OpenBDLM V1.0 reference manual

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

February 21, 2019

Contents

1 Getting started

1.1 What is OpenBDLM ?

OpenBDLM is a MATLAB open-source software developed to use Bayesian Dynamic Linear Models for time series analysis with time steps in the order of one hour or higher. OpenBDLM is capable of processing simultaneously several time series, enabling interpretation, monitoring and prediction over time. OpenBDLM includes an anomaly detection tool which allows detecting abnormal behavior in a fully probabilistic framework. In addition, 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.

1.2 Installing OpenBDLM

The following instructions show how to download and setup OpenBDLM on your local machine for direct use, testing, and development purposes.

1.2.1 Prerequisites

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

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

1.2.2 Installation

1. If an older OpenBDLM version is already installed, it is recommanded to remove from your MATLAB path any previous OpenBDLM versions.
2. Download and extract the ZIP file from `https://github.com/CivML-PolyMtl/OpenBDLM` (or clone the git repository) in your working directory.
3. Add "OpenBDLM-master" folder and all its subfolders to your MATLAB path through one of the following two options:
   - Using the "Set Path" dialog box in MATLAB, or
   - By running `addpath` function from the MATLAB command window

1.3 Starting OpenBDLM

- Set the current working directory to the folder "OpenBDLM-master"
- Type `OpenBDLM_main;` , and press the key enter ↵. 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 press the key enter ↵ 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;`

1.4 Demo

In the MATLAB command line, type `run_DEMO;` followed by pressing the key enter ↵ to run a demo. Some messages on the MATLAB command window show that the program has started (see Listing 2). Several figures identical to those in Figures 27-28 should popup on the screen[1]

```
     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`

1.5 OpenBDLM main menu

Once a project is loaded, the OpenBDLM main menu displays the list of actions which can be done (see Listing 3). 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.

[1]The demo runs in batch mode following the example described in Section 8.6.

```
----------------------------------------------
/    OpenBDLM main menu. 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

2 Inputs and outputs

2.1 Inputs

OpenBDLM has two operating modes: *interactive mode* and *batch mode*. In the *interactive mode*, OpenBDLM takes the required input for the project through MATLAB command line queries. Each input is validated after pressing the Enter key ↵. In *batch mode*, the inputs are provided in advance by the user and stored in a cell array of characters vector. OpenBDLM reads sequentially the inputs and performs the analysis. The batch mode requires the user to be familiar with the interactive mode because the set of inputs must be provided prior to the analysis. `OpenBDLM_main` can take three types of input:

- no input (*interactive mode*)

  ```
  OpenBDLM_main;
  ```

- the name of configuration file given as a character vector (*interactive mode*)

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

- a cell array of character vectors (*batch mode*)

  ```
  OpenBDLM_main({'''CFG_DEMO.m''', '2', '3', '1', '''Q'''});
  ```

2.2 Outputs

Four possible outputs can be obtained from `OpenBDLM_main.m`. These outputs are `data`, `model`, `estimation`, and `misc`.

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

2.2.1 data

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

- `labels`: See Section 4.1.1 for details.
- `timestamps`: See Section 4.1.1 for details.
- `values`: See Section 4.1.1 for details.

2.2.2 model

The output variable `model` stores all the information related to the model used in the analysis. This variables has 14 fields, amongst them are `hidden_states_names`, `A`, `C`, `Q`, `R`, `Z`, `components`, `parameter_properties`, `initX`, `initV`, `initS`.

- `hidden_states_names`: this field stores a $1 \times$ `S` cell array, where `S` $= \{1, 2\}$ is the number of model classes. Each cell array is a `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 $1 \times$ `S` cell array, where `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 $1 \times$ `S` cell array, where `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 $1 \times$ `S` cell array, where `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 $1 \times$ `S` cell array, where `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 3.3 for details.

- `parameter_properties`: See Section 3.4 for details.

- `initX`: See Section 3.5 for details.

- `initV`: See Section 3.5 for details.

- `initS`: See Section 3.5 for details.

2.2.3 estimation

The `estimation` structure stores the filtered or smoothed hidden states results. This variable has 11 fields, amongst them are `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` × `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` × `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 $1 \times$ `S` cell array, where `S` $= \{1, 2\}$ is the number of model classes. Each cell array stores `L` × `L` × `T` array, where `L` and `T` are the number of hidden states and the length of time vector, respectively.

2.2.4 misc

The `misc` structure stores the options and internal variables needed for running the software. This variable 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 3.6.

2.3 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 4.1.1). 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 3). 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.

3 Configuration file

The configuration file is a MATLAB script utilized for initializing the project (see Figure 1). The name of a configuration file can be given as an input to the function `OpenBLDM_main.m`, as mentionned in Section 2.1. The configuration file must follow a specific structures, which includes 6 sections: Project name, Data, Model structure, Model parameters, Initial states values and Options. The first three sections are mandatory, while the last three sections are optional.

```
%                    OpenBDLM configuration file
%          Autogenerated by OpenBDLM on 22-Nov-2018 17:18:09
%
%% A - Project name
misc.ProjectName='Example_DISP';
                                                Part 1: Project name
%% B - Data
dat=load('DATA_Example_DISP.mat');
data.values=dat.values;
                                                Part 2: Load data
data.timestamps=dat.timestamps;
data.labels={'Example_DISP'};

%% C - Model structure

% Model components
                                                Part 3: Model structure
% 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
                                                Part 4: 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,   0.97,      6 %#6
     '\sigma_w', 'AR',  '1',    '1', [0  Inf],       'N/A',  NaN, NaN,   0.0192,    7 %#7
     '\sigma_v', '',    '1',    '1', [0  Inf],       'N/A',  NaN, NaN,   7.425e-07, 8 %#8
};

%% E - Initial states values
% Initial hidden states mean for model 1:
model.initX{ 1 }=[      25.89    -0.202   -0.00305       0.0331  0.051   -0.00843   ]';

% Initial hidden states variance for model 1:
model.initV{ 1 }=diag([ 3.74E-05       6.85E-05       6.99E-05       5.73E-07
     5.73E-07       0.000485        ]);

% Initial probability for model 1
                                                Part 5: Initial hidden states
model.initS{1}=[1     ];

%% F - Options
misc.options.NaNThreshold=100;
misc.options.Tolerance=1e-06;
misc.options.trainingPeriod=[1  Inf];
                                                Part 6: Options
misc.options.isParallel=false;
misc.options.isMute=false;
misc.options.isMAP=false;
misc.options.maxTime=60;
```

Figure 1: Exemple of configuration file

3.1 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 variable `misc`.

