Object-oriented MATLAB toolbox for structural health monitoring

Technical documentation for version v1.0

By

Mihhail Samusev

Aalborg University
Department of Civil Engineering
Thomas Manns Vej 24
DK-9220 Aalborg

Contents

Chapter 1

Introduction

1.1 Purpose of the toolbox

The purpose of the toolbox is to provide a framework for the tasks characteristic to structural health monitoring (SHM) such as, data acquisition, basic signal processing, system identification in time and frequency domain, modal analysis and damage detection. The toolbox is being developed so that it contains classic SHM routines for educational purposes as well as allows for extensions with advanced routines for the research purposes.

1.2 Toolbox structure

The toolbox in its majority constructed based on the object-oriented programming (OOP) model, where the data and functions serving the same purpose are not treated separately, but logically grouped into user-defined data types known as classes. The model is widely used in modern software development, since it allows for better maintainability and extensibility of large project.

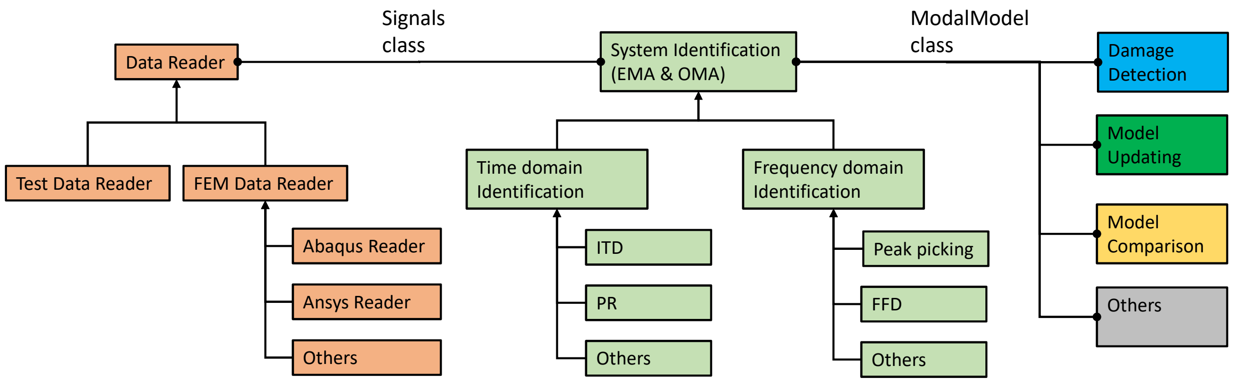


Figure 1.1: Conceptual structure of the SHM Toolbox.

The structure of the toolbox is presented in Figure 1.1, where different modules corresponding to classes their sub-classes are highlighted with the same color. Such structure allows for a organized work flow, from data reading to system identification and further to other relevant tasks. From Data Reader class the information is passed to System Identification class using a Signals class that contains the multi-channel signal together with important properties and functionality for signal processing. System Identification class outputs ModalModel class that contains the information and functionality relevant to the identified model. Several ModalModel objects (for example healthy and damaged) can be then be passed further to other modules, for instance for the damage identification.

1.3 Current version

Toolbox is under development and current release contains only a basic workflow for signal processing and time domain system identification. The release includes the following classes/functions:

- `Signals` - class, allows to store signals and perform basic low-, high, band-pass filtering, calculate correlation function matrix and power spectral density matrix. Relevant examples are included.
- `ID_TD_PolyReference` - function, time domain poly-reference (PR) system identification method (currently implemented as a standalone function, but should be a part of the class as shown in Figure 1.1)
- `ModalModel` - class, allows to store modal parameters of the identified model, calculate frequency response function (FRF) and calculate participation factors to fit the correlation function matrix of the data Brincker [2015]. Relevant examples are included.

