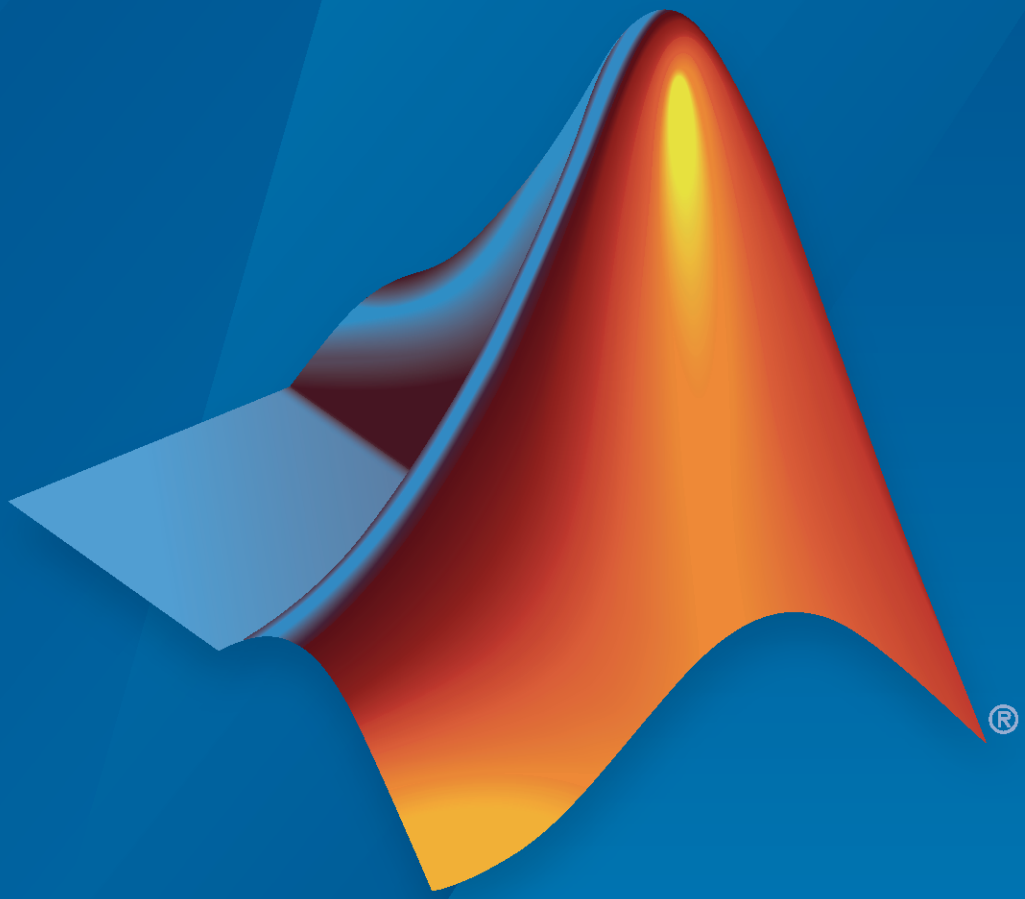
Predictive Maintenance Toolbox™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

Predictive Maintenance Toolbox™ Release Notes

Contents

R2022b

R2022a

R2021b

R2021a

R2020b

Bug Fixes

R2020a

R2019b

R2019a

R2018b

R2018a

R2022b

Version: 2.6

New Features

Bug Fixes

Diagnostic Feature Designer: Automatically generate a diverse feature set using Auto Features

You can now generate a diverse set of features using the new **Auto Features** capability. Previously, you needed to perform each feature extraction operation manually. When you use **Auto Features**, the app computes and ranks a set of features from the signal or spectrum variables that you select in the **Variables** pane.

For more information, see “Generate Features Automatically in Diagnostic Feature Designer”.

Diagnostic Feature Designer: Create custom features to use in the app

You can now create custom features to use with **Diagnostic Feature Designer**. Previously, you could extract only features that were built into the app. If you wanted to customize features, you had to create them outside of the app and then import them into the app to evaluate and rank them with other features.

For more information, see “Create Custom Features in Diagnostic Feature Designer”.

Diagnostic Feature Designer: Preview results when applying time series transformations to signal data

You can now preview your intermediate results when you use the **Time Series Processing** tab to stack transformation operations that convert ordinary signals into stationary time series. Previously, you had to create a new variable by clicking **Apply** to view how well a set of transformations worked.