3.2 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 4.1.1. The timestamp values, the amplitude values and the label values must be stored in the fields `timestamps`, `values`, and `labels` of the MATLAB structure named `data`.

3.3 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 $1 \times$ `S` cell array, where `S` $= \{1, 2\}$ is the number of model classes. Each cell array is a $1 \times$ `D` cell array of matrice, where `D` is the number of time series. Each block component is associated with a reference number:
  - 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: First-order autoregressive
  - 51: Kernel regression
  - 61: Level intervention
- `model.component.const`: it constrains model parameters between the block components from 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`: it defines the dependencies among the time series. The field `ic` stores a $1 \times$ `D` cell array of $1 \times ($`D`$- 1)$ matrix, 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 considered independent by default.

3.4 Model parameters

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

- column 1 must be a character vector that gives the name of the model parameters (e.g. `sigma_w`).
- column 2 must be a character vector that gives the reference name of the block associated with the parameter (e.g `LL`, see Section 10.5).
- 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 10.3).
- column 4 must be a character vector that gives the index corresponding to the observation associated with the parameter (e.g `3`).
- 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 4.5 and 10.4).
- column 6 must be a 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 4.5 and 10.4).
- 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 4.5 and 10.4).
- 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 4.5 and 10.4).
- column 9 must be a real number that gives the value of the model parameters.
- column 10 must be an integer that gives the reference number of the model parameters. The model parameters which share the same reference number are constrained to each other.

3.5 Initial states values

This part of the configuration file defines the initial states values (at time $t = 0$). The initial mean and covariance 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 × `S` cell array of array, where `S` $\in \{1, 2\}$ is the total number of model classes. Each array is `L` × 1 array of real number that stores the initial mean values associated with each hidden states variables, where `L` is the total number of hidden states variables associated with the model.

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

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

3.6 Options

This part of the configuration file defines the options that control different aspect of the software regarding the data pre-processing, optimization, hidden states estimation, and aspects related to graphical outputs. The options are stored in the field named options of the MATLAB variable `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 requires the MATLAB *Parallel Computing Toolbox*.
    Default: `true`.
  - `misc.options.maxIterations`: integer that gives the maximum number of iterations for the optimization procedure. Newton-Raphson only.
    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 so that 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. This option is used only for Stochastic Gradient optimization.
    Default: `false`.

- `misc.options.isLaplaceApprox`: logical so that if `isLaplaceApprox=true` the posterior covariance matrix is estimated using Laplace approximation around the optimized model parameters values. This option is used only for Newton-Raphson optimization.
  Default: `false`.
- `misc.options.NRTerminationTolerance`: real value determining the termination tolerance for the Newton-Raphson algorithm.
  Default: 10^{-7}.
- `misc.options.NRLevelsLambdaRef`: integer that controls the number of trial loop for a parameter being optimized for theNewton-Raphson algorithm.
  Default: 4.
- `misc.options.isMute`: logical so that if `isMute=true`, no message are displayed on screen during the optimization procedure.
  Default: `false`.
- `misc.options.maxEpochs`: integer that gives the maximum number of epochs the optimization procedure. This option is used only for Stochastic Gradient optimization.
  Default: `50`.
- `misc.options.Optimizer`: vector of character that defines the optimizer for Stochastic Gradient algorithm. It must be either `'MMT'`, `'ADAM'`, `'MMTbeta'`, `'ADAMbeta'`. This option is used only for Stochastic Gradient optimization.
  Default: `'MMT'`.
- `misc.options.SplitPercent`: real number that defines defines in percent the portion of the training data used for validation.
  Default: `30`.
- `misc.options.MiniBatchSizePercent`: real number that defines defines the size of mini-batch, in percent of the training data.
  Default: `20`.
- `misc.options.SGTerminationTolerance`: termination tolerance for the Stochastic gradient algorithm. This option is used only for Stochastic Gradient optimization.
  Default: `0.95`.

- 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 that gives in percent the amount of data, starting at $t = 1$ used for the estimation of the initial hidden states.
    Default: `100`.

- `misc.options.KRNumberControlPoints`: integer that gives the number of control points used for the periodic kernel regression component (see Section 10.5).
  Default: `100`.

- Options for the synthetic data creation
  - `misc.options.Seed`: integer that controls the random number generation used to create synthetic data. Synthetic data created with the same seed are identical (useful to replicate 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.isPlotEstimations`: logical. if `isPlotEstimations=true`, figures plotting the estimation results popup on screen each time the hidden states are estimated.
    Default: `true`.
  - `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 that are displayed in the plots. 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 by which the x-axis is shifted on each figure.
    Default: `0`.
  - `misc.options.isExportTEX`: logical so that if `isExportTEX=true`, figures are exported in LaTeX format.
    Default: `false`.

- `misc.options.isExportPNG`: logical so that if `isExportPNG=true`, figures are exported in PNG format.
  Default: `false`.
- `misc.options.isExportPDF`: logical so that if `isExportPDF=true`, figures are exported in PDF format.
  Default: `false`.

