# Prognostics Model Library: User Manual

Matthew Daigle
NASA Ames Research Center, Moffett Field, CA 94035
matthew.j.daigle@nasa.gov

August 18, 2016

## 1 Introduction

*Prognostics* is a technical problem dealing with predicting the future state of a system of interest. *Model-based prognostics* is a framework to solve this problem based on the use of dynamic system models. The *Prognostics Model Library* is a Matlab toolbox for developing models for prognostics to support this framework, and a growing library of prognostics models for different components. It defines a set of classes to construct models that can be simulated and used within model-based prognostics algorithms. The included prognostics models are for end-of-discharge prediction of lithium-ion batteries, and end-of-life prediction of centrifugal pumps and pneumatic valves.

The model-based prognostics paradigm is summarized in Fig. 1. There is a system being monitored, and one develops a model describing how the system evolves in time in response to its inputs. That model is used as the basis for state estimation, performed by an *observer*, and state prediction, performed by a *predictor*. This toolbox addresses the problem of defining models for this purpose. More details about the model structure are discussed later.

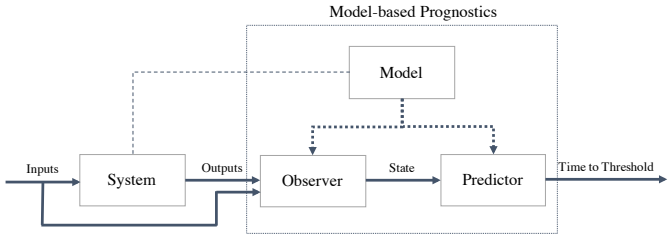


Figure 1: Model-based prognostics conceptual architecture.

### 1.1 Installation

The toolbox is packaged as `PrognosticsModelLibrary.mltbx`. To install the toolbox, open Matlab, double-click on the file, and click the install button. The toolbox files will be added to Matlab's user toolboxes directory and will make its path accessible to Matlab. The toolbox works with Matlab R2016a, but has been known to work down to R2012a.

### 1.2 Documentation

The toolbox is fully documented within Matlab, and can be accessed through Matlab's `help` and `doc` functions on any toolbox file, for example:

```
>> help Battery.StateEqn
>> doc Battery.StateEqn
```

Thus, the purpose of this user manual is to explain the theory behind the framework (Section 2), explain what the toolbox provides (Section 3), and demonstrate the toolbox through a detailed example (Section 4).

## 2 Background

For a given system model, the prognostics problem is focused on determining how the system will evolve in time, and when the system state will reach some region of interest. In defining a model for prognostics, one must define a state equation, an output equation, an input equation, and a threshold equation.

In the following, the system state is denoted by the vector $\mathbf{x}$, the inputs by the vector $\mathbf{u}$, and the outputs by the vector $\mathbf{z}$. The state equation defines how the state is updated from time $t$ to $t + \Delta t$, where $\Delta t$ is the sampling time:

$$\mathbf{x}(t + \Delta t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t)), \tag{1}$$

where $\mathbf{v}(t)$ is the process noise, and $\mathbf{f}$ is the state update function. The output equation defines how the system outputs are computed from the states and inputs:

$$\mathbf{z})(t) = \mathbf{h}(t, \mathbf{x}(t), \mathbf{u}(t), \mathbf{n}(t)), \tag{2}$$

where $\mathbf{n}(t)$ is the sensor noise, and $\mathbf{h}$ is the output function.

For prognostics, one is interested in predicting when a region of the state space is reached, and so this region must be defined. This is achieved through a *threshold equation*, $\mathbf{g}(t)$, that returns a Boolean indicating whether the system is in the given space or not. In general, this is expressed as a function of time, the states, and the inputs:

$$b = \mathbf{g}(t, \mathbf{x}(t), \mathbf{u}(t)), \tag{3}$$

where $b$ is the Boolean output (true/false).

In order to predict how the state evolves in time, the system inputs for all future time points must also be known. This is defined through the input equation, which computes what the system inputs are for a given time and a set of *input parameters*:

$$\mathbf{u}(t) = \mathbf{i}(t, \mathbf{p}), \tag{4}$$

where $\mathbf{i}$ is the input function and $\mathbf{p}$ are the input parameters. The input parameters are defined as a set of numbers that are used to parameterize what the system inputs are at a given time. For example, consider a battery model, where the input to the model is the power required by the load. This power depends on the system usage and could take one of a number of profiles. The input parameters could be used to define such profiles, e.g., by specifying magnitudes and durations for different load segments.

The prognostics problem is then defined as, given some initial system state, with some specified uncertainty, the future inputs to the system, with some specified uncertainty, and a description of the process noise, to determine any combination of the following:

- what is the probability of the threshold being reached within a given future time,
- when the system will enter a state-input space such that the threshold equation evaluates to true, and
- what is the state of the system when the threshold is reached.

In general, these are computed through a stocastic simulation, which is why a model toolbox is useful.

