bk,nk])`	`eye(Ny)`, where *Ny* is the number of outputs
`arx(data,[na nb nk],opt)`, where `opt` is an option set created using `arxOptions`	`opt.OutputWeight` If `opt.OutputWeight = []`, then `eye(Ny)`.
`arx(data,init_model)`, where `init_model` is an estimation initialization model	`init_model.NoiseVariance`
`arx(data,init_model,opt)`	`opt.OutputWeight` If `opt.OutputWeight = []`, then `init_model.NoiseVariance`.

Polynomial Structure

The new `Structure` property of `idpoly` models stores the adjustable parameters, which include:

- The active polynomials

  For example, consider the ARX model:

  ```
  A = [1 2 1];
  B = [0 3 4];
  sys = idpoly(A,B);
  ```

  `sys.Structure` lists the polynomials A and B as parameters. You can specify nominal values and constraints for these parameters.

  `sys.Structure` does not list the C, D, and F polynomials.
- The transport delays and integrate noise flag

  You can set these delays and the flag for models of any polynomial configuration.

You interact with the `Structure` property to specify constraints (such as maximum/minimum bounds) for the various parameters. To change only the values of the polynomials or the transport delays, use the relevant `idpoly` model property, viz `a`, `b`, `c`, `d`, `f`, `ioDelay`, and `IntegrateNoise`.

For more information, see `idpoly`.

Compatibility Considerations

The recommended usage and workflow has changed for some model parameters and functionality. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected functionality:

Functionality	What Happens in R2012a	Use This Instead
Model properties that store the polynomial order — `na`, `nb`, `nc`, `nd`, `nf`, and `nk`	You may still modify the value of these properties as long as their sizes are compatibility with the input/output sizes. The estimation commands for polynomial models will continue to support the specification of “in-model” delays using `nk`.	Use `idpoly` to create a new model of desired orders. Use `ioDelay` and `InputDelay` to specify delays separate from the B polynomial.
Model properties that store standard deviation information — `da`, `db`, `dc`, `dd`, and `df`	You may still access these model properties using dot notation. For example, `sys.da`.	Use the functions `getpvec` and `polydata` to access parameters and their standard deviations.

Functionality	What Happens in R2012a	Use This Instead

Treatment of the leading zeros of the B polynomials	If you have a discrete-time `idpoly` model that has `nk` leading zeros, then `nk-1` of them are treated as delays. When you convert such a model into another linear model, these delays are set to the appropriate delay related property.
	
	For example,
	
	`sys = idpoly([1 2],...`
	`[0 0 0 4]);`
	`% nk = 3`
	`sys2 = tf(sys);`
	
	The `ioDelay` property of `sys2` is 2, and the numerator is `{[0 4]}`.
		N/A

Model property — `InitialState`	Still works.	Use the option, `InitialCondition`, when creating the relevant option set for estimation, prediction, simulation, and comparison.
Storage of the B and F polynomials	For multi-input models, the `b` and `f` properties are no longer saved as a matrix of doubles. These properties will now be saved using cell arrays. To continue storing these properties as a matrix of doubles, use `setPolyFormat`	N/A

Treatment of the trailing zeros of the B and F polynomials	Trailing zeros in the *B* and *F* polynomials of a discrete-time `idpoly` model are not discarded.
	
	For example, in previous releases:
	
	`sys = idpoly([1 2],...`
	`[2 4 0 0 0]);`
	
	resulted in `[2 4]` as the *B* polynomial for `sys`.
	
	Now, the same code gives `[2 4 0 0 0]` as the B polynomial for `sys`. Similar considerations apply to leading zeros of *B*, *F* polynomials of a continuous-time model.
		N/A

State-Space Models

State-Space Model Estimator

The new function, `ssest`, can be used to estimate a discrete-time or continuous-time identified state-space model. You can use time-domain or frequency-domain data with `ssest` and perform both structured and unstructured model estimation. You can also choose a canonical form of the identified state-space model.

To configure the handling of initial conditions and other initialization choices, data offsets and search algorithm, use the associated option command, `ssestOptions`.

For more information, see `ssest` and `ssestOptions`.

For a structured state-space model, which is an idss model with finite parameters, you can use either `pem` or `ssest` to update the values of those parameters for measured input-output data.

n4sid Supports Canonical Forms

The subspace estimator function, `n4sid`, now supports new parameterization options, such as modal and companion canonical forms and the presence of feedthrough.

To configure the handling of initial conditions and other initialization choices and data offsets, use the associated option command, `n4sidOptions`.

For more information, see `n4sid` and `n4sidOptions`.

State-Space Structure

The new `Structure` property of `idss` models stores the adjustable parameters, which include the `a`, `b`, `c`, `d` and `k` matrices.

You interact with the `Structure` property to specify constraints (such as maximum/minimum bounds) for the various parameters. To only change the values of the matrices, use the relevant `idss` model property, viz `a`, `b`, `c`, `d`, and `k`.

For more information, see `idss`.

Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected model properties:

Model Property	What Happens in R2012a	Use This Instead

`X0`, `InitialState`	Still available.	Use the `InitialState` option for estimation and the `InitialCondition` option for prediction, simulation, and comparison.
		
		For example, replace:
		
		`sys = n4sid(data,2,...`
		`'InitialState','estimate');`
		
		with:
		
		`opt = n4sidOptions(...`
		`'InitialState','estimate');`
		`sys = n4sid(data,2,opt);`

`As`, `Bs`, `Cs`, `Ds`, `Ks`, and `X0s`	Still available.	Use the `Structure` property to specify constraints (such as maximum/minimum bounds) for A, B, C, D, and K. Use the `InitialState` estimation option to specify constraints on the initial state vector.
		
		For example, instead of:
		
		`sys = idss(A,B,C,D,K);`
		`sys.X0s = [nan;1]`
		`syse = pem(data, sys);`
		
		Use:
		
		`opt = ssestOptions;`
		`X0 = idpar([nan; 1]);`
		`X0.Free(2) = false;`
		`opt.InitialState = X0;`
		`sys = idss(A,B,C,D,K);`
		`syse = ssest(data, sys, opt);`

`da`, `db`, `dc`, `dd`, and `dk`	Still available.	Use the new function `idssdata` to obtain the state-space matrix standard deviations.