4 Workflow

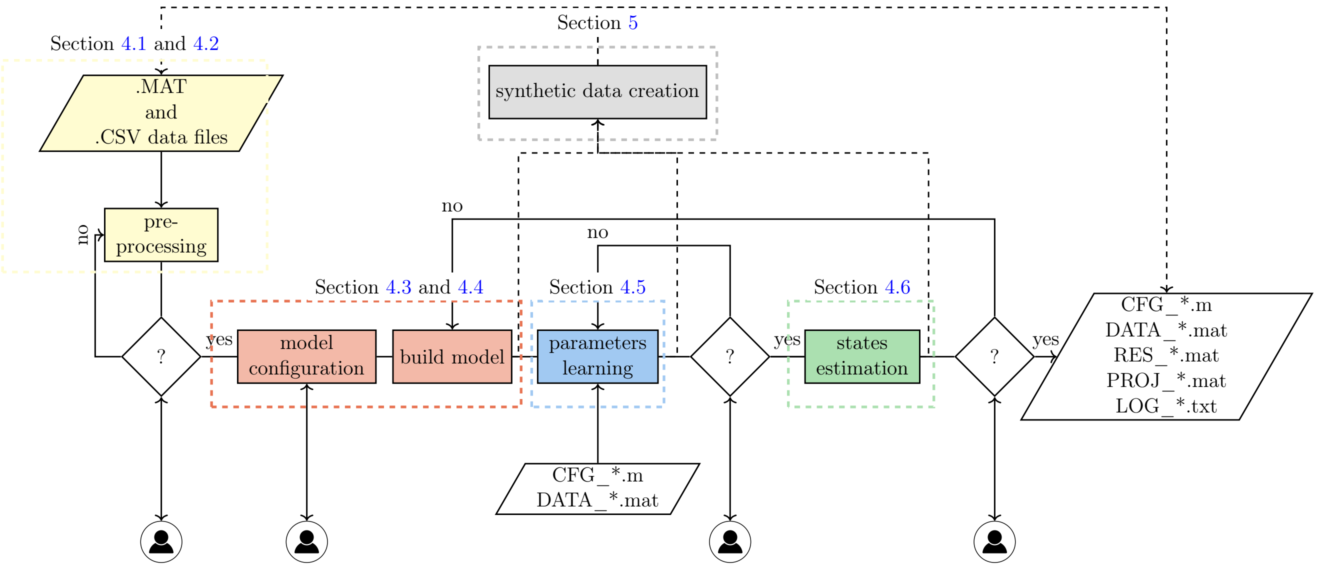


Figure 2: OpenBDLM workflow

4.1 Data loading

4.1.1 Input data format

OpenBDLM supports two types of input data format.

Comma Separated Values files (.CSV) One .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 field must contain the label of

```
'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

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 since January 0, 0000, 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.

MATLAB files (.MAT) 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 4.2 for details.)

4.1.2 Data loading functions

The data loading workflow is presented Figure 3. 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)
```

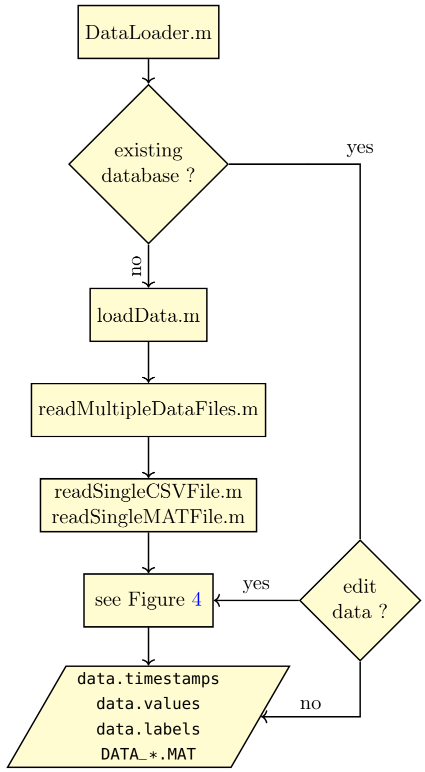


Figure 3: Data loading workflow

4.2 Data pre-processing

Data pre-processing is a step taken ahead of data analysis. In most cases, the set of available time series is heterogeneous because data is recorded from different systems of acquisition that are not synchronized with respect to time. Therefore, the raw data do not usually share the same timestamp vectors. This is an issue because BDLM is not capable to analyze asynchronous time series. Therefore, the main objective of data pre-processing is to synchronize the time series. Moreover, data pre-processing includes time series selection, data analysis time period selection, `NaN` removal, and data resampling.

```
- 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

4.2.1 Selection of time series

The selection of time series allows including a subset of the time series. Note that time series are automatically synchronized as they are added to (or removed from) the dataset.

4.2.2 Selection of data analysis time-window

The selection of the analysis time-window allows selecting a portion of data between two dates. The date format follows 'YYYY-DD-MM'. If the second date entered exceeds the date associated with the last data sample of the original dataset, padding with `NaN` values is performed. The timestep for the padding must be provided by the user.

4.2.3 Removing missing data

It is possible to control the maximum amount of missing data (`NaN`) 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 100% (see variable `misc.options.NaNThreshold`).

4.2.4 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 mid point of each time window.

4.2.5 Time synchronization options

By default, the time synchronization in OpenBDLM is done by padding with `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 100% and 10^{-6} days, respectively (see variables `misc.options.NaNThreshold` and `misc.options.tolerance`).

4.2.6 Data pre-processing functions

The data pre-processing workflow is presented Figure 4. The list of OpenBDLM functions used for data editing is:

- Control script to pre-process the 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 the timestep vector from the timestamps vector

  `[timesteps]=computeTimeSteps(timestamps)`

- Display information about stored data on screen

  `displayData(data,misc)`

- Display the list of DATA_*.mat files

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

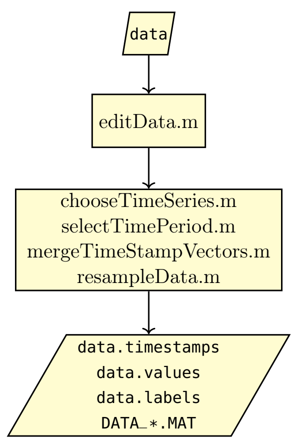


Figure 4: Data pre-processing workflow

4.3 Model configuration

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`.

4.3.1 Dependencies between time series

The field `model.component.ic` stores the information related to time series dependencies. `model.component.ic` stores a $1 \times$ `D` cell array of $(1 \times \mathtt{D} - 1)$ matrix, 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 independent. See the Reference Theory §10.7 for further details.

4.3.2 Block components

OpenBDLM supports 5 types of block components: (1) baseline, (2) periodic, (3) periodic kernel regression, (4) autoregressive, and (5) level intervention. (1) The baseline component models the local mean of the time series. There are three types of baseline supported in OpenBDLM: (i) level 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 is intended as a residual term to capture the time-dependent model errors. (5) The level intervention component allows estimating the magnitude of discrete jumps occurring in the local level at specific timestamps provided by the user. The field `block` stores a $1 \times$ `S` cell array, where $\mathtt{S} \in \{1, 2\}$ 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 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: First-order autoregressive
- 51: Kernel regression
- 61: Level intervention

The number of hidden states associated with each block component 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. See the Reference Theory §10.5 for further details.

4.3.3 Parameter constrains

The variable `model.component.const` stores a $1 \times$ `S` cell array, where $\mathtt{S} \in \{1, 2\}$ is the total number of model classes. It is defined only if $\mathtt{S} = 2$. 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.

4.3.4 Number of model class

OpenBDLM is capable of detecting regime changes in the dynamics in the baseline of the time series. Therefore, OpenBDLM 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, which includes 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).

4.3.5 Model configuration functions

The model configuration workflow is presented Figure 5. The list of OpenBDLM functions used for model configuration is:

- Controls script for model configuration

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

- Model configuration for real data

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

- Model 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)
```

