OpenBDLM

OpenBDLM V1.0 reference manual

Ianis Gaudot, Luong H. Nguyen, James-A. Goulet
Polytechnique Montreal

January 23, 2019

Contents

1 What is OpenBDLM ?

OpenBDLM is a MATLAB open-source software developed to use Bayesian Dynamic Linear Models for long-term time series analysis (i.e time step in the order of one hour or higher). OpenBDLM is capable to process simultaneously several time series data to interpret, monitor and predict their long-term behavior. OpenBDLM includes an anomaly detection tool which allows to detect abnormal behavior in a fully probabilistic framework. OpenBDLM handles time series with missing data and non-uniform timestep vector. OpenBDLM is available for download from GitHub at https://github.com/CivML-PolyMtl/OpenBDLM.

Keywords: time series analysis and forecasting, linear gaussian state-space models, time-series decomposition, anomaly detection, filtering, smoothing, bayesian analysis.

2 Installing OpenBDLM

These instructions shows how to download and set-up OpenBDLM on your local machine for direct use, testing and development purposes.

2.1 Prerequisites

MATLAB (version 2016a or higher) installed on Mac OSX or Windows.

The MATLAB *Statistics and Machine Learning Toolbox* is required.

2.2 Installing

1. Remove from your MATLAB path any previous OpenBDLM versions.

2. Extract the ZIP file from https://github.com/CivML-PolyMtl/OpenBDLM (or clone the git repository) in a folder you will be working from.

3. Add "OpenBDLM-master" folder and all its subfolders to your MATLAB path:

   - using the "Set Path" dialog box in MATLAB, or
   - by running `addpath` function from the MATLAB command window

3 Getting started

3.1 Call OpenBDLM

- Set the current working directory to the folder "OpenBDLM-master"
- Type `OpenBDLM_main;`, and type ↵. in the MATLAB command line.

The OpenBDLM starting menu should appear on the MATLAB command window (see Listing 1). Type `Q` in the command line, and type ↵ to quit the program.

```
_____________________________________________________
     Starting OpenBDLM_V1.0
_____________________________________________________
            Time series analysis using
            Bayesian Dynamic Linear Models
_____________________________________________________
- Start a new project:

     *      Enter a configuration filename
     0   -> Interactive tool

- Type D to Delete project(s), V for Version control, Q to Quit.

     choice >>
```

Listing 1: OpenBDLM starting menu on MATLAB command window when calling `OpenBDLM_main;`

3.2 Demo

In the MATLAB command line, type `run_DEMO;` followed by ↵ to run a demo. Some messages on the MATLAB command window show that the program runs properly (see Listing 2). Some figures as shown in Figure 28, and Figure 30 should popup on the screen [1]

[1] The demo runs in batch the example described in Section 22.

```
Starting OpenBDLM_V1.0...
Starting a new project...
Building model...
Simulating data...
Plotting data...
Saving database (binary format) ...
Saving database (csv format) ...
Saving project...
Printing configuration file...
Saving database (binary format) ...
Saving project...
Done ! See you soon !
```

Listing 2: Output on MATLAB command window when running `run_DEMO.m`

4 OpenBDLM inputs and outputs

4.1 Inputs

OpenBDLM_main accepts three types of input

- no input

```
OpenBDLM_main;
```

- the name of configuration file given as a character vector

```
OpenBDLM_main('CFG_DEMO.m');
```

- a cell array of character vectors

```
OpenBDLM_main({'''CFG_DEMO.m''','''Q'''});
```

4.2 Outputs

`OpenBDLM_main.m` has the possibility to return four output MATLAB structures containing the information about the internal variables `data`, `model`,`estimation`, `misc`.

```
[data, model, estimation, misc]=OpenBDLM_main;
```

4.2.1 data

The `data` structure stores the time series used for the analysis. The timestamps values, the amplitude values and the labels values are stored in the fields `timestamps`, `values`, and `labels`, respectively.

- `labels` See Section 9.1.2 for details.
- `timestamps` See Section 9.1.2 for details.
- `values` See Section 9.1.2 for details.

4.2.2 model

The `model` structure stores all the information about the model used for the analysis. This variables has 14 fields, amongst them `hidden_states_names`, `A`, `C`, `Q`, `R`, `Z`, `components`, `parameter_properties`, `initX`, `initV`, `initS`.

- `hidden_states_names`: this field stores a $1 \times \mathtt{S}$ cell array, where $\mathtt{S} = \{1, 2\}$ is the number of model classes. Each cell array is a $\mathtt{L} \times 3$ cell array, where `L` is the number of hidden states. The first column of the cell array stores the reference name of the hidden state, the second column indicate the index of the model class to which the hidden state belongs and the third column indicates the index of the time series to which the hidden states belongs.
- `A`: this field stores a $1 \times \mathtt{S}$ cell array, where $\mathtt{S} = \{1, 2\}$ is the number of model classes. The cell array contains function handles used to build the full transition matrix.
- `C`: this field stores a $1 \times \mathtt{S}$ cell array, where $\mathtt{S} = \{1, 2\}$ is the number of model classes. The cell array contains function handles used to build the full observation matrix.
- `Q`: this field stores a $1 \times \mathtt{S}$ cell array, where $\mathtt{S} = \{1, 2\}$ is the number of model classes. The cell array contains function handles used to build the full process noise covariance matrix.
- `R`: this field stores a $1 \times \mathtt{S}$ cell array, where $\mathtt{S} = \{1, 2\}$ is the number of model classes. The cell array contains function handles used to build the full observation noise covariance matrix.
- `Z`: this field stores a function handle used to build the transition probabilities matrix.

- `components`: See Section 6.3 for details.
- `parameter_properties`: See Section 6.4 for details.
- `initX`: See Section 6.5 for details.
- `initV`: See Section 6.5 for details.
- `initS`: See Section 6.5 for details.

4.2.3 estimation

The `estimation` structure stores the filtered or smoothed hidden states results. This variables has 11 fields, amongst them `x`, `V`, `y`, `Vy`, `S`, `LL`, `x_M`, `V_M`, `VV_M`.

- `x`: this field stores the mean of the estimated hidden states. `x` stores a `L` $\times$ `T` array, where `L` and `T` are the number of hidden states and the length of time vector, respectively.
- `V`: this field stores the variance of the estimated hidden states. `V` stores a `L` $\times$ `T` array, where `L` and `T` are the number of hidden states and the length of time vector, respectively.
- `y`: this field stores the mean of the estimated system responses. `y` stores a `D` $\times$ `T` array, where `D` and `T` are the number of time-series and the length of time vector, respectively.
- `Vy`: this field stores the variance of the estimated system responses. `Vy` stores a `D` $\times$ `T` array, where `D` and `T` are the number of time-series and the length of time vector, respectively.
- `S`: this field stores the probability of each model class. `S` stores a `T` $\times$ `S` array, where `D` and `S` are the number of time-series and the number of model classes, respectively.
- `LL`: this field stores the value of the log-likelihood.
- `x_M`: this field stores the mean of the estimated hidden states for each model class before the merging step. `x_M` stores a $1 \times$ `S` cell array, where where `S` $= \{1, 2\}$ is the number of model classes. Each cell array stores a `L` $\times$ `T` array, where `L` and `T` are the number of hidden states and the length of time vector, respectively.

- `V_M`: this field stores the variance and covariance of the estimated hidden states for each model class before the merging step. `V_M` stores a $1 \times \mathtt{S}$ cell array, where where $\mathtt{S} = \{1, 2\}$ is the number of model classes. Each cell array stores a $\mathtt{L} \times \mathtt{L} \times \mathtt{T}$ array, where `L` and `T` are the number of hidden states and the length of time vector, respectively.

4.2.4 misc

The `misc` structure stores the options and internal variables needed for running the software. This variables has 3 fields, amongst them `ProjectName`, `internalVars`, `options`.

- `ProjectName`: this field stores the name of the project as a character array.
- `internalVars`: this field stores internal variables which are needed for running the software.
- `options`: this field stores the options that control different aspects of the software. The list of options is given in Section 6.6.

4.3 OpenBDLM files types

OpenBDLM reads and/or creates five types of files: `DATA_`, `CFG_`, `PROJ_`, `RES_` and `LOG_`.

- `DATA_` files: the files named with the prefix `DATA_` are MATLAB MAT binary files that store the time series data (see Section 9.1.2). These files are located in the "data/mat" subfolder.
- `CFG_` files: the files named with the prefix `CFG_` are MATLAB scripts used to initialize and export a project (see Section 6). These files are located in the "config_files" subfolder.
- `PROJ_` files: the files named with the prefix `PROJ_` are MATLAB MAT binary files that stores a full project for further analysis. These files are located in the "saved_projects" subfolder.
- `RES_` files: the files named with the prefix `RES_` are MATLAB MAT binary files that stores the estimation results. These files are located in the "results/mat" subfolder.

- `LOG_` files: the files named with the prefix `LOG_` are plain TEXT files that record events occurring during the program run. These files are located in the "log_files" subfolder.

5 OpenBDLM running modes

5.1 Interactive mode

In the interactive mode, OpenBDLM requests and reads input from the user. The input must be provided from the keyboard and be typed in the Matlab command line. Each input is validated after typing ↵.

5.2 Batch mode

In the batch mode, the inputs must be provided in advance by the user and stored in a cell array of characters vector. OpenBDLM reads sequentially the provided input and it performs the analysis silently as requested from the inputs. The batch mode requires the user to be familiar with the interactive mode because the set of input must be provided prior to the analysis.

6 Configuration file

The configuration files are MATLAB scripts that are used to initialize a project. The name of a configuration file can be given as a input to the function `OpenBLDM_main.m`, as mentionned in Section 4.1. The configuration file must follow a specific organization, which includes 6 sections: Project name, Data, Model structure, Model parameters, Initial states values and Options. The first three sections Project name, Data, Model structure are required. The last three sections Model parameters, Initial states values and Options are optional.

6.1 Project name

This section of the configuration file defines the name of the project as a vector of characters stored in the field `ProjectName` of the MATLAB structure `misc`.

6.2 Data

This section of the configuration file defines information required for loading the data from a `DATA_` file located in "/data/mat" subfolder. The file must

follow the format described in Section 9.1.2. The timestamps values, the amplitude values and the labels values must be stored in the fields `timestamps`, `values`, and `labels` of the MATLAB structure named `data`.

6.3 Model structure

This part of the configuration file defines the model in a MATLAB structure named `model.component`. The structure `model.component` must have three fields, named `model.component.block`, `model.component.ic`, and `model.component.const`.

- `model.component.block`: it defines the block components associated with each time-series. The field `block` stores a $1 \times$ `S` cell array, where `S` $= \{1, 2\}$ is the number of model classes. Each cell array is a $1 \times$ `D` cell array of array, where `D` is the number of time series. Each block component is associated with a reference number:
  - 11: Local level
  - 12: Local level plus local trend
  - 13: Local level plus local trend plus local acceleration
  - 21: Local level compatible with local trend
  - 22: Local level compatible with local acceleration
  - 23: Local trend compatible with local acceleration
  - 31: Periodic
  - 41: First-order autoregressive
  - 51: Kernel regression
- `model.component.const`: it constrains model parameters between block components of different model classes. The field `const` stores a $1 \times$ `S` cell array, where `S` $= \{1, 2\}$ is the total number of model classes. It is defined only if `S` $= 2$. The first cell is empty, and the second cell is a $1 \times$ `D` cell array of array, where `D` is the number of time series. The array contains 0 and 1 to indicate which block components of the second model class has the same model parameters than the corresponding component of the first model class. A value of 1 indicates that the model parameters are constrained between the block components of the two model classes, 0 otherwise.

- `model.component.ic`: ifdefines the dependencies between the time-series. The field `ic` stores a $1 \times$ `D` cell array of $1 \times$ `D` $- 1$ array, where `D` is the number of time series. Each time-series depend on the time-series corresponding to the indexes given in the `D` arrays. If the array is empty, the time-series are modelled as independent.

6.4 Model parameters

This part of the configuration file aims at defining the model parameters properties. The model parameters properties are stored in the field named `model.param_properties` of the MATLAB structure `model`. The field `model.param_properties` stores `K` $\times$ 10 cell array, where `K` is the total number of model parameters.

- the column 1 must be a character vector that gives the name of the model parameters (e.g. `sigma_w`).
- the column 2 must be a character vector that gives the reference name of the block associated with the parameter (e.g `LL`, see Section 24.5).
- the column 3 must be a character vector that gives the index corresponding to the model class associated with the parameter (e.g either `1` or `2`) (see 24.3).
- the column 4 must be a character vector that gives the index corresponding to the observation associated with the parameter (e.g `3`).
- the column 5 must be a 1×2 array that gives the bound of the parameter(e.g `[NaN, NaN]`, `[0, Inf]`, `[0, 1]`). The bounds are used to transform (if necessary) model parameters from a bounded to an unbounded space during the optimization process (see Sections 13 and 24.4).
- the column 6 must be character vector that gives the type of the prior used during the optimization process (e.g either `N/A` or `normal`). `N/A` indicates that no prior is used (see Sections 13 and 24.4).
- the column 7 must be a real number that gives the mean of the prior when a prior of type `normal` is used, otherwise it must be set to `NaN` (see Sections 13 and 24.4).
- the column 8 must be a real number that gives the standard deviation of the prior when a prior of type `normal` is used, otherwise it must be set to `NaN` (see Sections 13 and 24.4).

- the column 9 must be a real number that gives the value of the model parameters.

- the column 10 must be an integer that gives the reference number of the model parameters. The model parameters sharing the same reference number are constrained to each other.

6.5 Initial states values

This part of the configuration file defines the initial (at time $t = 0$) states values. The mean and covariance initial hidden states values are stored in the `model.initX` and `model.initV` fields. The initial probability for the model class is stored in the field `model.initS`.

- `model.initX`: $1 \times$ `S` cell array of array, where `S` $= \{1, 2\}$ is the total number of model classes. Each array is a `L` $\times 1$ array of real number that stores the initial mean values associated with each hidden states components, where `L` is the total number of hidden states components associated with the model.

- `model.initV`: $1 \times$ `S` cell array of array, where `S` $= \{1, 2\}$ is the total number of model classes. Each array is a `L` $\times$ `L` array of real number that stores the initial variance and covariances values associated with each hidden states components.

- `model.initS`: $1 \times$ `S` cell array of array, where `S` $= \{1, 2\}$ is the total number of model classes. Each array is a 1×1 array of real number that gives the initial probability for the model class.

6.6 Options

This part of the configuration file defines the options that control different aspect of the software regarding the data pre-processing, the optimization, the state estimation, and the aspect of the graphical output. The options are stored in the field named options of the MATLAB structure `misc`.

- options for the data pre-processing

  - `misc.options.NaNThreshold`: real number that gives, in percent, the amount of missing data allowed at each time slice. Default: `100`.

- `misc.options.Tolerance`: real number that gives the gives the duration (in number of days) after which two timestamps are not considered equal. Default: 10^{-6}.

- Options for the model parameters estimation

  - `misc.options.trainingPeriod`: 1×2 array of real number that defines the training period, given in number of days since the first timestamp. Default: `[1 Inf]`.
  - `misc.options.isParallel`: logical that triggers or not the parallel computation for approximating the gradient in the optimization procedure. Note that parallel computation require the MATLAB *Parallel Computing Toolbox*. Default: `true`.
  - `misc.options.maxIterations`: integer that gives the maximum number of iterations for the optimization procedure. Default: `100`.
  - `misc.options.maxTime`: real number that gives, in minutes, the maximum amount of time to spend for the optimization procedure. Default: `60`.
  - `misc.options.isMAP`: logical that triggers or not the Maximum A Posteriori (MAP) estimation of the model parameters during the optimization procedure. MAP estimation includes prior information about the model parameters. Default: `false`.
  - `misc.options.isPredCap`: logical. if `isPredCap=true`, the Prediction Capacity (i.e. the log-likelihood over a test dataset) is used to drive the optimization process, otherwise the log-likelihood over the full dataset is used. For Stochastic Gradient only. Default: `false`.
  - `misc.options.isLaplaceApprox`: logical. if `isLaplaceApprox=true` the full Laplace approximation around the optimized model parameters values is computed. Default: `false`.
  - `misc.options.NRTerminationTolerance`: real. Termination tolerance for the Newton-Raphson algorithm. Default: 10^{-7}.
  - `misc.options.NRLevelsLambdaRef`: integer. Control the number of trial loop for a parameter being optimized for Newton-Raphson algorithm. Default: 4.
  - `misc.options.isMute`: logical. is `isMute=true`, no message are output during the optimization procedure. Default: `false`.

- Options for the estimation

  - `misc.options.MaxSizeEstimation`: real number that gives the maximum size, in Mb, for which the hidden states estimations are saved in the `PROJ_` file at the end of the analysis. Default: `100`.
  - `misc.options.MethodStateEstimation`: vector of character. It must be either `kalman` or `UD`. it gives the method used for the estimation of the hidden states. Default: `kalman`.
  - `misc.options.DataPercent`: real number. it gives in percent the amount of data, starting at $t = 1$ used for the estimation of the initial hidden states. Default: `100`.

- Options for the synthetic data creation

  - `misc.options.Seed`: integer. It controls the random number generation used to create synthetic data. Synthetic data created with the same seed are identical (useful to duplicate results). If `misc.options.Seed=[]`, the seed is based on current time and therefore, a different sequence of random number is generated at each run. Default: `12345`.

- Options for the graphical outputs

  - `misc.options.FigurePosition`: 1×4 array of real number that gives the location and size of the drawable area, specified as a vector of the form [left bottom width height] in the current units of MATLAB. Default: `[100, 100, 1300, 270]`
  - `misc.options.isSecondaryPlot`: logical. if `isSecondaryPlot=true`, a closeup over two weeks is plotted at the right of each figure. Default: `false`.
  - `misc.options.Subsample`: integer that controls the number of points to plot in the figure. The number of points to plot is divided by a factor given by the values of `misc.options.Subsample`. Default: `1`.
  - `misc.options.Linewidth`: real number that controls the width of the line plotted in the figure. Default: `1`.
  - `misc.options.ndivx`: integer that controls the number of labels for abscissa x-axis in each figure. Default: `4`.

- `misc.options.ndivy`: integer that controls the number of labels for ordinate y-axis in each figure. Default: `3`.
- `misc.options.Xaxis_lag`: real number that gives in number of days the amount of time the x-axis is shifted on each figure. Default: `0`.
- `misc.options.isExportTEX`: logical. if `isExportTEX=true`, the figure are exported in LaTeX format. Default: `false`.
- `misc.options.isExportPNG`: logical. if `isExportPNG=true`, the figure are exported in PNG format. Default: `false`.
- `misc.options.isExportPDF`: logical. if `isExportPDF=true`, the figure are exported in PDF format. Default: `false`.

7 OpenBDLM main menu

Once a project is loaded, the OpenBDLM main menu (see Listing 3) displays the list of actions which can be done. The OpenBDLM main menu appears each time a selected action is done; until the user types `Q` to save the project and quit the program.

- option `1` : estimates the model parameters
- option `2` : estimates the initial hidden states
- option `3` : estimate the hidden states
- option `11` : display the current model parameters values on the MATLAB command line window and allows the user to modify the model parameters values
- option `12` : display the initial hidden states values on the MATLAB command line window and allows the user to modify the initial hidden states values
- option `13` : display information about the current training period on the MATLAB command line window and allows the user to modify the training period
- option `14` : plot figures on the screen about the data in memory, the data availability, and the hidden states estimation results
- option `15` : display the content of the model matrices on the MATLAB command line window at a specific time

- option `16` : create synthetic data from the current information in memory
- option `17` : export data and results as well as the figures in PNG, PDF and LaTeX format
- option `18` : display the current options

```
_____________________________________________
/    Choose from
_____________________________________________

     1  ->  Learn model parameters values
     2  ->  Estimate initial hidden states values
     3  ->  Estimate hidden states values

     11 ->  Display and modify current model parameter values
     12 ->  Display and modify current initial hidden states values
     13 ->  Display and modify current training period
     14 ->  Plots
     15 ->  Display model matrices
     16 ->  Create synthetic data
     17 ->  Export
     18 ->  Display current options in configuration file format

     Type Q to Save and Quit

     choice >>
```

Listing 3: OpenBLDM main menu

8 OpenBDLM workflow

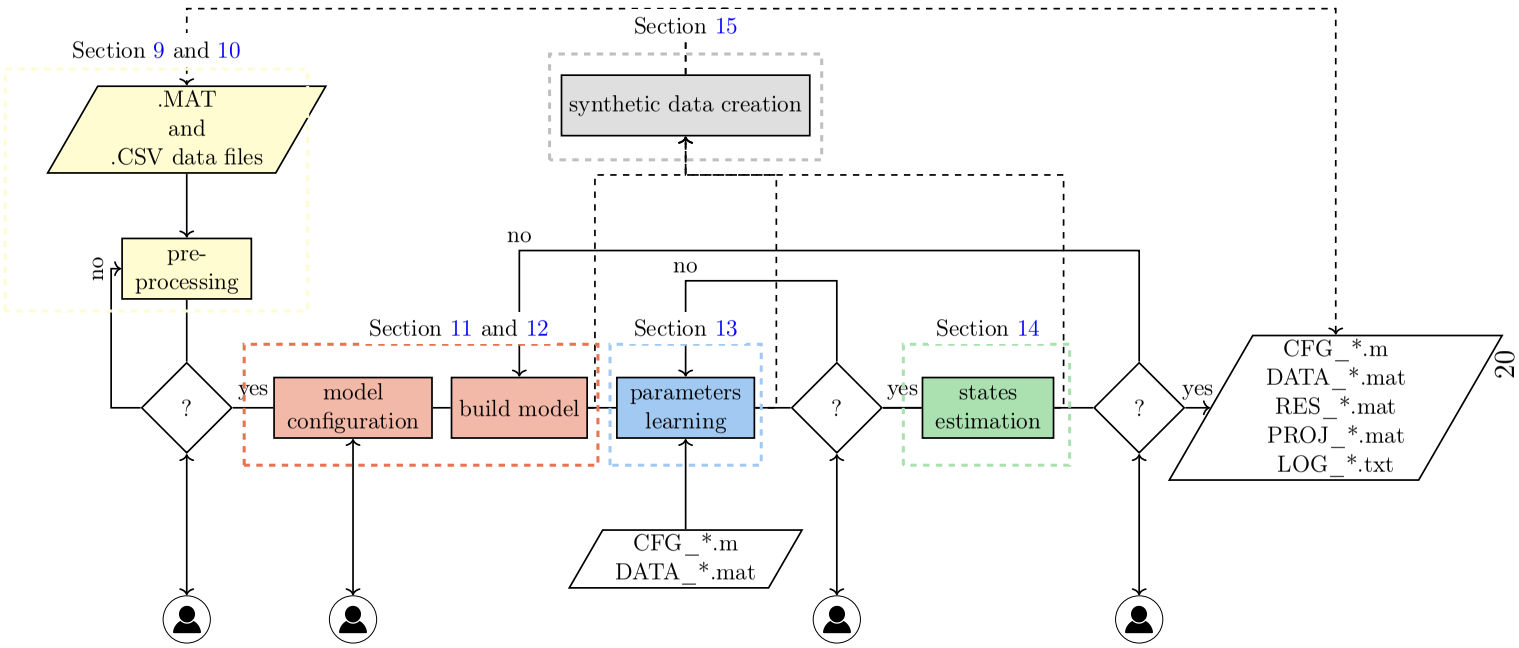


Figure 1: OpenBDLM workflow

9 Data loading

9.1 Input data format

OpenBDLM supports two types of input data format.

9.1.1 Comma Separated Values files

One Comma Separated Values (.csv) data file must be provided for each time series. The file must contain two columns that are organized as shown in Listing 4. The first line of the file is the header. In the header, the first

```
'name'          ,   '2000-01-01-22-00-00'
737422          ,   0.40
737423.5        ,   0.21
737424          ,   0.548
7374245.25      ,   NaN
7374246         ,   0.57
```

Listing 4: CSV data file example

field must contain the label of the time-series given as a quoted delimited string, as 'name'. The second field is the date of the first timestamp given as a quoted delimited string, formatted as 'YYYY-DD-MM-HH-MM-SS'. For the remaining lines, the first field is the date given as a serial date number in number of days, given as a real number, and the second field is the magnitude of the physical quantity measured, given as a real number. The missing data must be indicated as `NaN` number. The .csv files must be stored in the "OpenBDLM-master/data/csv" subfolder.

9.1.2 MATLAB .MAT files

The MATLAB binary .MAT file must contain three MATLAB variables called `labels`, `timestamps`, and `values`.

- `labels`: $1 \times$ `D` cell array containing the reference name associated with each time-series, where `D` is the number of time series.
- `timestamps`: `N` $\times 1$ array containing the timestamps given as serial date number from January 0, 0000, where `N` is the number of data samples.
- `values`: `N` $\times$ `D` array containing the data amplitude values.

MATLAB binary .mat files must be stored in the "OpenBDLM-master/data/mat" subfolder. Note that MATLAB binary .MAT files can be used to load at once several time-series which share the same timestep vector (i.e. synchronous time-series, see Section 10 for details.)