Model Property	What Happens in R2012a	Use This Instead

`nk`	Still available but **may cause a backward incompatibility**.
	
	If you previously specified both `nk` and `InputDelay`, you could see different results in this release. For example,
	
	`load iddata1 z1;`
	`sys = pem(z1,4,...`
	`'nk',5,'InputDelay,2);`
	
	In this release, `sys.nk` is 3, whereas `sys.nk` was 5 in earlier releases.
		For estimation, use the `InputDelay` and `Feedthrough` estimation properties instead. When creating an `idss` model, specify the `InputDelay` and `Structure.d` properties.
		
		`nk`, `InputDelay`, and `Feedthrough` are related:
		
		• `nk(j) = 0` means that the model has no delay for the j^{th} input. Therefore, `InputDelay` is `0`, and `Structure.d.Free(:,j)` is `true`.
		• `nk(j) = 1` means that the model has zero delay for the j^{th} input. Therefore, `InputDelay` is `0`, and there is no feedthrough. `Structure.d.Free(:,j)` is `false`, and `Structure.d.Value(:,j)` is zero.
		• `nk(j) = N, N>1` means that the model has nonzero delay for the j^{th} input. Therefore, `InputDelay` is `N-1`, and there is no feedthrough. `Structure.d.Free(:,j)` is `false`, and `Structure.d.Value(:,j)` is `0`.
		
		`nk > 1` can only be used for a discrete-time model.

Model Property	What Happens in R2012a	Use This Instead

`SSParameterization`	Still available.
	
	However, when you use `get` to obtain the value of `SSParameterization`, the software may report a canonical form as the structured form.
		• Use the `'form'/value` name-value pair when estimating using either `n4sid` or `ssest` to specify the form of the estimated model.
		• To change the structure of an existing model, use one of these methods:
		  • Change each matrix individually using the `Structure` property.
		  • Use `canon` to specify a canonical form.
		  • Use `ss2ss` and specify a transformation matrix.
		
		Note Parameter covariance is not translated in these operations.

`DisturbanceModel`	Still available.	For estimation, specify `DisturbanceModel` as an option for estimation. For changing the model structure, for its disturbance component, use `Structure.k.Value` and `Structure.k.Free` instead. For example, `DisturbanceModel = 'none'` corresponds to setting `model.Structure.k.Value` to zeros and `model.Structure.k.Free` to `false`.
`CanonicalIndices`	Still available if the model is in canonical form.	Use `canon` and `ss2ss` to change the state-space form.

Process Models

Process Model Estimator

The new function, `procest`, lets you estimate process models using time-domain or frequency-domain data. You can also specify the handling of input offsets and disturbances using an option set for this function using `procestOptions`.

For more information, see `procest` and `procestOptions`.

Multi-Output Support

You can now create and estimate multi-output process models.

For more information, see “Multi-Output Process Models” on page 23-3

Noise Transfer Function

Use the new property `NoiseTF` of `idproc` models to specify the value of the noise transfer function in numerical form. `NoiseTF` is a structure with the fields `num` (numerator) and `den` (denominator) representing the noise-transfer function. This property replaces the `DisturbanceModel` property.

Input Delay

The `InputDelay` property of `idproc` model represents input delays and is now independent of the `Td` property.

The `Td` property represents the transport delay, which is thus similar to the `ioDelay` property of `idpoly` and `idtf` models.

For more information, see `idproc`.

Process Model Structure

The `Structure` property of `idproc` models houses active parameters. These parameters are a subset of `Kp`, `Tp1`, `Tp2`, `Tp3`, `Tw`, `Zeta`, `Td`, and `Tz`, depending on the `Type` option used to create the model. `Structure` also contains the `Integration` property whose value determines if the model structure contains an integrator.

You use the `Structure` property to specify constraints (such as maximum/minimum bounds) for the various active parameters.

`Structure` is an `Ny`-by-`Nu` array, where `Ny` is the number of outputs and `Nu` is the number of inputs. The array specifies a transfer function for each input/output pair.

For example:

```
sys = idproc({'p2u' 'p0' 'p3zi'; 'p1' 'p2d' 'p2uz'});
```

In this case, `sys.Structure` is a 2-by-3 array. `sys.Structure(1,1).Zeta` is a parameter, while `sys.Structure(1,2)` does not have a `Zeta` field, as this parameter is inactive for the (1,2) output-input pair.

To change the list of active parameters, you must create a new model. However, you may change the `Integration` property at any time.

Lower Bound on Time Constants

The minimum value permitted for the time constants of an `idproc` model, `Tp1`, `Tp2`, `Tp3`, `Tw`, and `Zeta` is now `0`. In previous releases, you could not specify for these constraints a value smaller than `0.001`. For well-conditioned estimations, it is still recommended that you specify reasonable upper and lower bounds around the time-constant values.

Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected model properties:

Model Property	What Happens in R2012a	Use This Instead
`InputLevel`	Still available.	Use the `InputOffset` option for estimation using `procestOptions`. For advanced control, you can specify the `InputOffset` option as `'estimate'` or a `param.Continuous` object.

`InitialState`	Still available.	Use the `InitialCondition` option for estimation, prediction, simulation and comparison.
		
		For example, replace:
		
		`sys = pem(data,'p1d',...`
		`'InitialState','estimate');`
		
		with:
		
		`opt = procestOptions(...`
		`'InitialCondition','estimate');`
		`sys = procest(data,...`
		`'p1d',opt);`

Model Property	What Happens in R2012a	Use This Instead

`DisturbanceModel`	Still available.	The `DisturbanceModel` property of `idproc` models in previous releases represented both the estimation flag and as the actual value of the noise transfer function. The `DisturbanceModel` property has now been replaced by:
		• The `NoiseTF` property, which represents the value of the noise transfer function.
		• The `DisturbanceModel` estimation option, which is contained in the `procestOptions` option set. This option stores the flag, which determines how the noise transfer function is estimated.
		For example, replace:
		`load iddata1 z1;`
		`sys = pem(z1,'p1d',...`
		`'DisturbanceModel','arma1');`
		`NoiseTF = sys.DisturbaceModel{2};`
		with:
		`load iddata1 z1;`
		`opt = procestOptions(...`
		`'DisturbanceModel','arma1');`
		`sys = pem(z1,'p1d',opt);`
		`NoiseTF = sys.NoiseTF;`
		For more information, see `procestOptions`.