For more information about time series processing, see “Time Series Processing and Time Series Features”.

Diagnostic Feature Designer: Generate AR model-based features

You can now build autoregressive (AR) models to extract features from your signals. AR models are useful for rotating machinery that contains elements such as motors, gears, and bearings. The harmonics of these elements produce sharp peaks in the machine's power spectra. Faults in the machine elements are likely to disturb the normal harmonics and cause detectable changes in AR model characteristics, such as model coefficients.

To create AR model-based features, select a signal. Then, select **Time-Domain Features > Model-Based Features**. In the **Model-Based Features** tab, specify your model order. Then select the features that you want to generate. The tab provides features for model parameters (such as coefficients and natural frequencies), model fit, and estimation residuals.

Lithium-Ion Battery Fault Detection: Identify the worst cell in a battery pack using meanDifferenceModel

You can now find the worst cell in a serially connected battery pack using the `meanDifferenceModel` function. This function evaluates the consistency of the open-circuit voltages (OCVs) for the battery cells by estimating the deviation of the cell OCV from the mean OCV of the battery pack. Large deviations indicate a possible fault condition, such as an internal short

circuit. The function returns the index of the cell with the largest deviation, or the worst cell. The function can also return the estimated OCV deviations, as well as the internal resistance deviations from the resistance mean. If you omit output arguments, the function plots the OCV deviation for all cells.

For more information, see `meanDifferenceModel`.

New Example: Broken Rotor Fault Detection in AC Induction Motors Using Vibration and Electrical Signals

A new example illustrates broken rotor fault detection in three-phase AC induction motors using features from the **Diagnostic Feature Designer** and **Classification Learner** apps.

- “Broken Rotor Fault Detection in AC Induction Motors Using Vibration and Electrical Signals”

R2022a

Version: 2.5

New Features

Bug Fixes

Diagnostic Feature Designer: View session variables by type and obtain processing sequence and history

You can now view your variables by their type in the **Diagnostic Feature Designer Variables** and **Details** panes. Previously, the **Data Browser** pane listed variables in the order that you created them. The new **Variables** pane organizes your variables into four groups—Signals, Spectra, Features, and Condition Variables—that make it easier to find the variable you want to analyze next. Each group contains subgroups to separate full and framed signal data.

When you select a variable, you can view its processing history in the app, as well as additional information, in the new **Details** pane. Previously, to obtain processing history, you needed to activate a tooltip by pointing to the variable in the **Data Browser** pane. The **Details** pane displays the variable or variables from which your selected variable was computed. You can view the full processing sequence by clicking **History View**. You can view the parameters of the most recent processing step for the selected variable by clicking **Parameters**. This information is available for variables that have been computed during the current session. The **Details** pane also provides additional information about the selected variable, including its independent variable, its frame policy, and the data set that contains it.

For more information about using the **Variables** and **Details** panes, see Process Data and Explore Features in Diagnostic Feature Designer.

Diagnostic Feature Designer: Extract time series features from signal data

You can now transform your signals into stationary time series, and from the time series, extract specialized features in **Diagnostic Feature Designer**. Time series features provide unique insights into the data.

A stationary time series has no trends or periodic fluctuations, and constant variance and autocorrelation over time. Second-order stationary signals, which are usually sufficient for engineering applications, have zero mean and constant variance. The app provides a set of processing transformations that you can use to eliminate nonstationary components and reduce time-dependent variance. Once you have created the time series, you can extract features that include metrics on distribution, autocorrelation, and partial autocorrelation.

To transform signals into stationary time series in the app, select the source signal, and then, in the **Feature Designer** tab, in the **Data Processing** section, select **Residue Generation** > **Time-Series Processing**.

To extract time series features from stationary time series, select the time series, and then, in the **Feature Designer** tab, in the **Feature Generation** section, select **Time-Domain Features** > **Time-Series Features**.

For more information, see Time Series Processing and Time Series Features.

Remaining Useful Life (RUL) Prediction: Generate C/C++ code using MATLAB Coder for RUL prediction based on a similarity model

You can now use MATLAB® Coder™ functionality to generate C/C++ code using `hashSimilarityModel`, `pairwiseSimilarityModel`, or `residualSimilarityModel` RUL

models. Code generation is supported for the `predictRUL` function with these models. Previously, you could generate C/C++ code only for degradation-based and survival-based RUL models. Now, code generation is supported for all Predictive Maintenance Toolbox™ RUL model types.

