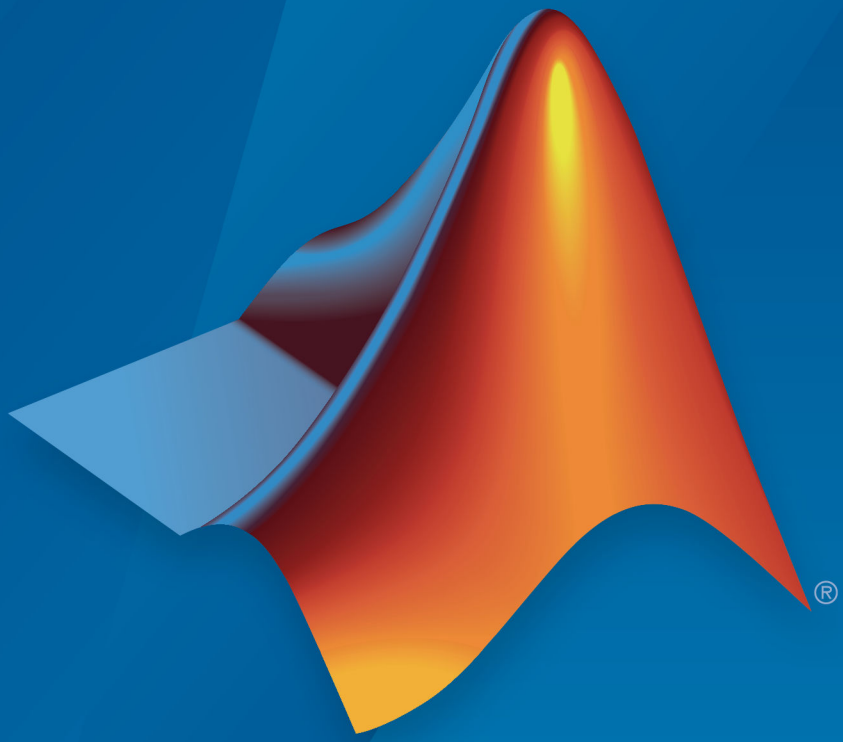
Predictive Maintenance Toolbox™ Release Notes



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

R2019a

R2018b

R2018a

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

Version: 1.1

New Features

Bug Fixes

Compatibility Considerations

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 .xlsx 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