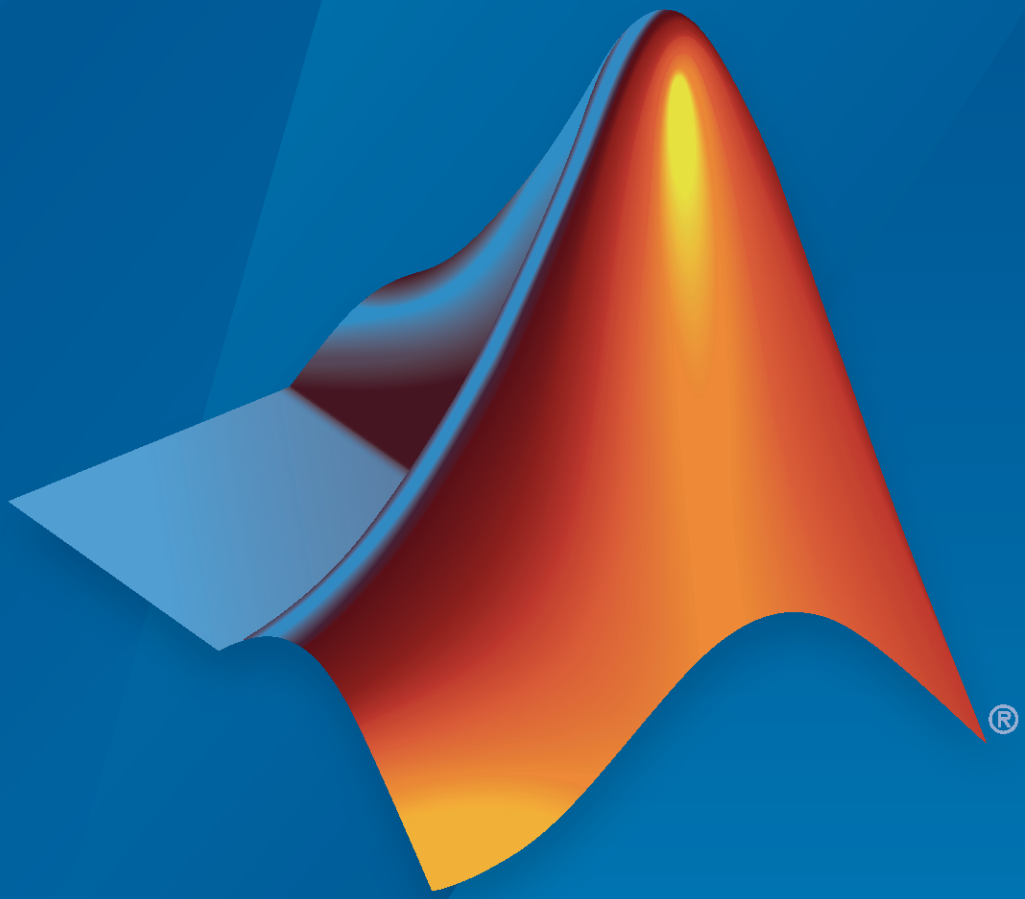
Predictive Maintenance Toolbox™ Release Notes



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

R2020a

R2019b

R2019a

R2018b

R2018a

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

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 .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