For an example of code generation for an RUL model, see Generate Code for Predicting Remaining Useful Life.

Rotating Machinery Metrics: Generate C/C++ code using MATLAB Coder to extract time-synchronous averaged signals

You can now use MATLAB Coder functionality to generate C/C++ code for workflows that use the time-synchronous averaging functions — `tsadifference`, `tsaregular`, and `tsaresidual`.

Syntaxes that generate plots are not supported for code generation.

Nonlinear Feature Extraction: Generate C/C++ code using MATLAB Coder to identify signal-based nonlinear features

You can now use MATLAB Coder functionality to generate C/C++ code to identify signal-based nonlinear features in time-series data using `phaseSpaceReconstruction`, `approximateEntropy`, `lyapunovExponent` or `correlationDimension` functions.

Syntaxes that generate plots are not supported for code generation.

Time-Frequency Feature Extraction: Generate C/C++ code using MATLAB Coder to extract spectral, temporal, and joint moments of time-frequency distributions from signals

You can now use MATLAB Coder functionality to generate C/C++ code for workflows that use the time-frequency moment functions—`tfsmoment`, `tftmoment`, and `tfmoment`.

For `tfsmoment` only, to generate C/C++ code, the input argument `order` must be a scalar and not a vector.

New Example: ThingSpeak Dashboard for Live RUL Estimation of a Servo Motor Gear Train

A new example illustrates setting up a ThingSpeak™ dashboard for real-time Remaining Useful Life (RUL) estimation and visualization of a servo motor gear train.

- ThingSpeak Dashboard for Live RUL Estimation of a Servo Motor Gear Train

R2021b

Version: 2.4

New Features

Bug Fixes

Diagnostic Feature Designer: Generate spectral features for characteristic fault frequency bands in rotating machinery

You can now generate spectral features, such as peak amplitude or band power, for specific frequency bands when you work with data from rotating machinery. Previously, you could extract spectral features only across a single frequency range. These specialized features allow you to focus your feature extraction within frequency bands that bound the characteristic fault harmonics for your system. You specify these fault bands by entering the physical characteristics of your machinery, such as the number of gear teeth or the bearing ball diameter. You can also specify custom bands that are physically dependent only on the primary rotational speed.

For more information about computing features using fault bands in the app, see Spectral Features Based on Fault Bands.

Diagnostic Feature Designer: Rank features extracted from unlabeled data

You can now perform ranking on features that you extract from unlabeled data. Previously, you could perform ranking only when your ensemble data contained labels such as `healthy` and `faulty`, or when you were performing prognostic ranking to support remaining useful life (RUL) prediction.

Ranking methods that use condition variables with data labels are now grouped under **Supervised Ranking**. These methods, formerly grouped under **Classification Ranking**, do not change. Ranking methods that do not use data labels are grouped under **Unsupervised Ranking**. The app offers two ranking options in this group.

1. **Laplacian Score** — Scores reflect how well features cluster with other features to form distinct groupings. For more information, see `fsulaplacian`.
2. **Variance** — Ranks features based on their variance. Features with low variances tend to add less useful information to a model.

Unsupervised ranking is available in **Diagnostic Feature Designer**, but not in **Classification Learner**. If you plan to export your features to **Classification Learner** to train a model, you must use ensemble data that includes labels.

For more information on feature ranking in the app, see **Feature Ranking Tab** in **Diagnostic Feature Designer**.

Diagnostic Feature Designer: Use a streamlined workflow for plotting, data processing, and feature extraction

The app now includes updates that can streamline your workflow. These updates include the following changes.

- Set general plot preferences — You can now directly specify plot options that apply to all the plots that you generate. For example, for all signals and spectra, you can specify condition-variable grouping and the number of curves to plot. Previously, you needed to generate a plot before setting preferences and could not set customized defaults for all your plots. You can make these specifications before you generate your first plot, or anytime during your session. You can override these preferences to customize specific plots, but still retain the preferences as a default for subsequent plots. For more information about plot options, see Import and Visualize Ensemble Data in Diagnostic Feature Designer.