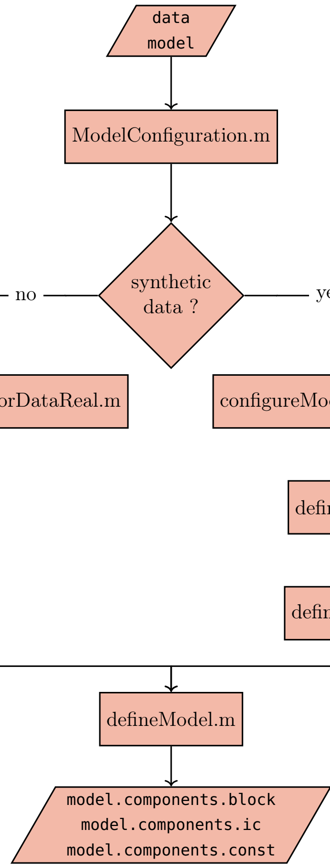


Figure 5: Model configuration workflow

4.4 Model construction

Model construction builds 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 are also considered during model construction.

4.4.1 Model construction functions

The model construction workflow is presented in Figure 6. The function in OpenBDLM used for model construction is:

- Builds the model

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

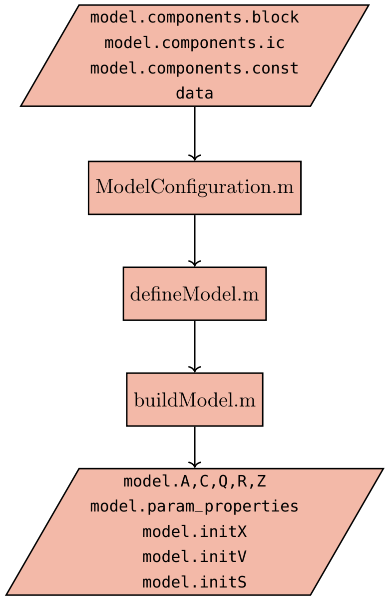


Figure 6: Model construction workflow

4.5 Model parameters estimation

The default values for the parameters are either defined from heuristic knowledge or from statistics computed on the data itself (see Table 1). Each value is only an initial guess which has to be refined using an optimization algorithm (see Section 10.4). The algorithm will estimate the model parameter values from the data.
In order to access the model parameter estimation menu from OpenBDLM, type `1` (see Listing 6). The optimization algorithm is either the Newton-Raphson (choice `1`) or the Stochastic Gradient Descent (choice `2`) algorithms (see Section 10.4). By default,

```
------------------------------------------------------------
/ 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

OpenBDLM uses all the data as the training set (see `misc.options.Trainingperiod=[1 Inf]`). OpenBDLM transform the model parameters to perform the optimization in an unbounded model parameter space (see Section 10.4.5). Undefined bounds (`[NaN, NaN]`) for a parameter means that the parameter is assumed to be known and thus, it will be excluded from the parameter estimation process. By default, OpenBLDM assumes that the parameters for the period of the periodic component and the process noise standard deviation associated with the baseline and the periodic component have values fixed to 0 and are not optimized.
The model parameter can be learned by maximizing either the likelihood (Maximum

Table 1: Default values for model parameters. $\hat{\sigma}_{y_{obs(1:\mathtt{T})}}$ corresponds to the empirical standard deviation calculated using observed data from the first data sample to the last data sample of index `T`. Note that default value for synthetic data are different, see Table 3 for details.

$\boldsymbol{\theta}$ name	$\boldsymbol{\theta}^0$	$\boldsymbol{\theta}$ bounds
$\sigma_w^{\mathtt{LL}}$	0	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{LT}}$	$10^{-7} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{LA}}$	$10^{-8} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{LcT}}$	$10^{-7} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{LcA}}$	$10^{-7} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{TcA}}$	$10^{-8} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{P}}$	0	$[\mathrm{NaN}, \mathrm{NaN}]$
$\sigma_w^{\mathtt{AR}}$	$10^{-1} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[0, \mathrm{Inf}]$
$\sigma_{w,0}^{\mathtt{KR}}$	$10^{-1} \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[0, \mathrm{Inf}]$
$\sigma_{w,1}^{\mathtt{KR}}$	0	$[\mathrm{NaN}, \mathrm{NaN}]$
σ_v	$0.05 \times \hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[0, \mathrm{Inf}]$
$\phi^{\mathtt{AR}}$	0.75	$[0, 1]$
$p^{\mathtt{P}}$	$[365.24, 1, 182.62]$[2]	$[\mathrm{NaN}, \mathrm{NaN}]$
$p^{\mathtt{KR}}$	365.24	$[\mathrm{NaN}, \mathrm{NaN}]$
$\ell^{\mathtt{KR}}$	$2/\mathrm{L}^{\mathtt{KR}}$ (see §10.5.8)	$[0, \mathrm{Inf}]$
$\phi^{\cdot\vert\cdot}$	0.01	$[-\mathrm{Inf}, \mathrm{Inf}]$
$\mu_b^{\mathtt{LI}}$	0	$[-\mathrm{Inf}, \mathrm{Inf}]$
$\sigma_b^{\mathtt{LI}}$	$\hat{\sigma}_{y_{obs(1:\mathtt{T})}}$	$[0, \mathrm{Inf}]$

Likelihood Estimation, MLE), or the posterior function (Maximum A Posteriori estimation, MAP) (see Section 10.4). Note that MAP requires a valid prior for each unknown model parameters. In the case of of MAP, OpenBDLM supports gaussian prior only. The default option is MLE (see `misc.options.isMAP=false`). There is the possibility to use the prediction capacity to drive the optimization process[3] (see `misc.options.isPredCap` and `misc.options.SplitPercent`). The model parameter estimation framework computes point estimate of the model parameters. However, it is also possible provide confidence intervals around the point estimate using the Laplace approximation[4] (see `misc.options.isLaplaceApproximation = true`). Note that computing the Laplace approximation can significantly increase the computation time and it is not recommended 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. Therefore, it is advised to run the optimizations several times with different starting model parameters values in order to check if the proposed solution is the best attainable solution. If `misc.options.isMute = false`, outputs on the MATLAB command line window allows to monitor the optimization process (see Listings 7-8). 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 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).

[3] The use of the prediction capacity is only available using the Stochastic Gradient Descent optimization.
[4] Laplace approximation is currently only available using the Newton-Raphson optimization.

The optimization stops when all the parameters converged, or if the maximum number of iterations (see `misc.options.maxIterations`) (or epochs for Stochastic Gradient Descent algorithm, see `misc.options.maxEpochs`) / the maximum time (see `misc.options.maxTime`) is reached. The values of the parameters are saved in the variable `model` MATLAB.