`X0`	Still available.	There is no replacement for this model property as `idproc` is not a state-space model. Continuing to use `X0` may produce bad results.

Model Property	What Happens in R2012a	Use This Instead

Kp, Tp1, Tp2, Tp3, Tw, Zeta, Td, and Tz	**Backward incompatibility.**
	
	These properties are now saved as double matrices. In previous releases, they were stored as structures.
	
	Assigning the value of these parameters to structures will continue to work:
	
	`model = idproc('p1','Tp1',1,'Kp',2)`
	`model.Tp1.value = 5;`
	
	In previous releases, you could obtain the value of a parameter as a structure and access its fields. Now, you will receive an error.
	
	`model = idproc('p1','Tp1',1,'Kp',2)`
	`Tp1 = model.Tp1;`
	`Tp1.status % throws error`
	
	However, subreferencing for a field of the old parameter structure will continue to work:
	
	`model = idproc('p1','Tp1',1,'Kp',2)`
	`model.Tp1.status`
	`% returns {'estimate'}`
		Use the `Structure` property to specify parameter constraints.
		
		`Structure` replaces the specification of process model parameter bounds. See Call Replacements.

Call Replacements

Replace a Call Like...	With...
`model.Tp1.status = {'estimate'}`	`model.Structure.Tp1.Free = true;`
`model.Tp1.status = {'zero'}`	`model.Structure.Tp1.Free = false;` `model.Structure.Tp1.Value = 0;`
`model.Tp1.status ={ 'fixed'}`	`model.Structure.Tp1.Free = false;`
`model.Tp1.min = value`	`model.Structure.Tp1.Minimum = value`
`model.Tp1.max = value`	`model.Structure.Tp1.Maximum = value`
`model.Tp1.value = value`	`model.Structure.Tp1.Value = value`
For multi-input models: `model.Tp1.status{2} = 'estimate'`	`model.Structure(1,2).Tp1.Free = true;`
For multi-input models: `model.Tp1.value(2)= value`	`model.Structure(1,2).Tp1.Value = value`

Linear Grey-Box Models

Linear Grey-Box Model Estimator

The new function `greyest` lets you estimate the parameters of a linear grey-box model. You can specify an option set for the estimation by using the function, `greyestOptions`.

For more information, see `greyest` and `greyestOptions`.

Complex Parameters Support

You can now parameterize a real system using complex-conjugate pairs of parameters in an `idgrey` model.

When the parameters of such a system are estimated, they continue to be complex conjugates. Thus, symmetry is maintained across the real axis.

For more information, see the related example in the `greyest` reference page.

ODE file API

You can now specify an arbitrary number of parameters as independent input arguments to the ODE file. In previous releases, the parameters of the model had to be consolidated into a single vector that was then passed as the first input argument of the ODE file. Now, you can pass independent parameters as separate input arguments. The same holds true for the optional input arguments.

Old syntax:

```
ODEFUN(ParameterVector, Ts, OptionalArg)
```

New syntax:

```
ODEFUN(Par1, Par2,  …, ParN, Ts, OptArg1, OptArg2, …)
```

If all the model parameters are scalars, you can still combine them into a single vector and pass them as a single input argument to the ODE file.

Also, specifying the value for the output arguments `K` and `X0` is now optional. In earlier releases, you were required to set a value for `K` and `X0` even if you did not want to parameterize them. Now, you can omit them entirely from the output argument list. For more information, see `idgrey`.

Linear Grey-Box Model Structure

The `Structure` property of the `idgrey` model stores information on the ODE function and its parameters. `Structure` contains the following properties:

Property	Role
`FcnType`	The sample time handling behavior of the linear ODE model. `FcnType` specifies whether the ODE file returns state-space data that corresponds to one of the following: • `'c'` — A continuous-time model. • `'d'` — A discrete-time model. • `'cd'` — A continuous-time model if the sample time is `0` and a discrete-time model if the sample time greater than `0`. **Compatibility Consideration:** Use instead of the `CDmfile` property.
`Function`	Name or function handle to the MATLAB function that parameterizes the state-space structure. **Compatibility Consideration:** Use instead of the `MfileName` property.
`Parameters`	Vector of parameter objects, with an entry for each model parameter. Use the parameter object to specify initial values and minimum/maximum constraints. You can also indicate whether the parameter is a free- or fixed- estimation parameter.
`ExtraArgs`	Option input arguments used by the ODE file to compute the state-space data. **Compatibility Consideration:** Use instead of the `FileArgument` property.
`StateName`	Model state names.
`StateUnit`	Model state units.

Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected model properties:

Model Property	What Happens in R2012a	Use This Instead
`MfileName`	Still available.	Use the `Structure.Function` property to specify the ODE function name or function handle instead.

Model Property	What Happens in R2012a	Use This Instead
`X0`	Still available.	Use the `InitialState` option when you create an estimation option set using `greyestOptions`.
`dA`, `dB`, `dC`, `dD`, `dK` and `dX0`	Still available.	Use the functions `getpvec` and `idssdata` to access parameters and their standard deviations.
`FileArgument`	Still available.	Use the `Structure.ExtraArgs` property to specify the additional ODE function arguments.
`CDmfile`	Still available.	Use the `Structure.FcnType` property to specify sample time handling behavior.
`InitialState`	Still available.	Use the `InitialState` option for estimation and the `InitialCondition` option for prediction, simulation and comparison.
`DisturbanceModel`	Still available.	Use the `DisturbanceModel` estimation option in the option set created using `greyestOptions`.

Identified Frequency-Response Data Models

Specify InterSample Behavior of Inputs

You can use the new `InterSample` property of `idfrd` models to specify the behavior of the input signals between samples for model transformations between discrete-time and continuous-time. This property is relevant only for discrete-time `idfrd` models.

For more information, see the `InterSample` property information in `idfrd`.

Frequency Unit

Use the new property `FrequencyUnit` of `idfrd` models to specify the units for frequency-domain data.

For a list of the supported units for `FrequencyUnit`, see `idfrd`.

Compatibility Consideration: The `FrequencyUnit` property replaces the `Unit` property.

Compatibility Considerations

Input Delay Treatment (**Backward incompatibility.**) When you convert an identified model into an `idfrd` model, its `InputDelay` and `ioDelay` properties are translated into the corresponding properties of the `idfrd` model. In previous releases, the delays were absorbed into the `ResponseData` property as additional phase lag.