- Preselect signals or spectra for both data processing and feature generation — Previously, for feature generation, you needed to open a feature dialog box first and then manually select the source signal or spectrum. For more information about the data processing and feature generation workflow, see Process Data and Explore Features in Diagnostic Feature Designer.
- Specify frame-based processing directly — Access frame parameters using **Frame Policy** in the Computation section of the **Feature Designer** tab. Previously, you needed to open the **Computation Options** dialog box to access frame specification parameters. To select an independent variable or to use parallel processing, use the **Options** list that is also in the Computation section.
- Specify results handling for ensemble datastores during import — For simulation and file ensemble datastores, you can now specify during import whether to write processing results to the external data files or app memory. Previously, you made this selection in **Computation Options** after you imported an ensemble datastore. For more information, see Import Ensemble Datastore in Import Data into Diagnostic Feature Designer.

Remaining Useful Life (RUL) Prediction: Generate C/C++ code using MATLAB Coder for RUL prediction that is based on a survival model

You can now use MATLAB Coder functionality to generate C/C++ code using a `reliabilitySurvivalModel` or `covariateSurvivalModel` object. Previously, you could generate C/C++ code only for RUL degradation models. For the survival models, code generation is supported for the `predictRUL` function.

To generate code for a prediction algorithm using a survival model, follow the same general steps that you take for a degradation model. Use the `saveRULModelForCoder` command to save the model for code generation. Then, use the `loadRULModelForCoder` to load the model in your entry-point function. For an example of generating code for an RUL model, see Generate Code for Predicting Remaining Useful Life.

Rotating Machinery Metrics: Generate C/C++ code using MATLAB Coder for gear condition metrics and fault band metrics

You can now use MATLAB Coder functionality to generate C/C++ code for the `faultBandMetrics` and `gearConditionMetrics` commands. The following instances are not supported for C/C++ code generation:

- Data stored in `fileEnsembleDatastore` and `workspaceEnsemble` objects, as well as data in the form of a tall array.
- Position arguments for the `gearConditionMetrics` command. For instance, the syntax `M = gearConditionMetrics(T,sigvar,difvar,regvar,resvar)` is not supported.
- Sorting a table by a table column with values wrapped in cells for the `gearConditionMetrics` command. For instance, the values of the column `Order` must be scalar for the syntax `M = gearConditionMetrics(T,'SortBy','Order')`.

For more information, see the `faultBandMetrics` and `gearConditionMetrics` reference pages.

New Example: Live RUL Estimation of a Servo Gear Train using ThingSpeak

A new example illustrates real-time Remaining Useful Life (RUL) estimation of a servo motor gear train using ThingSpeak.

- Live RUL Estimation of a Servo Gear Train using ThingSpeak

New Example: Fault detection and diagnosis using artificial intelligence

Two new examples illustrate the application of artificial intelligence techniques to fault detection and diagnosis.

- Rolling Element Bearing Fault Diagnosis Using Deep Learning — Diagnose faults in a rolling element bearing using a pretrained network.
- Anomaly Detection in Industrial Machinery Using Three-Axis Vibration Data — Detect anomalies in industrial-machine vibration data using machine learning and deep learning.

R2021a

Version: 2.3

New Features

Bug Fixes

Diagnostic Feature Designer: Import data using an updated interface with more flexible options

You can now import data using a single dialog box that provides increased flexibility. Use **New Session** to initiate the data import process. **New Session** replaces **Import Data**, and initiates a process that replaces the previous multiple dialog boxes with a single dialog box that allows you to perform all your import specifications in one place. Within this dialog box, you can now do the following:

- Select a single data source from your workspace and view all the workspace variables that have the same internal variables and member format. Use this option when you are importing multiple data sets. The app simplifies your task by displaying the names of all the compatible data sources that can be combined with your initial selection.
- Import power and order spectra from a `table`. Previously, you needed to import spectral data in an `idfrd` object.
- Generate, rather than import, a virtual independent variable (IV) such as time or sample index. This option is the default when your import data does not include time or another IV.

For more information, see Import Data into Diagnostic Feature Designer.

Diagnostic Feature Designer: Preselect signals and spectra to process

You can now preselect the variable in the data browser that you want to use for data processing and view compatible processing options. Previously, you could use your data browser selection only for plotting. For more information, see Process Data and Explore Features in Diagnostic Feature Designer and the Data Processing parameter description in **Diagnostic Feature Designer**.

Diagnostic Feature Designer: Use tooltips to obtain the history of processing and data sources for derived variables

When you perform a sequence of operations in the app, you produce a set of derived variables, each of which is a unique result of the processing history and source variables. You can now obtain that history from a tooltip that appears when you point to a variable name in the data browser. Previously, the variable names encapsulated the processing history, and a multistep processing sequence resulted in a long variable name. For more information, see Process Data and Explore Features in Diagnostic Feature Designer.