# 3 Toolbox Contents

The toolbox provides one main package, the `Model` package, which defines two classes:

- `Model`: a class defining a system model consisting mainly of state, input, and output variables, a model parameters structure, and a set of equations defining the model (state, output, and input equations).
- `PrognosticsModel`: a subclass of the `Model` class that also provides a threshold equation.

In addition, several example models are provided as additional packages:

- Battery: an electrochemistry-based lithium-ion battery model, for end-of-discharge prediction
- BatteryCircuit: an equivalent circuit-based lithium-ion battery model, for end-of-discharge prediction

- CentrifugalPump: a centrifugal pump model, including damage progression models, for end-of-life prediction
- PneumaticValve: a pneumatic valve model, including damage progression models, for end-of-life prediction

Each example model provides several files, that may include the following:

- `Create.m`: a function that creates the `PrognosticsModel` object
- `StateEqn.m`: a function defining the state equation
- `OutputEqn.m`: a function defining the output equation
- `InputEqn.m`: a function defining the input equation
- `ThresholdEqn.m`: a function defining the threshold equation
- `Initialize.m`: a function computing the initial system state given system inputs and outputs
- `Parameters.m`: a function defining the model parameters structure
- `test.m`: a function demonstrating the use of the example model under various conditions

# 4 Example

In this section, we go through the `Battery` example in detail. The code from the test function is provided below:

```
1  function test
2  % test    Test battery model for various conditions.
3  %
4  %   Copyright (c) 2016 United States Government as represented by
5  %   the Administrator of the National Aeronautics and Space Admin-
6  %   istration. All Rights Reserved.
7
8  % Create battery model
9  battery = Battery.Create;
10
11 % Simulate for default conditions
12 [T1,X1,U1,Z1] = battery.simulate(3100,'printTime',60);
13
14 % Plot results
15 figure;
16 battery.plotOutputs(T1,Z1);
17
18 % Determine end-of-discharge time for these conditions
19 T = battery.simulateToThreshold();
20 fprintf('EOD time is %g s\n',T(end));
21
22 % Simulate for a variable load profile
23 % Specified by a sequence of pairs of numbers, where the first is
```

```
% the load (in Watts) and the second is the duration (in seconds).
loads = [8; 10*60; 4; 5*60; 12; 15*60; 5; 20*60; 10; 10*60];
battery.inputEqnHandle = @(P,t)Battery.InputEqn(P,t,loads);
[T2,X2,U2,Z2] = battery.simulate(3150,'printTime',60);

% Plot results
figure;
battery.plotOutputs(T2,Z2);

% Determine end-of-discharge time for these conditions
T = battery.simulateToThreshold();
fprintf('EOD time is %g s\n',T(end));

% Plot results comparing the two simulations
figure;
battery.plotOutputs(T1,Z1,T2,Z2);
```

First, the `PrognosticsModel` object for the battery model is created. Then, the `simulate` method of the `Model` class is used, followed by its `plotOutputs` method. The resulting figure is shown as Fig. 2.

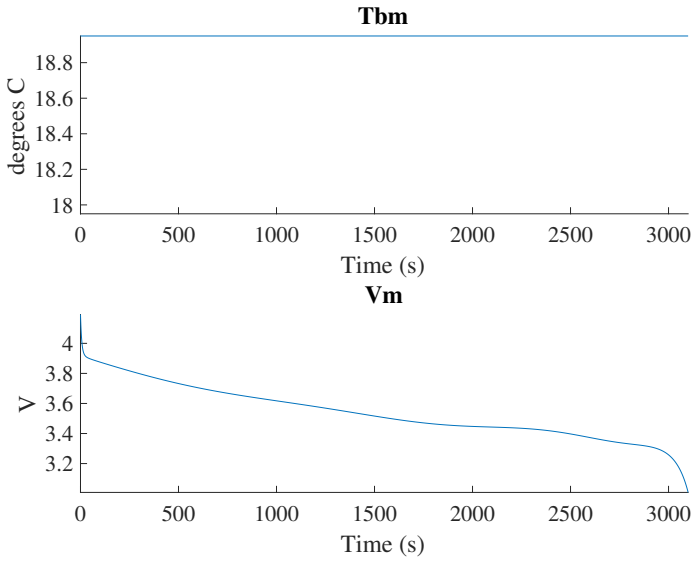


Figure 2: Battery model outputs for a constant load profile.

The default input is a constant load of 8 W. Temperature changes are not currently modeled, so the temperature stays flat. The voltage decreases immediately at time 0 due to the presence of the load, and then decreases gradually until state-of-charge becomes low and the discharge curve becomes nonlinear.

Next, the end-of-discharge time is computed using the `simulateToThreshold` method of `PrognosticsModel`. In this case, the voltage threshold is 3.0 V, and is computed as 3102 s.

Next, a variable load profile is set up. For the battery, the input parameters

consist of pairs of load magnitude and durations, i.e., the battery will see first a load of 8 W for 10 minutes, followed by a load of 4 W for 5 minutes, etc. The default input equation for the model is then modified by resetting the input equation handle property. An anonymous function is created so that the input parameters are part of the function, and it becomes the new default (without having to specify the input parameters again). The results are then plotted, and the resulting figure is shown as Fig. 2.