The `OutputDelay` property of an identified model is converted to the `ioDelay` property of an `idfrd` model.

Identification Data Objects

Frequency-Domain Data Units

Use the new property `FrequencyUnit` of `iddata` objects to specify the units for frequency-domain data.

For a list of the supported units for `FrequencyUnit`, see `iddata`.

Compatibility Consideration: The `FrequencyUnit` property replaces the `Unit` property.

Impulse and Step Response Plots

Plot the impulse or step response for `iddata` objects by estimating a discrete-time transfer function model using `impulseest`. Use the resulting model as the input argument for `impulse` or `step`.

In the previous release, you could plot the step response without first estimating a discrete-time transfer function model:

```
load iddata1 z1;
step(z1);
```

where `z1` is an `iddata` object.

Now, you must use `impulseest` to estimate a discrete-time transfer function. Then, plot the appropriate response for the model. For example:

```
load iddata1 z1;
sys = impulseest(z1);
step(sys);
```

For more information, see `impulseest`.

Compatibility Consideration: Backward incompatibility. To see the step or impulse response for negative time values, use the `noncausal` input argument with `impulseest`. In previous releases, you could call `impulse(data)` to do this.

Compatibility Considerations

Supported Units for TimeUnit Property You can now specify the `TimeUnit` property of an `iddata` object as only one of the supported units. Supported units include: `'nanoseconds'`, `'microseconds'`, `'milliseconds'`, `'seconds'`, `'minutes'`, `'hours'`, `'days'`, `'weeks'`, `'months'`, and `'years'`.

Analysis Commands

Function	What Has Changed in R2012a
`predict`	• `predict` now returns a data object of the same type as the input data. • You can now specify an infinite prediction horizon with time-series models. When you specify the prediction horizon as `Inf`, `predict` returns the initial condition response of the model. • **Compatibility Consideration:** For a multi-output system, the predictor model is now returned as a dynamic system. In previous releases it was returned as a cell array.

`compare`	• When using FRD validation data, `compare` plots the magnitude and phase response. The fit percentage shown corresponds to the closeness of the complex frequency response of the system to that of the data (using normalized root mean square, NRMSE).
	• For complex-valued validation data or model, `compare` plots the real and imaginary parts on separate axes.
	• You can now use `compare` to compare data sets. The data sets may be either `iddata` or `frd` objects.
	• You can interactively change the prediction horizon for time-domain comparison plots. You can also interactively change the initial conditions. Right-click on the plot to select the appropriate option.
	• You can now compare arrays of systems to a validation data set.
	• You can now specify the initial conditions and sample range for comparison using the option set created by the new function `compareOptions`. For more information, see `compareOptions`.
	• **Compatibility Consideration: Backward incompatibility.** The format of the outputs has changed when you call `compare` using the syntax:
	`[yh,fit,x0] = compare(data,...`
	`sys1,...,sysn,m,options)`
	For example, `fit` is a cell array rather than a 3-d numeric array when comparing responses of multiple systems or when using multi-experiment validation data.

Function	What Has Changed in R2012a

`step`	• You can specify an option set for the generated plot using the function `stepDataOptions`.
	• You can customize a step plot by creating a plot using `stepplot`. Then, to display confidence intervals on the plot programmatically, use `showConfidence`.
	• **Compatibility Considerations:**
	  • Specify the number of standard deviations for the confidence region using the new `ConfidenceRegionNumberSD` option in the corresponding option set. In previous releases, you used the `'sd'/N` name-value pair to specify the number of standard deviations.
	  • **Backward incompatibility.** Using a 2-element double vector to indicate the plot time range is no longer supported. You can only specify a scalar, the final time, or a vector containing the time instants to be plotted.
	  • **Backward incompatibility.** The third output argument now returns the state trajectory. In previous releases, the third output argument was the response standard deviation, which is now returned as the fourth output argument.

Function	What Has Changed in R2012a

`impulse`	• You can specify an option set for the generated plot using the function, `timeoptions`. For more information, see `timeoptions`.
	• You can customize an impulse plot by creating a plot using `impulseplot`. Then, to display confidence intervals on the plot programmatically, use `showConfidence`.
	• **Compatibility Considerations:**
	  • Specify the number of standard deviations for the confidence region using the new `ConfidenceRegionNumberSD` option in the corresponding option set. In previous releases, you used the `'sd'/N` name-value pair to specify the number of standard deviations.
	  • **Backward incompatibility.** Using a 2-element double vector to indicate the plot time range is no longer supported. You can only specify a scalar, the final time, or a vector containing the time instants to be plotted.
	  • **Backward incompatibility.** The third output argument now returns the state trajectory. In previous releases, the third output argument was the response standard deviation, which is now returned as the fourth output argument.

Function	What Has Changed in R2012a

`bode`	• To customize a bode plot, use `bodeplot`. You can specify an option set for the generated plot using the function `bodeoptions`. For more information, see `bodeplot` and `bodeoptions`.
	
	To display confidence intervals on a bode plot programmatically, use `showConfidence`.
	• **Compatibility Considerations:**
	  • Specify the number of standard deviations for the confidence region using the new `ConfidenceRegionNumberSD` option in the corresponding option set. In previous releases, you used the `'sd'/N` name-value pair to specify the number of standard deviations.
	  • The plot input arguments `'fill'`,`'mode'`, and `'AP'` are no longer supported. Use the plot options, `bodeoptions`,`getoptions` and `setoptions`, instead. Alternatively, you may interactively change these options by right-clicking on the plot and choosing the appropriate options.
	  • **Backward incompatibility.** You can no longer specify the frequency range using `w = {wmin, wmax,np}`. Instead, use `logspace(wmin,wmax,np)`.
	  • Do not use `bode` for plotting time-series models. Instead, use the new function `spectrum`. For more information, see `spectrum`.