The R2021a history tracking process is fully compatible with saved sessions that use the previous variable-naming approach. If you open a session that you saved prior to R2021a, the app treats each saved variable as an original data source and preserves the original concatenated name as a single source name. When you derive additional variables, the app preserves the original names in their entirety, but appends processing steps to the new variable name and populates the tooltip according to the R2021a approach.

Ensemble Datastores: Use the subset function to extract ensemble members that you specify from an existing ensemble into a new ensemble

You can now create a new ensemble datastore from a subset of an existing ensemble datastore by extracting the ensemble members that correspond to the indices you specify.

Use `subset` when you want to perform ensemble operations on a specific ensemble member or group of ensemble members, and when using a sequence of `read` commands with the source ensemble does not provide the ensemble members that you want to process.

For more information, see `subset`.

Remaining Useful Life (RUL) Prediction: Generate C/C++ code using MATLAB Coder for the prediction, update, and restart of an RUL prediction that is based on a degradation model

You can now use MATLAB Coder functionality to generate C/C++ code using a `linearDegradationModel` or an `exponentialDegradationModel`. Code generation is supported for the `predictRUL`, `update`, and `restart` functions.

To generate code for a prediction algorithm using a degradation model, use the new `saveRULModelForCoder` command to save the model for code generation. Then, use the `loadRULModelForCoder` to load the model in your entry-point function. If you update the model at run time, you can use the new `readState` and `restoreState` commands to preserve the updated model state. For examples, see:

- Generate Code for Predicting Remaining Useful Life
- Generate Code that Preserves RUL Model State for System Restart

Live Editor Tasks: Interactively define fault frequency bands and extract spectral metrics

Use the new **Extract Spectral Features** Live Editor task to interactively define fault frequency bands of interest and extract spectral metrics like peak amplitude, peak frequency, and band power from power spectrum data, without writing code. You can define and configure bearing, gear mesh, and custom fault frequency bands from which targeted spectral metrics of the power spectrum data can be obtained. The task generates a plot of the frequency bands and power spectrum data that lets you interactively explore the effects of changing parameter values and options. The task also automatically generates code that becomes part of your live script.

For more information, see **Extract Spectral Features**. For an example, see Analyze Gear Train Data and Extract Spectral Features Using Live Editor Tasks.

RUL Examples: Predict RUL using artificial intelligence

New examples illustrate RUL prediction using techniques of machine learning and deep learning.

- Battery Cycle Life Prediction From Initial Operation Data
- Remaining Useful Life Estimation using Convolutional Neural Network

R2020b

Version: 2.2.1

Bug Fixes

R2020a

Version: 2.2

New Features

Bug Fixes

Diagnostic Feature Designer: Generate MATLAB code in the app

You can now generate MATLAB code in the app to automate data processing, feature extraction, and feature ranking computations that you initially performed interactively. Apply this code to any data set that includes the same variables as the data set that you imported into the app when you generated the code. For example, you can use this code to compute a feature set for a larger set of measurement data than the measurement data set that you worked with in the app, or to update the feature set if you obtain new data.

For more information, see Automatic Feature Extraction Using Generated MATLAB Code.

R2019b

Version: 2.1

New Features

Bug Fixes

Live Editor Tasks: Interactively perform phase space reconstruction and extract signal-based condition indicators

Use new Live Editor tasks to perform phase space reconstruction and to extract the approximate entropy, correlation dimension, and Lyapunov exponent without writing code. The tasks can generate plots that let you interactively explore the effects of changing parameter values and options. They also automatically generate code that becomes part of your live script.

In R2019b, Predictive Maintenance Toolbox includes four tasks:

- **Reconstruct Phase Space** — Reconstruct the phase space with specified or automatically computed lag and embedding dimension
- **Estimate Approximate Entropy** — Estimate the regularity of a nonlinear time series
- **Estimate Correlation Dimension** — Estimate the chaotic signal complexity of a nonlinear time series
- **Estimate Lyapunov Exponent** — Estimate the rate of separation of infinitesimally close trajectories

To use the tasks in the Live Editor, on the **Live Editor** tab, in the **Task** menu, select a task. Alternatively, in a code block in a live script, begin typing the task name and select the task from the suggested command completions. For an example of using multiple Live Editor tasks in a workflow, see Reconstruct Phase Space and Estimate Condition Indicators Using Live Editor Tasks.