As well as the following examples:

- `SignalsClassEx1` - creating an instance of the Signals class
- `SignalsClassEx2` - detrending and removing the noise with band pass filter
- `SignalsClassEx3` - calculating correlation, power spectral density and plotting (based on data from Brincker [2015])
- `ModalModelClassEx1` - creating an instance of the ModalModel class
- `ModalModelClassEx2` - obtaining ModalModel from a system identification technique and using it to fit data correlation function matrix (based on data from Brincker [2015])

Chapter 2

Getting started

2.1 Prerequisites for the toolbox

The toolbox is written in MATLAB version 2018a and the compatibility is successfully tested on versions 2014a. The users with version below R2008a will not be able to use the toolbox due to the old syntax in object-oriented programming style.

The toolbox often favours well optimized standard MATLAB function implementations to own implementation if such standard implementation is available. However, some of the standard functions require the corresponding MATLAB Toolbox, that a user might or might not have by default. For example, a correlation function matrix calculation utilizes the standard `xcorr` function of the Signal Processing Toolbox. The user can check the list of available toolboxes in his version of MATLAB by typing `ver` to the command window.

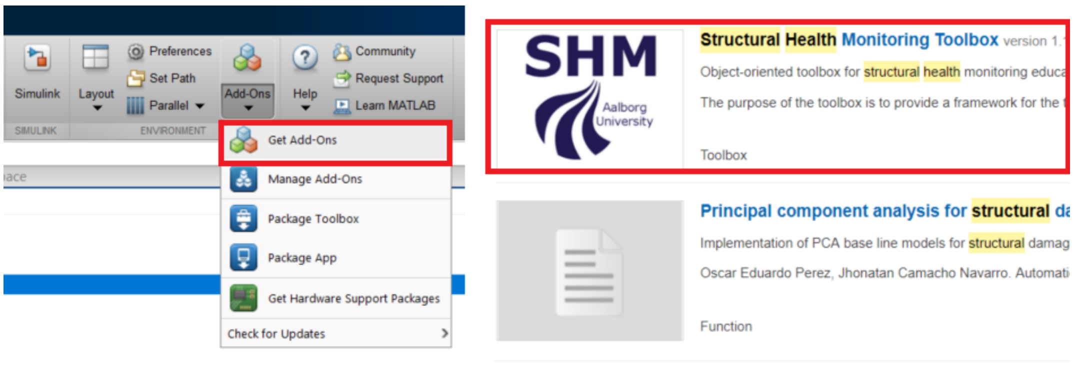
List of required standard MATLAB toolboxes for the correct performance of the SHM Toolbox:

- Signal Processing Toolbox

2.2 Installation

If the user already has an installation file with `*.mltbx` extension then the procedure is as simple as running the file. The installation chooses a standard path for add-on toolboxes: `\MATLAB\Add-Ons\Toolboxes\*` The latest version of the toolbox is uploaded to MATLAB File Exchange or GITHUB. To obtain the installation file proceed to the link, click *download* and choose *toolbox* from the drop down menu.

The toolbox can also be installed from inside MATLAB. It is done by opening Add-Ons Explorer by clicking *HOME - ENVIRONMENT - Get Add-Ons* as shown in Figure 2.1. The toolbox can be found by its name and logo. Finally, the user has to open the toolbox description page and click *Add* on the right side of the window. The list of added toolboxes can be viewed and edited by clicking *Manage Add-Ons* (See Figure 2.1).



Structural Health Monitoring Toolbox version 1.
Object-oriented toolbox for structural health monitoring educa
The purpose of the toolbox is to provide a framework for the
Toolbox

Principal component analysis for structural da
Implementation of PCA base line models for structural damag
Oscar Eduardo Perez, Jhonatan Camacho Navarro. Automati
Function

Figure 2.1: Opening Add-Ons Explorer to browse the toolbox

2.3 How to use in-built documentation

Inbuilt MATLAB documentation through `help` function is the main source of documentation for the toolbox. In order to get started type `help shmtoolbox` in the command window to see the available contents of the current version of the toolbox.

From there, the user can continue typing `help` in front the functions or classes. This will call a more detailed individual documentation for a chosen function or class with references to other related content of the toolbox.

