Release Notes for System Identification Toolbox™

Contents

R2012b

Regularized estimates of impulse response, specification of transport delays and estimation options using `impulseest` **2**

`translatecov` command for translating model covariance across transformations **3**

`ssform` command for quick configuration of state-space model structure **4**

`Feedthrough` specification for discrete-time transfer function model estimation **5**

R2012a

Summary **8**

New Features in This Version **9**

Changes Introduced in This Version **27**

R2011b

R2011a

R2010b

No New Features or Changes

R2010a

New Ability to Use Discrete-Time Linear Models for Nonlinear Black-Box Estimation **68**
New Cell Array Support for B and F Polynomials of Multi-Input Polynomial Models **69**
Functions and Function Elements Being Removed **70**

R2009b

No New Features or Changes

R2009a

Enhanced Handling of Offsets and Trends in Signals **74**
Ability to Get Regressor Values in Nonlinear ARX Models **76**

R2008b

Functions and Properties Being Removed **78**

R2008a

Simulating Nonlinear Black-Box Models in Simulink Software **80**
Linearizing Nonlinear Black-Box Models at User-Specified Operating Points **81**
Estimating Multiple-Output Models Using Weighted Sum of Least Squares Minimization Criterion **82**
Improved Handling of Initial States for Linear and Nonlinear Models **83**
Improved Algorithm Options for Linear Models **84**

New Block Reference Pages **86**
Functions and Properties Being Removed **87**

R2007b

New Polynomial Nonlinearity Estimator for Hammerstein-Wiener Models **90**

R2007a

New Nonlinear Black-Box Modeling Options **92**
New Nonlinear Grey-Box Modeling Option **93**
Optimization Toolbox Search Method for Nonlinear Estimation Is Supported **94**
New Getting Started Guide **95**
Revised and Expanded User's Guide **96**

R2006b

MATLAB Compiler Support **98**

R2006a

balred Introduced for Model Reduction **102**
Search Direction for Minimizing Criteria Can Be Computed by Adaptive Gauss-Newton Method **103**
Maximum Number of Bisections Used by Line Search Is Increased **104**
Functions and Properties Being Removed **105**

R14SP3

No New Features or Changes

R14SP2

No New Features or Changes

R2012b

Version: 8.1
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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 mdbut 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

Version: 8.0
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

New features this release include:

- "Continuous-Time Transfer Function Identification for Time- and Frequency-Domain Data " on page 9
- "Time-Series Modeling and Forecasting, Including Generating ARIMA Models" on page 10
- "Estimation of Multi-Output Polynomial and Process Models" on page 10
- "Interactive Response Plots with Better Look and Feel" on page 11
- "Models Created with System Identification Toolbox Can Be Used Directly with Control System Toolbox Functions" on page 11
- "Improved Reliability of Numerical Computations" on page 12
- "Estimating Functions and Estimation Option Sets" on page 12
- "Model Analysis and Validation Option Sets" on page 14
- "Identified Linear Models" on page 15
- "System Identification Tool GUI" on page 25

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 29.

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 prefiction error method, PEM, but have calling syntaxes that are adjusted to the particular model type.

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

Algorithm Property Field	Options Set Field
`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`.
`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`	`SearchOption.AdvancedSearchMethod` and `SearchOptions` are offered only for iterative estimation methods. For example, `arxOptions` does not provide these fields, but `tfestOptions` does.

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 27.

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 12.

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

Property	Model Types Affected	What Happens in R2012a	Use This Instead
`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 27.
`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

Property	Model Types Affected	What Happens in R2012a	Use This Instead
`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 a 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)` would return a transfer function model with two inputs. The first input would correspond to the measured component, `B/A`. The second input would correspond to the noise component, `C/A`. Although `size(sys,2)` is `1`, `size(sys_tf,2)` would be `2`. Thus, `sys` had one input, while `sys_tf` would have two inputs.

In this release, executing `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 using 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` will have 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` 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:

- Import a data set into the System Identification Tool GUI.
- 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:

- Import a data set into the System Identification Tool GUI.
- 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:

- Import a data set into the System Identification Tool GUI.
- 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:

- Import a data set into the System Identification Tool GUI.
- 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

Compatibility Considerations: Yes

Changes introduced in this version:

- "Reorganization of Estimation Reports" on page 27
- "Polynomial Models" on page 29
- "State-Space Models" on page 34
- "Process Models" on page 39
- "Linear Grey-Box Models" on page 44
- "Identified Frequency Response Data Models" on page 48
- "Identification Data Objects" on page 49
- "Analysis Commands" on page 50
- "Other Functionality Being Removed or Changed" on page 60

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`.
`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.IterationsTermination` information is not provided for models estimated using a non-iterative estimation function, such as `arx` or `n4sid`.

EstimationInfo Field	Report Field
`InitialState`	Either: • `InitiateState` (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 10.

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. 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.OutputWeightIf 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.OutputWeightIf 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);`
		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]);`
		N/A

Functionality	What Happens in R2012a	Use This Instead
	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.	

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');`

Model Property	What Happens in R2012a	Use This Instead
		`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.