For more information about Live Editor tasks generally, see Add Interactive Tasks to a Live Script (MATLAB).

Spectral Analysis: Define frequency bands and extract spectral features

Faults in electrical motor and rotating machinery components manifest in the spectrum of the motor current or in drivetrain vibration signals. By analyzing spectral patterns (such as the peak amplitude or band power) within certain characteristic frequency bands of the signal spectrum, various types of component faults can be detected or their degradation monitored. Predictive Maintenance Toolbox offers the following four new commands for generating spectral metrics within specified frequency bands:

- `faultBands` — Define fault frequency bands around characteristic fault harmonics and sidebands within the frequency range of the signal spectrum. For more information, see `faultBands`.
- `bearingFaultBands` — Construct frequency components that define bearing faults in the outer and inner race, rolling element, and cage of a bearing. For more information, see `bearingFaultBands`.
- `gearMeshFaultBands` — Construct frequency components that define gear mesh faults. For more information, see `gearMeshFaultBands`.
- `faultBandMetrics` — Extract spectral features like peak amplitude, peak frequency, and band power from a signal spectrum using the fault frequency bands obtained using one the above commands. For more information, see `faultBandMetrics`.

For an example that demonstrates the use of motor current signature analysis (MCSA) to identify gear faults, see Motor Current Signature Analysis for Gear Train Fault Detection.

Prognostic Ranking in Diagnostic Feature Designer: Rank features to determine best indicators of system degradation in Diagnostic Feature Designer

You can now use the monotonicity, trendability, and prognosability methods to determine which features are the best indicators of system degradation and contribute the most to accurately predicting remaining useful life (RUL). These methods were first introduced as feature metrics for the command line in R2018b. Use these methods when you have system run-to-failure data to determine which condition indicators best track the system degradation process.

To access these methods once you have calculated features, on the **Feature Ranking** tab, click **Prognostic Ranking**. To access one of the ranking methods that were previously available in the app, on the **Feature Ranking** tab, click **Classification Ranking**.

For more information on prognostic ranking in the app, see the **Prognostic Ranking** parameter description in **Diagnostic Feature Designer**.

For more information on the prognostic RUL metrics, see `monotonicity`, `trendability`, and `prognosability`.

Machine-Specific Rotation Speeds: Filter TSA signals using machine-specific rotation speeds in Diagnostic Feature Designer

You can now compute machine-specific rotation speeds when you perform time-synchronous averaging (TSA). Apply these speed values when you filter the resulting TSA signals. Previously, you could specify only one constant rotation speed value when you filtered TSA signals. Use this approach to tune the filtered signal more accurately for each TSA signal when their individual rotation speeds vary.

For an example showing how to work with individual RPM values, see Isolate a Shaft Fault Using Diagnostic Feature Designer.

generateSimulationEnsemble: Control display of simulation progress when generating a simulation ensemble

You can now control whether `generateSimulationEnsemble` displays a simulation progress line in the MATLAB command window. Previously, `generateSimulationEnsemble` always displayed progress. To disable the progress display, set the `ShowProgress` name-value pair argument to `false`.

For more information, see `generateSimulationEnsemble`.

R2019a

Version: 2.0

New Features

Bug Fixes

Diagnostic Feature Designer: Interactively extract, visualize, and rank features from measured or simulated data for machine diagnostics and prognostics

The **Diagnostic Feature Designer** app allows you to interactively explore and extract features from ensemble data that contains signals, spectra, and condition labels from multiple members. The app provides tools for visualization, analysis, feature generation, and feature ranking. You design and compare features interactively, and then determine which features are best at discriminating between data from nominal systems and from faulty systems.

To open the **Diagnostic Feature Designer**, type `diagnosticFeatureDesigner` at the command line.

For more information, see **Diagnostic Feature Designer**.

Gear Condition Metrics: Extract standard gear condition indicators from time-synchronous averaged signals

You can now use the `gearConditionMetrics` command to extract standard gear condition indicators from a set of raw, difference, regular, and residual time-synchronous averaged (TSA) signals.