`pzmap`	**Compatibility Considerations:** • **Backward incompatibility.** For multi-input, multi-output systems, `pzmap` now shows the system poles and transmission zeros. In previous releases, `pzmap` showed the poles and zeros of individual input/output pairs. To plot the poles and zeros for individual input/output pairs, use `iopzmap` and `iopzplot`. For more information, enter `help function_name` at the MATLAB command prompt. • The `'sd/N'` name-value input argument for displaying the pole-zero confidence regions is no longer supported. Instead, use `iopzmap` and its corresponding options set (`pzoptions`). Use the `ConfidenceRegionNumberSD` option to specify the standard deviations for the confidence regions. You can also use the `showConfidence` command to view the confidence regions programmatically.

Function	What Has Changed in R2012a

`nyquist`	• You can customize a nyquist plot by creating the plot using `nyquistplot`. Then, to display confidence intervals on the plot programmatically, use `showConfidence`.
	• **Compatibility Considerations:**
	  • The `'sd/N'` name-value input argument for displaying the confidence ellipses is no longer supported. Create an option set using `nyquistoptions`. Use the `ConfidenceRegionNumberSD` option to specify the standard deviations for the confidence ellipses. Use the `ConfidenceRegionDisplaySpacing` option to specify the spacing of the confidence ellipses. For more information, see `nyquistoptions`.
	  • **Backward incompatibility.** You can no longer obtain the complex frequency response and its uncertainty as the outputs of `nyquist`. Instead, use `freqresp` to obtain these values.
	  `nyquist` now returns the real and imaginary parts of the frequency response and their individual uncertainties. For more information, see `nyquist`.
	  • **Backward incompatibility.** You can no longer specify the frequency range using `w = {wmin, wmax,np}`. Instead, use `logspace(wmin,wmax,np)`.
	  • The plot input name-value pair`'mode'/'same'`is no longer supported. Use the plot options instead (see `nyquistoptions`,`getoptions` and `setoptions`). Alternatively, you may interactively change these options by right-clicking on the plot and choosing the appropriate options.

Function	What Has Changed in R2012a

`c2d`	• You can now use the conversion methods `'tustin'`, `'matched'` and `'impulse'` without requiring the Control System Toolbox software.
	• You can specify the conversion method and associated option for `c2d` using `c2dOptions`. For more information, see `c2dOptions`.
	• **Compatibility Considerations:**
	  • Parameter covariance translation is no longer supported by `c2d`. Therefore, the `'CovarianceMatrix'-'none'` name-value pair is no longer supported.
	  • **Backward incompatibility.** Grey-box models of `FcnType'c'` cannot be discretized directly. Instead, convert such models to `idss` models before using `c2d`.
	  • **Backward incompatibility.** Process models cannot be discretized directly. You must first convert your process model to an `idpoly` model or an `idtf` model and then discretize the new model.

`d2c`	• You can now use the conversion methods `'tustin'` and `'matched'` without requiring the Control System Toolbox software.
	• You can specify the conversion method and associated option for `d2c` using `d2cOptions`. For more information, see `d2cOptions`.
	• **Compatibility Consideration:**
	  • Parameter covariance translation is no longer supported by `d2c`. Therefore, the `'CovarianceMatrix'-'none'` name-value pair is no longer supported.
	  • **Backward incompatibility.**
	  Grey box models of `FcnType'd'` cannot be converted into continuous-time models directly. Instead, convert such models to `idss` models before using `d2c`.
	  • The input name-value pair `'InputDelay'/0` are no longer supported. Input delays are now handled uniformly, as described in Continuous-Discrete Conversion Methods.

Function	What Has Changed in R2012a
`ssdata`	• Use the new function `idssdata` to fetch state-space matrices for identified linear models. If `idssdata` is used for a model other than `idss` or `idgrey`, it returns empty matrices for uncertainty outputs. For more information, see `idssdata`. • You can still call the `ssdata` command with six or more output arguments to fetch the state-space matrices and related uncertainty information. However, this syntax of `ssdata` may be removed in the future and it is recommended to use `idssdata` instead. • **Compatibility Consideration:Backward incompatibility.** `ssdata` now returns the sampling time, `Ts`, as the fifth output when it is called with five outputs. In previous releases, `ssdata` returned the disturbance matrix, *K*, as the fifth output.
`tfdata`	**Compatibility Consideration: Backward incompatibility.** `tfdata` now returns the sampling time, `Ts`, as the third output. In previous releases, `tfdata` returned the numerator standard deviation as the third output. The new syntax is: `[num,den,Ts,sdnum,sdden] = tfdata(sys);` `sdnum` and `sdden` are `[]` if `sys` does not contain uncertainty information or for multi-output polynomial models with a nondiagonal *A* polynomial array.
`zpkdata`	**Compatibility Consideration: Backward incompatibility.** `zpkdata` now returns the sampling time, `Ts`, as its fourth output argument. In previous releases, `zpkdata` returned the standard deviations of the zeros. The new syntax is: `[z,p,k,z,Ts,covz,covp,covk] = zpkdata(sys)` where `covz`, `covp` and `covk` are the covariance of the zeros, poles and gain of `sys`.
`canon`	You can use the new function `canon` to transform `idss` models into various canonical forms. For more information, see `canon`.

Function	What Has Changed in R2012a
`findstates(idParametric)`	You can now specify arbitrary prediction horizons for `findstates`. You can use an option set to specify the option for `findstates`. Use the new function `findstatesOptions` to create the option set. For more information, see `findstatesOptions`.
`ffplot`	`ffplot` is no longer supported. Use `bodeplot` instead. Use `bodeoptions` to set the frequency units and scale.
`setstruc`	`setstruct` is no longer supported. Use the `Structure` property of the `idss` model to configure the model parameters.
`setpname`	`setpname` is no longer supported. Use the `Info.Label` field of the `Structure` property associated with the model parameter.
`idprops`	`idprops` is no longer supported. For information regarding a model, enter `doc model_name`.
`idhelp`	`idhelp` is no longer supported. For information regarding a model or function, enter `doc model_or_function_name`.

Other Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`sys.LinearModel`, for `idnlhw` model, `sys`	Returns an `idpoly` model.	N/A	The `LinearModel` property of `idnlhw` models is no longer returned as a state-space model for multi-output models. Instead, `idnlhw` returns an `idpoly` model.

R2011b

Version: 7.4.3

Bug Fixes

R2011a

Version: 7.4.2

Bug Fixes

R2010b

Version: 7.4.1

No New Features or Changes

R2010a

Version: 7.4

New Features

Compatibility Considerations

New Ability to Use Discrete-Time Linear Models for Nonlinear Black-Box Estimation

You can now use the following discrete-time linear models for initializing a nonlinear black-box estimation.