9.2 Data loading functions

The data loading workflow is presented Figure 2. The list of OpenBDLM functions used for data loading is:

- Control script to load data

```
[data,misc,dataFilename]=DataLoader(misc)
```

- Loads data

```
[dataOrig,misc]=loadData(misc)
```

- Reads data from multiple data files

```
[dataOrig,misc]=readMultipleDataFiles(misc)
```

- Reads a single CSV file

```
[dat,label]=readSingleCSVFile(FileToRead,varargin)
```

- Reads a single MAT file

```
[dat,label]=readSingleMATFile(FileToRead,varargin)
```

- Saves data in a DATA_ MATLAB MAT file

```
[misc,dataFilename]= saveDataBinary(data,misc,varargin)
```

- Saves data in separate CSV files

```
[misc]= saveDataCSV(data,misc,varargin)
```

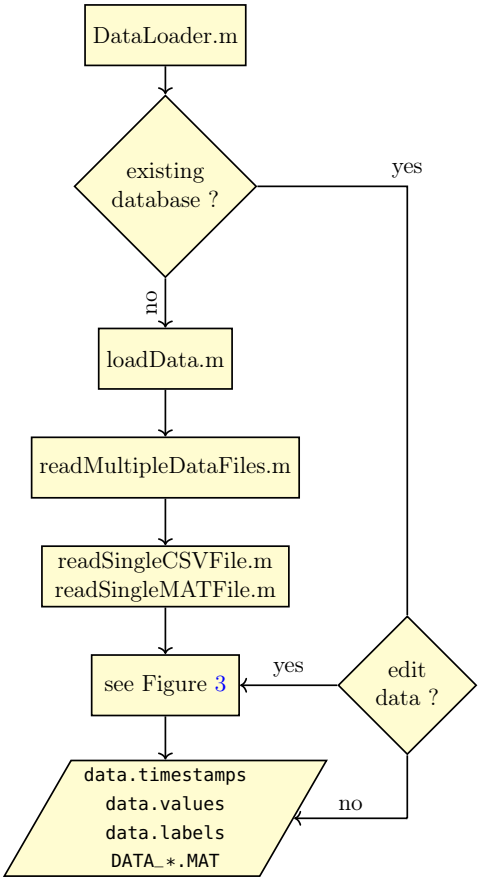


Figure 2: Data loading workflow

10 Data editing and pre-processing

Data editing prepares the data for analysis. In most cases, the set of available time-series is heterogeneous, in the sense that each time-series does not originate from the same system of acquisition. Therefore, the raw data do not usually share the same time vectors. This is an issue because BDLM techniques are not capable to analyze asynchronous time-series. Therefore, when multiple time series are processed simultaneously, the main objective of data pre-processing is to synchronize the time-series. Data editing includes time series selection, data analysis time period selection, missing data removal, and data resampling. The time synchronization is performed automatically through the data editing process.

```
— Data editing and preprocessing. Choose from:

    1  —>  Select time series
    2  —>  Select data analysis time period
    3  —>  Remove missing data
    4  —>  Resample
    5  —>  Change synchronization options

    6  —>  Reset changes
    7  —>  Save changes and continue analysis

    choice >>
```

Listing 5: OpenBDLM data editing menu

10.1 Selection of time-series

The selection of time-series allows selecting only a subset of the time series in memory. The time series are automatically synchronized as time-series are added to (or removed from) the dataset.

10.2 Selection of data analysis time period

The selection of the period of analysis allows selecting only the data between two dates, given as 'YYYY-DD-MM' format. If the second requested date exceeds the date corresponding to the last timestamp of the original dataset, padding with `NaN` values are performed. The timestep for the `NaN` padding must be provided by the user.

10.3 Removing missing data

It is possible to control the maximum amount of `NaN` missing data allowed at each time slice. The maximum amount of `NaN` allowed at each time slice is given in percent with respect to the total number of time-series. By default, the maximum amount of missing data is given by the value in `misc.options.NaNThreshold`.

10.4 Data resampling

Data resampling changes the sampling rate of the time-series according to a given timestep provided by the user. If the requested timestep is higher than the original data timestep, `NaN` values are added. Conversely, if the requested

timestep is lower than the original data timestep, OpenBDLM averages the data amplitude values within non-overlapping fixed time windows, each having the duration of the requested timestep. The first time window starts at the first timestamp, and the new timestamps are assigned at the times corresponding to the middle of each time window.

10.5 Time synchronization options

By default, the time synchronization in OpenBDLM is done by adding `NaN` values. The time synchronization is controlled by the `NaNThreshold` and `tolerance` variables. `NaNThreshold` is given in percent with respect to the total number of time-series. The variable `tolerance` gives the duration (in number of days) after which two timestamps are not considered equal. The default values for `NaNThreshold` and `tolerance` are given by `misc.options.NaNThreshold` and `misc.options.tolerance`.

10.6 Data editing functions

The data editing workflow is presented Figure 3. The list of OpenBDLM functions used for data editing is:

- Control script to edit dataset (selection, resampling, etc..)

```
[data,misc,dataFilename]=editData(data,misc,varargin)
```

- Requests the user to select some time series

```
[data,misc]=chooseTimeSeries(data,misc)
```

- Creates a single time vector from a set of time series

```
[data,misc]=mergeTimeStampVectors(dataOrig,misc,varargin)
```

- Resamples dataset according to a given timestep

```
[data_resample,misc]=resampleData(data,misc,varargin)
```

- Selects data between two dates

```
[data,misc]=selectTimePeriod(data,misc)
```

- Computes timestep vector from timestamps vector

```
[timesteps]=computeTimeSteps(timestamps)
```

- Display on screen the content of the data in memory

```
displayData(data,misc)
```

- Display the list of DATA_*.mat files

```
[FileInfo]= displayDataBinary(misc,varargin)
```

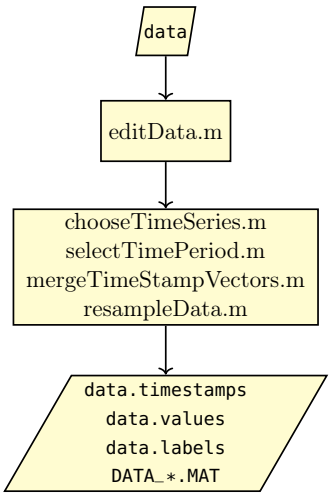


Figure 3: Data editing and pre-processing workflow

11 Model configuration

After loading and pre-processing the data, the next step of the analysis is the model configuration. The model configuration includes, (i) defining the number of model class, (ii) defining the dependencies between the time series in case of multiple time-series analysis, (iii) defining the block components which are assigned to each time series and model class, (iv) defining possible constrain between model parameters. The MATLAB structure named `model.component` stores all the information about the model configuration. The structure `model.component` has three fields named `model.component.ic`, `model.component.block`, and `model.component.const`.

11.1 Dependencies between time-series

The field `model.component.ic` stores the information between time-series dependencies. `model.component.ic` stores a $1 \times \mathtt{D}$ cell array of $1 \times \mathtt{D} - 1$ array, where `D` is the number of time series. Each time-series depend on the time-series corresponding to the indexes given in the `D` arrays. If the array is empty, the time-series is independent.

11.2 Block components

OpenBDLM supports four types of block components: (1) baseline, (2) periodic, (4) periodic kernel regression, and (3) autoregressive (1) The baseline component models the local mean of the time series. There are three types of baseline proposed in the software: (i) level only model, (ii) trend model, (iii) acceleration model. (2) The periodic component models harmonic periodic phenomena. (3) The periodic kernel regression models non-harmonic periodic pattern. (4) The autoregressive component models the time dependent model error. The field `block` stores a $1 \times \mathtt{S}$ cell array, where $\mathtt{S} = \{1, 2\}$ is the number of model classes. Each cell array is a $1 \times \mathtt{D}$ cell array of array, where `D` is the number of time series. Each block component is associated with a reference number:

- 11: Local level
- 12: Local level plus local trend
- 13: Local level plus local trend plus local acceleration
- 21: Local level compatible with local trend
- 22: Local level compatible with local acceleration
- 23: Local trend compatible with local acceleration
- 31: Periodic
- 41: First-order autoregressive
- 51: Kernel regression

The number of hidden states associated with each block component may can be different. Each block component can be replicated, each having its own set of model parameters. For instance, two periodic components with periods of 365 days and 1 day can be used to model seasonal and daily variations, respectively.

11.3 Parameter constrains

The field `model.component.const` stores a $1 \times$ `S` cell array, where `S` $= \{1, 2\}$ is the total number of model classes. It is defined only if `S` $= 2$. The first cell is empty, and the second cell is a $1 \times$ `D` cell array of array, where `D` is the number of time series. The array contains 0 and 1 to indicate which block components of the second model class has the same model parameters than the corresponding component of the first model class. A value of 1 indicates that the model parameters are constrained between the block components of the two model classes, 0 otherwise.

11.4 Number of model class

OpenBDLM is capable to detect changes of behavior due to changes in the dynamics in the baseline of the time-series. Therefore, the software handles model switching between the three types of baseline dynamics, that is, local level, local trend, and local acceleration models. OpenBDLM supports a maximum of two model dynamics, the software supports six types of model switch: (1) from local level model to local trend model (and reverse), (2) from local level model to acceleration model (and reverse), (3) from local trend to acceleration model (and reverse).

11.5 Model configuration functions

The model configuration workflow is presented Figure 4. The list of OpenBDLM functions used for model configuration is:

- Controls script for model configuration

```
[data,model,estimation,misc]=ModelConfiguration(data,model,
    estimation,misc)
```

- Models configuration for real data

```
[data,model,estimation,misc]=configureModelForDataReal(data,model
    ,estimation,misc)
```

- Models configuration for synthetic data (for synthetic data creation only)

```
[data,model,estimation,misc]=configureModelForDataSimulation(data
    ,model,estimation,misc)
```

- Requests user's input to configure the model

  ```
  [model,misc]=defineModel(data,misc)
  ```

- Requests user's input to define time series labels/reference name (for synthetic data creation only)

  ```
  [data,misc]=defineDataLabels(data,misc)
  ```

- Requests user's input to define data timestamps (for synthetic data creation only)

  ```
  [data,misc]=defineTimestamps(data,misc)
  ```

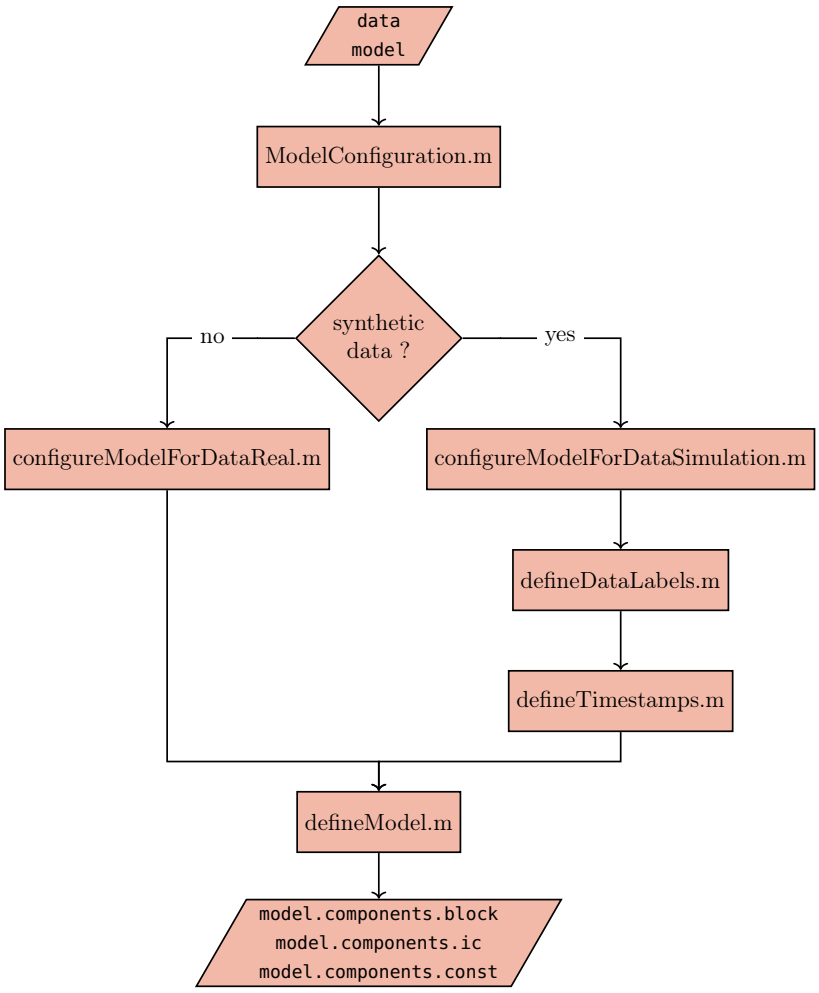


Figure 4: Model configuration workflow

12 Model construction

Once the model has been configured, the next step is to build the model. The aim of the model building, or model construction, is to build the full model matrices A, C, Q, R by assembling the sub-matrices associated with each block component and each time-series. The corresponding values for the model parameters must also be considered. Because the model matrices are often time-dependent, OpenBDLM creates a function for each matrix, that depends on the time t, and model parameters $\boldsymbol{\theta}$.

12.1 Model construction functions

The model construction workflow is presented Figure 5. The OpenBDLM function used for model construction is:

- Builds the model

```
[model, misc]=buildModel(data, model, misc)
```

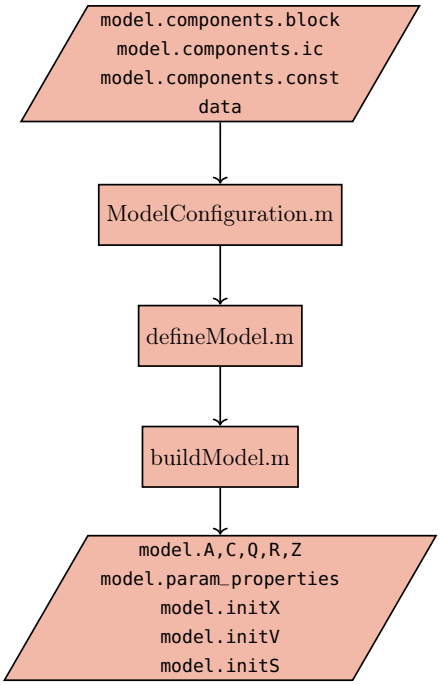


Figure 5: Model construction workflow

13 Model parameters estimation

OpenBDLM provides default values of the parameters from heuristic knowledge or from statistics on the data. Those values are usually poor guesses which have to be refined using an optimization algorithm that estimates the model parameters from the data.

From the main OpenBLDM menu, type `1` to access the model parameter estimation menu (see Listing 6). The optimization algorithm is either the Newton-Raphson (choice `1`) or the Stochastic Gradient Descent (choice `2`) algorithms (see Section 24.4).
By default, OpenBDLM uses all the data as the training set (`misc.options.Trainingperiod=[1 Inf]`. Moreover, OpenBDLM transform the model parameters to perform the optimization in an unbounded model parameter space (see Section 24.4.6). Undefined bounds (`[NaN, NaN]`) for a parameter means that the parameter is assumed to be known and it does not require to be learned from the data. By default, OpenBLDM assumes that all parameters are unknown, except for the period of the periodic component and the process noise standard deviation associated with the baseline and the periodic component. Those model parameters have values fixed to 0.
The model parameter can be learned by maximizing either the likelihood (Maximum Likelihood Estimation, MLE), or the posterior function (Maximum A Posteriori estimation, MAP) (see Section 24.4). OpenBDLM supports gaussian prior only. Note that MAP requires a valid prior for each unknown model parameters. The default option is MLE (see `misc.options.isMAP=false`).

```
_____________________________________________
/ Learn model parameters
_____________________________________________


     1 ->  Newton-Raphson
     2 ->  Stochastic Gradient Ascent

     Type R to return to the previous menu

     choice >>
```

Listing 6: OpenBDLM model parameter estimation menu

Model parameter estimation computes point estimate of the model parameters. It may also provide confidence intervals around the point estimate using the Laplace approximation (see `misc.options.isLaplaceApproximation = true`). Note that computing the Laplace approximation can significantly increase the computation time when the number of model parameters is large.
Note also that Newton-Raphson and Stochastic Gradient Descent techniques are sensitive to the initial model parameters values. The algorithm can reach

a local maximum, instead of the global maximum. It is advised to run the optimizations several times with different starting model parameters values in order to check that the proposed solution is stable.
If `misc.options.isMute = false`, some messages output on the MATLAB command line window allow to monitor the optimization process (see Listing 7). At each iteration, the quantity displayed are: the current value of the target function, the name as well as the current value of the unknown model parameter(s) being optimized, the change in model parameters, the change in the target function, and the convergence status (`1` if the model parameter converged according to the convergence criteria, `0` otherwise). The optimization stops when all the parameters converged, or if the maximum number of iterations (see `misc.options.maxIterations`) or the maximum time (see `misc.options.maxTime`) is reached. The values of the parameters are saved in the `model` MATLAB structure.

13.1 Model parameter estimation functions

- Pilote function for optimization

```
[data,model,estimation,misc]=piloteOptimization(data,model,
    estimation,misc)
```

- Estimates model parameter using Newton-Raphson algorithm

```
[optim,model]=NewtonRaphson(data,model,misc)
```

- Estimates model parameter using Stochastic Gradient Descent algorithm

```
[optim,model] = SGD(data,model,misc,varargin)
```

- Reads model parameter properties

```
[arrayOut]=readParameterProperties(cellIn,Position)
```

- Writes model parameter properties

```
[cellOut]=writeParameterProperties(cellIn,arrayIn,Position)
```

- Approximates the target function, as well as the first and second derivative of the logarithm of the target function with respect to parameter values

```
[logpdf,Glogpdf,Hlogpdf, delta_grad] = logPosteriorPE(data,model,
    misc,varargin)
```

```
    \Start Newton-Raphson max. algorithm (finite difference method)

      Training period:                              1-Inf [days]
      Maximal number of iteration:                  100
      Total time limit for calibration :            60 [min]
      Convergence criterion:                        1e-07*LL
      Nb. of search levels for \lambda:             4*2

           Initial LL: 36626.8381
                       AR|M1|1         AR|M1|1         |M1|1
      parameter names: \phi            \sigma_w        \sigma_v
       initial values: +7.50e-01       +1.74e-02       +8.70e-03
-------------------------
    Loop #1 : |M1|1 | \sigma_v
       delta_param: -0.0040755
    log-likelihood : 36994.6374
    param change   : 0.0087002 -> 0.0046247

                     AR|M1|1         AR|M1|1         |M1|1
   parameter names: \phi             \sigma_w        \sigma_v
    current values: +7.50e-01       +1.74e-02       +4.62e-03
  current f.o. std: +0.00e+00       +0.00e+00       +1.93e-04
      previous dLL: +1.00e+06       +1.00e+06       +3.68e+02
         converged: +0.00e+00       +0.00e+00       +0.00e+00
-------------------------
    Loop #2 : AR|M1|1 | \sigma_w
       delta_param: 0.0046034
    log-likelihood : 40998.3934
    param change   : 0.0174 -> 0.022003

                     AR|M1|1         AR|M1|1         |M1|1
   parameter names: \phi             \sigma_w        \sigma_v
    current values: +7.50e-01       +2.20e-02       +4.62e-03
  current f.o. std: +0.00e+00       +5.26e-05       +1.93e-04
      previous dLL: +1.00e+06       +4.00e+03       +3.68e+02
         converged: +0.00e+00       +0.00e+00       +0.00e+00
-------------------------
```

Listing 7: OpenBLDM output on MATLAB command window when running Newton-Raphson algorithm.

- Computes the gradient and hessian of the gaussian prior distribution of each model parameter

```
[logprior,Glogprior,Hlogprior]= logPriorDistr(P,Mu,Sigma,varargin
    )
```

- Computes numerical hessian H of a function

```
H=numerical_hessian(x,fX,varargin)
```

- Defines transformation functions and their derivatives according to provided bounds for the model parameters

```
[fct_TR,fct_inv_TR,grad_TR2OR,hessian_TR2OR]=
    parameter_transformation_fct(model,param_idx_loop)
```

- Performs Switching Kalman filter on time series

```
[x,V,VV,S,loglik,U,D]=SwitchingKalmanFilter(data,model,misc)
```

- Computes the first and second derivative of the logarithm of the likelihood function with respect to parameter values

```
[grad,hessian,fail_gradHess,delta_grad] = gradHess(data, model,
    misc,pTR,pOR,log_lik_0,grad_TR2OR,delta_grad,param_idx_loop)
```

- Computes the optimal parameter step size for the approximation of the numerical derivative of the likelihood and compute the terms required to approximate the numerical derivative using finite differene method

```
[delta_grad,fail_delta_grad,log_lik_1,log_lik_2] =
    StepSizeOptimization(model,data,misc,pOR,log_lik_0,delta_grad
    ,param_idx_loop)
```

- Splits the full dataset into train and test dataset (for Stochastic Gradient Descent only)

```
[data_train,data_valid] = dataSplit(data,idxTrain,alpha_split,
    varargin)
```

- Computes the metric function (for Stochastic Gradient Descent only)

```
[metricVL,idxMaxM,logpdf_test,logpdf_train] = metricFct(
    data_train,data_test,model,misc,parameterSearch,
    parameterSearchTR)
```

- paramGrid (for Stochastic Gradient Descent only)

```
[xM,momentumM,RMSpropM,gradM,learningRateM, mmtHessM, hessM]=
    paramGrid(x,momentum,RMSprop,grad,learningRate,mmtHess,hess)
```

- ADAM optimizer (for Stochastic Gradient Descent only)

```
[xsearch,xsearchTR,momentumTR,RMSpropTR] = ADAM(xsearchTRprev,
    momentumTRprev,RMSpropTRprev,grad,step,beta_1,beta_2,epsilon,
    Niter,fctInvTR)
```

- MMT optimizer (for Stochastic Gradient Descent only)

```
[xsearch,xsearchTR,momentumTR] = MMT(xsearchTRprev,momentumTRprev
    ,grad,step,beta,fctInvTR)
```

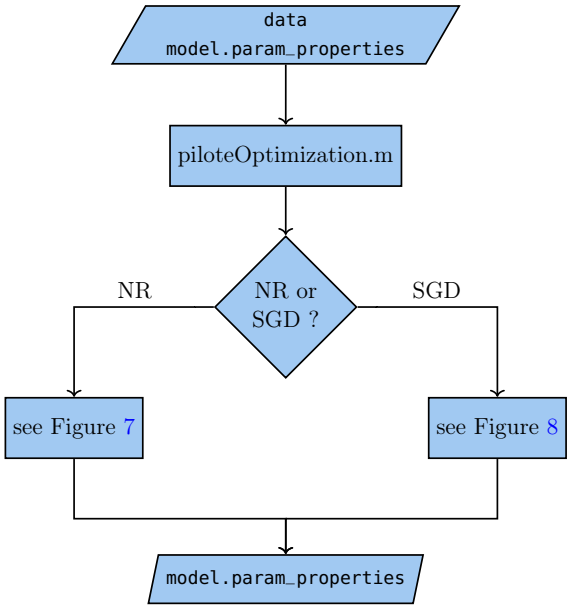


Figure 6: Model parameter estimation workflow

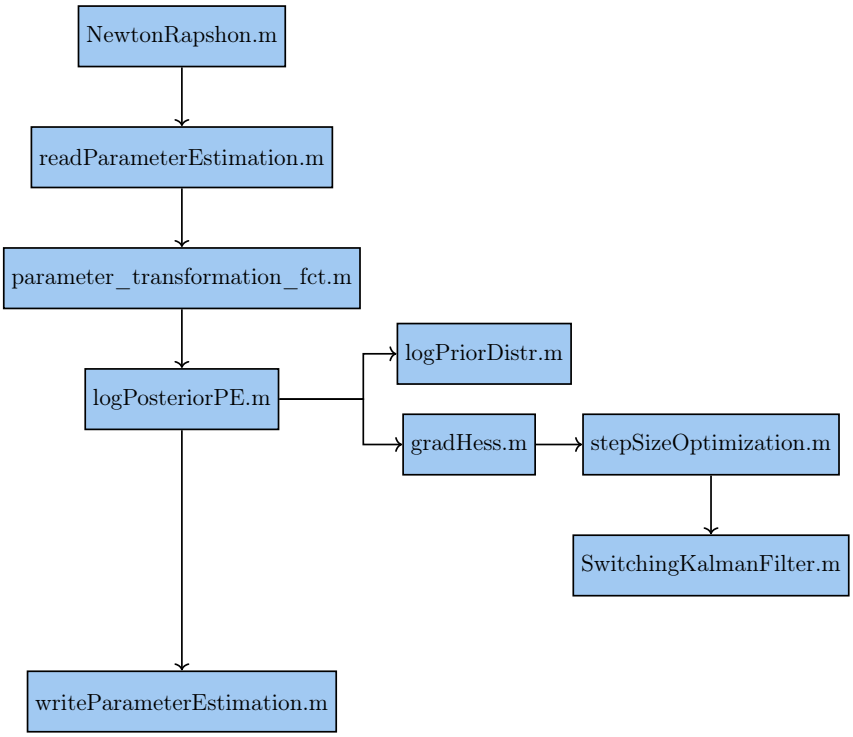


Figure 7: Model parameter estimation Newton-Raphson workflow

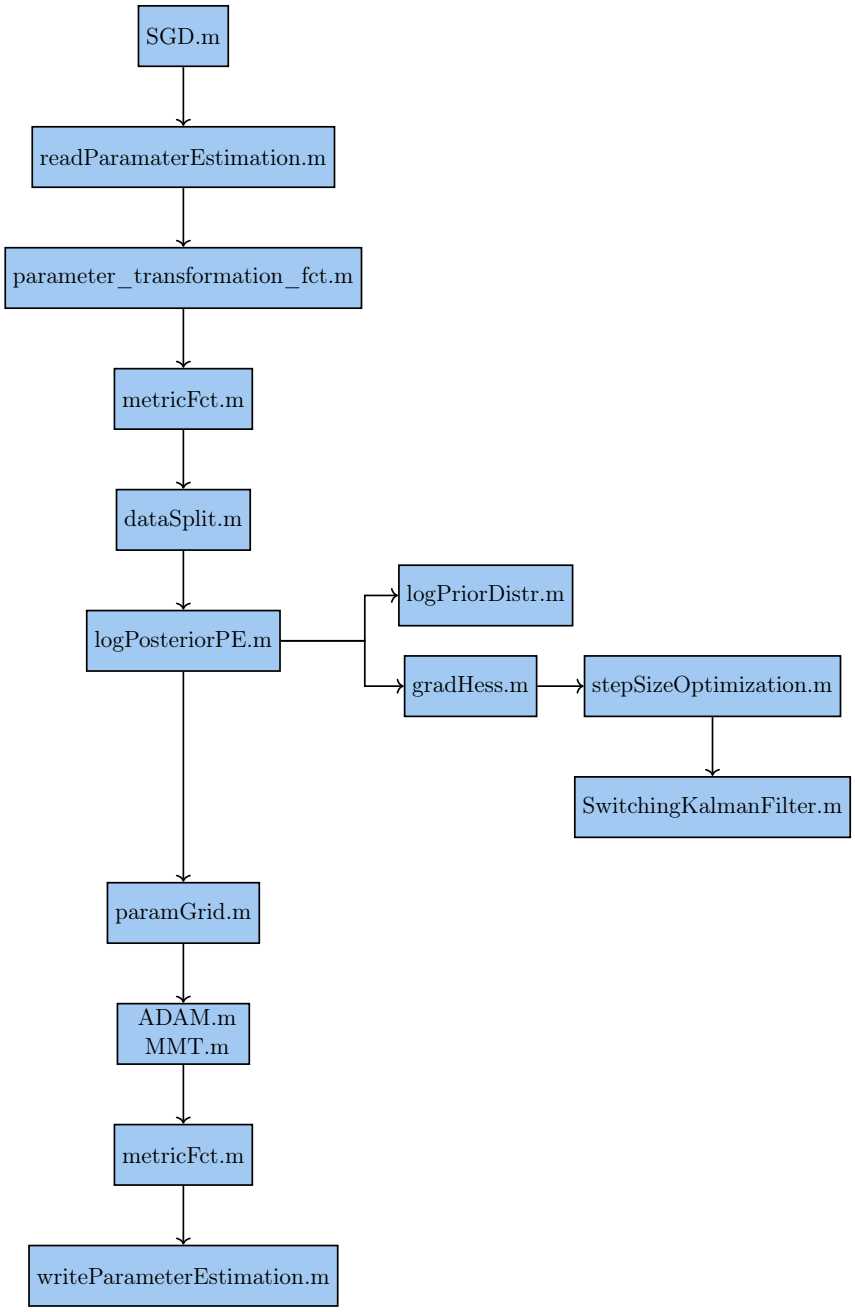


Figure 8: Model parameter estimation Stochastic gradient descent workflow

14 Hidden states estimation

The estimation of the hidden states is the core of OpenBLDM. The hidden state estimation is performed recursively using the Kalman filter/smoother or the UD filter/smoother (see Section 24.2). Once a project is loaded, typing `3` opens the hidden state estimation menu (see Listing 8).

```
_____________________________________________________
/    State estimation
_____________________________________________________

     1 ->  Filter
     2 ->  Smoother

     Type R to return to the previous menu

     choice >>
```

Listing 8: OpenBDLM hidden states estimation menu

Type `1` runs the Kalman or UD filter, and typing `2` runs the Kalman or UD smoother. The Kalman is the default state estimation method (see `misc.options.MethodStateEstimation`), but UD computations are more stable in particular in the case of missing data or in case of a sudden increase of the time step.