For more information, see `gearConditionMetrics` and Condition Indicators for Gear Condition Monitoring.

fileEnsembleDatastore: Specify list of ensemble datastore file names

`fileEnsembleDatastore` now lets you explicitly specify a list of files to include in the ensemble datastore. Previously, you could provide only a single location folder, and the ensemble datastore included all files at that location with a specified extension. The new functionality lets you specify a subset of files in a folder to include, or include files from more than one folder. You can also specify files using a wildcard character (*). To specify files to include, use the `location` input argument when you create the ensemble datastore. For more information, see `fileEnsembleDatastore`.

R2018b

Feature Selection Metrics: Evaluate features to determine best indicators of system degradation and improve accuracy of remaining useful life predictions

Selecting appropriate estimation parameters out of all available features is the first step in building a reliable remaining useful life (RUL) prediction engine. Predictive Maintenance Toolbox offers three feature selection metrics for accurate RUL prediction: monotonicity, trendability, and prognosability. Use these metrics when you have run-to-failure data of systems to determine which condition indicators best track the degradation process.

For more information, see the `monotonicity`, `trendability`, and `prognosability` reference pages.

Features for Rotating Machinery: Extract the residual, difference, and regular signals from a time-synchronous averaged signal to generate diagnostic feature

You can now use the `tsaresidual`, `tsadifference`, and `tsaregular` commands to extract the residual, difference, and regular signals from a time-synchronous averaged (TSA) signal, respectively. These features detect changes in the TSA signal that are indicative of a change in the machine state.

For more information, see the `tsaresidual`, `tsadifference`, and `tsaregular` reference pages.

fileEnsembleDatastore Object: Read all variable types from ensemble member while loading file only once

When you use a `fileEnsembleDatastore` object, use the new `ReadFcn` property to specify one function for reading all ensemble variables. The `read` command calls this function to read all data variables, independent variables, and condition variables that are specified in the `SelectedVariables` property of the ensemble datastore.

Previously, you had to specify separate functions `DataVariablesFcn`, `IndependentVariablesFcn`, and `ConditionVariablesFcn` for reading data variables, independent variables, and condition variables, respectively. Therefore, the `read` operation accessed each member file in the ensemble up to three separate times to read all selected variables. `ReadFcn` increases efficiency by allowing `read` to read all variables in a member file in a single operation.

For more information about using the new property, see the `fileEnsembleDatastore` reference page.

Compatibility Considerations

The `DataVariablesFcn`, `IndependentVariablesFcn`, and `ConditionVariablesFcn` properties of `fileEnsembleDatastore` will be removed in a future release. Use the `ReadFcn` property instead. For more details, see `fileEnsembleDatastore`.

Ensemble Datastore Objects: Read multiple ensemble members in one operation

You can now configure both `simulationEnsembleDatastore` and `fileEnsembleDatastore` objects to read more than one ensemble member per call to the `read` function. By default, calling

`read` returns a single table row containing data from one ensemble member. To read multiple ensemble members at once, set the new `ReadSize` property to a positive integer value. For example, if you set `ReadSize` to 3, then calling `read` returns a three-row table containing data from the next three ensemble members. The `read` operation also sets the `LastMemberRead` to a string vector containing the file paths of the corresponding three files.

For more information and examples, see the `simulationEnsembleDatastore` and `fileEnsembleDatastore` reference pages.

fileEnsembleDatastore Object: Create ensembles of files with multiple file extensions

You can now create a `fileEnsembleDatastore` object to manage an ensemble of files that do not all have the same file extension. For instance, suppose that you have some data stored in .xls files, and some stored in .s files. You can create a `fileEnsembleDatastore` object for these files using a string array of both file extensions, as follows.

```
extension = [".xls",".xlsx"];
fensemble = fileEnsembleDatastore(location,extension)
```

Both `fileEnsembleDatastore` and `SimulationEnsembleDatastore` objects also have a new read-only `Files` property, which is a string vector containing the file names of all ensemble members.

For more information about managing files with ensemble datastore objects, see the `fileEnsembleDatastore` and `simulationEnsembleDatastore` reference pages.

Functionality being removed or changed

DataVariablesFcn, IndependentVariablesFcn, and ConditionVariablesFcn properties of fileEnsembleDatastore will be removed

Still runs

The `DataVariablesFcn`, `IndependentVariablesFcn`, and `ConditionVariablesFcn` properties of `fileEnsembleDatastore` will be removed in a future release. Use the `ReadFcn` property instead.