`nk`	Still available but **may cause a backward incompatibility**.However, 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);`
		For estimation, use the `InputDelay` and `Feedthrough` estimation properties instead. When creating an `idss` model, specify the `InputDelay` and `Structure.f` properties. `nk`, `InputDelay`, and `Feedthrough` are related:
		
		• `nk(j) = 0` means that the model has no delay for the jth input. Therefore,

Model Property	What Happens in R2012a	Use This Instead
	In this release, `sys.nk` is 3, whereas `sys.nk` was 5 in earlier releases.	`InputDelay` is `0`, and `Structure.f.Free(:,j)` is `true`. • `nk(j) = 1` means that the model has zero delay for the jth input. Therefore, `InputDelay` is `0`, and there is no feedthrough. `Structure.f.Free(:,j)` is `false`, and `Structure.f.Value(:,j)` is zero. • `nk(j) = N, N>1` means that the model has nonzero delay for the jth input. Therefore, `InputDelay` is `N-1`, and there is no feedthrough. `Structure.f.Free(:,j)` is `false`, and `Structure.f.Value(:,j)` is `0`. `nk > 1` can only be used for a discrete-time model.
`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:

Model Property	What Happens in R2012a	Use This Instead
		- 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 11

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.

Compatibility Consideration: See the `DisturbanceModel` entry in .

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`. You can set the `InputOffset` option to `estimate` or to a `param.Continuous` object for advanced control.

`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);`

`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

Model Property	What Happens in R2012a	Use This Instead

		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.
`Kp`, `Tp1`, `Tp2`, `Tp3`, `Tw`, `Zeta`, `Td`, and `Tz`	**May cause a backward incompatibility.** These properties are now saved as double matrices. In previous releases, they were stored as structures.	Use the `Structure` property to specify parameter constraints.`Structure` replaces the specification of process model parameter

Model Property	What Happens in R2012a	Use This Instead

	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'} ``
		bounds. See Call Replacements on page 43.

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 estimating by using the function, `greyestOptions`.

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

Complex Parameters Supported

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.
`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.

Model Property	What Happens in R2012a	Use This Instead
`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 (May cause a 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

Function	What Has Changed in R2012a
	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 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)`

Function	What Has Changed in R2012a
	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.
`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`.

Function	What Has Changed in R2012a
`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. For more information, enter `doc function_name` at the MATLAB command prompt.

Function	What Has Changed in R2012a
`nyquist`	• You can customize a nyquist plot by creating a 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 (see `nyquistoptions`,`getoptions` and `setoptions` instead. Alternatively, you may interactively change these options by

Function	What Has Changed in R2012a
	right-clicking on the plot and choosing the appropriate options.
`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,

Function	What Has Changed in R2012a
	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.
`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 then returns empty matrices for uncertainty outputs. For more information, see `idssdata`. • When `ssdata` is called with six or more outputs, the standard deviation information for the model parameters is returned as well. • **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.

Function	What Has Changed in R2012a
`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`.
`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`.

Function	What Has Changed in R2012a
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
New Features: No
Bug Fixes: Yes

R2011a

Version: 7.4.2
New Features: No
Bug Fixes: Yes

R2010b

Version: 7.4.1
New Features: No
Bug Fixes: No

No New Features or Changes

R2010a

Version: 7.4
New Features: Yes
Bug Fixes: No

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

Compatibility Considerations: Yes

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 69.

R2009b

Version: 7.3.1
New Features: No
Bug Fixes: No

No New Features or Changes

R2009a

Version: 7.3
New Features: Yes
Bug Fixes: No

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
New Features: No
Bug Fixes: No

Functions and Properties Being Removed
Compatibility Considerations: Yes

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

Version: 7.2
New Features: Yes
Bug Fixes: No

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

Compatibility Considerations: Yes

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 81.
`model.Algorithm.SearchDirection`	Still runs	`model.Algorithm.SearchMethod`	See “Improved Algorithm Options for Linear Models” on page 84.
`gns` option of `model.Algorithm.SearchDirection`	Still runs	`gn`	See “Improved Algorithm Options for Linear Models” on page 84.
`GnsPinvTol` of `model.Algorithm.Advanced`	Still runs	`GnPinvConst`	See “Improved Algorithm Options for Linear Models” on page 84.

R2007b

Version: 7.1
New Features: Yes
Bug Fixes: No

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 in the System Identification Toolbox documentation.

R2007a

Version: 7.0
New Features: Yes
Bug Fixes: No

- “New Nonlinear Black-Box Modeling Options” on page 92
- “New Nonlinear Grey-Box Modeling Option” on page 93
- “New Getting Started Guide” on page 95
- “Revised and Expanded User’s Guide” on page 96

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: Yes
Bug Fixes: No

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: Yes
Bug Fixes: No

- "balred Introduced for Model Reduction" on page 102
- "Search Direction for Minimizing Criteria Can Be Computed by Adaptive Gauss-Newton Method" on page 103
- "Maximum Number of Bisections Used by Line Search Is Increased" on page 104

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

Compatibility Considerations: Yes

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 102.

R14SP3

Version: 6.1.2
New Features: No
Bug Fixes: No

No New Features or Changes

R14SP2

Version: 6.1.1
New Features: No
Bug Fixes: No

No New Features or Changes