14.1 Hidden state estimation functions

The hidden state estimation workflow is presented Figure 9. The OpenBLDM functions used for hidden state estimation are:

- Pilote function for hidden state estimation

```
[data,model,estimation,misc]=piloteStateEstimation(data,model,
    estimation,misc)
```

- Runs state estimation

```
[estimation]=StateEstimation(data,model,misc,varargin)
```

- Runs switching Kalman filter for all time

```
[x,V,VV,S,loglik,U,D]=SwitchingKalmanFilter(data,model,misc)
```

- Performs Rauch-Tung-Striebel switching smoother for all time

```
[x,V,VV,S,x_prior_smoothed,V_prior_smoothed,VV_prior_smoothed,
    S_prior_smoothed]=RTS_SwitchingKalmanSmoother(data,model,
    estimation)
```

- Performs one step of the Kalman filter

```
[xnew,Vnew,VVnew,loglik]=KalmanFilter(A,C,Q,R,y,x,V,varargin)
```

- Performs one step of the UD filter

```
[xnew,Vnew,VVnew,U_post,D_post,loglik]=UDFilter(A,C,Q,R,y,x,V,
    U_post,D_post)
```

- Computes UD decomposition

```
[U,D] = myUD(mat,varargin)
```

14.2 Estimation of the initial hidden states

OpenBDLM computes default values for the initial (at $t = 0$) mean (`model.initX`) and covariance (`model.initV`) value, as well as the initial model probabilities (`model.initS` . Those initial values are usually poor guesses, which may be refined using Kalman smoothing up to $t = 0$. The percent of data used to estimate initial hidden states using Kalman smoothing is controlled by the value provided in `misc.options.DataPercent`. From the OpenBDLM main menu, type `2` in the MATLAB command window to estimate the initial hidden states using Kalman smoothing. The hidden state estimation workflow is presented Figure 10. The OpenBLDM functions used for initial hidden state estimation are:

- Pilote function for initial state estimation

```
[data,model,estimation,misc]=piloteInitialStateEstimation(data,
    model,estimation,misc)
```

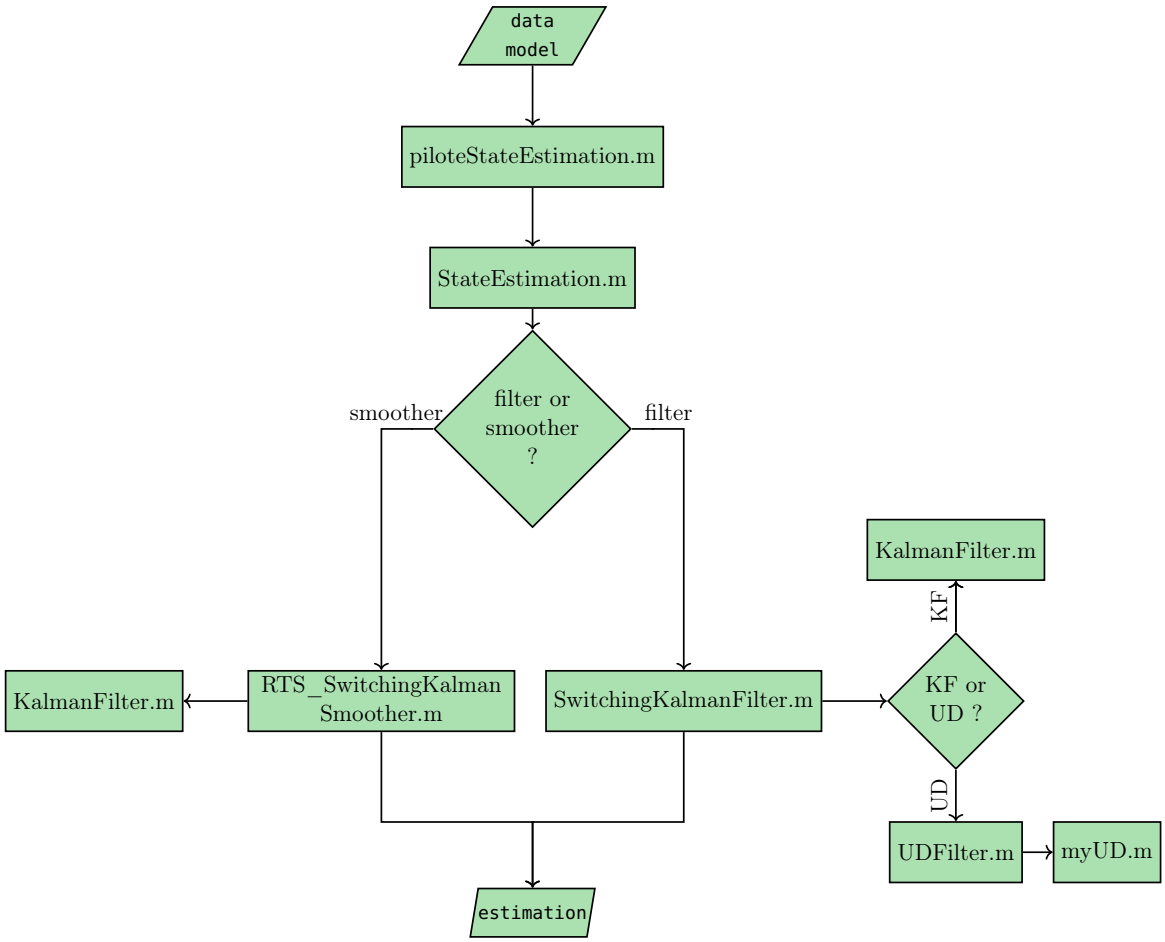


Figure 9: Hidden states estimation workflow

- Runs state estimation

```
[estimation]=StateEstimation(data,model,misc,varargin)
```

- Performs Rauch-Tung-Striebel switching smoother for all time

```
[x,V,VV,S,x_prior_smoothed,V_prior_smoothed,VV_prior_smoothed,
    S_prior_smoothed]=RTS_SwitchingKalmanSmoother(data,model,
    estimation)
```

- Performs one step of the Kalman filter

```
[xnew,Vnew,VVnew,loglik]=KalmanFilter(A,C,Q,R,y,x,V,varargin)
```

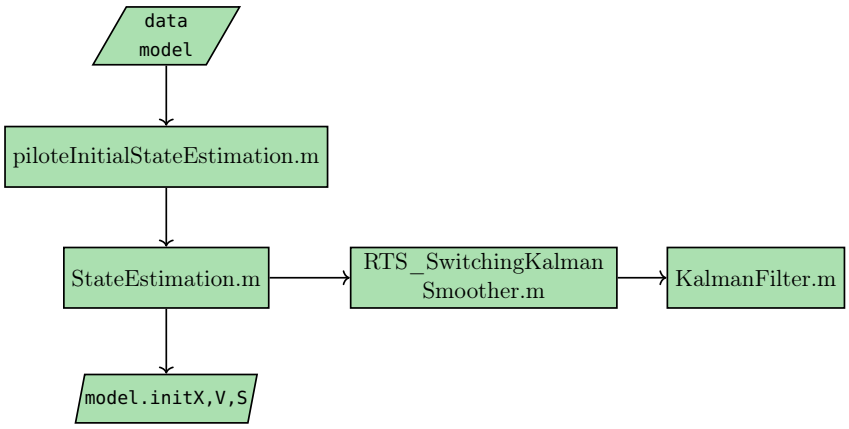


Figure 10: Initial hidden states estimation workflow

15 Creating synthetic data

The creation of synthetic data is possible using OpenBDLM. The analysis of synthetic data is useful for validation, test and debugging purposes because the true value of the hidden states and model parameters are known. OpenBDLM uses the transition model of the state-space modelling approach (see Section 24.1) to create realistic synthetic data. There are two ways for creating synthetic data using OpenBDLM:

- from the interactive tool
- from an existing project.

15.1 Synthetic data creation using the interactive tool

The creation of the synthetic data from the interactive tool (option `0` from the starting menu) enables to create synthetic data from scratch. OpenBDLM requests the user to provide the number of time series, and to define the time vector (starting time, end time, timestep). In the next step, the user has to define the time-series dependence (if applicable), to provide the number of model class, and to choose the block component for each time-series, as well as model constrains between model classes (if applicable). Default values for initial hidden states mean values and model parameters are automatically assigned for each block component. In the case of two model classes, the synthetic baseline will switch between the first and the

second model class according to the transition probability values (see Section 24.3). The amplitude of each synthetic anomaly (i.e. change of the local trend) is sampled randomly in a normal distribution of zero mean and standard deviation σ_w^{12} as defined in the switching process noise transition matrix. Alternatively, the user may choose to create *custom anomalies*. In such case, the beginning (in sample index), duration (in number of samples) and amplitude (in change of the local trend) of each anomaly is user's specified. The information about custom anomaly are stored in the field `custom_anomalies` of the structure variable `misc`:

- `misc.custom_anomalies.start_custom_anomalies`: this field stores a $1 \times$ `A` vector of integers, where `A` is the total number of synthetic anomaly. Each value indicates the sample index of the anomaly start.
- `misc.custom_anomalies.duration_custom_anomalies`: this field stores a $1 \times$ `A` vector of integers, where `A` is the total number of synthetic anomaly. Each value indicates the anomaly duration in number of samples.
- `misc.custom_anomalies.amplitude_custom_anomalies`: this field stores a $1 \times$ `A` vector of real number, where `A` is the total number of synthetic anomaly. Each value indicates the amplitude of the anomaly in change of the local trend.

The synthetic data are saved in `DATA_*.mat` and `*.csv` data files, and a `PROJ_*.mat` project file is created that stores the information about the model (structure variable `model`), and the true hidden states (see structure variable `estimation.ref`).

15.2 Synthetic data creation from an existing project

Once a project is loaded, it is possible to create synthetic data from it (option `16` from the main menu (see Listing 3). The synthetic data time vector will be the same as the time vector in memory, and missing data will be replicated. The model used to create the synthetic data will be the same as the model of current project, including current initial hidden states as well as model parameters values. The creation of synthetic data in this way is particularly useful to closely mimic real dataset. The synthetic data are saved in `DATA_new_*.mat` and `*.csv` data files, and a `PROJ_new_*.mat` new project file is created that stores the information about the model (structure variable `model`), and the true hidden states (see structure variable `estimation.ref`).

	$\boldsymbol{\theta}$	$\boldsymbol{\mu}_0$	$\mathrm{diag}(\boldsymbol{\Sigma}_0)$
LL	$\sigma_w^{\mathtt{LL}} = 0$	$[10]$	$[0.1^2]$
LT	$\sigma_w^{\mathtt{LT}} = 10^{-7}$	$[10, -0.1 \times 10^{-2}]$	$[0.1^2, 0.1^2]$
LA	$\sigma_w^{\mathtt{LA}} = 10^{-8}$	$[10, -0.1 \times 10^{-2}, -0.1 \times 10^{-5}]$	$[0.1^2, 0.1^2, 0.1^2]$
P	$p = [365.24, 1, 182.62]$, $\sigma_w^{\mathtt{P}} = 0$	$[10, 10]$	$[0.2^2, 0.2^2]$
KR	$p = [365.24]$, $\ell = 0.5$, $\sigma_{w,0}^{\mathtt{KR}} = \sigma_{w,1}^{\mathtt{KR}} = 0$	$[-0.97, 1.65, 1.73, -1.91, 0.23, 0.37, -2.89, -0.22, 0.73, -1.83]$	$[0.01^2, 0.01^2, 0.01^2, 0.01^2, 0.01^2, 0.01^2, 0.01^2, 0.01^2, 0.01^2, 0.01^2]$
AR	$\phi^{\mathtt{AR}} = 0.75$, $\sigma_w^{\mathtt{AR}} = 0.01$	$[0]$	$[0.1^2]$

Table 1: Default value of model parameters and initial hidden state $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ for synthetic data creation.

15.3 Synthetic data creation functions

The synthetic data creation workflow is presented Figure 11. The OpenBLDM functions used for synthetic data creation are:

- Pilote function for synthetic data creation

```
[data,model,estimation,misc]=piloteSimulateData(data,model,
    estimation,misc)
```

- Creates synthetic data

```
[data,model,estimation,misc]=SimulateData(data,model,misc,
    varargin)
```

- Create synthetic data from transition probabilities

```
[data, model, estimation, misc]=
    simulateDataFromTransitionProbabilities(data,model,misc)
```

- Create synthetic data from custom anomalies (for two model classes only)

```
[data,model,estimation,misc]=simulateDataFromCustomAnomalies(data
    ,model,misc)
```

- Models configuration for synthetic data (for synthetic data creation from interactive tool only)

```
[data,model,estimation,misc]=configureModelForDataSimulation(data
    ,model,estimation,misc)
```

- Requests user's input to define the number of synthetic time series to create (for synthetic data creation from interactive tool only)

```
  [data,misc]=defineDataLabels(data,misc)
```

- Requests user's input to define synthetic data time vector (for synthetic data creation from interactive tool only)

```
[data,misc]=defineTimestamps(data,misc)
```

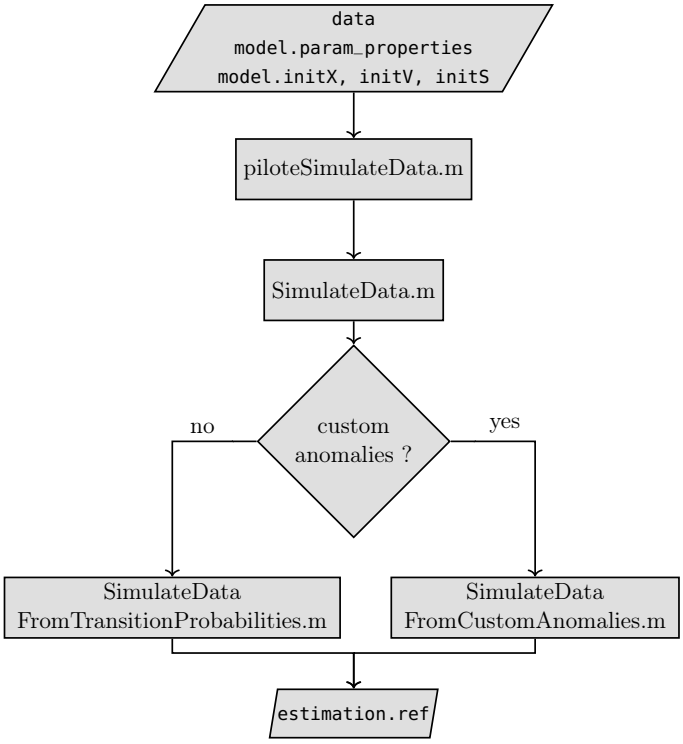


Figure 11: Synthetic data creation workflow

16 Visualizing results

The figures that popup on screen aim to represent the data, the data summary, and the hidden states results for each step of the analysis. Note that in contrast with the data plot, the data summary plot offers a way to visualize the amplitude, the timesteps and the data availability in a compact way. By default, a solid line connects two successive real-valued measurement, whatever the timestep. Missing data (NaN) are not represented in the plot of the observed data, thus resulting in gap for large period of time with missing data. The data availability in the data summary plot indicates each missing data with a red cross, thus making it useful to detect sparse missing data which are invisible in the data amplitude plots. The working period of the sensor is represented by a thick green line in the data availability plot. The plot appearance may be controlled from the dedicated `misc.options`.

Generating figures at any time The option `14` from the main menu allows generating the different type of figure at any time (see 9).

Saving figures It is not advised to save figures "manually" using the Matlab. It would most likely not save figures as seen on the screen. Instead, use the OpenBDLM export facilities described in the Section 17.4, or set the `misc.options.isExportTEX`, `misc.options.isExportPDF`, `misc.options.isExportPDF` to `true` to automatically save the figure in a specific format each time a figure is created [2]. The workflow for visualization is shown in Figure 12. The functions used to visualize the data and results are:

```
_____________________________
/    Plot
_____________________________

     1 ->  Plot data
     2 ->  Plot data summary
     3 ->  Plot hidden states

     Type R to return to the previous menu

     choice >>
```

Listing 9: OpenBDLM plot menu

- Pilote function to plot data and estimations

```
[misc] = pilotePlot(data,model,estimation,misc)
```

- Plot data amplitude values and data timestep

```
[FigureNames] = plotData(data,misc,varargin)
```

- Plot data amplitude, data time step, and data availability

```
plotDataSummary(data, misc, varargin)
```

- Plot hidden states, predicted data, and model probability

[2] Automatic figure saving is not recommanded because it is computationally expensive.

```
plotEstimations(data,model,estimation,misc,varargin)
```

- Plot true and estimated hidden states

```
[FigureNames] = plotHiddenStates(data,model,estimation,misc,
    varargin)
```

- Plot observed and predicted data

```
[FigureNames] = plotPredictedData(data,model,estimation,misc,
    varargin)
```

- Plot true and estimated model probability

```
[FigureNames] = plotModelProbability(data,model,estimation,misc
    ,varargin)
```

- Waterfall plot for kernel regression component

```
[FigureNames] = plotWaterfallKRegression(data,model,estimation,
    misc,varargin)
```

- Export the current figure in TeX file using matlab2tikz

```
exportPlot(FigureName,varargin)
```

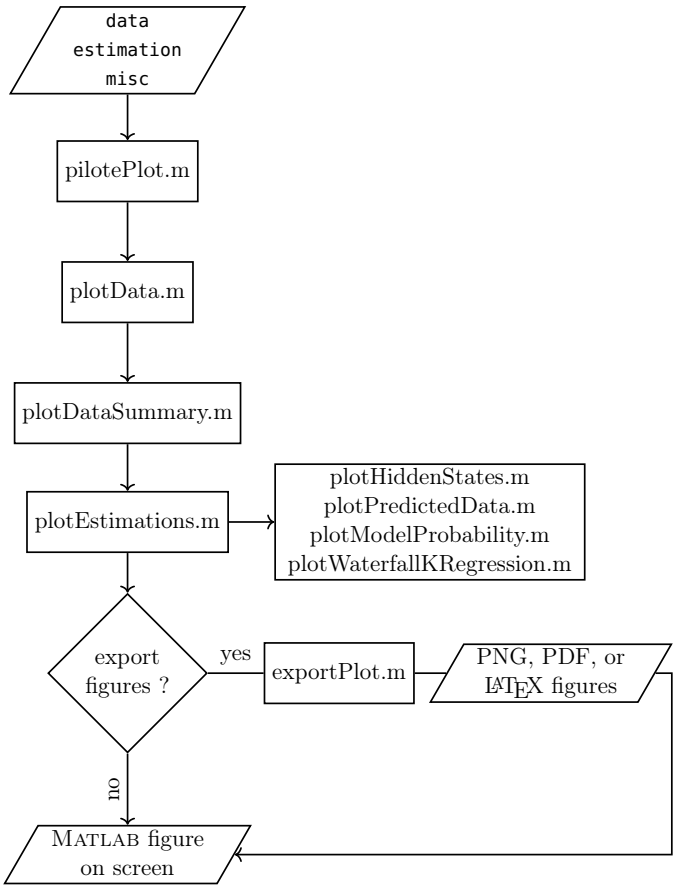


Figure 12: Visualization results workflow

17 Exploring results

Exploring results (raw data and figures), is essential for interpretation, validation and reporting during and after the analysis. During the analysis, the MATLAB binary RES_*.mat, PROJ_*.mat and DATA_*.mat files are automatically created for this purpose. Moreover, OpenBDLM enables exporting results in user's specified format using the export menu (option `17` from the main menu). For instance, the results can be exported in CSV format for direct use in a third party software. The figures can be exported in PDF, PNG, and LaTeX to create publication-quality figures.

17.1 RES_*.mat result file

The `RES_*.mat` results file are located in the "results/mat" subfolder. The MATLAB binary .MAT contain four MATLAB variables called `timestamps`, `Mean`, `StandardDeviation`, and `labels`.

- `timestamps`: $\mathtt{N} \times 1$ array containing the time vector.
- `Mean`: $\mathtt{N} \times (\mathtt{L} + \mathtt{D} + 1)$ array containing the filtered or smoothed posterior mean values of the hidden states, where `L` is the number of hidden states, `D` is the number of time series.
- `StandardDeviation`: $\mathtt{N} \times (\mathtt{L} + \mathtt{D} + 1)$ array containing the filtered or smoothed posterior standard deviation values of the hidden states.
- `labels`: $1 \times (\mathtt{L} + \mathtt{D} + 1)$ cell array containing the label of each column of the `Mean` and `StandardDeviation` arrays.

The function used to save the result is:

• Save results in a .mat file

```
[misc]=saveResultsMAT(data,model,estimation,misc,varargin)
```

17.2 PROJ_*.mat project file

The `PROJ_*.mat` project file are located in the "saved_projects/mat" subfolder. This files contains the internal variables `model`,`estimation`, `misc`. The content of those internal variables is described in Section 4. The function used to save the project is:

• Save the variables model, estimation, misc in a .mat project file

```
saveProject(model, estimation,misc,varargin)
```

17.3 DATA_*.mat data file

The `DATA_*.mat` data file are located in the "data/mat" subfolder. This file contains three variables called `labels`, `timestamps`, and `values` that fully

describe the time series data. The content of those variables is described in Section 9.1.2. The function used to save the data is:

- Save data in a binary Matlab .mat file

```
[misc, dataFilename] = saveDataBinary(data,misc,varargin)
```

17.4 Export the results

The option `17` from the main menu offers a way to export the data, the results, the project and the figures in specific format (see 10). It is possible to export the figures in PDF, PNG[3] and LaTeX[4,5]. It is possible to export the results and the data in .CSV files. OpenBDLM creates one three-columns CSV file for each posterior filtered or smoothed hidden states for each time series, as well as CSV files for the predicted data for each time series, and a CSV file for the model probability. The first column gives the timestamp, the second column the mean, and the third column the standard deviation. The first line of each file is the header, that gives the reference name of the time-series associated with the hidden states as well as the date of the first timestamp in the 'YYYY-DD-MM-HH-MM-SS' format. It is also possible to export the project in a configuration file that respects the format described in Section 6.
The export workflow is shown in Figure 13, and the functions used to export the results are:

- Pilote function to export data, estimations and project

```
[misc]=piloteExport(data,model,estimation,misc)
```

- Create and print a configuration file from project

[3] Yair Altman, 2011, export_fig, https://www.mathworks.com/matlabcentral/fileexchange/23629-export_fig

[4] Nico Schlömer,2013, matlab2tikz, https://www.mathworks.com/matlabcentral/fileexchange/22022-matlab2tikz-matlab2tikz

[5] All the time-series data figures shown in this document have been created from LaTeX files output from OpenBDLM. Minor post-processing have been done on the LaTeX file. Compilation using LuaLatex.

```
_____________________________________
/    Export
_____________________________________

     1 ->  Export the project in a configuration file
     2 ->  Export data in CSV format
     3 ->  Export results in CSV format
     4 ->  Create and export figures

     Type R to return to the previous menu

     choice >>
```

Listing 10: OpenBDLM export menu

```
[configFilename] = printConfigurationFile(data,model,estimation
    ,misc,varargin)
```

- Save time series data in separate .csv files

```
[misc] = saveDataCSV(data,misc,varargin)
```

- Save results in .CSV files

```
[misc]=saveResultsCSV(data, model, estimation, misc, varargin)
```

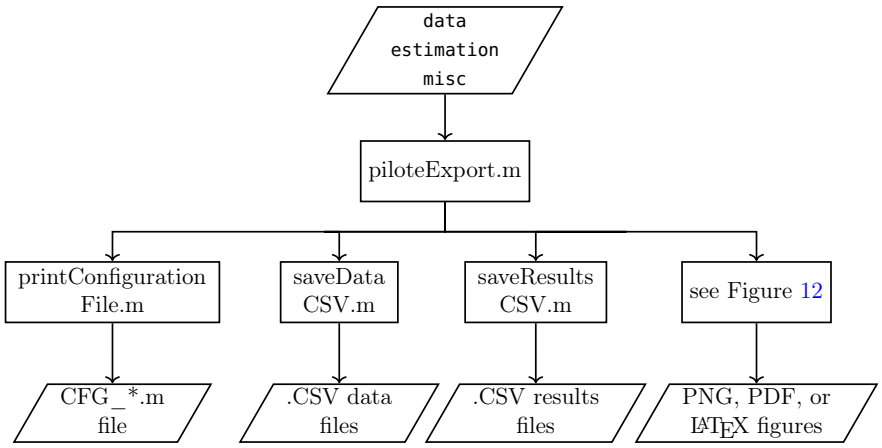


Figure 13: Export options workflow

18 Version control

For the users, version control tests verifies that the program runs properly on your machine. For development purpose, version control tests verifies that changes you have made are still compatible with the previous stable OpenBDLM version. To run version control, type `OpenBDLM_main;` in the MATLAB command line, and then type `V` . If program runs properly, you should get in the Matlab command window some messages as shown in Listing 11.