The `ReadFcn` property, introduced in R2018b, lets you specify one function to read all variable types from your ensemble datastore. Formerly, you had to designate functions separately for data variables, independent variables, and condition variables. An advantage of using `ReadFcn` is that the `read` operation accesses each member file only once to read all the variables. With separate functions for each variable type, `read` opens the file up to three times to read all variable types. Thus, designating a single `ReadFcn` is a more efficient way to access the datastore.

Update Code

To update your code to use the new property:

1. Rewrite your `fileEnsembleDatastore` read functions into one new function that reads variables of all types. (See Create and Configure File Ensemble Datastore for an example of such a function.)
2. Set `DataVariablesFcn`, `IndependentVariablesFcn`, and `ConditionVariablesFcn` to `[]` to clear them.

3 Set `ReadFcn` to the new function.

currentValue syntax of predictRUL not recommended
Still runs

The following syntax of the `predictRUL` command is not recommended:

```
estRUL = predictRUL(mdl,currentValue,threshold)
```

For a trained degradation model `mdl`, this syntax estimates the remaining useful life (RUL) based on the current measured value `currentValue` of a condition indicator. A more reliable way to estimate RUL for degradation models is to update the model with each successive measurement of the condition indicator using the `update` command. Then, use the updated model to estimate the RUL.

Update Code

Suppose that you store successive condition indicator measurements in an array `TestData`. The array contains measurements at regular intervals at least up to the time `currentTime` for which `currentValue` is the condition indicator measurement. To update your code, replace:

```
estRUL = predictRUL(mdl,currentValue,threshold)
```

with the following code:

```
for t = 1:CurrentTime
    update(mdl,TestData(t,:))
end
estRUL = predictRUL(mdl,threshold)
```

For an example, see the `predictRUL` reference page.

R2018a

Version: 1.0

New Features

Survival, similarity, and time-series models for remaining useful life (RUL) estimation

Remaining useful life (RUL) is the expected value of time to failure conditional on the history of the component known by sensor measurements and auxiliary output information. Predictive Maintenance Toolbox provides similarity models, degradation models, and survival models for RUL estimation. For more information on these types of RUL estimation, see Models for Predicting Remaining Useful Life.

Time, frequency, and time-frequency domain feature extraction methods for designing condition indicators

A condition indicator is a feature of system data whose behavior changes in a predictable way as the system degrades or operates in different operational modes. Such features are useful for distinguishing normal from faulty operation or for predicting remaining useful life. Predictive Maintenance Toolbox supplements existing functionality in MATLAB and Signal Processing Toolbox™ with additional functions that can be useful for designing condition indicators. For more information, see Condition Indicators for Monitoring, Fault Detection, and Prediction.

Managing and labeling of sensor data imported from local files, Amazon S3, Windows Azure Blob Storage, and Hadoop Distributed File System

You may have collected measurements on systems using sensors for healthy operation or faulty condition and stored them in local files, cloud storage platforms or in distributed file systems. You can organize, read, and manage such measured data using the `fileEnsembleDatastore` object and use it for designing your predictive maintenance algorithms. For more information, see File Ensemble Datastore With Measured Data.

Managing and labeling of simulated machine data from Simulink models

Instead of data from physical systems, you may have a Simulink® model that represents a range of healthy and faulty operating conditions. The `generateSimulationEnsemble` function helps you generate such data from your model. Then use the `simulationEnsembleDatastore` object to organize, read, and manage the data for designing your predictive maintenance algorithms. For more information, see Generate and Use Simulated Data Ensemble.

Examples for developing predictive maintenance algorithms for motors, gearboxes, batteries, and other machines

This release includes the following examples on data generation, fault detection and diagnosis, and RUL prediction:

- Data Generation
  - Using Simulink to Generate Fault Data
  - Multi-Class Fault Detection Using Simulated Data
- Fault Detection and Diagnosis

    - Rolling Element Bearing Fault Diagnosis
    - Fault Diagnosis of Centrifugal Pumps using Steady State Experiments
    - Fault Diagnosis of Centrifugal Pumps using Residual Analysis
    - Fault Detection Using an Extended Kalman Filter
    - Fault Detection Using Data Based Models
    - Detect Abrupt System Changes Using Identification Techniques
- Prediction
    - Similarity-Based Remaining Useful Life Estimation
    - Wind Turbine High-Speed Bearing Prognosis
    - Condition Monitoring and Prognostics Using Vibration Signals
    - Nonlinear State Estimation of a Degrading Battery System