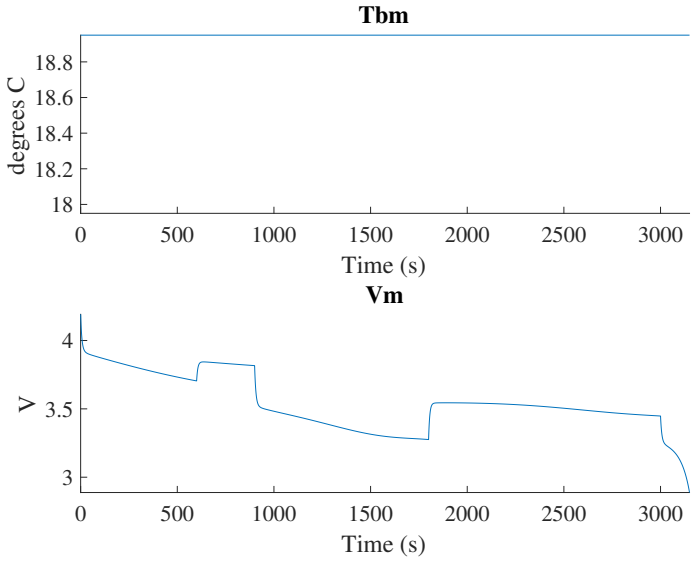


Figure 3: Battery model outputs for a variable loading profile.

Now, the voltage changes when the load changes. The end-of-discharge for this case is then computed as 3134 s. Outputs from both scenarios are then plotted on the same figure for comparison, shown as Fig. 4.

# 5 Notices



## 5.1 Disclaimers

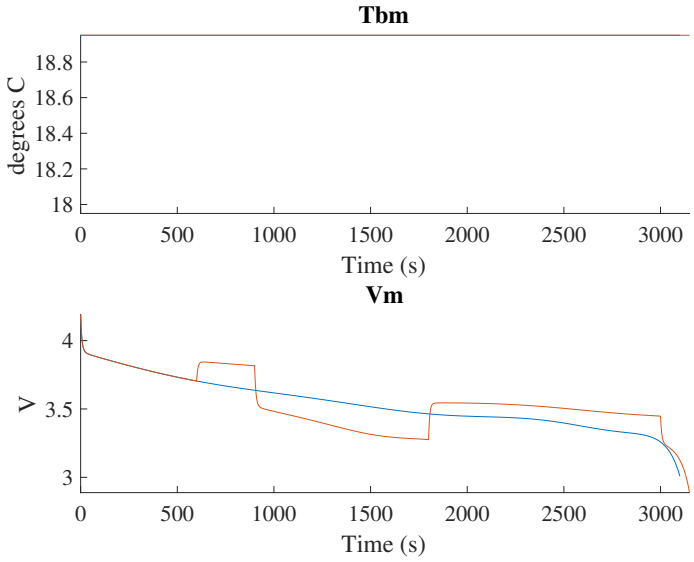


Figure 4: Comparison of battery model outputs for constnt and variable loading profiles.

THAT THE SUBJECT SOFTWARE WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL BE ERROR FREE, OR ANY WARRANTY THAT DOCUMENTATION, IF PROVIDED, WILL CONFORM TO THE SUBJECT SOFTWARE. THIS AGREEMENT DOES NOT, IN ANY MANNER, CONSTITUTE AN ENDORSEMENT BY GOVERNMENT AGENCY OR ANY PRIOR RECIPIENT OF ANY RESULTS, RESULTING DESIGNS, HARDWARE, SOFTWARE PRODUCTS OR ANY OTHER APPLICATIONS RESULTING FROM USE OF THE SUBJECT SOFTWARE. FURTHER, GOVERNMENT AGENCY DISCLAIMS ALL WARRANTIES AND LIABILITIES REGARDING THIRD-PARTY SOFTWARE, IF PRESENT IN THE ORIGINAL SOFTWARE, AND DISTRIBUTES IT"AS IS."

Waiver and Indemnity: RECIPIENT AGREES TO WAIVE ANY AND ALL CLAIMS AGAINST THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS, AS WELL AS ANY PRIOR RECIPIENT. IF RECIPIENT'S USE OF THE SUBJECT SOFTWARE RESULTS IN ANY LIABILITIES, DEMANDS, DAMAGES, EXPENSES OR LOSSES ARISING FROM SUCH USE, INCLUDING ANY DAMAGES FROM PRODUCTS BASED ON, OR RESULTING FROM, RECIPIENT'S USE OF THE SUBJECT SOFTWARE, RECIPIENT SHALL INDEMNIFY AND HOLD HARMLESS THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND

SUBCONTRACTORS, AS WELL AS ANY PRIOR RECIPIENT, TO THE EXTENT PERMITTED BY LAW. RECIPIENT'S SOLE REMEDY FOR ANY SUCH MATTER SHALL BE THE IMMEDIATE, UNILATERAL TERMINATION OF THIS AGREEMENT.