The `help shmtools` also contains the list of examples that the user can explore to get started with the toolbox. An example file can be opened a script by writing `open` or `edit` in front of the example file name.

2.4 Working with objects in MATLAB

As mentioned before, the toolbox is being developed using object-oriented programming model. Objects allow to hold data together with the functions that operate on that data. As an example, an object of type `Signals` (in other words belonging to `Signals` class) is created in Listing 2.1. The input that takes part in the creation of the object is according to class constructor whose documentation can be viewed typing `help Signals`.

Listing 2.1: Creating an object of type Signals

```
y = rand(100,3); % random 3 channel signal
Fs = 50;         % sampling frequency
s = Signals(y, Fs, '3 channels')

whos('s')
  Name      Size            Bytes  Class     Attributes

  s         1x1                 8  Signals
```

Calling the object reveals its properties. It can be seen in Listing 2.2 that the class constructor calculates additional properties that are relevant for the object. All the visible properties can be accessed using dot notation.

Listing 2.2: Properties of `s`

```
s =
  Signals with properties:
    Label: '3 channels'
        N: 100
    nChan: 3
    ydata: [100x3 double]
       Fs: 50
       dt: 0.0200
        T: 2
    tdata: [100x1 double]
        R: []
      PSD: []

s.dt     % access properties using dot notation
ans =
    0.0200
```

Similar to properties, user can access the functionality (methods) of the class with the dot notation. An example in Listing 2.3 shows how an object property `R` (correlation function matrix) can be initialized.

Listing 2.3: Using a class method to initialize a property

```
nLag = 10;         % number of time lags (excl. zero)
s.BuildCF(nLag); % build correlation function matrix
s =
  Signals with properties:
...
    tdata: [100x1 double]
        R: [3x3x11 double] % correlation function matrix
      PSD: []
```

The full list of available properties and methods of a particular class can be viewed by pressing a TAB key after the object name and a dot. However, this method provides neither a description of the properties/methods nor the inputs to the methods. Therefore it is recommended to use `help Signals` to get a better picture of the class capabilities.

It should be noted that an object should not be confused with a structure. Unlike object properties, structure properties can be added by incorrect spelling as shown in Listing 2.4. Both names and ability of the user to access and modify the object properties are strictly defined by the class.

Listing 2.4: Incorrect spelling leads to new structure property

```
s = struct('Time',10)
s =
  struct with fields:
    Time: 10

   s.time = 15 % attempt to change s.Time with incorrect case
s =
  struct with fields:
    Time: 10
    time: 15
```

Another major difference is that the objects are passed by reference and structures are passed by value similar to other most of MATLAB data types such as `string`, `double`, etc. Consider an example in Listing 2.5. The instance of an object `s` is assigned to a variable `sCopy`. The assignment happens by reference, which means that `sCopy` gets assigned an address of `s` in computer memory rather than a copy of `s` (which would be called an assignment by value). Since now both `s` and `sCopy` are located on the same memory address they become dependent on each other and the changes in one of the objects will be applied to the other.

Listing 2.5: Incorrect copy of an object

```
s = Signals(rand(100,3), 50, '3 channels')
s =
  Signals with properties:
    Label: '3 channels'
        N: 100
        ...

sCopy = s; % assign s to sCopy (assigns reference and not a copy of s)
sCopy.SetLabel('Label has been changed'); % change label

s =
  Signals with properties:
    Label: 'Label has been changed' % label of original s has changed
        N: 100
        ...
```

Note that if the user copieas a single property of the class to a variable, basic behaviour (passing by value) is used, and the variable is initialized with an independent copy of the information stored in the property.

2.5 Bug Reports

Open source software is, unfortunately, difficult to keep completely free from bugs. Should you find a bug in the toolbox, please contact msa@civil.aau.dk.

Bibliography

Brincker, R. (2015). *Introduction to operational modal analysis*. Wiley, 1. ed. edition.