How does version control work? In the folder "version_control", there are families of three files corresponding to a given project named CTL_00X , where X refers to the version control test index (`CFG_CLT_00X.m`, `DATA_CTL_00X.mat` and `PROJ_CTL_00X.mat`). For each family, the project file `PROJ_CTL_00X.mat` contains hidden states estimations computed from the data and model described in the files `DATA_CTL_00X.mat` and `CFG_CTL_00X.m`, respectively; using a stable version of OpenBDLM. The version control uses the pair of files `CFG_CTL_00X.m` and `DATA_CTL_00X.mat` to compute new hidden states estimations using the current OpenBDLM version installed on your machine. Therefore, if the hidden states estimations from the previous stable OpenBDLM version does not match the estimations from the current OpenBDLM version (RMS value above a given threshold), or if the code

crashes, the version control test fails[6].

```
- Version control test #1

    Starting OpenBDLM_V1.0...
    Loading configuration file...
    Saving data...
    Building model...
    Computing hidden states ...
    Saving project...
    Saving project...
    Done ! See you soon !

    ==> Version control test 3: PASS
```

Listing 11: OpenBDLM version control output

The functions used for version control are:

- Pilote function for version control

  `piloteVersionControl(misc)`

- Version control for OpenBDLM

  `[controlOut]=versionControl(misc, varargin)`

19 Step-by-step example: single time-series analysis

This example uses a single time series data that mimics displacement data measured on a bridge (synthetic data).

19.1 Step 1: start a project

In the MATLAB command window, type `OpenBDLM_main;` , and press the key "enter" ↵. The `OpenBDLM_main` starting menu appears. The user has to provide an input from the command line each time a `choice >>` appears.

[6] It is possible to add or modify CFG, PROJ and DATA files to design new version control tests.

```
_____________________________________________
    Starting OpenBDLM_V1.0
_____________________________________________
           Time series analysis using
           Bayesian Dynamic Linear Models
_____________________________________________
- Start a new project:

    *      Enter a configuration filename
    0   -> Interactive tool

- Type D to Delete project(s), V for Version control, Q to Quit.

    choice >> 0

_________________________________________
         Starting a new project...
_________________________________________

- Enter a project name (max 25 characters):
    choice >> Example_DISP

- Does this project aim to create synthetic data ? (y/n)
    choice >> no

    Load data...

- Choose a database

    0   -> Build a new database

    choice >> 0
```

Listing 12: OpenBDLM command line interaction when calling `OpenBDLM_main;` from MATLAB command line

First, choose the interactive tool by typing `0`. Secondly, provide a project name (e.i `Example_DISP`). Then, answer `no` to indicate that you are not concerned with creating synthetic data. The next step is to type `0` to indicate that you aim to load new data.

19.2 Step 2: load the data

At this stage, a graphical user interface [7] should appear on screen. Browse the "examples/Example_DISP" folder to select the csv file named `Example_DISP_DISP.csv`. Then, click on the Add button, and then the Done button, as highlighted in Figure 14. You will notice that some basic information regarding the loaded time-series, such as the time series index, the reference name and the number of data points are now displayed in the MATLAB command window, as depicted in Listing 13. At this time, three MATLAB figures as those represented in Figure 15 should popup on screen. The first figure represents the data amplitude; the second figure represents the data timestep, and the last figure the data availability. The figures show that data points exist between August 2013 and October 2015 (see Figure 15a). The timestep is non-uniform; it varies from 1 hour to 25 hours (see Figure 15b). The most frequent (i.e referent) time step is 1 hour. There is no missing data (see Figure 15c).

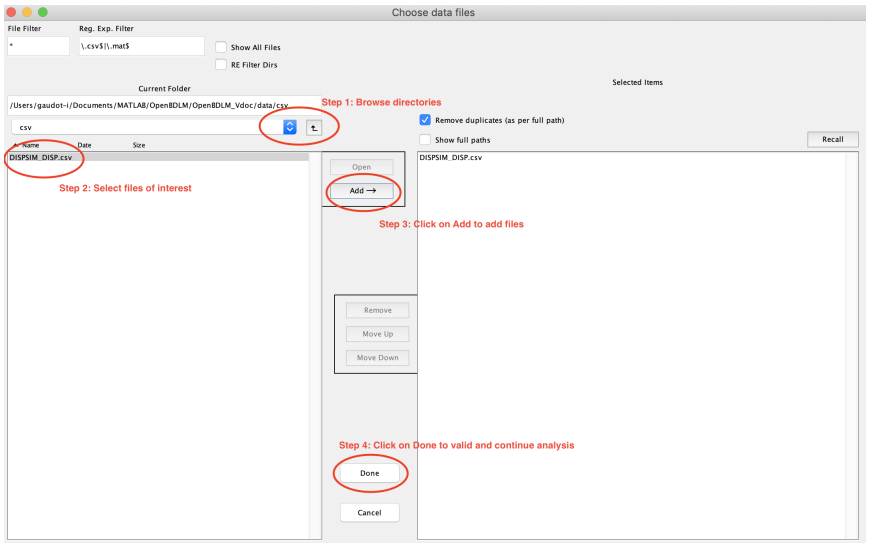


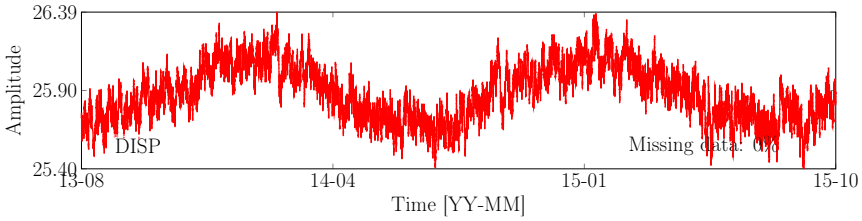
Figure 14: Interactive data loading using the graphical user interface.

.

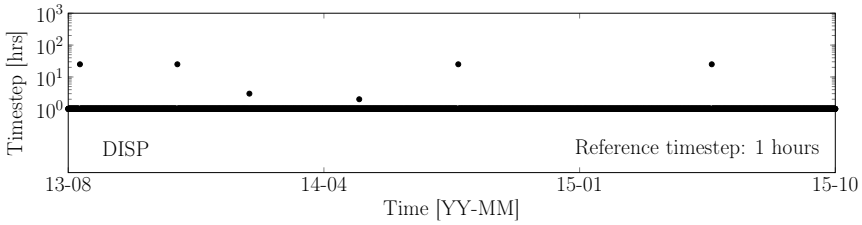
[7] Douglas M. Schwarz, 2007, uipickfiles https://www.mathworks.com/matlabcentral/fileexchange/10867-uipickfiles-uigetfile-on-steroids

```
— Data available:

    Time series number #      Reference name            Size
    ____________________________________________________________
    1                         DISP                      [19366x1]
    ____________________________________________________________
```

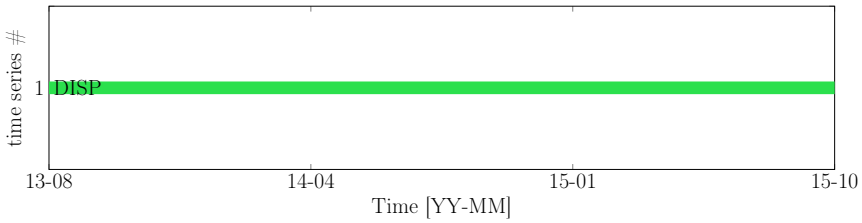
Listing 13: MATLAB command window output after the loading of a data file.



(a) Amplitude



(b) Timestep



(c) Availbility

Figure 15: Data used in the example presented Section 19.

19.3 Step 3: edit and pre-process the data

The data editing and preprocessing menu as depicted in Listing 5 appear. In this example, the objective is to use the data as such. Therefore, no editing or preprocessing is done. Therefore, type `7` to save the data as such, and

continue analysis.

19.4 Step 4: configure the model

First, the program requests the number of model class. In this example, the time series data looks stationary and we are not interested in anomaly detection, so type type `1`. Secondly, OpenBDLM asks for the type of block component. Type `[11 31 31 41]` to define a model with a local level, a yearly periodic component, a daily periodic component and an autoregressive component. The output on MATLAB command window during interactive model configuration is presented in Listing 14. The model is then built, a `DATA_Example_DISP.mat` binary data file, a `CFG_Example_DISP.m` configuration file, as well as a `PROJ_Example_DISP.mat` project file are created. The OpenBLDM main menu must appear on the MATLAB command window (see Listing 3). Type `Q` to save and quit.

19.5 Step 5: open the configuration file

After the data loading and the model configuration, a configuration file named `CFG_Example_DISP.csv` is automatically created and saved in "config_files" folder. Open the configuration file from MATLAB command line by typing `edit CFG_Example_DISP.m`. The first part of this configuration file as it should appear on the MATLAB editor is shown in Listing 15. The Model parameters section of the configuration file shows that the model totalizes 8 model parameters, that is

$$\boldsymbol{\theta} = \{\sigma_w^{LL}, p^{\text{PD1}}, \sigma_w^{\text{PD1}}, p^{\text{PD2}}, \sigma_w^{\text{PD2}}, \phi^{AR}, \sigma_w^{AR}, \sigma_v\}.$$

The default model parameters values are

$$\boldsymbol{\theta}^{\text{default}} = \{0, 365.2422, 0, 1, 0, 0.75, 0.0174, 0.0087002\}.$$

The default hidden states mean and covariance values are

$$\begin{aligned} \boldsymbol{\mu}_0^{\text{default}} &= [25.8, 5, 0, 5, 0, 0]^\intercal, \text{and} \\ \text{diag}(\boldsymbol{\Sigma}_0^{\text{default}}) &= [0.121, 0.121, 0.121, 0.121, 0.121, 0.0303], \end{aligned}$$

respectively.

19.6 Step 6: estimate the hidden states

Type `OpenBDLM_main('CFG_Example_DISP.m');` in the MATLAB command line. Once, the main menu appears, type `3`, then `1` to estimate the filtered hidden states using the default model parameters and default initial hidden states values. The value of the log-likelihood is 36626, and the estimated hidden states are presented in Figure 16.

19.7 Step 7: estimate the model parameters from the data

Type `OpenBDLM_main('CFG_Example_DISP.m');` in the MATLAB command line. Once, the main menu appears, type `1`, then `1` to estimate the model parameters using Newton-Raphson (type `2` to use the Stochastic Gradient Descent instead). The model parameters learning procedure starts, and messages are printed on MATLAB command window to monitor the convergence through iterations (see Listing 7). Note that, by default, OpenBDLM considers that the parameters σ_w^{LL}, p^{PD1}, σ_w^{PD1} , p^{PD2}, σ_w^{PD2} are known. Therefore, there are three model parameters to be learned from the data in this example. The estimation of the model parameters may take several hours. Therefore, press combinations `Ctrl` + `c` to abort the process. Once the algorithm is converged, the optimized model parameters values should be close to [8]

$$\boldsymbol{\theta}^* = \{0, 365.2422, 0, 1, 0, 0.97, 0.0192, 7.4258 \times 10^{-7}\}.$$

19.8 Step 8: estimate the hidden states using the optimized model parameters values

In the "examples/Example_DISP" folder, there is a configuration file named `CFG_Example_DISP_optim.m'` that contains optimized model parameters estimated using the Newton-Raphson algorithm. Copy and paste `CFG_Example_DISP_optim.m'` from the "examples/Example_DISP" subfolder to the "config_files" folder. Type `OpenBDLM_main('CFG_Example_DISP_optim.m');` in the MATLAB command line to load the configuration file `CFG_Example_DISP_optim.m`. Once the main menu appears, type `3`, then `1` to estimate the filtered hidden states using the optimized model parameters and default initial hidden states values. The value of the log-likelihood is now 48819. The estimated hidden states are presented in Figure 17.

[8] Note that it is possible to get slightly different value of parameters with the same performance.

19.9 Step 9: estimate the initial hidden states

Type `OpenBDLM_main('CFG_Example_DISP_optim.m');` in the MATLAB command line. Then, type `2`, to optimize the initial hidden states value. The estimated initial hidden states mean and covariance values are

$$\begin{aligned}\boldsymbol{\mu}_0^* &= [25.9, -0.204, -0.00288, 0.0341, 0.0521, -0.0436]^\intercal, \text{and} \\ \text{diag}(\boldsymbol{\Sigma}_0^*) &= [3.74 \times 10^{-5}, 6.87 \times 10^{-5}, 7 \times 10^{-5}, 5.73 \times 10^{-7}, 5.73 \times 10^{-7}, \\ &\quad 0.000493],\end{aligned}$$

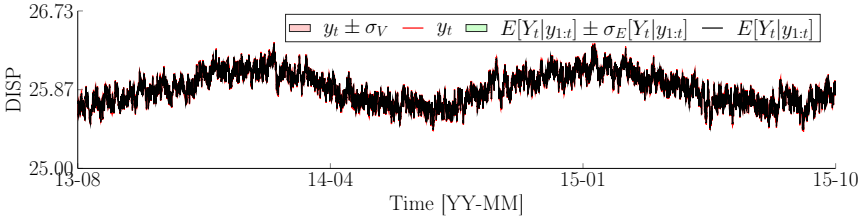
respectively. Once it is done, type `3`, and then `1` to compute the filtered hidden states using the optimized model parameters and optimized initial hidden states. The value of the log-likelihood is 49056. The estimated hidden states are presented in Figure 18.

```
- How many model classes do you want for each time-series?
    choice >> 1

    ------------------------------------------------------------
    BDLM Component reference numbers
    ------------------------------------------------------------
    11: Local level
    12: Local trend
    13: Local acceleration
    21: Local level compatible with local trend
    22: Local level compatible with local acceleration
    23: Local trend compatible with local acceleration
    31: Periodic
    41: Autoregressive process (AR(1))
    51: Kernel regression
    61: Level Intervention
    ------------------------------------------------------------

- Identify components for time series #1; e.g. [11 31 41]
    choice >> [11 31 31 41]

    Building model...
    Saving project...
    Project saved in saved_projects/PROJ_Example_DISP.mat.
    Printing configuration file...
    Saving data...

    Database saved in data/mat/DATA_Example_DISP.mat
    Configuration file saved in config_files/CFG_Example_DISP.m.
```

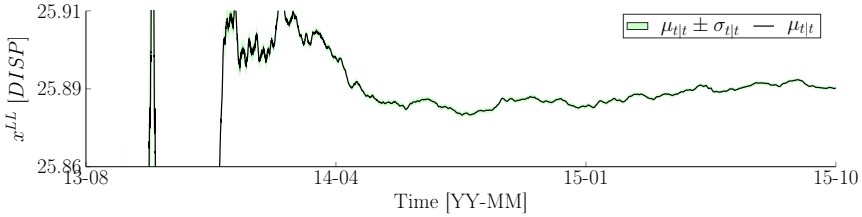
Listing 14: MATLAB command window output during interactive model configuration.

```
%                    OpenBDLM configuration file
%          Autogenerated by OpenBDLM on 22-Nov-2018 17:18:09
%
%% A - Project name
misc.ProjectName='Example_DISP';

%% B - Data
dat=load('DATA_Example_DISP.mat');
data.values=dat.values;
data.timestamps=dat.timestamps;
data.labels={'Example_DISP'};

%% C - Model structure
% Components reference numbers
% 11: Local level
% 12: Local trend
% 13: Local acceleration
% 21: Local level compatible with local trend
% 22: Local level compatible with local acceleration
% 23: Local trend compatible with local acceleration
% 31: Periodic
% 41: Autoregressive
% 51: Kernel regression

% Model components
% Model 1
model.components.block{1}={[11 31 31 41] };

% Model component constrains | Take the same  parameter as model class #1

% Model inter-components dependence | {[components form dataset_i depends on components
    from  dataset_j]_i,[...]}
model.components.ic={[ ] };
%
%% D - Model parameters
model.param_properties={
    % #1       #2       #3    #4    #5         #6       #7     #8     #9    #10
    % Param name Block name  Model Obs Bound Prior Mean Std Values Ref
    '\sigma_w', 'LL',  '1',   '1', [NaN  NaN],    'N/A',  NaN, NaN,  0,          1 %#1
    'p',        'PD1', '1',   '1', [NaN  NaN],    'N/A',  NaN, NaN, 365.24,      2 %#2
    '\sigma_w', 'PD1', '1',   '1', [NaN  NaN],    'N/A',  NaN, NaN,   0,         3 %#3
    'p',        'PD2', '1',   '1', [NaN  NaN],    'N/A',  NaN, NaN,   1,         4 %#4
    '\sigma_w', 'PD2', '1',   '1', [NaN  NaN],    'N/A',  NaN, NaN,   0,         5 %#5
    '\phi',     'AR',  '1',   '1', [0  1],        'N/A',  NaN, NaN,   0.75,      6 %#6
    '\sigma_w', 'AR',  '1',   '1', [0  Inf],      'N/A',  NaN, NaN,   0.0174,    7 %#7
    '\sigma_v', '',    '1',   '1', [0  Inf],      'N/A',  NaN, NaN,   0.0087002, 8 %#8
};

%% E - Initial states values
% Initial hidden states mean for model 1:
model.initX{ 1 }=[      25.8    5       0       5       0       0     ]';

% Initial hidden states variance for model 1:
model.initV{ 1 }=diag([         0.121   0.121   0.121   0.121   0.121   0.0303 ]);

% Initial probability for model 1
model.initS{1}=[1     ];
```

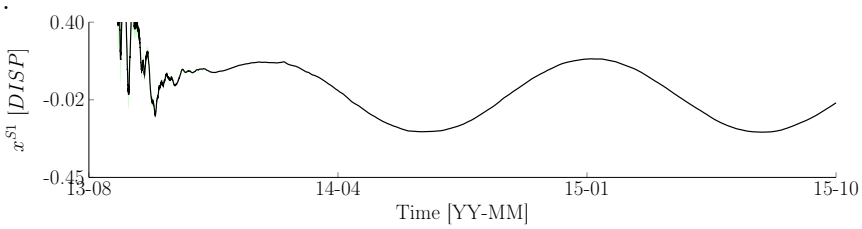
Listing 15: Example of a configuration file



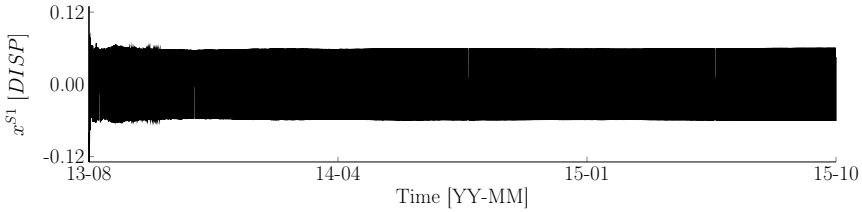
(a) Observed and estimated displacement data



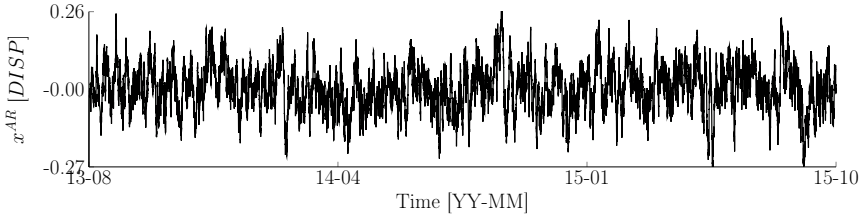
(b) Estimated displacement local level component



(c) Estimated displacement yearly periodic component

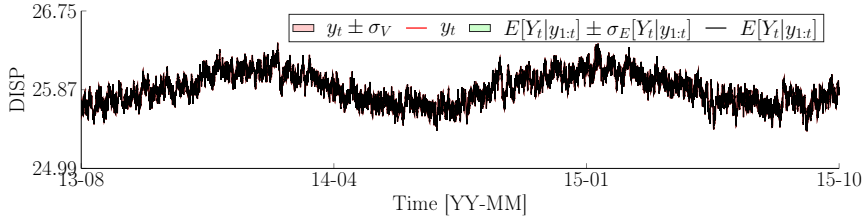


(d) Estimated displacement daily periodic component

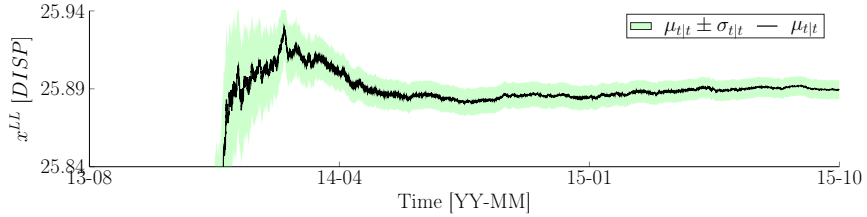


(e) Estimated displacement autoregressive component

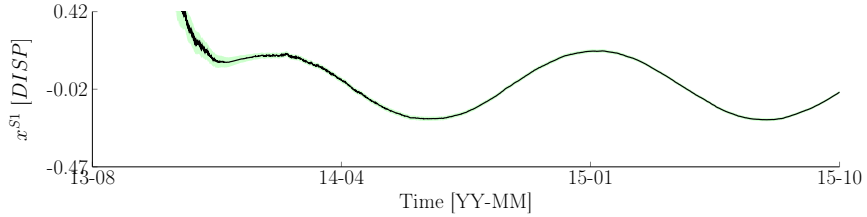
Figure 16: Estimated results using OpenBDLM defaults model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 15a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.



(a) Observed and estimated displacement data

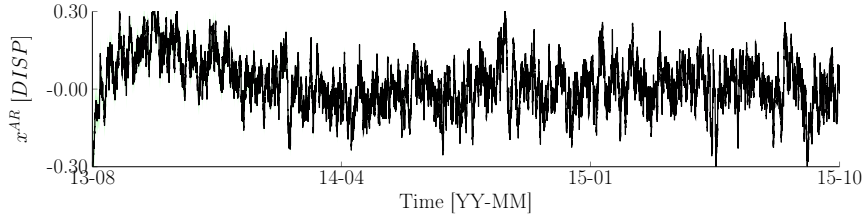


(b) Estimated displacement local level.



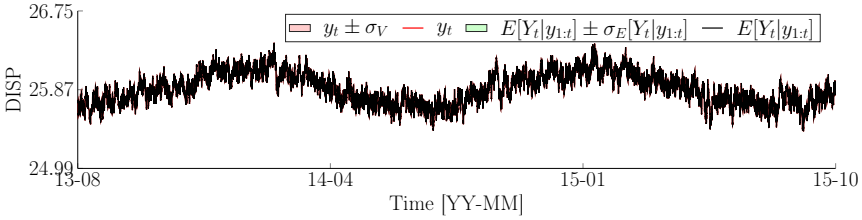
(c) Estimated displacement yearly periodic component (first hidden state)

(d) Estimated displacement daily periodic component (first hidden state)

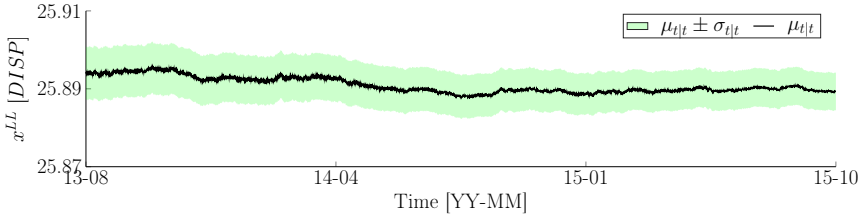


(e) Estimated displacement autoregressive component

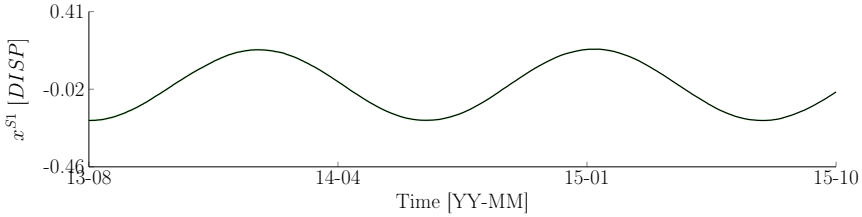
Figure 17: Estimated results using OpenBDLM optimized model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 15a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.



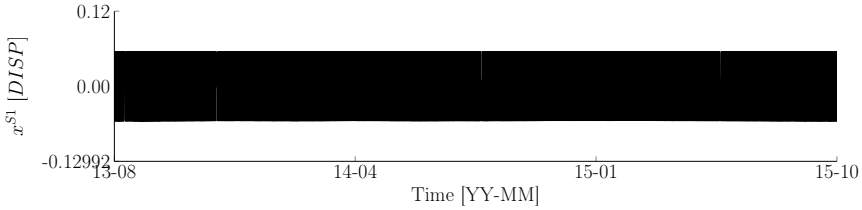
(a) Observed and estimated displacement data



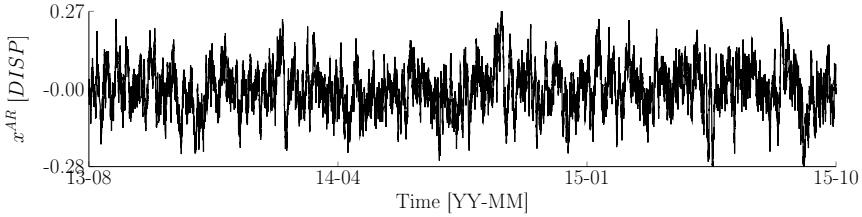
(b) Estimated displacement local level component.