Discrete-time Linear Model	Use for Initializing...
Single-output polynomial model of ARX structure (`idpoly`)	Single-output nonlinear ARX model estimation
Multi-output polynomial model of ARX structure (`idarx`)	Multi-output nonlinear ARX model estimation
Single-output polynomial model of Output-Error (OE) structure (`idpoly`) or state-space model with no disturbance component (`idss`) object with K = 0	Single-output Hammerstein-Wiener model estimation
State-space model with no disturbance component (`idss` object with K = 0)	Multi-output Hammerstein-Wiener model estimation

During estimation, the software uses the linear model orders and delay as initial values of the nonlinear model orders and delay. For nonlinear ARX models, this initialization always provides a better fit to the estimation data than the linear ARX model.

You can use a linear model as an alternative approach to using model orders and delay for nonlinear estimation of the same system.

You can estimate or construct the linear model and then use this model for constructing (see `idnlarx` and `idnlhw`) or estimating (see `nlarx` or `nlhw`) the nonlinear model. For more information, see Using Linear Model for Nonlinear ARX Estimation, and Using Linear Model for Hammerstein-Wiener Estimation in the *System Identification Toolbox User's Guide*.

New Cell Array Support for B and F Polynomials of Multi-Input Polynomial Models

You can now use cell arrays to specify the *B* and *F* polynomials of multi-input polynomial models. The *B* and *F* polynomials are represented by the `b` and `f` properties of an `idpoly` object These properties are currently double matrices.

For multi-input polynomial models, these polynomials will be represented by cell arrays only in a future version. If your code performs operations on the `b` and `f` properties, make one of the following changes in the code:

- When you construct the model using the `idpoly` command, use cell arrays to specify the *B* and *F* polynomials. Using cell arrays causes the `b` and `f` properties to be represented by cell arrays.
- After you construct or estimate the model, use the new `setPolyFormat` command to:
  - Convert `b` and `f` properties to cell arrays.
  - Make the model backward compatible to continue using double matrices for `b` and `f` properties. This operation ensures that operations on `b` and `f` properties that use matrix syntax continue to work without errors in a future version.

When you use cell arrays, you must also update your code to use cell array syntax on `b` and `f` properties instead of matrix syntax.

Note For single-input polynomial models, the `b` and `f` properties continue to be double row vectors.

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When you Use the Function or Element?	Use This Instead	Compatibility Considerations
Double matrix support for `b` and `f` properties of multi-input `idpoly` models.	Warns	Use cell array to specify the `b` and `f` properties of multi-input polynomial models.	If your code performs operations on the `b` and `f` properties, update the code to be compatible with a future release. See “New Cell Array Support for B and F Polynomials of Multi-Input Polynomial Models” on page 27-2.

R2009b

Version: 7.3.1

No New Features or Changes

R2009a

Version: 7.3

New Features

Enhanced Handling of Offsets and Trends in Signals

This version of the product includes new and expanded functionality for handling offsets and trends in signals. This data processing operation is necessary for estimating more accurate linear models because linear models cannot capture arbitrary differences between the input and output signal levels.

The previous version of the product let you remove mean values or linear trends from steady-state signals using the GUI and the `detrend` function. For transient signals, you had to remove offsets and trends using matrix manipulation.

The GUI functionality for removing means and linear trends from signals is unchanged. However, you can now do the following at the command line:

- Save the values of means or linear trends removed during detrending using a new `detrend` output argument. You can use this saved trend information to detrend other data sets. You can also restore subtracted trends to the output simulated by a linear model that was estimated from detrended data.

  For example, this syntax computes and removes mean values from the data, and saves these values to the output variable `T`: `[data_d,T]=detrend(data)`. `T` is an object with properties that store offset and slope information for input and output signals.
- Remove any offset or linear trend from the data using a new `detrend` input argument. This is useful for removing arbitrary nonzero offsets from transient data or applying previously saved trend information to any data set.

  For example, this syntax removes an offset or trend specified by `T`: `data_d = detrend(data,T)`.
- Add an arbitrary offset or linear trend to data signals. This is useful when you want to simulate the response of a linear model about a nonzero equilibrium input-output level and this model was estimated from detrended data.

  For example, this syntax adds trend information to a simulated model output `y_sim`, which is an `iddata` object: `y = retrend(y_sim,T)`. `T` specifies the offset and slope information for inputs and outputs.

For more information, see Handling Offsets and Trends in Data.

Ability to Get Regressor Values in Nonlinear ARX Models

The `getreg` command can now return the numerical values of regressors in nonlinear ARX models and provides an intermediate output of nonlinear ARX models.

This advanced functionality converts input and output values to regressors, and passes the regressor values to the `evaluate` command to compute the model response. This incremental step lets you gain insight into the propagation of information through the nonlinear ARX model.

For more information, see the `getreg` reference page. To learn more about the nonlinear ARX model structure, see Nonlinear Black-Box Model Identification.

R2008b

Version: 7.2.1

Compatibility Considerations

Functions and Properties Being Removed

Function or Property Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`model.Algorithm.Trace`	Still runs	`model.Algorithm.Display`	Using `model.Algorithm.Trace` results in a warning.

R2008a

Simulating Nonlinear Black-Box Models in Simulink Software

You can now simulate nonlinear ARX and Hammerstein-Wiener models in Simulink using the nonlinear ARX and the Hammerstein-Wiener model blocks in the System Identification Toolbox block library. This is useful in the following situations:

- Representing dynamics of a physical component in a Simulink model using a data-based nonlinear model
- Replacing a complex Simulink subsystem with a simpler data-based nonlinear model

Note Nonlinear ARX Model and Hammerstein-Wiener Model blocks read variables from the MATLAB (base) workspace or model workspace. When the MATLAB workspace and model workspace contain a variable with the same name and this variable is referenced by a Simulink block, the variable in the model workspace takes precedence.

If you have installed Real-Time Workshop® software, you can generate code from models containing nonlinear ARX and the Hammerstein-Wiener model blocks. However, you cannot generate code when:

- Hammerstein-Wiener models use the `customnet` estimator for input or output nonlinearity.
- Nonlinear ARX models use custom regressors or use the `customnet` or `neuralnet` nonlinearity estimator.