```
    \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               0               0
___________________________
    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               0               0
___________________________
```

Listing 7: OpenBLDM output example when running Newton-Raphson algorithm.

```
  \Start SGD algorithm (finite difference method)

    Optimization mode                          MLE
    Optimizer                                  MMT
    Metric                                     logpdf
    Learning Rate mode                         hessian
    Training period:                           1 — Inf [days]
    Validation set portion:                    30 [%]
    Training set:                              13556 [data points]
    Validation set:                            5810 [data points]
    Mini batch:                                3873 [data points]
    Number of max epoch:                       30+1 [epochs]
    Total time limit for calibration:          60 [min]

  Epoch #1
           Metric: 25972.5904
                  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

__________________________
  Epoch #2
          Metric: 33933.8856
 parameter names: AR|M1|1        AR|M1|1        |M1|1
  current values: +9.61e—01      +2.00e—02      +5.66e—03
    param change: +2.11e—01      +2.61e—03      —3.04e—03
initialize param:         0              0              0


__________________________
  Epoch #3
          Metric: 33933.8856
 parameter names: AR|M1|1        AR|M1|1        |M1|1
  current values: +9.61e—01      +2.00e—02      +5.66e—03
    param change: +0.00e+00      +0.00e+00      +0.00e+00
initialize param:         1              0              0
```

Listing 8: OpenBLDM output example when running Stochastic Gradient Ascent algorithm.

4.5.1 Model parameter estimation functions

- Pilot 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)
```

- 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-difference 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)
```

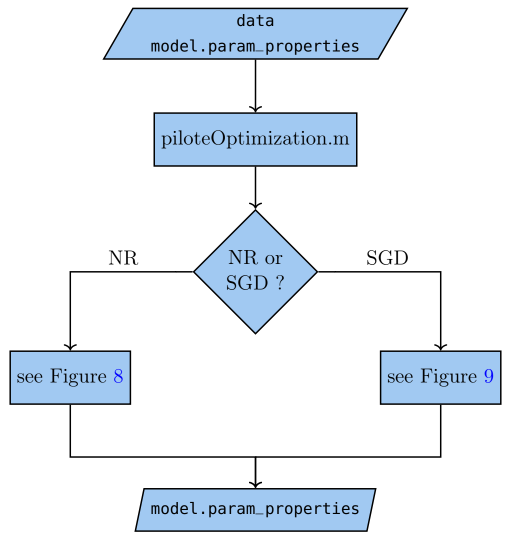


Figure 7: Model parameter estimation workflow

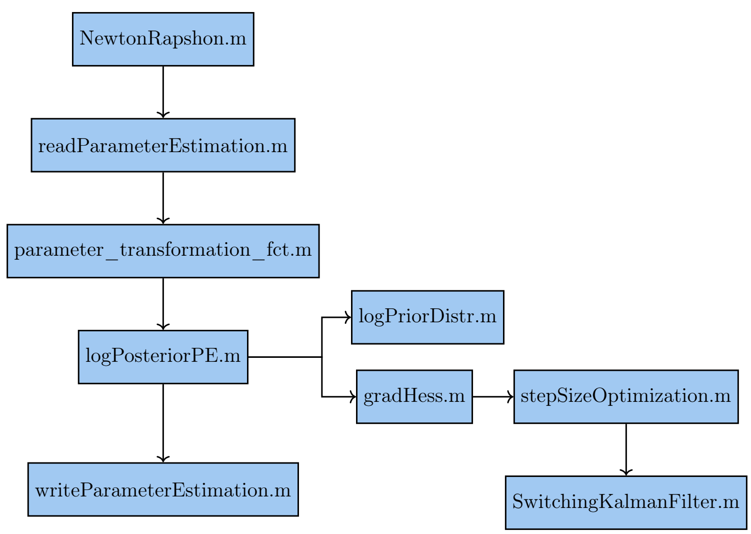


Figure 8: Model parameter estimation Newton-Raphson workflow

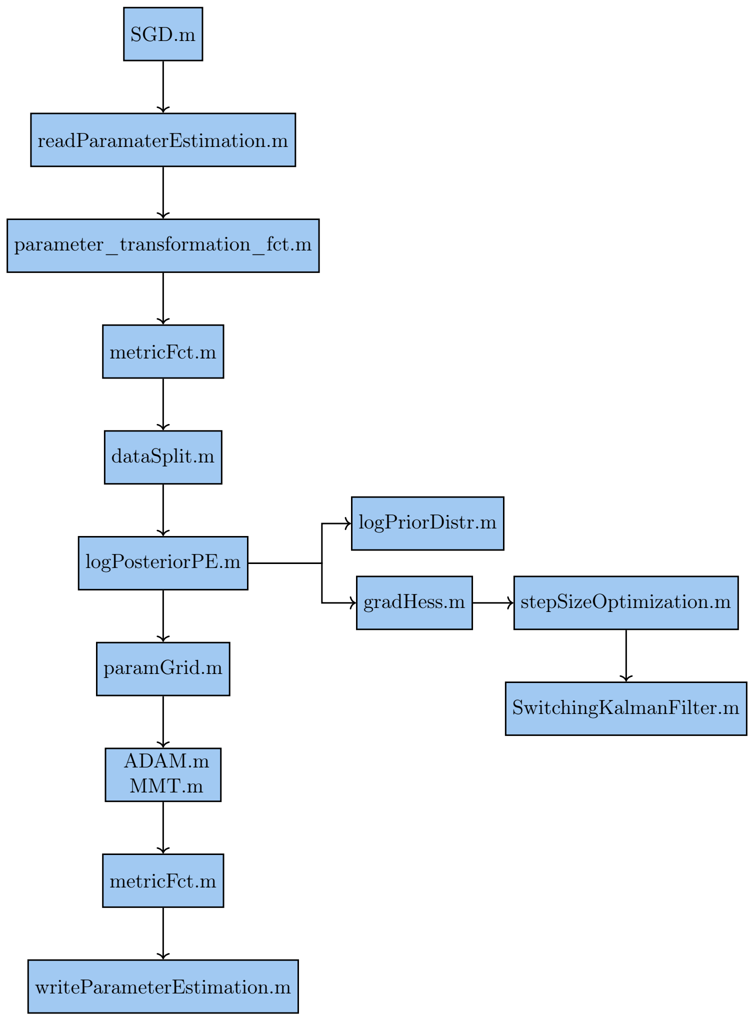


Figure 9: Model parameter estimation Stochastic gradient descent workflow

4.6 Hidden states estimation

The estimation of hidden states is the core of OpenBLDM. The hidden state estimation is performed recursively using the Kalman filter/smoother or the UD filter (see Section 10.2). Once a project is loaded, typing `3` opens the hidden state estimation menu (see Listing 9).