(c) Estimated displacement yearly periodic component (first hidden state)



(d) Estimated displacement daily periodic component (first hidden state)



(e) Estimated displacement autoregressive component

Figure 18: Estimated results using OpenBDLM optimized model parameters and optimized initial hidden states. The hidden states are estimated from the data presented in Figure 15a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

20 Step-by-step example: dependence model between two time-series

This example uses two time series data that mimics displacement and temperature data measured on a bridge (synthetic data).

20.1 Step 1: start a project

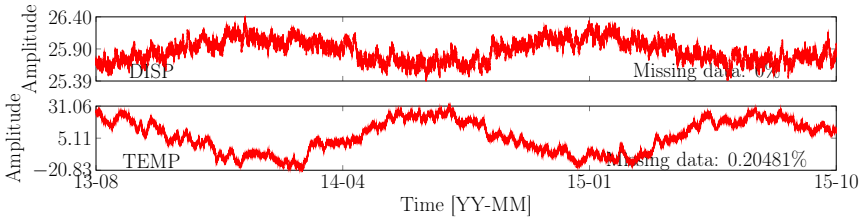
First, choose the interactive tool by typing `0`. Secondly, provide a project name (e.i `Example_DISPTEMP`). Then, answer `no` to indicate that you are not concerned with creating synthetic data. The next step is to type `0` to indicate that you aim to load new data.

20.2 Step 2: load the data

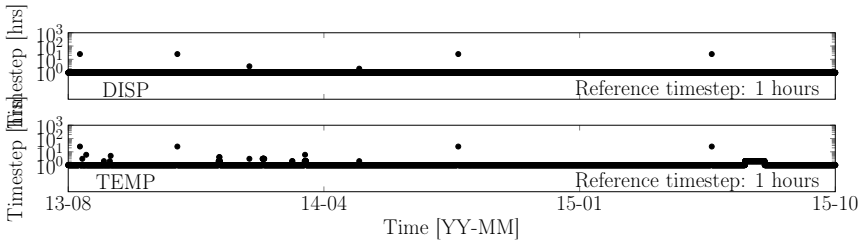
At this stage, a graphical user interface should appear on screen. Browse the "examples/Example_DISPTEMP" folder to select the csv files named `Example_DISPTEMP_DISP.csv` and `Example_DISPTEMP_TEMP.csv`. Then, click on the Add button, and then the Done button, as highlighted in Figure 14. You will notice that some basic information regarding the loaded time-series, such as the time series index, the reference name and the number of data points are now displayed in the MATLAB command window, as depicted in Listing 17. At this time, three MATLAB figures as those represented in Figure 15 should popup on screen. The first figure represents in red the data amplitude of each time series data; the second figure represents the data timestep, and the last figure the data availability. The figures show that data points exist between August 2013 and October 2015 (see Figure 19a). The timestep is non-uniform; it varies from 1 hour to 25 hours (see Figure 19b). The timestep vector is not identical on each time series. It means that the time-series are not synchronized between each other. The most frequent (i.e referent) time step is 1 hour for both time-series. There is no missing data on the displacement time-series, but there are missing data on the temperature time series as indicated by the red crosses on the Figure 19c. Each red cross indicate the presence of Not a Number (NaN) value in the time series data.

20.3 Step 3: edit and pre-process the data

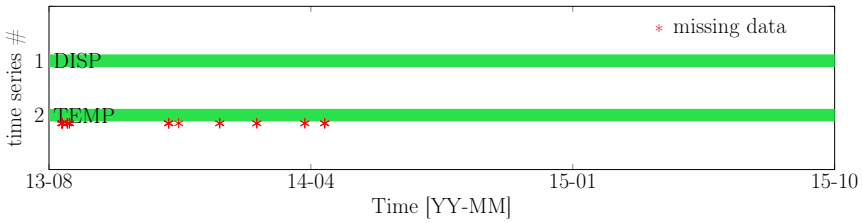
The next step of the analysis consists in editing and preprocessing the data. The data editing and preprocessing menu as depicted in Listing 5 should appear on the MATLAB command window. In this example, there are two



(a) Amplitude



(b) Timestep



(c) Availability

Figure 19: Data used in Section 20

.

```
— Data available:

    Time series number #      Reference name            Size
    ------------------------------------------------------------------
    1                         DISP                      [19366x1]
    2                         TEMP                      [19042x1]
    ------------------------------------------------------------------
```

Listing 16: MATLAB command window output after selected data files. from MATLAB command line

time-series which are not synchronized between each other. Therefore, a pre-processing is required. Custom pre-processing can be done using the

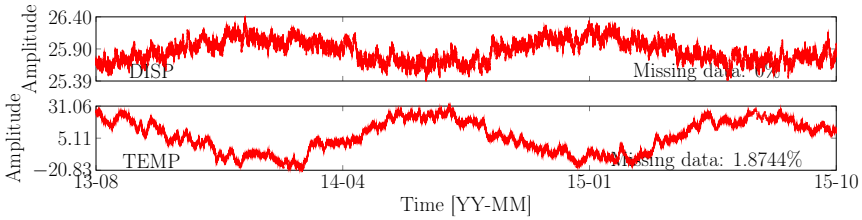
data editing and preprocessing OpenBDLM tools (see Listing 5). In this example, we are interested in using the default OpenBDLM preprocessing. In such case, the option `7` will automatically synchronized the two time-series between each other and create a new dataset. After typing `7`, a figure should popup on the screen that shows the processed data which are now synchronized between each other (see Figure 20). The displacement and temperature time series data now share same timestep vector, and therefore they both have the same number of data samples, as shown in the Listing 17.

```
— Data available:

    Time series number #      Reference name            Size
    ------------------------------------------------------------------------
    1                         DISP                      [19366x1]
    2                         TEMP                      [19366x1]
    ------------------------------------------------------------------------
```

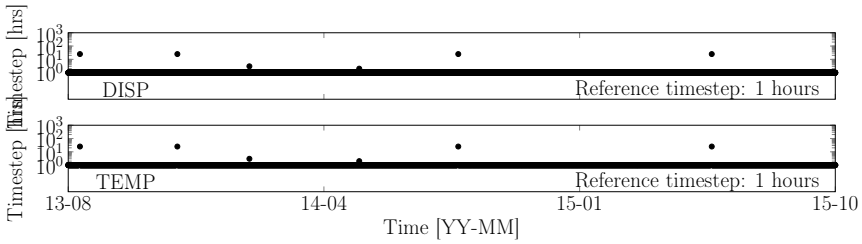
Listing 17: MATLAB command window output after selected data files

20.4 Step 4: configure the model

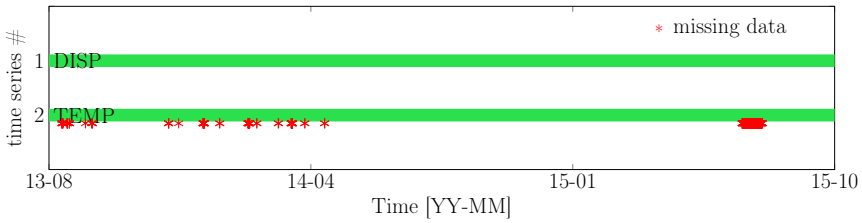
The next step is to configure the model. First, the program requests the number of model class. In this exemple, the time series data looks stationary and we are not interested in anomaly detection, and therefore we type `1`. Secondly, because there are several time series, OpenBDLM needs to know if there are dependencies between the time-series. Typing `2` for the first time-series, and `0` for the second time series means that the model will consider the irreversible components (if any) of the second (temperature) time-series as covariates to describe the irreversible patterns observed in the first (displacement) time-series. Then, OpenBDLM asks for the type of block component for each time-series. Type `[11 41]` for the displacement time series and `[11 31 31 41]` for the temperature time series. The yearly and daily periodic patterns observed in the displacement time-series are modelled using the dependence on the periodic components defined for modelling the temperature time series data. Note that because an autoregressive component is chosen for the displacement time series data, the time-dependent model error on the displacement is modelled using a dependence on the autoregressive component of the temperature time series data, as well as an independent autoregressive component. The output on MATLAB command window during interactive model configuration is



(a) Amplitude



(b) Timestep



(c) Availability

Figure 20: Data used in Section 20

.

presented in Listing 18. Type ↵ to valid. The model is then built, a `DATA_DISPTEMP.mat` binary data file, a `CFG_DISPTEMP.m` configuration file, as well as a `PROJ_DISPTEMP.mat` project file are created. The OpenBLDM main menu must appear on the MATLAB command window (see Listing 3). Type `Q` to save and quit.

20.5 Step 5: open the configuration file

After the data loading and the model configuration, a configuration file named `CFG_Example_DISPTEMP.m` configuration file is automatically created and saved in "config_files" folder. Open the configuration file from MATLAB command line by typing `edit CFG_Example_DISPTEMP.m`. The Model parameters section of the configuration file shows that the model totalizes 15

```
- Identifies dependence between time series; use [0] to indicate no
    dependence
    time serie #1 depends on time series # >> [2]

- Identifies dependence between time series; use [0] to indicate no
    dependence
    time serie #2 depends on time series # >> [0]

- How many model classes do you want for each time-series?
     choice >> 1

     ____________________________________
          BDLM Component reference numbers
     ____________________________________
     11: Local level
     12: Local trend
     13: Local acceleration
     21: Local level compatible with local trend
     22: Local level compatible with local acceleration
     23: Local trend compatible with local acceleration
     31: Periodic
     41: Autoregressive process (AR(1))
     51: Kernel regression
     61: Level Intervention
     ____________________________________

- Identify components for time series #1; e.g. [11 31 41]
     choice >> [11 41]

- Identify components for time series #2; e.g. [11 31 41]
     choice >> [11 31 31 41]

     Building model...
     Saving project...
     Project saved in saved_projects/PROJ_Example_DISPTEMP.mat.
     Printing configuration file...
     Saving data...
     Database saved in data/mat/DATA_Example_DISPTEMP.mat
     Configuration file saved in config_files/CFG_Example_DISPTEMP.m.
```

Listing 18: MATLAB command window output during model configuration

model parameters, that is

$$\boldsymbol{\theta} = \{\sigma^{LL}_{w,\mathrm{D}}, \phi^{AR}_{\mathrm{D}}, \sigma^{AR}_{w,\mathrm{D}}, \sigma_{v,\mathrm{D}},$$
$$\sigma^{LL}_{w,\mathrm{T}}, p^{\mathrm{PD1}}_{\mathrm{T}}, \sigma^{\mathrm{PD1}}_{w,\mathrm{T}}, p^{\mathrm{PD2}}_{T}, \sigma^{\mathrm{PD2}}_{w,\mathrm{T}}, \phi^{AR}_{\mathrm{T}}, \sigma^{AR}_{w,\mathrm{T}}, \sigma_{v,\mathrm{T}}, \phi^{\mathrm{D|T}}_{PD1}, \phi^{\mathrm{D|T}}_{PD2}, \phi^{\mathrm{D|T}}_{AR}\}.$$

The default model parameters values are

$$\boldsymbol{\theta}^{\text{default}} = \{0, 0.75, 0.017, 0.0087,$$
$$0, 365.24, 0, 1, 0, 0.75, 1.2905, 0.64526, 0.5, 0.5, 0.5\}.$$

The default initial hidden states mean and covariance values are

$$\boldsymbol{\mu}_0^{\text{default}} = [25.7, 0, 16.7, 5, 0, 5, 0, 0]^\intercal, \text{and}$$
$$\text{diag}(\boldsymbol{\Sigma}_0^{\text{default}}) = [0.122, 0.0305, 666, 666, 666, 666, 666, 167],$$

respectively. In the Options section, change `misc.options.MethodStateEstimation='kalman'` to `misc.options.MethodStateEstimation='UD'`. In this specific example, the presence of missing data requires the use of UD computations instead of the standard, default, Kalman computation. Note that the choice about UD or Kalman is problem dependent.

20.6 Step 6: estimate the hidden states

Type `OpenBDLM_main('CFG_Example_DISPTEMP.m');` in the MATLAB command line. Once, the main menu appears, type `3`, then `1` to estimate the filtered hidden states using the default model parameters and default initial hidden states values. The value of the log-likelihood is -3800091. The estimated hidden states are presented in Figure 21.

20.7 Step 7: estimate the model parameters from the data

Type `OpenBDLM_main('CFG_Example_DISPTEMP.m');` in the MATLAB command line. Once, the main menu appears, type `1`, then `1` to estimate the model parameters using Newton-Raphson (type `1` to use the Stochastic Gradient instead). The model parameters learning procedure should start (see for example Listing 7). Note that, by default, OpenBDLM considers that the parameters $\sigma_{w,\text{D}}^{LL}$, $\sigma_{w,\text{T}}^{LL}$, $p_{\text{T}}^{\text{PD1}}$, $\sigma_{w,\text{T}}^{\text{PD1}}$, $p_{\text{T}}^{\text{PD2}}$, $\sigma_{w,\text{T}}^{\text{PD2}}$ are known. Therefore, there are nine model parameters to be learned from the data in this example. The estimation of the model parameters may take several hours. Therefore, press combinations `Ctrl` + `c` to abort the process. Once the algorithm is converged, the optimized model parameters values should be close to [9]

$$\boldsymbol{\theta}^* = \{0, 0.97, 0.019, 7.42 \times 10^{-7}, 0, 365.2422, 0, 1, 0, 0.99, 0.43, 2.67 \times 10^{-5},$$
$$-0.011, 0.0711, 0.000292\}$$

[9] Note that it is possible to get slightly different value of parameters with the same performance.

20.8 Step 8: estimate the hidden states using the optimized model parameters. values

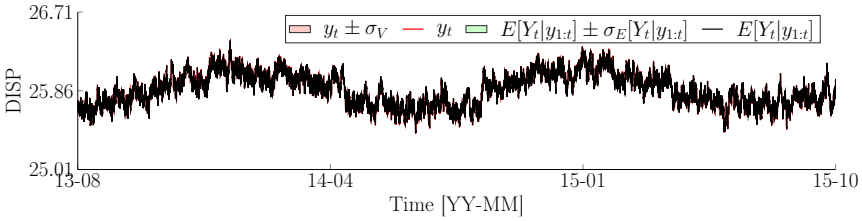
In the "examples/Example_DISPTEMP" folder, there is a configuration file named `CFG_Example_DISPTEMP_optim.m` that contains optimized model parameters estimated using the Newton-Raphson algorithm. Copy and paste `CFG_Example_DISPTEMP_optim.m` from the "examples/Example_DISPTEMP" subfolder to the "config_files" folder. Type `OpenBDLM_main('CFG_Example_DISPTEMP_optim.m');` in the MATLAB command line to load the configuration file `CFG_Example_DISPTEMP_optim.m`. Once the main menu appears, type `3`, then `1` to estimate the filtered hidden states using the optimized model parameters and default initial hidden states values. The value of the log-likelihood is now 38085. The estimated hidden states are presented in Figure 22.

20.9 Step 9: estimate the initial hidden states

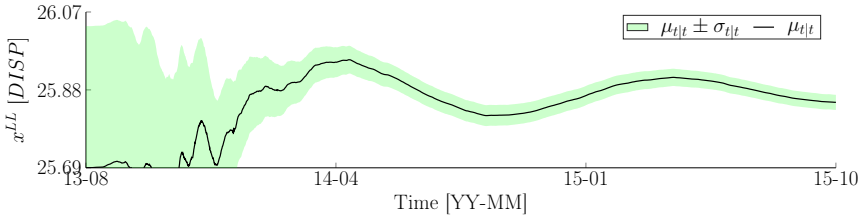
Type `OpenBDLM_main('CFG_Example_DISPTEMP_optim.m');` in the MATLAB command line. Then, type `2`, to optimize the initial hidden states value. The estimated initial hidden states mean and covariance values are

$$\begin{aligned}\boldsymbol{\mu}_0^* &= [25.9, -0.0595, 5.45, 17.6, -0.934, 0.678, 0.41, 2.1]^\intercal, \text{and} \\ \text{diag}(\boldsymbol{\Sigma}_0^*) &= [3.71 \times 10^{-5}, 0.000457, 0.287, 0.263, 0.265, 8.14 \times 10^{-5}, \\ &\quad 8.14 \times 10^{-5}, 0.71],\end{aligned}$$

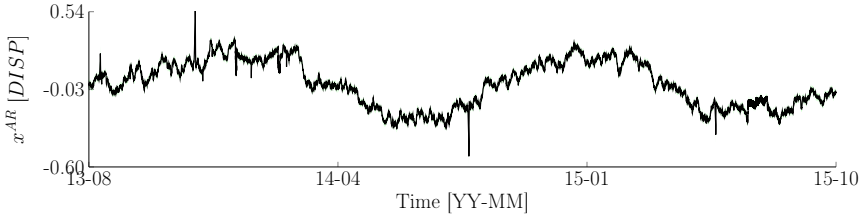
respectively. Once it is done, type `3`, and then `1` to compute the filtered hidden states using the optimized model parameters and optimized initial hidden states. The value of the log-likelihood is 38120. The estimated hidden states are presented in Figure 23.



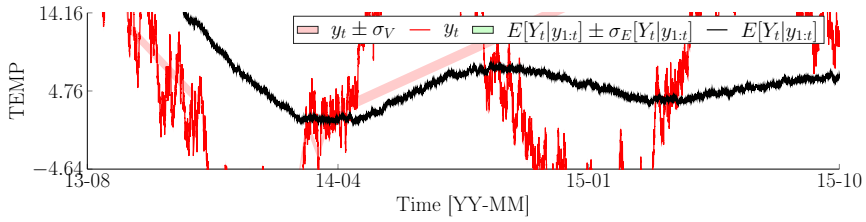
(a) Observed and estimated displacement data



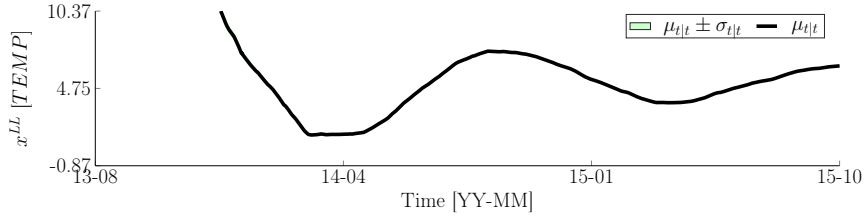
(b) Estimated displacement local level component.



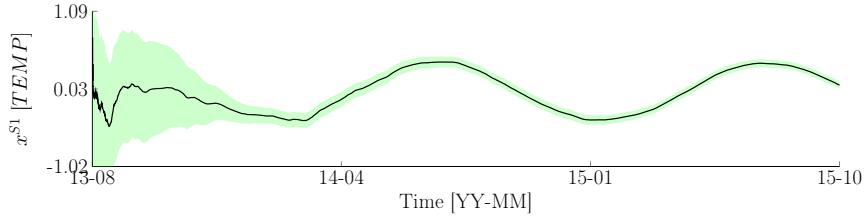
(c) Estimated displacement autoregressive component.



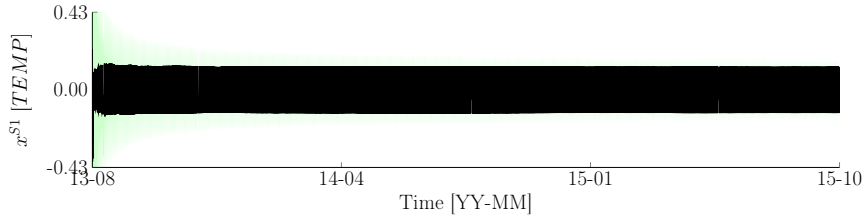
(d) Observed and estimated temperature data



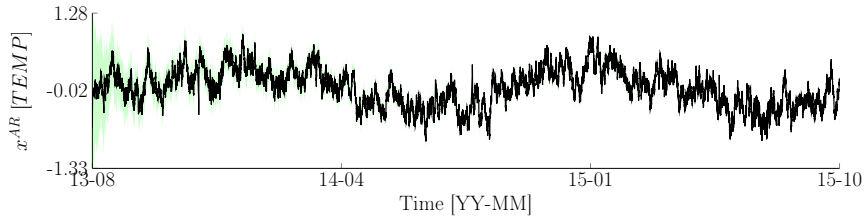
(e) Estimated temperature local level component.



(f) Estimated temperature yearly component (first hidden state)

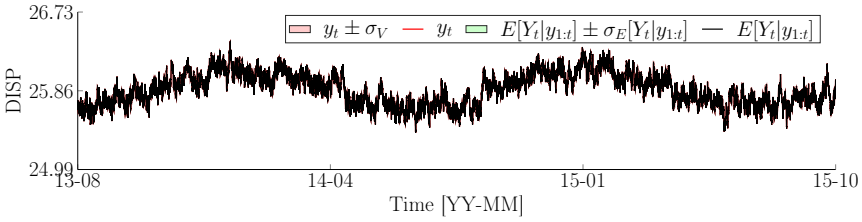


(g) Estimated temperature daily component (first hidden state)

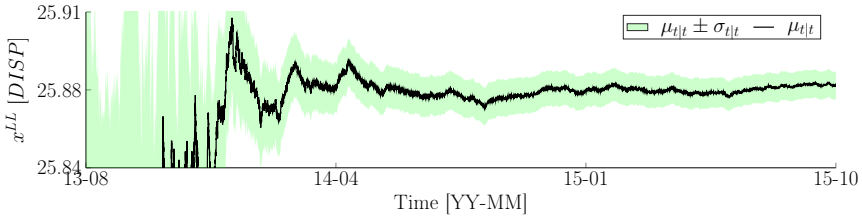


(h) Estimated temperature autoregressive component

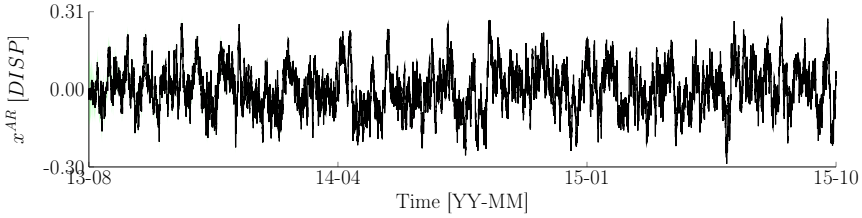
Figure 21: Estimated results using OpenBDLM default model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 20a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.



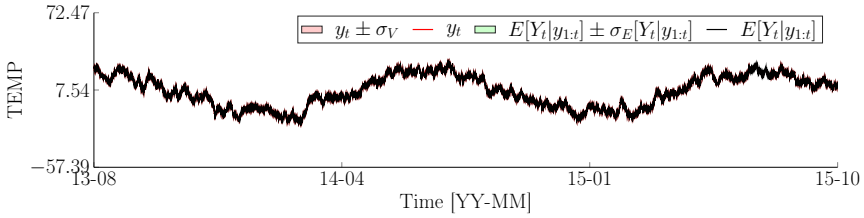
(a) Observed and estimated displacement data



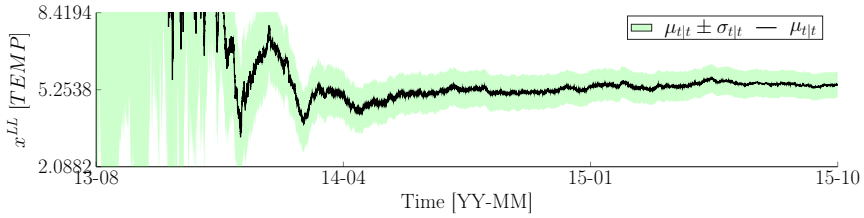
(b) Estimated displacement local level component.



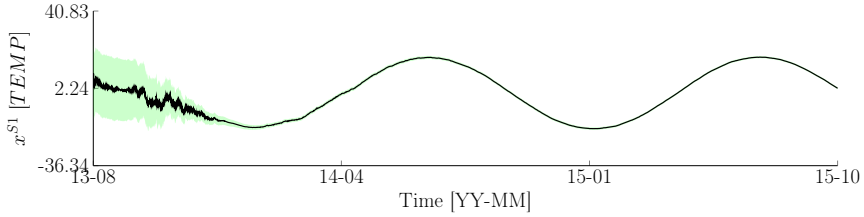
(c) Estimated displacement autoregressive component.



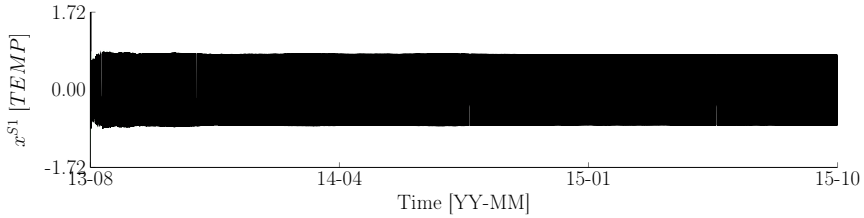
(d) Observed and estimated temperature data



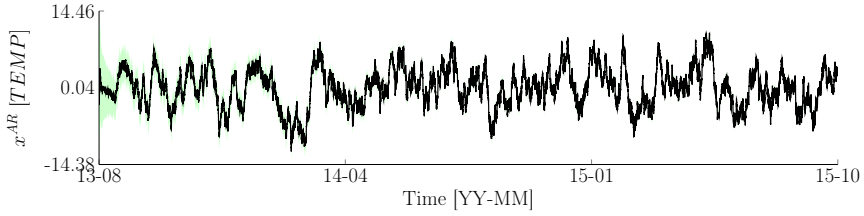
(e) Estimated temperature local level component.