You can access the new System Identification Toolbox blocks from the Simulink Library Browser. For more information about these blocks, see the IDNLARX Model (nonlinear ARX model) and the IDNLHW Model (Hammerstein-Wiener model) block reference pages.

Linearizing Nonlinear Black-Box Models at User-Specified Operating Points

You can now use the `linearize` command to linearize nonlinear black-box models, including nonlinear ARX and Hammerstein-Wiener models, at specified operating points. Linearization produces a first-order Taylor series approximation of the system about an operating point. An *operating point* is defined by the set of constant input and state values for the model.

If you do not know the operating point, you can use the `findop` command to compute it from specifications, such as steady-state requirements or values of these quantities at a given time instant from the simulation of the model.

For nonlinear ARX models, if all of the steady-state input and output values are known, you can map these values to the model state values using the `data2state` command.

`linearize` replaces `lintan` and removes the restriction for linearizing models containing custom regressors or specific nonlinearity estimators, such as `neuralnet` and `treepartition`.

If you have installed Simulink Control Design software, you can linearize nonlinear ARX and Hammerstein-Wiener models in Simulink after importing them into Simulink.

For more information, see:

- Linear Approximation of Nonlinear Black-Box Models about computing operating points and linearizing models

- Simulating Identified Model Output in Simulink about importing nonlinear black-box models into Simulink

Estimating Multiple-Output Models Using Weighted Sum of Least Squares Minimization Criterion

You can now specify a custom weighted trace criterion for minimization when estimating linear and nonlinear black-box models for multiple-output systems. This feature is useful for controlling the relative importance of output channels during the estimation process.

The `Algorithm` property of linear and nonlinear models now provides the `Criterion` field for choosing the minimization criterion. This new field can have the following values:

- `det` — (Default) Specify this option to minimize the determinant of the prediction error covariance. This choice leads to maximum likelihood estimates of model parameters. It implicitly uses the inverse of estimated noise variance as the weighting function. This option was already available in previous releases.
- `trace` — Specify this option to define your own weighing function that controls the relative weights of output signals during the estimation. This criterion minimizes the weighted sum of least square prediction errors. You can specify the relative weighting of prediction errors for each output using the new `Weighting` field of the `Algorithm` property. By default, `Weighting` is an identity matrix, which means that all outputs are weighed equally. Set `Weighting` to a positive semidefinite symmetric matrix.

For more information about Algorithm fields for nonlinear estimation, see the `idnlarx` and `idnlhw` reference pages.

Note If you are estimating a single-output model, `det` and `trace` values of the `Criterion` field produce the same estimation results.

Improved Handling of Initial States for Linear and Nonlinear Models

The following are new options to handle initial states for nonlinear models:

- For nonlinear ARX models (`idnlarx`), you can now specify a numerical vector for initial states when using `sim` or `predict` by setting the `Init` argument. For example:

  ```
  predict(model,data,'init',[1;2;3;4])
  ```

  where the last argument is the state vector.

  For more information, see the `sim` and `predict` reference pages.
- For Hammerstein-Wiener models (`idnlhw`), you can now choose to estimate the initial states when using `predict` or `nlhw` by setting `INIT='e'`.

  For more information, see the `predict` and `nlhw` reference pages.

If you want to specify your own initial states, see the corresponding model reference pages for a definition of the states for each model type.

If you do not know the states, you can use the `findop` or the `findstates` command to compute the states. For more information about using these commands, see the `findop(idnlarx)`, `findop(idnlhw)`, `findstates(idnlarx)`, and `findstates(idnlhw)` reference pages.

To help you interpret the states of a nonlinear ARX model, you can use the `getDelayInfo` command. For more information, see the `getDelayInfo` reference page.

The `findstates` command is available for all linear and nonlinear models. Also see the `findstates(idnlgrey)` reference page.

Improved Algorithm Options for Linear Models

The following improvements are available for the `Algorithm` property of linear models to align linear and nonlinear models (where appropriate) and improve robustness for default settings:

- The `SearchDirection` field (`model.Algorithm.SearchDirection`) has been renamed to `SeachMethod` (`model.Algorithm.SearchMethod`) to be consistent with the nonlinear models, where the corresponding field is `SeachMethod`.
- The new `lsqnonlin` option for specifying `SearchMethod` is available. `model.Algorithm.SearchMethod='lsqnonlin'` uses the `lsqnonlin` optimizer from the Optimization Toolbox software. You must have Optimization Toolbox software installed to use this option.
- The improved `gn` algorithm (subspace Gauss-Newton method) is available for specifying `SearchDirection`. The updated `gn` algorithm better handles the scale of the parameter Jacobians and is also consistent with the algorithm used for nonlinear model estimation.
- The default values for the `LimitError` field of the `Algorithm` property (`modelname.Algorithm.LimitError`) is changed to 0, which is consistent with the corresponding option for estimating nonlinear models. In previous releases, `LimitError` default value was 1.6, which robustified the estimation process against data outliers by associating a linear penalty for large errors, rather than a quadratic penalty. Now, there is no robustification by default (`LimitError=0`). You can estimate the model with the default setting and plot the prediction errors using `pe(data,model)`. If the resulting plot shows occasional large values, repeat the estimation with `model.Algorithm.LimitError` set to a value between 1 and 2.
- The `model.Algorithm.Advanced` property has a new tolerance field `GnPinvConst` corresponding to the `gn` `SearchMethod`. `GnPinvConst` specifies that singular values of the Jacobian that are smaller than `GnPinvConst*max(size(J))*norm(J)*eps` are discarded when computing the search direction. You can assign a positive real value for this field. Default value is `1e4`.
- The default value of `model.Algorithm.Advanced.Zstability` property has been changed from `1.01` to `1+sqrt(eps)`. The new default reduces the possibility of a situation where the estimation algorithm does not converge (predictor becomes unstable) while still allowing enough flexibility to capture lightly damped modes.

New Block Reference Pages

New documentation for System Identification Toolbox blocks is provided. For more information, see Block Reference in the System Identification Toolbox reference documentation.