```
----------------------------------------------------
/    State estimation
----------------------------------------------------
     Current state estimation method: UD

     1 -> Filter
     2 -> Smoother

     3 -> Change estimation method to kalman

     Type R to return to the previous menu

     choice >>
```

Listing 9: 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. It is possible to switch between the Kalman and UD computation (and conversely) by typing `3` from the state estimation menu (see Listing 9).

4.6.1 Hidden state estimation functions

The hidden state estimation workflow is presented Figure 10. The OpenBLDM functions used for hidden state estimation are:

- Pilot 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 steps

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

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

  ```
  [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)
```

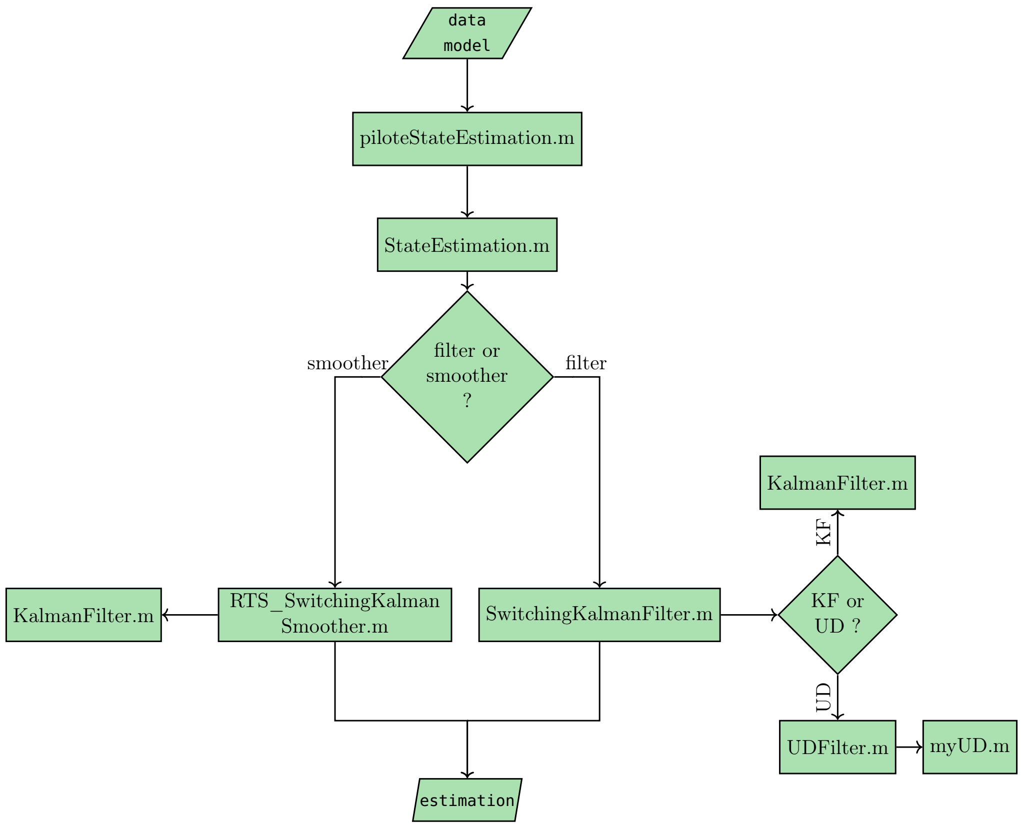


Figure 10: Hidden states estimation workflow

4.6.2 Estimation of the initial hidden states

OpenBDLM computes default values for the initial (at $t = 0$) mean (`model.initX`) and covariance (`model.initV`), as well as the initial model probabilities (`model.initS`) (see Table 2). Those initial values are usually rough 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 11. The OpenBLDM functions used for initial hidden state estimation are:

- Pilot function for initial state estimation

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

- 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)
```

Table 2: Default values for the initial hidden states $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$. $\hat{\sigma}_{y_{obs(1:\mathtt{T})}}$ corresponds to the empirical standard deviation estimated using the observed data from the first data sample to the last data sample of index `T`. $\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}}$ corresponds to the empirical average of the first ten percent of data samples. Note that default values for synthetic data are different from these ones, see Table 3 for details.