(f) Estimated temperature yearly component (first hidden state)

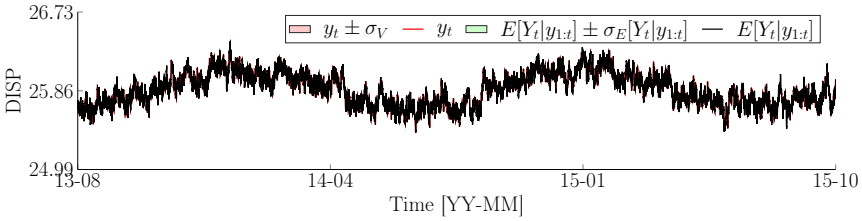


(g) Estimated temperature daily component (first hidden state)

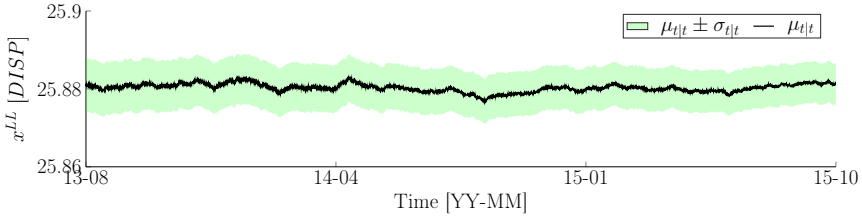


(h) Estimated temperature autoregressive component

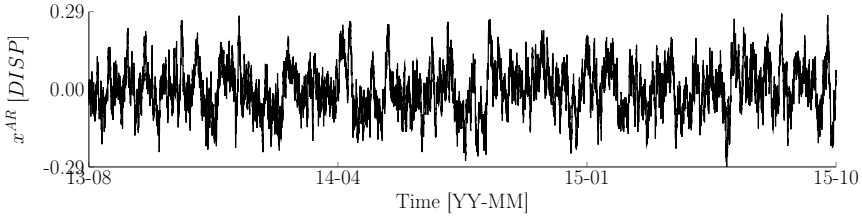
Figure 22: Estimated results using OpenBDLM optimized model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 20a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.



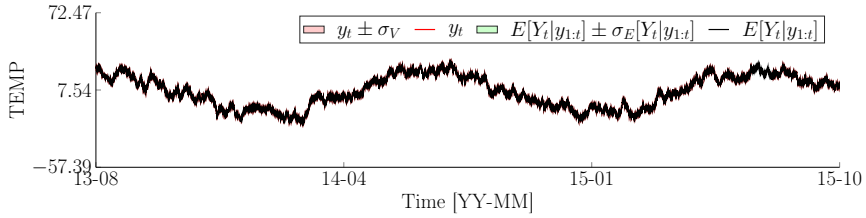
(a) Observed and estimated displacement data



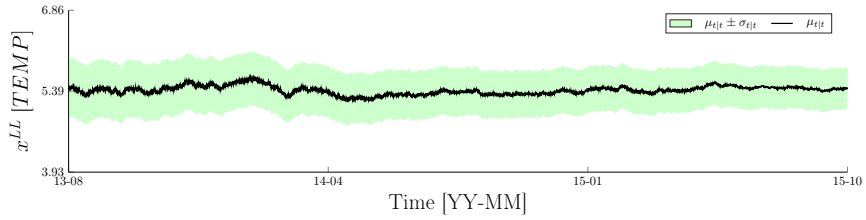
(b) Estimated displacement local level component.



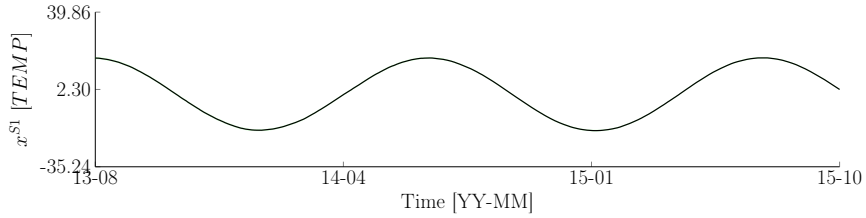
(c) Estimated displacement autoregressive component.



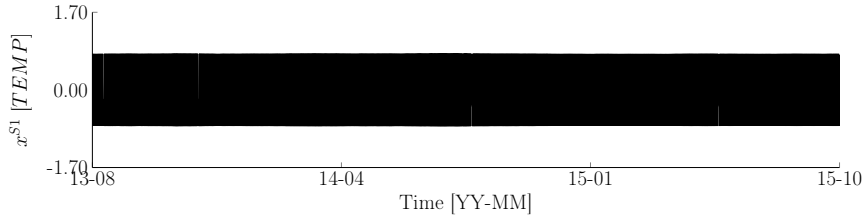
(d) Observed and estimated temperature data



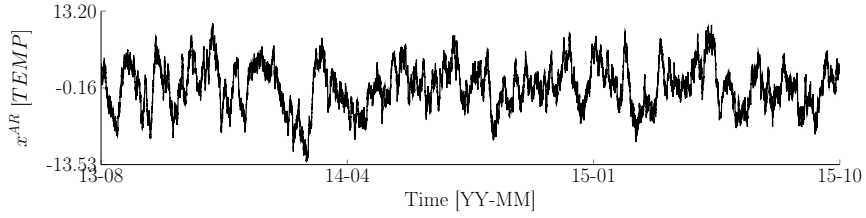
(e) Estimated temperature local level component.



(f) Estimated temperature yearly component (first hidden state)



(g) Estimated temperature daily component (first hidden state)



(h) Estimated temperature autoregressive component

Figure 23: Estimated results using OpenBDLM optimized model parameters and optimized initial hidden states. The hidden states are estimated from the data presented in Figure 20a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

21 Step-by-step example: time-series with anomaly

This example uses a time series data that mimics displacement data measured on a bridge (synthetic data). The baseline switches between a trend stationary to acceleration stationary dynamics during a specific time window to mimic a fictitious anomaly. The anomalous time windows is of 26 days duration, starting on June 25, 2010.

21.1 Step 1: start a project

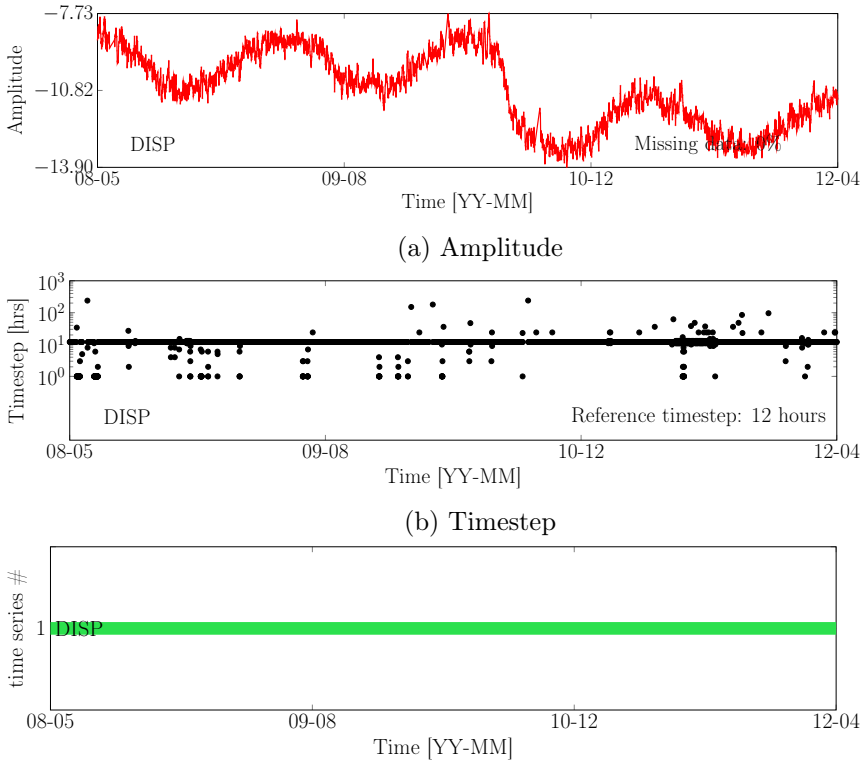
First, choose the interactive tool by typing `0`. Secondly, provide a project name (i.e. `Example_DISP_ANOMALY`). Then, answer `no` to indicate that you are not concerned with creating synthetic data. The next step is to type `0` to indicate that you aim to load new data.

21.2 Step 2: load the data

At this stage, a graphical user interface should appear on screen. Browse the "data/csv" folder to select the csv file named `Example_DISP_ANOMALY_DISP.csv`. Then, click on the Add button, and then the Done button, as highlighted in Figure 14. You will notice that some basic information regarding the loaded time-series, such as the time series index, the reference name and the number of data points are now displayed in the MATLAB command window. At this time, three MATLAB figures as those represented in Figure 24 should popup on screen. The first figure represents in red the data amplitude of each time series data; the second figure represents the data timestep, and the last figure the data availability. The figures show that data points exist between May 2008 and April 2012 (see Figure 24a). The timestep is non-uniform; it varies from 1 hour to 10 days (see Figure 24b). The most frequent (i.e referent) time step is 12 hour. There is no missing data on the displacement time-series.

21.3 Step 4: configure the model

First, the program requests the number of model class. In this example, a change is clearly observed in the data, and we are therefore interested to model non-stationary data and eventually detect the change (i.e. anomaly detection). In such case, we type `2` to build a switching model with two model classes. Then, OpenBDLM asks for the type of block component for



(a) Amplitude

(b) Timestep

(c) Availability

Figure 24: Data used in Section 21

.

the first and second model class. The data baseline looks level or trend stationary, and there is a clear yearly periodic pattern superimposed to it. Therefore, we choose `[23 31 41]` for the first model class and `[13 31 41]` for the second model class. This model considers a trend stationary model with a yearly periodic pattern for the first model class, and an acceleration stationary model with a yearly periodic pattern for the second model class. The next step is to define the model parameter constrain between the two model classes. Here, it is reasonable to consider that the two model classes only differs in the baseline dynamics, not in the periodic or autoregressive component. Therefore, we type `[0 1 1]` to indicate that the two model classes share the same model parameter, except for the baseline component. The output on MATLAB command window during interactive model configuration is presented in Listing 19. Type ↵ to valid. The model is then

built, a `DATA_Example_DISP_ANOMALY.mat` binary data file, a `CFG_DISP_ANOMALY.m` configuration file, as well as a `PROJ_Example_DISP_ANOMALY.mat` project file are created. The OpenBLDM main menu must appear on the MATLAB command window (see Listing 3). Type `Q` to save and quit.

21.4 Step 5: open the configuration file

After the data loading and the model configuration, a configuration file named `CFG_Example_DISP_ANOMALY.m` configuration file is automatically created and saved in "config_files" folder. Open the configuration file from MATLAB command line by typing `edit CFG_Example_DISP_ANOMALY.m`. The Model parameters section of the configuration file shows that the model totalizes 12 model parameters, that is

$$\boldsymbol{\theta} = \{\sigma_{w,\mathrm{D}}^{TcA,1}, p_{\mathrm{D}}^{\mathrm{PD1}}, \sigma_{w,\mathrm{D}}^{\mathrm{PD1}}, \phi_{\mathrm{D}}^{AR}, \sigma_{w,\mathrm{D}}^{AR},$$
$$\sigma_{w,\mathrm{D}}^{LA,2}, \sigma_{w,\mathrm{D}}^{TcA,12}, \sigma_{w,\mathrm{D}}^{LA,21}, \sigma_{v,\mathrm{D}}^{1}, \sigma_{v,\mathrm{D}}^{2}, Z^{11}, Z^{22}\}$$

The default model parameters values are

$$\boldsymbol{\theta}^{\mathrm{default}} = \{1.5018 \times 10^{-8}, 365.24, 0, 0.75, 0.075092,$$
$$1.5018 \times 10^{-8}, 1.5018 \times 10^{-8}, 1.5018 \times 10^{-5}, 0.075092, 0.075092, 0.99986, 0.99986\}.$$

The default initial hidden states mean, covariance and model probability values for model class 1 are

$$\begin{aligned} \boldsymbol{\mu}_0^{1,\mathrm{default}} &= [-8.72, 0, 0, 5, 0, 0]^\intercal, \text{and} \\ \mathrm{diag}(\boldsymbol{\Sigma}_0^{1,\mathrm{default}}) &= [9.02, 2.26, 2.26, 9.02, 9.02, 2.26], \text{and} \\ \pi_0^{1,\mathrm{default}} &= 0.5. \end{aligned}$$

respectively. The default initial hidden states mean, covariance and model probability values for model class 2 are

$$\begin{aligned} \boldsymbol{\mu}_0^{2,\mathrm{default}} &= [-8.72, 0, 0, 5, 0, 0]^\intercal, \text{and} \\ \mathrm{diag}(\boldsymbol{\Sigma}_0^{2,\mathrm{default}}) &= [9.02, 2.26, 2.26, 9.02, 9.02, 2.26], \text{and} \\ \pi_0^{2,\mathrm{default}} &= 0.5. \end{aligned}$$

respectively.

21.5 Step 6: estimate the hidden states

Type `OpenBDLM_main('CFG_Example_DISP_ANOMALY.m');` in the MATLAB command line. Once, the main menu appears, type `3`, then `1` to estimate the filtered hidden states using the default model parameters and default initial hidden states values. The value of the log-likelihood is -409. The estimated hidden states are presented in Figure 25.

21.6 Step 7: estimate the model parameters from the data

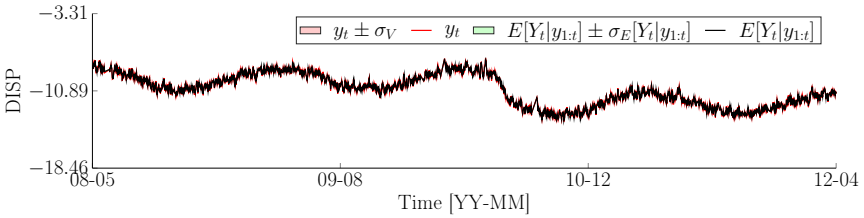
Type `OpenBDLM_main('CFG_Example_DISP_ANOMALY.m');` in the MATLAB command line. Once, the main menu appears, type `1`, then `1` to estimate the model parameters using Newton-Raphson (type `1` to use the Stochastic Gradient instead). The model parameters learning procedure should start (see for example Listing 7). Note that, by default, OpenBDLM considers that the parameters P^{PD1}, σ_w^{PD1} are known. Therefore, there are ten model parameters to be learned from the data in this example. The estimation of the model parameters may take several hours. Therefore, press combinations `Ctrl` + `c` to abort the process. Once the algorithm is converged, the optimized model parameters values should be close to [10]

$$\boldsymbol{\theta}^* = \{0, 365.24, 0, 0.75, 0.213,$$
$$0.001, 8.1224 \times 10^{-6}, 1.5018 \times 10^{-5}, 0.10671, 0.10671, 0.99997, 0.99957\}.$$

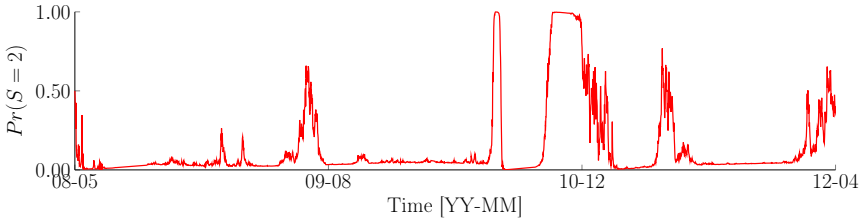
21.7 Step 8: estimate the hidden states using the optimized model parameters values

In the "examples/Example_DISP_ANOMALY" folder, there is a configuration file named `CFG_Example_DISP_ANOMALY_optim.m` that contains optimized model parameters estimated using the Newton-Raphson algorithm. Copy and paste `CFG_Example_DISP_ANOMALY_optim.m` from the "examples/Example_DISP_ANOMALY" subfolder to the "config_files" folder. Type `OpenBDLM_main('CFG_Example_DISP_ANOMALY_optim.m');` in the MATLAB command line to load the configuration file `CFG_Example_DISP_ANOMALY_optim.m`. Once the main menu appears, type `3`, then `1` to estimate the filtered hidden states using the optimized model parameters and default initial hidden states values. The value of the log-likelihood is now 172. The estimated hidden states are presented in Figure 26.

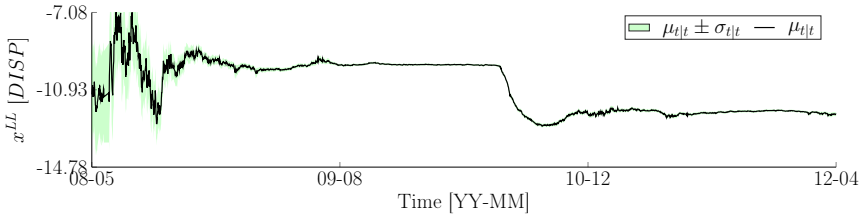
[10] Note that it is possible to get slightly different value of parameters with the same performance.



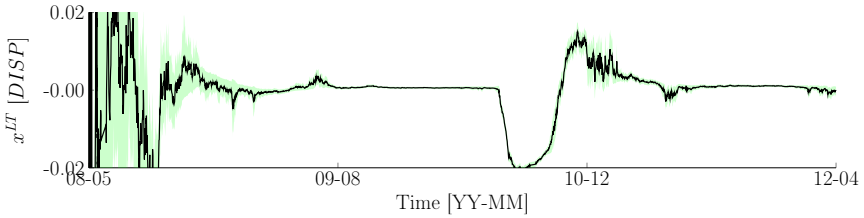
(a) Observed and estimated displacement data



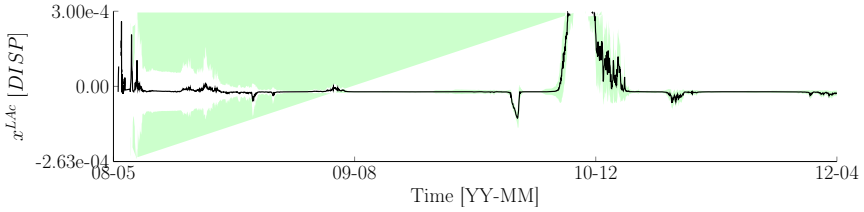
(b) Estimated probability of anomaly (probability of model class 2)



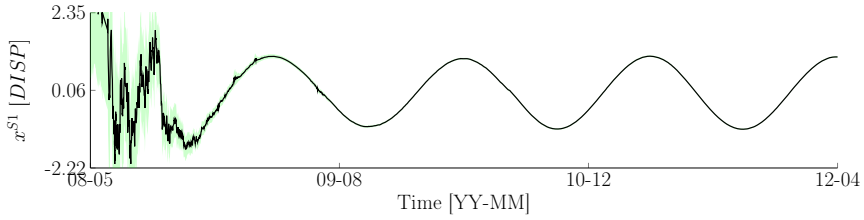
(c) Estimated displacement local level component



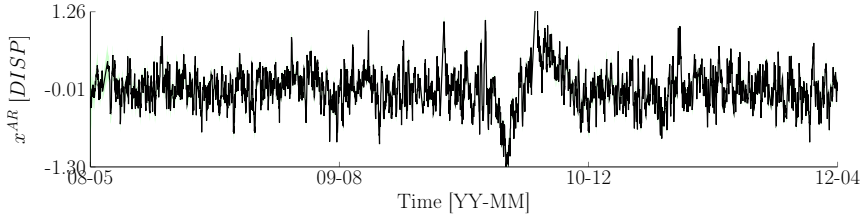
(d) Estimated displacement local trend component.



(e) Estimated displacement local acceleration component.



(f) Estimated displacement yearly periodic component (first hidden state)



(g) Estimated displacement autoregressive component

Figure 25: Estimated results using OpenBDLM default model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 24a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

21.8 Step 9: estimate the initial hidden states

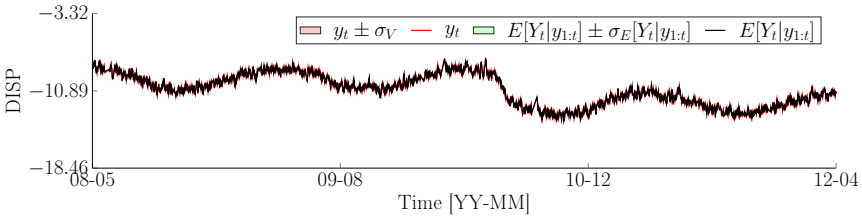
Type `OpenBDLM_main('CFG_Example_DISP_ANOMALY_optim.m');` in the MATLAB command line. Then, type `2`, to optimize the initial hidden states value. The optimized initial hidden states mean, covariance and model probability values for model class 1 are

$$\begin{aligned} \boldsymbol{\mu}_0^{1,*} &= [-9.43, -0.0109, -5.03 \times 10^{-9}, 1.08, -0.00634, -0.548]^\intercal, \text{and} \\ \text{diag}(\boldsymbol{\Sigma}_0^{1,*}) &= [0.139, 0.000877, 2.26, 0.00224, 0.00105, 0.32], \text{and} \\ \pi_0^{1,*} &= 0.997, \end{aligned}$$

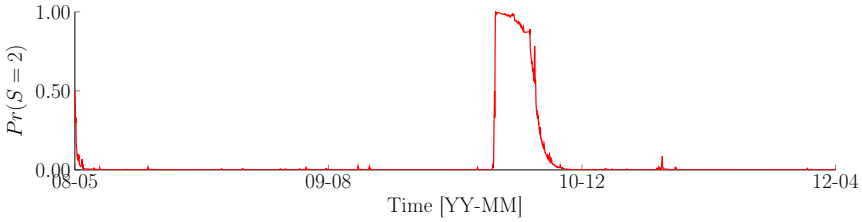
respectively. The optimized initial hidden states mean, covariance and model probability values for model class 2 are

$$\begin{aligned} \boldsymbol{\mu}_0^{2,*} &= [-9.53, 0.102, -0.0599, 1.08, -0.00636, -0.482]^\intercal, \text{and} \\ \text{diag}(\boldsymbol{\Sigma}_0^{2,*}) &= [0.674, 0.418, 0.803, 0.00224, 0.00105, 0.759], \text{and} \\ \pi_0^{2,*} &= 0.00279, \end{aligned}$$

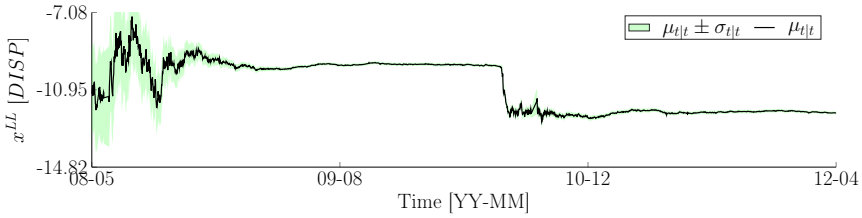
respectively. Once it is done, type `3`, and then `1` to compute the filtered hidden states using the optimized model parameters and optimized initial



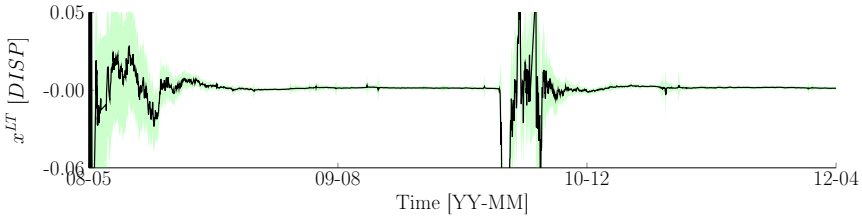
(a) Observed and estimated displacement data



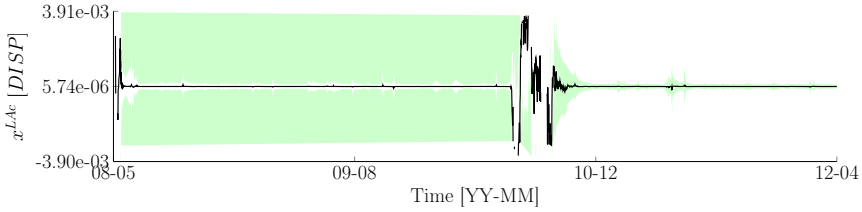
(b) Estimated probability of anomaly (probability of model class 2)



(c) Estimated displacement local level component

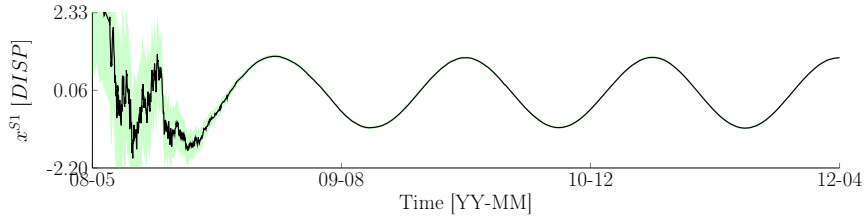


(d) Estimated displacement local trend component.

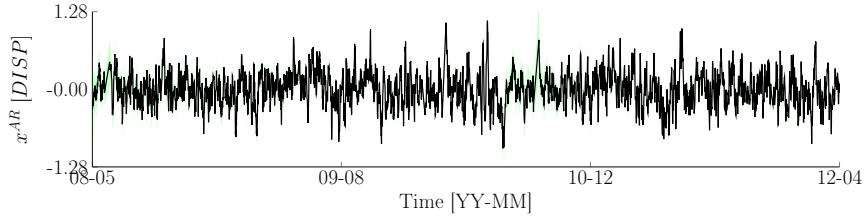


(e) Estimated displacement local acceleration component.

hidden states. The value of the log-likelihood is 188. The estimated hidden



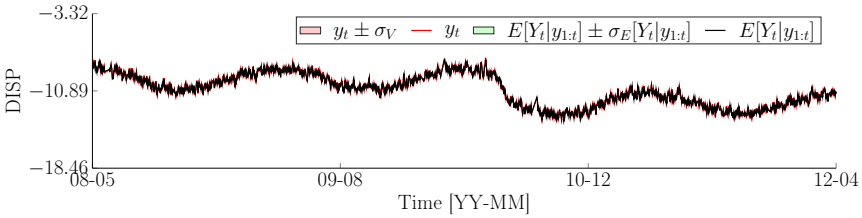
(f) Estimated displacement yearly periodic component (first hidden state)



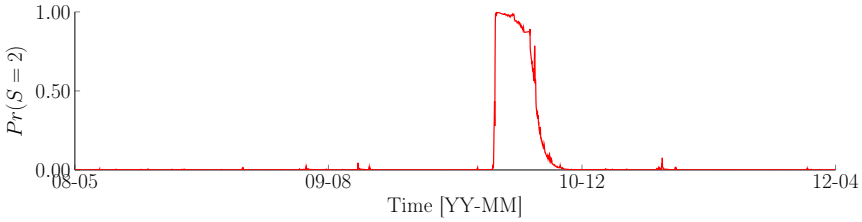
(g) Estimated displacement autoregressive component

Figure 26: Estimated results using OpenBDLM optimized model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 24a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

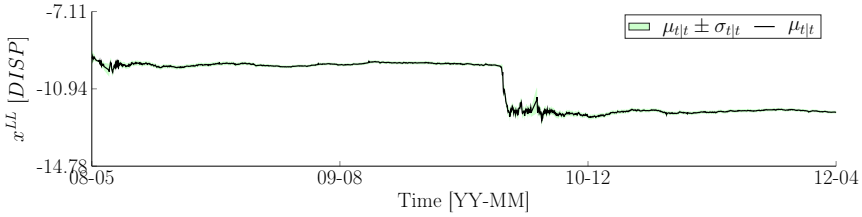
states are presented in Figure 27.