Functions and Properties Being Removed

Function or Property Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`lintan`	Still runs	`linearize(idnlhw)` `linearize(idnlarx)`	See "Linearizing Nonlinear Black-Box Models at User-Specified Operating Points" on page 31-2.
`model.Algorithm.SearchDirection`	Still runs	`model.Algorithm.SearchMethod`	See "Improved Algorithm Options for Linear Models" on page 31-4.
`gns` option of `model.Algorithm.SearchDirection`	Still runs	`gn`	See "Improved Algorithm Options for Linear Models" on page 31-4.
`GnsPinvTol` of `model.Algorithm.Advanced`	Still runs	`GnPinvConst`	See "Improved Algorithm Options for Linear Models" on page 31-4.

R2007b

Version: 7.1

New Features

New Polynomial Nonlinearity Estimator for Hammerstein-Wiener Models

You can now estimate nonlinearities for Hammerstein-Wiener models using a single-variable polynomial at either the input or the output. This nonlinearity estimator is available at the command line.

For more information, see the `poly1d` reference pages. For more information about estimating Hammerstein-Wiener models, see Identifying Hammerstein-Wiener Models.

R2007a

Version: 7.0

New Features

New Nonlinear Black-Box Modeling Options

You can now estimate nonlinear discrete-time black-box models for both single-output and multiple-output time-domain data. The System Identification Toolbox product supports the following types of nonlinear black-box models:

- Hammerstein-Wiener
- Nonlinear ARX

To learn how to estimate nonlinear black-box models using the System Identification Tool GUI or commands in the MATLAB Command Window, see the System Identification Toolbox documentation.

Note You can estimate Hammerstein-Wiener black-box models from input-output data only. These models do not support time-series data, where there is no input.

New demos are available to help you explore nonlinear black-box functions. For more information, see the collection of demos in the `Tutorials on Nonlinear ARX and Hammerstein-Wiener Model Identification` category.

New Nonlinear Grey-Box Modeling Option

You can now estimate nonlinear discrete-time and continuous-time models for arbitrary nonlinear ordinary differential equations using single-output and multiple-output time-domain data, or time-series data (no measured inputs). Models that you can specify as a set of nonlinear ordinary differential equations (ODEs) are called *grey-box models*.

To learn how to estimate nonlinear grey-box models using the commands in the MATLAB Command Window, see System Identification Toolbox documentation.

Specify the ODE in a function or a MEX-file. The template file for writing the MEX-file, `IDNLGREY_MODEL_TEMPLATE.c`, is located in `matlab/toolbox/ident/nlident`.

To estimate the equation parameters, first construct an `idnlgrey` object to specify the ODE file and the parameters you want to estimate. Use `pem` to estimate the ODE parameters. For more information, see the `idnlgrey` and `pem` reference pages.

New demos are available to help you explore nonlinear grey-box functions. For more information, see the collection of demos in the `Tutorials on Nonlinear Grey-Box Model Identification` category.

Optimization Toolbox Search Method for Nonlinear Estimation Is Supported

If you have Optimization Toolbox software installed, you can specify the `lsqnonlin` search method for estimating black-box and grey-box nonlinear models in the MATLAB Command Window.

```
model.algorithm.searchmethod='lsqnonlin'
```

For more information, see the `idnlarx`, `idnlhw`, and `idnlgrey` reference pages.

New Getting Started Guide

The System Identification Toolbox product now provides a new Getting Started Guide. This guide introduces fundamental identification concepts and provides the following tutorials to help you get started quickly:

- Tutorial – Identifying Linear Models Using the GUI — Tutorial for using the System Identification Tool graphical user interface (GUI) to estimate linear black-box models for single-input and single-output (SISO) data.
- Tutorial – Identifying Low-Order Transfer Functions (Process Models) Using the GUI — Tutorial for using the System Identification Tool graphical user interface (GUI) to estimate low-order transfer functions to fit single-input and single-output (SISO) data.
- Tutorial – Identifying Linear Models Using the Command Line — Tutorial for estimating models using System Identification Toolbox objects and methods for multiple-input and single-output (MISO) data.

Revised and Expanded User's Guide

The System Identification Toolbox documentation has been revised and expanded.

R2006b

Version: 6.2

New Features

MATLAB Compiler Support

The System Identification Toolbox product now supports the MATLAB Compiler product.

You can use MATLAB Compiler to take MATLAB files as input and generate redistributable, standalone applications that include System Identification Toolbox functionality, including the following:

- Creating data and model objects
- Preprocessing and manipulating data
- Simulating models
- Transforming models, including conversions between continuous and discrete time and model reduction
- Plotting transient and frequency response

To use these features, write a function that uses System Identification Toolbox commands. Use the MATLAB Compiler software to create a standalone application from the MATLAB Compiler file. For more information, see the MATLAB Compiler documentation.

Standalone applications that include System Identification Toolbox functionality have the following limitations:

- No access to the System Identification library in the Simulink software (`slident`)
- No support for model estimation

R2006a

Version: 6.1.3

New Features

Compatibility Considerations

balred Introduced for Model Reduction

Use `balred` to perform model reduction instead of `idmodred`.

Search Direction for Minimizing Criteria Can Be Computed by Adaptive Gauss-Newton Method

An adaptive Gauss-Newton method is now available for computing the direction of the line search for cost-function minimization. Use this method when you observe convergence problems in the estimation results, or as an alternative to the Levenberg-Marquard (`lm`) method.

The `gna` search method was suggested by Adrian Wills, Brett Ninness, and Stuart Gibson in their paper "On Gradient-Based Search for Multivariable System Estimates", presented at the IFAC World Congress in Prague in 2005. `gna` is an adaptive version of `gns` and uses a cutoff value for the singular values of the criterion Hessian, which is adjusted adaptively depending on the success of the line search.

Specify the `gna` method by setting the `SearchDirection` property to `'gna'`. For example:

```
m = pem(data,model_structure,'se','gna')
```

The default initial value of `gamma` in the `gna` search is `10^-4`. You can set a different value using the `InitGnaTol` property.

Maximum Number of Bisections Used by Line Search Is Increased

The default value for the `MaxBisections` property, which is the maximum number of bisections along the search direction used by line search, is increased from `10` to `25`. This increases the number of attempts to find a lower criterion value along the search vector.

Functions and Properties Being Removed

Function or Property Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`idmodred`	Still runs	`balred`	See “balred Introduced for Model Reduction” on page 35-2.

R14SP3

Version: 6.1.2

No New Features or Changes

R14SP2

Version: 6.1.1

No New Features or Changes