	$\boldsymbol{\mu}_0$	$\mathrm{diag}(\boldsymbol{\Sigma}_0)$
`LL`	$[\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}}]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2]$
`LT`	$[\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}},0]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`LA`	$[\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}},0,0]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}},\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`LcT`	$[\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}},0]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`LcA`	$[\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}},0,0]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}},\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`TcA`	$[\hat{\mu}_{y_{obs(1:\mathtt{T}/10)}},0,0]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}},\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`P`	$[5,0]$	$[(2\times\hat{\sigma}_{y_{obs(1:\mathtt{T})}})^2,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`AR`	$[0]$	$[\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`KR`	$[0,0,\ldots,0]$	$[\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}},\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}},\ldots,\hat{\sigma}^2_{y_{obs(1:\mathtt{T})}}]$
`LI`	$[0]$	$[10^{-20}]$

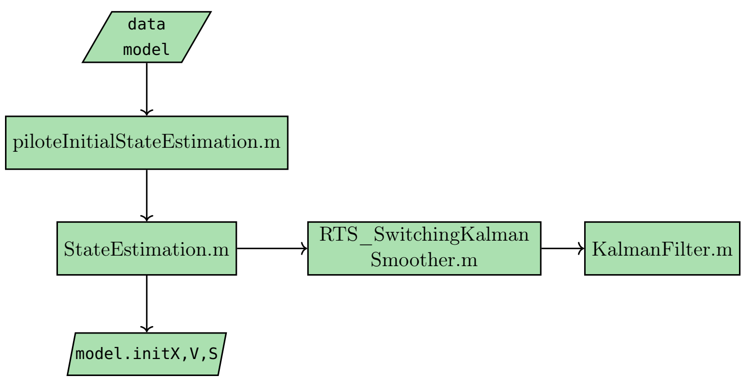


Figure 11: Initial hidden states estimation workflow

5 Generate 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 10.1) to create realistic synthetic data. There are two ways for creating synthetic data using OpenBDLM:

- From the interactive tool
- From an existing project.

5.1 Generate synthetic data using the interactive tool

The creation of the synthetic data from the interactive tool (option `0` from the starting menu) enables the creation of 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 define a set of block components 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 10.3). 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. Alternately, the user may choose to create *custom anomalies*. In such a case, the beginning (in sample index), duration (in number of samples) and amplitude (in change of the local trend) of each anomaly is user 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`).

Table 3: Default value of model parameters and initial hidden states $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ for synthetic data generation.

	$\boldsymbol{\theta}$	$\boldsymbol{\mu}_0$	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	$\phi^{\mathtt{AR}} = 0.75$, $\sigma_w^{\mathtt{AR}} = 0.01$	$[0]$	$[0.1^2]$

5.2 Generate synthetic data 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 the 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`).

5.3 Synthetic data generation functions

The synthetic data creation workflow is presented Figure 12. The OpenBLDM functions used for synthetic data creation are:

- Pilot 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 inputs 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 inputs to define synthetic data time vector (for synthetic data creation from interactive tool only)

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

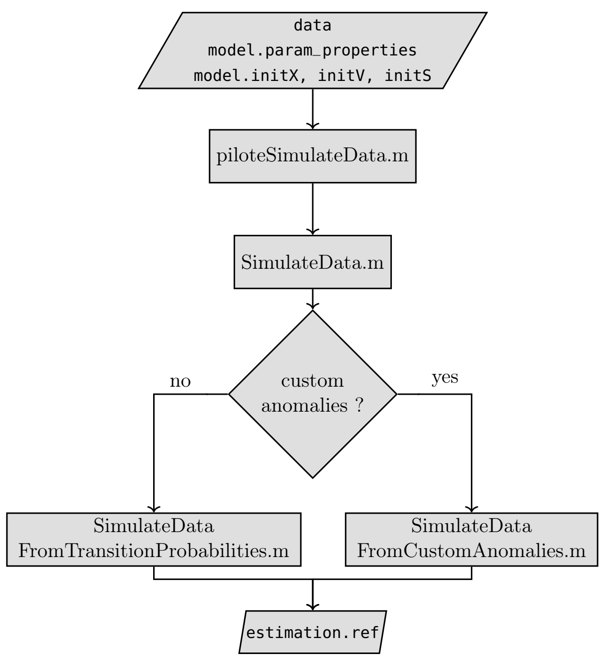


Figure 12: Synthetic data creation workflow

6 Results

6.1 Visualizing results

The figures that popup on screen 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 length. 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`.

6.1.1 Generating figures at any time

The option `14` from the main menu allows generating the different type of figure at any time (see Listing 10).

```
-------------------------------------
/    Plot
-------------------------------------


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

     Type R to return to the previous menu

     choice >>
```

Listing 10: OpenBDLM plot menu

6.1.2 Saving figures

It is not advised to save figures "manually" using MATLAB's menus. It would most likely not save figures as seen on the screen. Instead, use the OpenBDLM export facilities described in the Section 6.2.4, or set the `misc.options.isExportTEX`, `misc.options.isExportPDF`, `misc.options.isExportPDF` to `true` to automatically save the figures in a specific format each time a figure is created[5]. The workflow for visualization is shown in Figure 13. The functions used to visualize the data and results are:

- 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)
```

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

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

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

- Plot hidden states, predicted data, and model probability

```
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 LaTeX (tikz) file using matlab2tikz