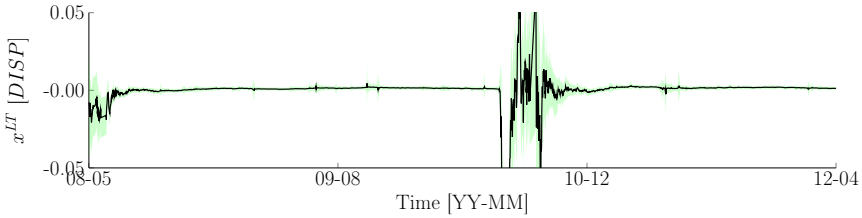
(a) Observed and estimated displacement data



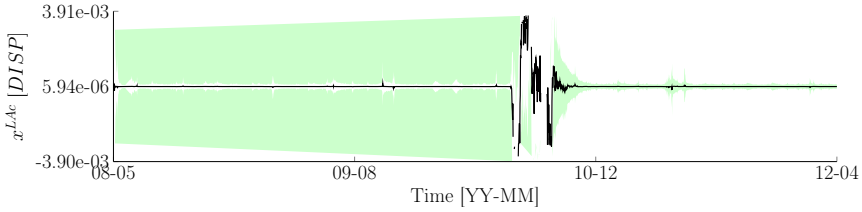
(b) Estimated probability of anomaly (probability of model class 2)



(c) Estimated displacement local level component



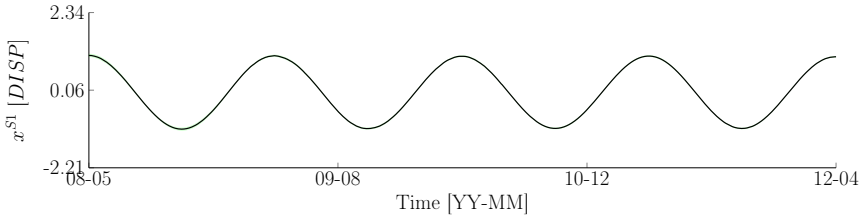
(d) Estimated displacement local trend component.



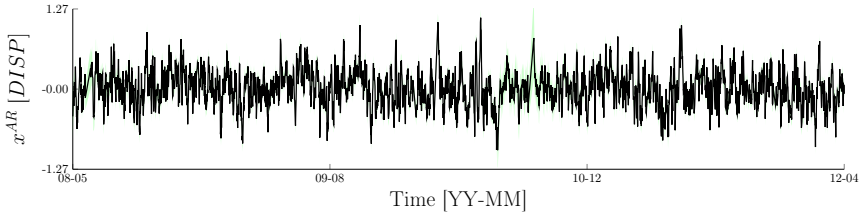
(e) Estimated displacement local acceleration component.

```
- How many model classes do you want for each time-series?
    choice >> 2

    ___________________________________
    BDLM Component reference numbers
    ___________________________________
    11: Local level
    12: Local trend
    13: Local acceleration
    21: Local level compatible with local trend
    22: Local level compatible with local acceleration
    23: Local trend compatible with local acceleration
    31: Periodic
    41: Autoregressive process (AR(1))
    51: Kernel regression
    61: Level Intervention
    ___________________________________

- Model class # 1

- Identify components for time series #1; e.g. [11 31 41]
    choice >> [23 31 41]

- Model class # 2

- Identify components for time series #1; e.g. [11 31 41]
    choice >> [13 31 41]

- Identify the shared parameters between the components of the model
   classes 1 and 2 for time  series #1 ; e.g. [0 1 1]
    choice >> [0 1 1]

    Building model...
    Saving project...
    Project saved in saved_projects/PROJ_Example_DISP_ANOMALY.mat.
    Printing configuration file...
    Saving data...

    Database saved in data/mat/DATA_Example_DISP_ANOMALY.mat
    Configuration file saved in config_files/CFG_Example_DISP_ANOMALY
        .m.
```

Listing 19: MATLAB command window output during model configuration



(f) Estimated displacement yearly periodic component (first hidden state)



(g) Estimated displacement autoregressive component

Figure 27: Estimated results using OpenBDLM optimized model parameters and optimized initial hidden states. The hidden states are estimated from the data presented in Figure 24a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

22 Step-by-step example: generate and analyze synthetic data

The objective is to create synthetic time series data with an acceleration stationary baseline, and a yearly periodic pattern as well as a autoregressive process superimposed into it. This example corresponds to the OpenBDLM demo presented in Section 3.

22.1 Step 1: start a project

First, choose the interactive tool by typing `0`. Secondly, provide a project name (i.e. `Example_SYNTHETIC`). Then, answer `yes` to indicate that you would like to create synthetic data. Finally, type `1` to create one single synthetic time series.

22.2 Step 2: define the time step vector

Type `2000-01-01`, `2005-01-01` and `1`, to provide the date corresponding of the first and last data sample of the synthetic data, as well as the timestep in days.

22.3 Step 3: configure the model

First, the program requests the number of model class. In this example, we would like to create synthetic data with no anomaly. In such case, we type `1` to choose a single model class. Then, OpenBDLM asks for the type of block component. As mentionned earlier, the objective is to create synthetic time series data with an acceleration stationary baseline, and a yearly periodic pattern superimposed into it. Therefore, we choose `[13 31 41]`. This model considers a trend stationary model with a yearly periodic pattern, and an autoregressive process to be more realistic. At this time, three figures that represent the amplitude, timestep and availability of the newly created synthetic data (as shown in Figure 28) should popup on the screen. Type `Q` to save and quit.

22.4 Step 4: explore the model

From the main menu, type `11` to see what are the values of model parameters which have been assigned to create the synthetic data. The

model totalizes 6 model parameters, that is

$$\boldsymbol{\theta} = \{\sigma_{w,\mathrm{D}}^{LA}, p_{\mathrm{D}}^{\mathrm{PD1}}, \sigma_{w,\mathrm{D}}^{\mathrm{PD1}}, \phi_{\mathrm{D}}^{AR}, \sigma_{w,\mathrm{D}}^{AR}, \sigma_{v,\mathrm{D}}\}$$

The default model parameters values are

$$\boldsymbol{\theta}^{\mathrm{default}} = \{1 \times 10^{-8}, 365.24, 0, 0.75, 0.01, 0.01\},$$

in agreement with the values indicated in Table 1. Type `R` to return to the main menu. Then, type `12` the see that the default initial hidden states mean, covariance and model probability values are

$$\begin{aligned} \boldsymbol{\mu}_0^{1,\mathrm{default}} &= [10, -1 \times 10^{-5}, -0.001, 10, 10, 0]^\intercal, \text{and} \\ \mathrm{diag}(\boldsymbol{\Sigma}_0^{1,\mathrm{default}}) &= [0.01, 0.01, 0.01, 0.04, 0.04, 0.01], \text{and} \\ \pi_0^{1,\mathrm{default}} &= 1. \end{aligned}$$

respectively, in agreement with the values indicated in Table 1. Type `R` to return to the main menu.

22.5 Step 5: estimate the hidden states

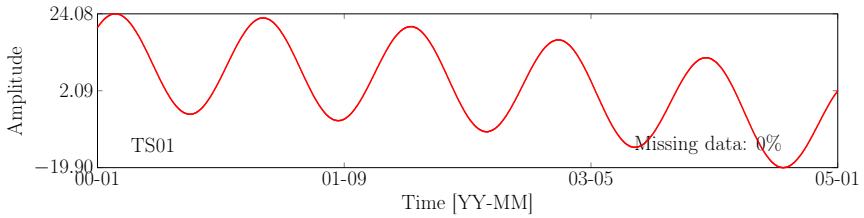
From the main menu, type `3`, then `1` to estimate the filtered hidden states using the default model parameters and default initial hidden states values. The value of the log-likelihood is 4976, and the estimated hidden states are presented in Figure 29. For each figure, the red dashed line represents the true known values of the hidden states.

22.6 Step 6: estimate the initial hidden states

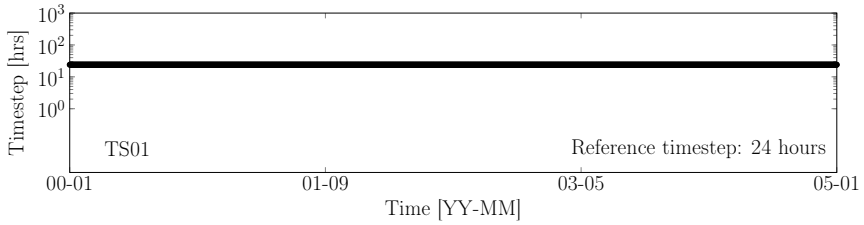
From the main menu, type `2`, to optimize the initial hidden states value. The estimated initial hidden states mean and covariance values are

$$\begin{aligned} \boldsymbol{\mu}_0^* &= [10, -0.00103, -9.68 \times 10^{-6}, 10, 10, -0.0106]^\intercal, \text{and} \\ \mathrm{diag}(\boldsymbol{\Sigma}_0^*) &= [2.35 \times 10^{-5}, 1.33 \times 10^{-9}, 3.3 \times 10^{-14}, 1.9 \times 10^{-6}, 2.03 \times 10^{-6}, 0.000353], \end{aligned}$$

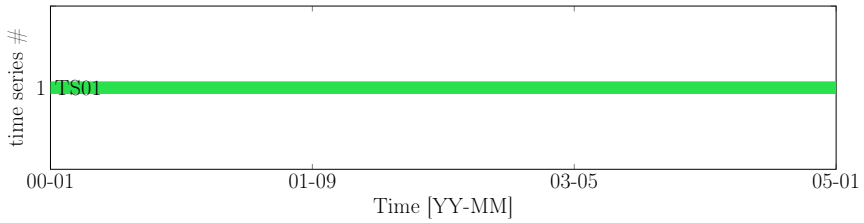
respectively. Once it is done, type `3`, and then `1` to compute the filtered hidden states using the optimized model parameters and optimized initial hidden states. The value of the log-likelihood is 5011. The estimated hidden states are presented in Figure 30.



(a) Amplitude



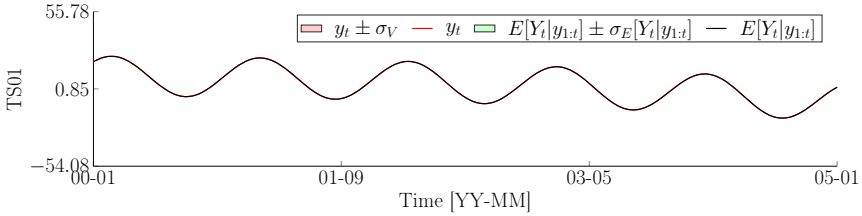
(b) Timestep



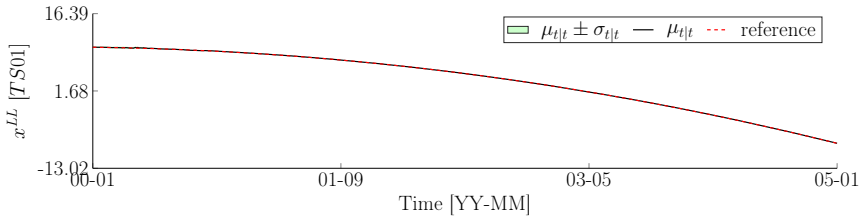
(c) Availability

Figure 28: Data used in Section 22

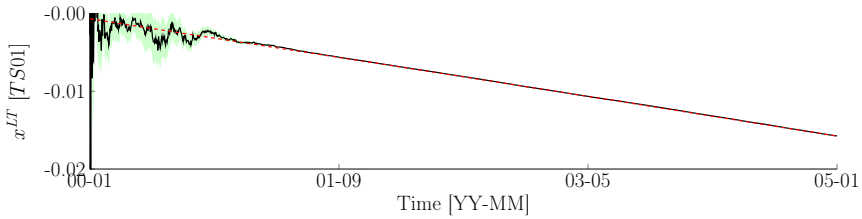
.



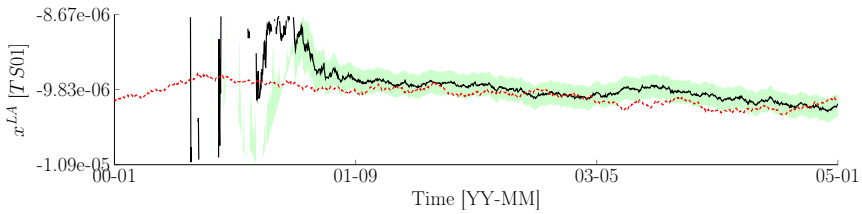
(a) Observed and estimated displacement data



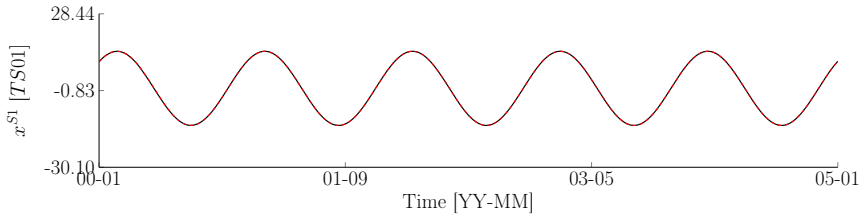
(b) Estimated displacement local level component



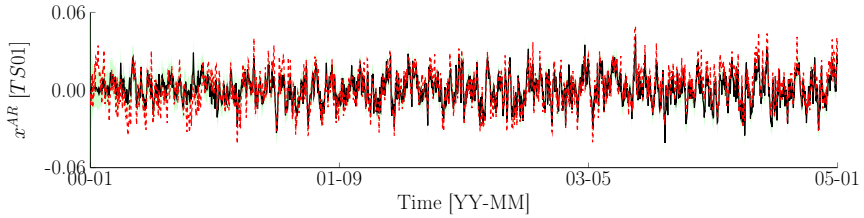
(c) Estimated displacement local trend component.



(d) Estimated displacement local acceleration component.

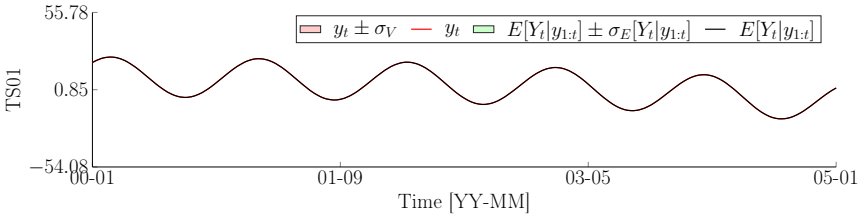


(e) Estimated displacement yearly periodic component (first hidden state)

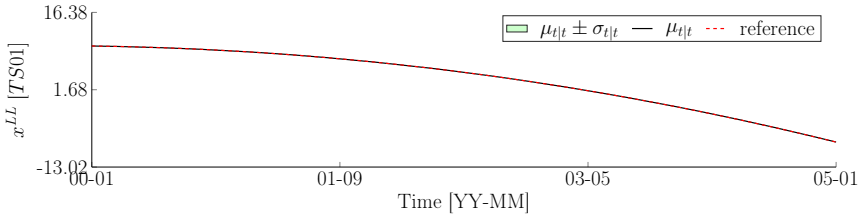


(f) Estimated displacement autoregressive component

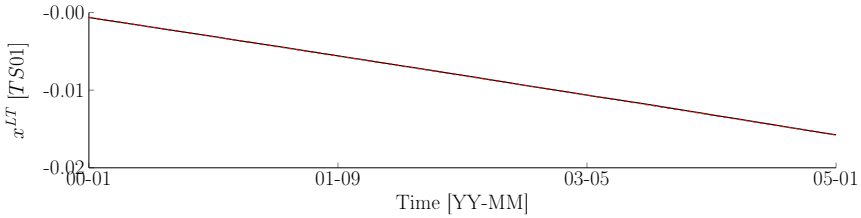
Figure 29: Estimated results using OpenBDLM default model parameters and default initial hidden states. The hidden states are estimated from the data presented in Figure 28a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively. The red dashed line represent the true the hidden state value.



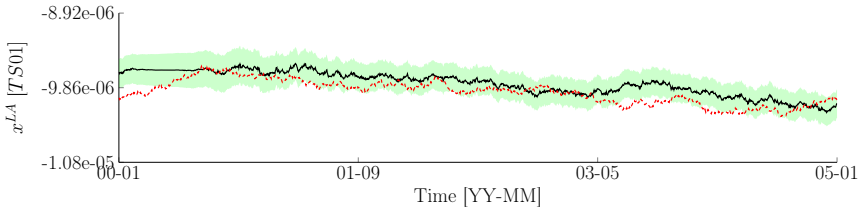
(a) Observed and estimated displacement data



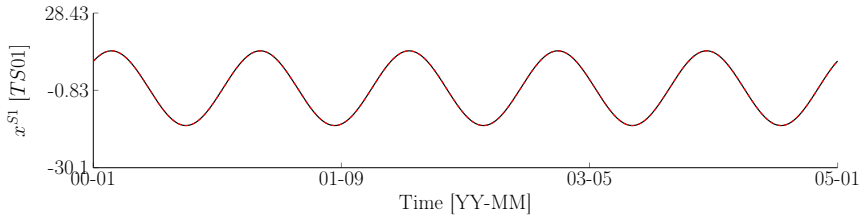
(b) Estimated displacement local level component



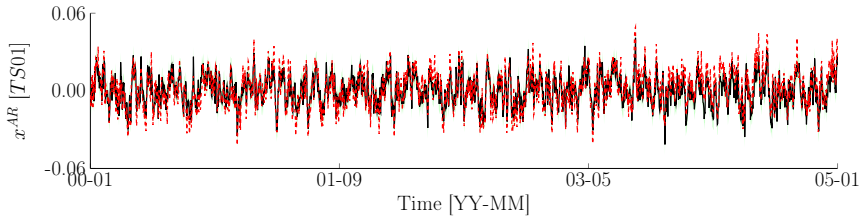
(c) Estimated displacement local trend component.



(d) Estimated displacement local acceleration component.



(e) Estimated displacement yearly periodic component (first hidden state)



(f) Estimated displacement autoregressive component

Figure 30: Estimated results using OpenBDLM default model parameters and optimized initial hidden states. The hidden states are estimated from the data presented in Figure 28a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively. The red dashed line represent the true the hidden state value.

23 FAQ Troubleshooting

To be completed

24 Reference Theory

This section presents a summary of the theory behind Bayesian Dynamic Linear Models. For in-depth details, the reader should consult the following references:

A Kernel-based Method for Modeling Non-Harmonic Periodic Phenomena in Bayesian Dynamic Linear Models
Nguyen, L.H., Gaudot, I., Shervin Khazaeli and Goulet, J.-A.
Frontiers in Built Environment. Vol. X, Issue X, pp, 2019
[PDF] [ ] [ ] [ ] [1]

Uncertainty quantification for model parameters and hidden state variables in bayesian dynamic linear models
Nguyen, L.H., Gaudot, I., and Goulet, J.-A.
Structural Control and Health Monitoring. Vol. X, Issue X, pp.e2136, 2018
[PDF] [ ] [ ] [DOI link] [2]

Anomaly Detection with the Switching Kalman Filter for Structural Health Monitoring
Nguyen, L.H. and Goulet, J.-A.
Structural Control and Health Monitoring. Vol. 24, Issue 4, pp.e2136, 2018
[PDF] [Endnote] [BibTeX] [DOI link] [3]

Structural health monitoring with dependence on non-harmonic periodic hidden covariates
Nguyen, L.H. and Goulet, J.-A.
Engineering Structures, 166:187 ÃŘ 194., 2018
[PDF] [Endnote] [BibTeX] [DOI link] [4]

Empirical validation of Bayesian Dynamic Linear Models in the context of Structural Health Monitoring
Goulet, J.-A. and Koo, K.
Journal of Bridge Engineering. Vol. 23, Issue 2, pp. 05017017, 2018
[PDF] [Endnote] [BibTeX] [DOI link] [5]

Bayesian dynamic linear models for structural health monitoring
Goulet, J.-A.
Structural Control and Health Monitoring. Vol. 24, Issue 12, pp.e2025, 2017
[PDF] [Endnote] [BibTeX] [DOI link] [6]

24.1 Linear gaussian state-space model

OpenBDLM builds on Bayesian dynamic linear models (BDLMs). Bayesian dynamic linear models [7] are a class of linear gaussian state-space models which can be described from the transition and the observation equations. The transition equation describes the dynamics of the system, and is formulated as

$$\mathbf{x}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{w}_t, \quad \begin{cases} \mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \\ \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t), \end{cases} \tag{1}$$

where, for each each time $t = 1, \ldots, \mathtt{T}$, the variables $\mathbf{x}_t$ follow a Gaussian distribution with mean $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$, $\mathbf{A}_t$ is the transition matrix, and $\mathbf{w}_t$ represents Gaussian model errors with zero mean and covariance matrix $\mathbf{Q}_t$. The variables $\mathbf{x}_t$ are usually referred to as hidden states because they are not directly observed. The relationship between the observations $\mathbf{y}_t$ and the hidden states $\mathbf{x}_t$ is given by the observation equation, such as

$$\mathbf{y}_t = \mathbf{C}_t \mathbf{x}_t + \mathbf{v}_t, \quad \{ \ \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t), \tag{2}$$

where $\mathbf{C}_t$ is the observation matrix, and $\mathbf{v}_t$ is the Gaussian measurement error with zero mean and covariance matrix $\mathbf{R}_t$. BDLMs are capable of analyzing multiple time series simultaneously. In case of dependencies between the time series, regression coefficients are added in $\mathbf{C}_t$ (see Section 24.7 and [6]). One particularity of BDLMs is their capacity to update the current estimated state with the current observations, thus allowing to perform online state inference of non-stationary time series.

24.2 Kalman filter & UD filters

The analytical solutions for the prediction, observation and update step are available through either the Kalman filter (KF) or the UD filter, which can be expressed in its short form as:

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \mathcal{L}_t) = \text{Filter}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \mathbf{y}_t, \mathbf{A}_t, \mathbf{Q}_t, \mathbf{C}_t, \mathbf{R}_t) \tag{3}$$

where $\mathcal{L}_t$ is the marginal likelihood describing the probability of observing observations $\mathbf{y}_t$ at time t given all the observations up to time $t-1$ [8]. Note that the UD and Kalman filter are two different methods for calculating the same results. On one hand, the Kalman filter is faster and computationally

simpler to implement and on the other hand, the UD filter is more robust toward numerical instabilities.
The standard Kalman filter expressed in Eq. 3 can process stationary, trend stationary, and acceleration stationary time series, but it is not capable of handling non-stationary time-series, which is needed when it comes to anomaly detection. The generalization of the Kalman Filter for non-stationary time-series is found in the Switching Kalman filter (SKF) equations.

24.3 Switching Kalman filter

We may be interested in anomaly detection, that is, modelling and detecting the changes of behavior due to changes in the dynamics of the baseline response of the time-series. One way to model changing dynamics is to run in parallel a collection of $\mathtt{S}$ linear models, each having their own system dynamics $\mathbf{A}_t$ and $\mathbf{Q}_t$. In such approach, a discrete markovian switching variable $s_t = 1, .., j, .., \mathtt{S}$ with a transition probabilities matrix $\mathbf{Z}_t$ and probabilities $\boldsymbol{\pi}_t$ is introduced to indicate which dynamics is used at time t. The problem of incorporating switching dynamics into the model is that the state vector grows in a way that the dimension of the state vector at time t is $\mathtt{S}^t$. Therefore, the estimation quickly becomes intractable. One solution is to merge at each time t the states sharing the same dynamics using gaussian mixture. This technique, known as the Switching Kalman filter, allows to keep the dimension of the state vector equal to S at each time t [9]. The SKF algorithm can be divided into two successive steps, (i) the "Filter" and, (ii) the "Collapse" step. Following the notation used in Eq. 3 the first step can be expressed in its short form as:

$$(\boldsymbol{\mu}_{t|t}^{i(j)}, \boldsymbol{\Sigma}_{t|t}^{i(j)}, \mathcal{L}_t^{i(j)}) = \text{Filter}(\boldsymbol{\mu}_{t-1|t-1}^{i}, \boldsymbol{\Sigma}_{t-1|t-1}^{i}, \mathbf{y}_t, \mathbf{A}_t^j, \mathbf{Q}_t^{i(j)}, \mathbf{C}_t^j, \mathbf{R}_t^j) \quad (4)$$

where the superscripts $i(j)$ indicates that the current state at time t is $s_t = j$ given the state at time $t-1$ is $s_{t-1} = i$, and $\mathcal{L}_t^{i(j)}$ the marginal likelihood that describes the probability of observing observations $\mathbf{y}_t$ at time t given all the observations up to time $t-1$, and given the state at time t_1 was $s_{t-1} = i$ and that it switches to $s_t = j$ at time t. The state probability $\pi_{t|t}^j$ at each time t is computed from the previous state probabilities $\boldsymbol{\pi}_{t-1|t-1}$, the likelihood $\mathcal{L}_t^{i(j)}$, and the transition probability $Z_t^{i(j)}$, such as

$$\pi_{t|t}^j = \sum_{i=1}^{\mathtt{S}} \frac{\mathcal{L}_t^{i(j)} \pi_{t-1|t-1}^i Z_t^{i(j)}}{c}, \quad (5)$$

where c is a normalization constant ensuring that $\sum_{j=1}^{\mathtt{S}} \pi_{t|t}^{j} = 1$. Moreover, the state switching probability is defined as

$$W_{t-1|t}^{i(j)} = \frac{\mathcal{L}_t^{i(j)} \pi_{t-1|t-1}^{i} Z_t^{i(j)}}{c \pi_{t|t}^{j}}. \tag{6}$$

$W_{t|t-1}^{i(j)}$ are required to perform the "Collapse" step, which can be expressed in its short form as:

$$(\boldsymbol{\mu}_{t|t}^{j}, \boldsymbol{\Sigma}_{t|t}^{j}) = \text{Collapse}(\boldsymbol{\mu}_{t|t}^{i(j)}, \boldsymbol{\Sigma}_{t|t}^{i(j)}, W_{t-1|t}^{i(j)}); \tag{7}$$

where state switching probabilities $W_{t|t-1}^{i(j)}$ are used as weighting factors for the gaussian mixture. From Eq. 7, the SKF algorithm provides a set a $\mathtt{S}$ state vectors at each time t. However, for the ease of interpretation, it is generally more convenient to have a single state vector at each time t. Therefore, we hereafter introduce the "Merge" step. Similarly to the "Collapse" step of the SKF algorithm, the "Merge" step uses the gaussian mixture technique, and it can be expressed in its short form as:

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}) = \text{Merge}(\boldsymbol{\mu}_{t|t}^{j}, \boldsymbol{\Sigma}_{t|t}^{j}, \pi_{t|t}^{j}); \tag{8}$$

where the state probabilities $\pi_{t|t}^{j}$ is used as weighting factors for the gaussian mixture [3].

24.4 Model parameter estimation

The matrices $\mathbf{A}_t$, $\mathbf{Q}_t$, $\mathbf{C}_t$ and $\mathbf{R}_t$ depend on a set of model parameters $\boldsymbol{\theta}$. In most cases, $\boldsymbol{\theta}$ are unknown, and they can be learned from a training dataset $\mathbf{y}_{1:\mathtt{Tr}}$. The procedure of learning the model parameters is hereafter referred to as model parameters estimation.

24.4.1 Maximum log A Posteriori (MAP)

The log a posteriori probability density function (PDF) is defined as

$$\ln p(\boldsymbol{\theta}|\mathbf{y}_{1:\mathtt{Tr}}) \propto \ln p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}), \tag{9}$$

where $p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta})$ is the likelihood, $p(\boldsymbol{\theta})$ is the prior PDF. The likelihood PDF is the joint prior probability of observations, that is, plausibility of the available observations $\mathbf{y}_{1:\mathtt{Tr}}$ given the parameter vector $\boldsymbol{\theta}$. Assuming that the

observations are independent from each other, the joint log-likelihood function is defined as the sum of the marginal log-likelihoods, such as

$$\ln p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta}) = \sum_{t=1}^{\mathtt{Tr}} \ln p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \boldsymbol{\theta}) = \sum_{t=1}^{\mathtt{Tr}} \ln \left[\sum_{j=1}^{\mathtt{S}} \sum_{i=1}^{\mathtt{S}} \mathcal{L}_t^{i(j)} \pi_{t-1|t-1}^{i} Z_t^{i(j)} \right], \tag{10}$$

where $\mathcal{L}_t^{i(j)}$ and $\pi_{t-1|t-1}^{i}$ are computed at each time t from the Switching Kalman Filter; $\mathtt{S}$ is the total number of model class, and the values of $Z_t^{i(j)}$ are known from the current set of model parameters. The maximum log a posteriori procedure consists in identifying the point estimates by maximizing the log A Posteriori PDF, such as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left[\ln p(\boldsymbol{\theta}|\mathbf{y}_{1:\mathtt{Tr}}) \right],$$

where $\boldsymbol{\theta}^*$ are the optimized model parameters values.

24.4.2 Maximum log Likelihood Estimation (MLE)

The Maximum log Likelihood Estimation (MLE) is a special case of the MAP where the prior PDF $p(\boldsymbol{\theta})$ is assumed to be uniform [10]. Therefore, the Maximum log Likelihood procedure consists in identifying the point estimates by maximizing the log likelihood PDF, such as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left[\ln p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta}) \right],$$

where $\boldsymbol{\theta}^*$ are the optimized model parameters values.

24.4.3 Laplace Approximation

The MAP and MLE are point estimation methods which do not take into account the uncertainty in the parameter estimates $\boldsymbol{\theta}^*$. The estimation of the uncertainties in the model parameters estimates can be addressed using the Laplace approximation [10] so that

$$p(\boldsymbol{\theta}|\mathbf{y}_{1:\mathtt{Tr}}) \approx \mathcal{N}\left(\boldsymbol{\theta}; \boldsymbol{\theta}^*, -\mathbf{H}(\boldsymbol{\theta}^*)^{-1}\right)$$

where $\mathbf{H}(\boldsymbol{\theta}^*)$ is the second derivative of the log a posteriori or log likelihood PDF evaluated at the optimal parameter values $\boldsymbol{\theta}^*$.

24.4.4 Parameter-wise Newton-Raphson

The parameter-wise Newton-Raphson [10] algorithm is an iterative approach which can be used to find the model parameters that correspond to the maximum of a target PDF, hereafter noted $\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta})$. The function $\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta})$ is either the log a posteriori or the log likelihood PDF. One iteration of the Newton-Raphson algorithm is

$$\theta^i_{\text{new}} = \theta^i_{\text{old}} - \lambda \frac{\nabla \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})}{\nabla^2 \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})}, \tag{11}$$

where i is the index of the parameter being learned, λ is the learning rate, ∇ the first derivative, ∇^2 the second derivative. $\boldsymbol{\theta}_{\text{old}}$ and $\boldsymbol{\theta}_{\text{new}}$ are the previous and updated vector of model parameters. One parameter is updated at each iteration. The convergence of each model parameters is reached when the following conditions are satisfied

$$\begin{cases} \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}}) & < & \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{new}}) \\ \left|\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{new}}) - \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})\right| & \leqslant & \tau \cdot \left|\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})\right| \end{cases}, \tag{12}$$

where τ is a termination tolerance. The Newton-Raphson algorithm stops when each model parameters has reached the convergence. In most cases. the derivatives of $\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta})$ can not be computed analytically, and the derivatives are approximated numerically using the central differentiation scheme, such as

$$\begin{aligned} \nabla \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i) &= \frac{\partial \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta})}{\partial \theta^i} \approx \frac{\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta} + \Pi(i)\Delta\theta^i) - \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta} - \Pi(i)\Delta\theta^i)}{2\Delta\theta^i} \\ \nabla^2 \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i) &= \frac{\partial^2 \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta})}{\partial^2 \theta^i} \approx \frac{\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta} + \Pi(i)\Delta\theta^i) - 2\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}) + \mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta} - \Pi(i)\Delta\theta^i)}{(\Delta\theta^i)^2}, \end{aligned} \tag{13}$$

where $\Delta\theta^i$ is a small perturbation to the value of the i^{th} model parameters and $\Pi(i)$ is an indicator vector for which all values are equal to 0, except the i^{th} value which is equal to one.

24.4.5 Stochastic gradient descent

To be completed

24.4.6 Model parameter space transformation

There are some model parameters which are defined in a bounded interval. For instance, the standard deviation model parameters are real numbers that lie in the $[0, +\infty]$ interval. Therefore, during the learning procedure, it may happen that new model parameters $\boldsymbol{\theta}_{\text{new}}$ are proposed outside their valid interval. Those parameters must be rejected, which strongly hinders the computational efficiency of the learning algorithm. The solution employed in OpenBDLM is to transform the bounded parameters space into an unbounded parameters space, where the parameters lie in the $[-\infty, +\infty]$ interval, and to perform the learning procedure in the transformed, unbounded, space. The transformation is done using a function $g(.)$ so that,

$$\theta^{i,\text{tr}} = g(\theta^i), \quad \theta^{i,\text{tr}} \in [-\infty, +\infty]. \tag{14}$$

The choice of the function $g(.)$ depends on the bound of θ. Three cases generally occur:

- $\theta \in [-\infty, +\infty]$, $g(\theta) = 1$, so that $\theta^{\text{tr}} = \theta$ and $\theta = \theta^{\text{tr}}$
- $\theta \in [0, +\infty]$, $g(\theta) = \ln(\theta)$, so that $\theta^{\text{tr}} = \ln(\theta)$ and $\theta = e^{\theta^{\text{tr}}}$
- $\theta \in [\text{a}, \text{b}]$, $g(\theta) = \text{sigmoid}(\theta)$, so that $\theta^{\text{tr}} = -\ln\left(\frac{b-a}{\theta-a} - 1\right)$, and $\theta = \left(\frac{b-a}{1+e^{-\theta^{\text{tr}}}} + a\right)$

For instance, the standard deviation model parameters are real numbers that lie in the $[0, +\infty]$ interval and the logarithm transformation is used. Moreover, the autoregression coefficient model parameters are real numbers that lie in the $[0, 1]$ interval, and the sigmoid transformation is used.

24.5 Block components

The block components are pieces of the full model. Each block component is used to describe a given dynamics for a given time-series. Therefore, each block component has its own transition and observation model, which are associated with some model parameters. Each block component can be associated with one or more hidden states variables. The block components are then assembled to build the full model. The block components associated with irreversible change in the time series belongs to the *baseline* component. The other block components are associated with reversible change in the time-series. The *compatible* block component are needed to model switching dynamics in the baseline of the time-series.

24.5.1 Local level (baseline)

The local level block component describes the local mean of a stationary time-series (no trend and no acceleration) [6]. The local level describes irreversible changes.
Number of hidden states: 1
Hidden states vector: $\mathbf{x}^{\mathtt{LL}} = [x^{\mathtt{LL}}]$
Transition matrix: $\mathbf{A}^{\mathtt{LL}} = [1]$
Observation matrix: $\mathbf{C}^{\mathtt{LL}} = [1]$
Process noise covariance matrix: $\mathbf{Q}^{\mathtt{LL}} = [(\sigma_w^{\mathtt{LL}})^2]$
Model parameters: $\boldsymbol{\theta}^{\mathtt{LL}} = [\sigma_w^{\mathtt{LL}}]$
$\sigma_w^{\mathtt{LL}}$ is the process noise standard deviation which can be learned from the data.

24.5.2 Local trend (baseline)

The local trend block component describes the local mean of a trend-stationary time-series (trend and no acceleration) [6]. The local level describes irreversible changes.
Number of hidden states: 2
Hidden states vector: $\mathbf{x}^{\mathtt{LT}} = [x^{\mathtt{LL}}, x^{\mathtt{LT}}]^\intercal$
Transition matrix: $\mathbf{A}^{\mathtt{LT}} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$
Observation matrix: $\mathbf{C}^{\mathtt{LT}} = [1, 0]$
Process noise covariance matrix: $\mathbf{Q}^{\mathtt{LT}} = (\sigma_w^{\mathtt{LT}})^2 \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}$
Model parameters: $\boldsymbol{\theta}^{\mathtt{LT}} = [\sigma_w^{\mathtt{LT}}]$
$\sigma_w^{\mathtt{LT}}$ is the process noise standard deviation, which can be learned from the data, and Δt is the local timestep computed from the data.

24.5.3 Local acceleration (baseline)

The local acceleration block component describes the local mean of a acceleration-stationary time-series [6]. It describes irreversible changes.
Number of hidden states: 3
Hidden states vector: $\mathbf{x}^{\mathtt{LA}} = [x^{\mathtt{LL}}, x^{\mathtt{LT}}, x^{\mathtt{LA}}]^\intercal$
Transition matrix: $\mathbf{A}^{\mathtt{LA}} = \begin{bmatrix} 1 & \Delta t & \Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix}$
Observation matrix: $\mathbf{C}^{\mathtt{LA}} = [1, 0, 0]$

Process noise covariance matrix: $\mathbf{Q}^{\texttt{LA}} = (\sigma_w^{\texttt{LA}})^2 \begin{bmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}$

Model parameters: $\boldsymbol{\theta}^{\texttt{LA}} = [\sigma_w^{\texttt{LA}}]$

$\sigma_w^{\texttt{LA}}$ is the process noise standard deviation, which can be learned from the data, and Δt is the local timestep computed from the data.

24.5.4 Local level trend compatible (baseline)

The local level trend compatible component must be used in case of model switching between a local level model and a local trend model [3]. The local level trend compatible block component describes the local mean of a stationary time-series. It describes irreversible changes.

Number of hidden states: 1

Hidden states vector: $\mathbf{x}^{\texttt{LLTc}} = [x^{\texttt{LL}}, 0]^\intercal$

Transition matrix: $\mathbf{A}^{\texttt{LLTc}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

Observation matrix: $\mathbf{C}^{\texttt{LLTc}} = [1, 0]$

Process noise covariance matrix: $\mathbf{Q}^{\texttt{LLTc}} = (\sigma_w^{\texttt{LLTc}})^2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

Model parameters: $\boldsymbol{\theta}^{\texttt{LLTc}} = [\sigma_w^{\texttt{LLTc}}]$

$\sigma_w^{\texttt{LLTc}}$ is the process noise standard deviation, which can be learned from the data, and Δt is the local timestep computed from the data.

24.5.5 Local level acceleration compatible (baseline)

The local level acceleration compatible component must be used in case of model switching between a local level model and a local acceleration model [3]. The local level acceleration compatible block component describes the local mean of a stationary time-series. It describes irreversible changes.

Number of hidden states: 1

Hidden states vector: $\mathbf{x}^{\texttt{LLAc}} = [x^{\texttt{LL}}, 0, 0]^\intercal$

Transition matrix: $\mathbf{A}^{\texttt{LLAc}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Observation matrix: $\mathbf{C}^{\texttt{LLAc}} = [1, 0, 0]$

Process noise covariance matrix: $\mathbf{Q}^{\texttt{LLAc}} = (\sigma_w^{\texttt{LLAc}})^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Model parameters: $\boldsymbol{\theta}^{\texttt{LLAc}} = [\sigma_w^{\texttt{LLAc}}]$

$\sigma_w^{\texttt{LLAc}}$ is the process noise standard deviation, which can be learned from the data, and Δt is the local timestep computed from the data.

24.5.6 Local trend acceleration compatible (baseline)

The local trend acceleration compatible component must be used in case of model switching between a local trend model and a local acceleration model [3]. The local trend acceleration compatible block component describes the local mean of a trend-stationary time-series. It describes irreversible changes.
Number of hidden states: 2
Hidden states vector: $\mathbf{x}^{\texttt{LTAc}} = [x^{\texttt{LL}}, x^{\texttt{LT}}, 0]^\intercal$
Transition matrix: $\mathbf{A}^{\texttt{LTAc}} = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Observation matrix: $\mathbf{C}^{\texttt{LTAc}} = [1, 0, 0]$
Process noise covariance matrix: $\mathbf{Q}^{\texttt{LTAc}} = (\sigma_w^{\texttt{LTAc}})^2 \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} & 0 \\ \frac{\Delta t^2}{2} & \Delta t & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Model parameters: $\boldsymbol{\theta}^{\texttt{LTAc}} = [\sigma_w^{\texttt{LTAc}}]$
$\sigma_w^{\texttt{LTAc}}$ is the process noise standard deviation, which can be learned from the data, and Δt is the local timestep computed from the data.

24.5.7 Periodic (Fourier form)

The periodic (Fourier form) block component describes a periodic pattern in the time-series using Fourier form [7, 6]. The periodic Fourier form allows modelling sine-like periodic pattern in time-series. It describes reversible changes.
Number of hidden states: 2
Hidden states vector: $\mathbf{x}^{\texttt{P}} = [x^{\texttt{P}_1}, x^{\texttt{P}_2}]^\intercal$
Transition matrix: $\mathbf{A}^{\texttt{P}} = \begin{bmatrix} \cos\omega & \sin\omega \\ -\sin\omega & \cos\omega \end{bmatrix}$
Observation matrix: $\mathbf{C}^{\texttt{P}} = [1, 0]$
Process noise covariance matrix: $\mathbf{Q}^{\texttt{P}} = (\sigma_w^{\texttt{P}})^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Model parameters: $\boldsymbol{\theta}^{\texttt{P}} = [\sigma_w^{\texttt{P}}, p^{\texttt{P}}]$
$\sigma_w^{\texttt{P}}$ is the process noise standard deviation and p the period in days, which can be learned from the data, and Δt is the local timestep computed from the data. $\omega = \frac{2\pi\Delta t}{p}$ is the angular of frequency defined from the period p, given in days.

24.5.8 Periodic (Kernel regression form)

The periodic (Kernel regression form) block component describes a periodic pattern in the time-series using periodic kernel regression [1]. The periodic Kernel regression form allows modelling form-free periodic pattern in time-series. It describes reversible changes. The periodic kernel measures the similarity between pairs of covariates, and it is defined as

$$k(t_i, t_j) = \exp\left[-\frac{2}{\ell^2}\sin\left(\pi\frac{t_i - t_j}{p}\right)^2\right].$$

The kernel output $k(t_i, t_j) \in (0, 1)$ measures the similarity between two timestamps t_i and t_j as a function of the distance between these, as well as a function of two parameters; the period and kernel length, $\boldsymbol{\theta} = \{p, \ell\}$.
Number of hidden states: $\mathtt{L}^{\mathtt{KR}} + 1$
Hidden states vector: $\mathbf{x}^{\mathtt{KR}} = [x_0^{\mathtt{KR}}, x_1^{\mathtt{KR}}, \ldots, x_{\mathtt{L}^{\mathtt{KR}}}^{\mathtt{KR}}]^\intercal$
Transition matrix: $\mathbf{A}^{\mathtt{KR}} = \begin{bmatrix} 0 & \tilde{\boldsymbol{k}}^{\mathtt{KR}}(t, \mathbf{t}^{\mathtt{KR}}) \\ \mathbf{0} & \mathbf{I}_{\mathtt{L}^{\mathtt{KR}}} \end{bmatrix}$
Process noise covariance matrix: $\mathbf{Q}_t^{\mathtt{KR}} = \begin{bmatrix} (\sigma_{w,0}^{\mathtt{KR}})^2 & \mathbf{0} \\ \mathbf{0} & (\sigma_{w,1}^{\mathtt{KR}})^2 \cdot \mathbf{I}_{\mathtt{L}^{\mathtt{KR}}} \end{bmatrix}$,
Observation matrix: $\mathbf{C}^{\mathtt{KR}} = [1, 0, \ldots, 0]$
Model parameters: $\boldsymbol{\theta}^{\mathtt{KR}} = [p^{\mathtt{KR}}, \ell^{\mathtt{KR}}, \sigma_{w,0}^{\mathtt{KR}}, \sigma_{w,1}^{\mathtt{KR}}]$
In the transition matrix, $\tilde{\boldsymbol{k}}^{\mathtt{KR}}(t, \mathbf{t}^{\mathtt{KR}})$ corresponds to the normalized kernel, $k(t, \mathbf{t}^{\mathtt{KR}})/\sum_t k(t, \mathbf{t}^{\mathtt{KR}})$. $\tilde{\boldsymbol{k}}^{\mathtt{KR}}(t, \mathbf{t}^{\mathtt{KR}})$ is parameterized by the kernel width $\ell^{\mathtt{KR}}$, its period $p^{\mathtt{KR}}$, and a vector of $\mathtt{L}^{\mathtt{KR}}$ timestamps $\mathbf{t}^{\mathtt{KR}} = [t_1^{\mathtt{KR}}, \cdots, t_{\mathtt{L}^{\mathtt{KR}}}^{\mathtt{KR}}]$ where each timestamp $t_i^{\mathtt{KR}}$ is associated with a hidden control point value $x_i^{\mathtt{KR}}$. $\sigma_{w,1}^{\mathtt{KR}}$ controls the increase in the variance of the hidden control points between successive time steps and $\sigma_{w,0}^{\mathtt{KR}}$ controls the time-independent process noise in the hidden predicted pattern. $p^{\mathtt{KR}}$ and $\ell^{\mathtt{KR}}$ give the period and correlation length of the kernel, respectively.

24.5.9 First order autoregressive

The first order autoregressive component describes the time-dependent model errors (i.e the residual between the model prediction and the data) [6]. It describes reversible changes.
Number of hidden states: 1
Hidden states vector: $\mathbf{x}^{\mathtt{AR}} = [x^{\mathtt{AR}}]$
Transition matrix: $\mathbf{A}^{\mathtt{AR}} = [\phi^{\mathtt{AR}}]$
Observation matrix: $\mathbf{C}^{\mathtt{AR}} = [1]$

Process noise covariance matrix: $\mathbf{Q}^{\texttt{AR}} = [(\sigma_w^{\texttt{AR}})]$
Model parameters: $\boldsymbol{\theta}^{\texttt{AR}} = [\sigma_w^{\texttt{AR}}, \phi^{\texttt{AR}}]$
$\sigma_w^{\texttt{AR}}$ is the process noise standard deviation, and $\phi^{\texttt{AR}}$ the autoregressive coefficient.

24.6 Handling non-uniform time vector and missing data

LabelSS:HandlingNonUniformMissingData

24.6.1 Non-uniform time vector

Non uniform time vector occurs when the time between two successive data measurements (i.e. the timestep) varies with time. In order to accommodate non-uniform time vector, a reference time step Δt^{ref} must be defined [6]. The reference time-step is a value corresponding to the most frequent time step in the time series. All parameter values in the parameter set $\boldsymbol{\theta}$ are estimated for the reference time step. Therefore, for local time step Δt different than the reference timestep Δt^{ref}, the parameters value must be adapted accordingly. It is proposed to scale the model error standard deviations σ_w in $\mathbf{Q}_t$ proportionally to the ratio between the current time step and the reference time step so that,

$$\sigma_w^{\Delta t} = \sigma_w^{\Delta t^{\text{ref}}} \frac{\Delta t}{\Delta t^{\text{ref}}}.$$

Therefore, the amount of process noise in the prediction model increases as the local time step increase with respect to the reference time step.
The transition matrix $\mathbf{A}^{\texttt{AR}}$ contains the autoregressive coefficients $\phi^{\texttt{AR}}$ that are recursively multiplied with the hidden state at each time step. To account for time step changes, the autoregressive coefficients are elevated to the power of the ratio between the current time step and the reference time step, such as

$$\phi^{\texttt{AR},\Delta t} = (\phi^{\texttt{AR},\Delta t^{\text{ref}}})^{\frac{\Delta t}{\Delta t^{\text{ref}}}}.$$

Therefore, the autocorrelation between successive data samples in the autoregressive prediction model decreases as the local time step increase with respect to the reference time step.

24.6.2 Missing data (NaN)

The presence of missing data (NaN) for specific time impedes to perform the full Kalman computations as the update step can not be performed [6].

However, the prediction step using the current transition model can be done. Therefore, BDLM can fill the gap when data are missing using the transition model.

24.7 Dependencies between time-series

The dependencies between time-series are handled by adding regression coefficients $\phi^{i|j}$ in the observation matrix. For a dataset with `D` time-series, the observation matrix is

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}^1 & \mathbf{C}^c_{1,2} & \cdots & \mathbf{C}^c_{1,j} & \cdots & \mathbf{C}^c_{1,\mathtt{D}} \\ \mathbf{C}^c_{2,1} & \mathbf{C}^2 & \cdots & \mathbf{C}^c_{2,j} & \cdots & \mathbf{C}^c_{2,\mathtt{D}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}^c_{i,1} & \mathbf{C}^c_{i,2} & \cdots & \mathbf{C}^c_{i,j} & \cdots & \mathbf{C}^c_{i,\mathtt{D}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}^c_{\mathtt{D},1} & \mathbf{C}^c_{\mathtt{D},2} & \cdots & \mathbf{C}^c_{\mathtt{D},j} & \cdots & \mathbf{C}^{\mathtt{D}} \end{bmatrix}.$$

The dependence matrix is a matrix with 0 and 1 which is used to indicate which time series have dependencies between each others, such as

$$\mathbf{D} = \begin{bmatrix} 1 & d_{1,2} & \cdots & d_{1,j} & \cdots & d_{1,\mathtt{D}} \\ d_{2,1} & 1 & \cdots & d_{2,j} & \cdots & d_{2,\mathtt{D}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{i,1} & d_{i,2} & \cdots & 1 & \cdots & d_{i,\mathtt{D}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{\mathtt{D},1} & d_{\mathtt{D},2} & \cdots & d_{\mathtt{D},j} & \cdots & 1 \end{bmatrix}.$$

Then,

- if $d_{i,j} = 0$, $\mathbf{C}^c_{i,j} = [\mathbf{0}]$
- if $d_{i,j} = 1$, $\mathbf{C}^c_{i,j} = \left[\phi^{i|j}_1, \phi^{i|j}_2, \cdots, \phi^{i|j}_{k_j}\right]$ where k_j is the number of hidden states associated with the j^{th} time-series.

The regression coefficient $\phi^{i|j}_k$ gives the linear dependence between the k^{th} hidden states of the j^{th} time series and the i^{th} time series. In OpenBDLM, a dependence model between time-series assigns regression coefficient for the observed hidden states associated with block component describing reversible behavior (periodic and autoregressive patterns).

References

[1] Luong Ha Nguyen, Ianis Gaudot, and James-A. Khazaeli, Shervin Goulet. A kernel-based method for modeling non-harmonic periodic phenomena in bayesian dynamic linear models. *Frontiers in Built Environment*, 0(0), 2019.

[2] Luong Ha Nguyen, Ianis Gaudot, and James-A. Goulet. Uncertainty quantification for model parameters and hidden state variables in bayesian dynamic linear models. *Structural Control and Health Monitoring*, 0(0):e2309, 2018. e2309 stc.2309.

[3] L. H. Nguyen and J.-A. Goulet. Anomaly detection with the switching kalman filter for structural health monitoring. *Structural Control and Health Monitoring*, pages e2136–n/a, 2018.

[4] L.H. Nguyen and J-A. Goulet. Structural health monitoring with dependence on non-harmonic periodic hidden covariates. *Engineering Structures*, 166:187 – 194, 2018.

[5] James-A. Goulet and Ki Koo. Empirical validation of bayesian dynamic linear models in the context of structural health monitoring. *Journal of Bridge Engineering*, 23(2):05017017, 2018.

[6] J.-A. Goulet. Bayesian dynamic linear models for structural health monitoring. *Structural Control and Health Monitoring*, 24:e2035–n/a, 2017.

[7] M. West and J. Harrison. *Bayesian Forecasting and Dynamic Models*. Springer Series in Statistics. Springer New York, 1999.

[8] Simo Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

[9] K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[10] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian data analysis*. CRC Press, 3 edition, 2014.