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

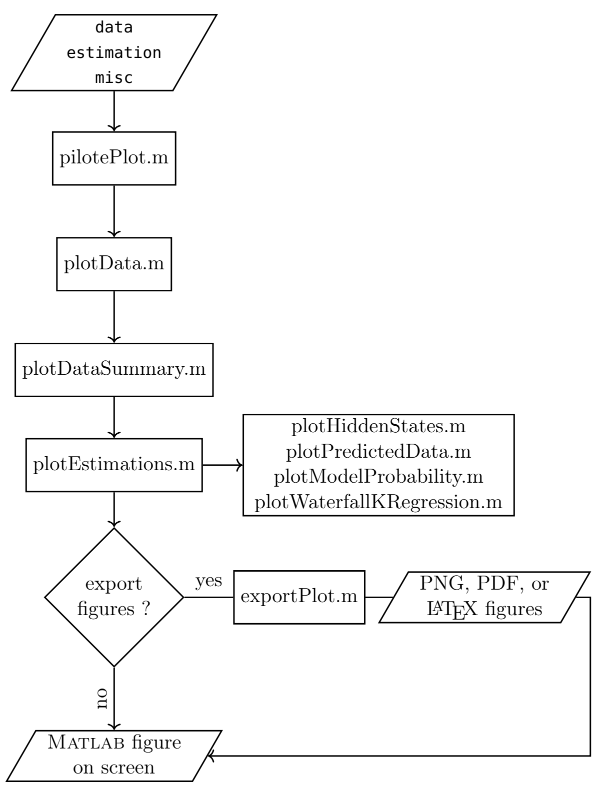


Figure 13: Visualization results workflow

6.2 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 (tikz) to create publication-quality figures.

6.2.1 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)
```

6.2.2 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 2. The function used to save the project is:

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

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

6.2.3 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 4.1.1. The function used to save the data is:

- Save data in a binary Matlab .mat file

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

6.2.4 Exporting 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 Listing 11). It is possible to export figures in PDF, PNG[6] and LaTeX[7] [8]. 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 3.

```
______________________________________
/    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 11: OpenBDLM export menu

The export workflow is shown in Figure 14, and the functions used to export the results are:

- Pilote function to export data, estimations and project

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

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

[8] All the time-series figures shown in this document have been created from LaTeX (tikz) files output from OpenBDLM. Minor post-processing have been done on the LaTeX file. Note that you should compile the .tec files using the LuaLatex compiler. If you choose to use the default Latex compiler, you must comment `\RequirePackage{luatex85}` in the preamble of .tex files.

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

- Create and print a configuration file from project

```
[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)
```

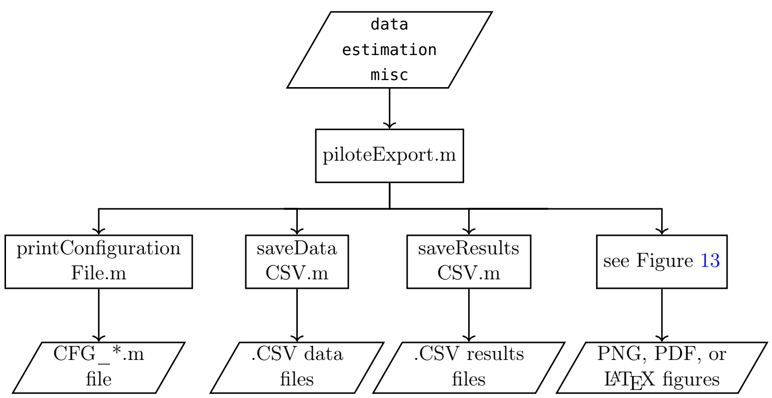


Figure 14: Export options workflow

7 Version control

For the users, version control tests verify that the program runs properly on your machine. For development purpose, version control tests verifie that changes you have made are still compatible with the current 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 12.

How does version control work ? In the folder "version_control", there are sets of three files corresponding to a given project named CTL_00X , where X refers to the version control test index (`CFG_CTL_00X.m`, `DATA_CTL_00X.mat` and `PROJ_CTL_00X.mat`). For each set, 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, and 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[9].

```
— 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 12: OpenBDLM version control output

The functions used for version control are:

- Pilot function for version control

  `piloteVersionControl(misc)`

- Version control for OpenBDLM

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

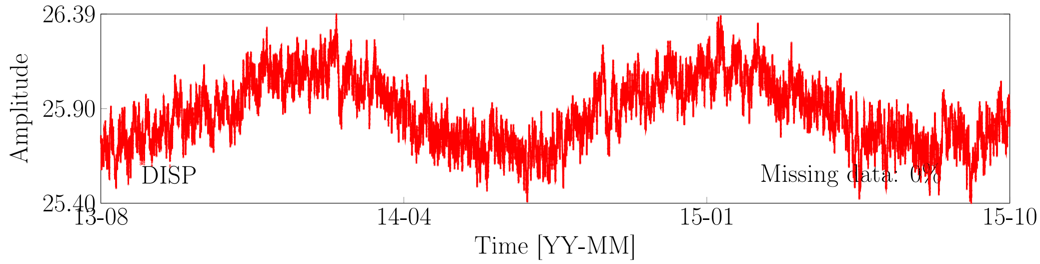
[9]It is possible to add or modify CFG, PROJ and DATA files to design new version control tests.

8 Examples

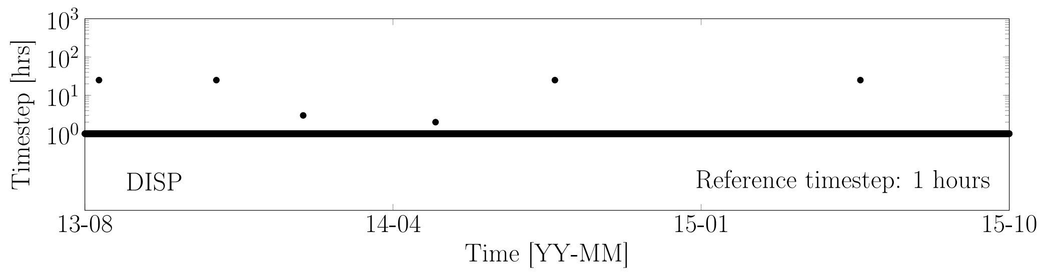
8.1 Example #1: single time series analysis

8.1.1 Data description

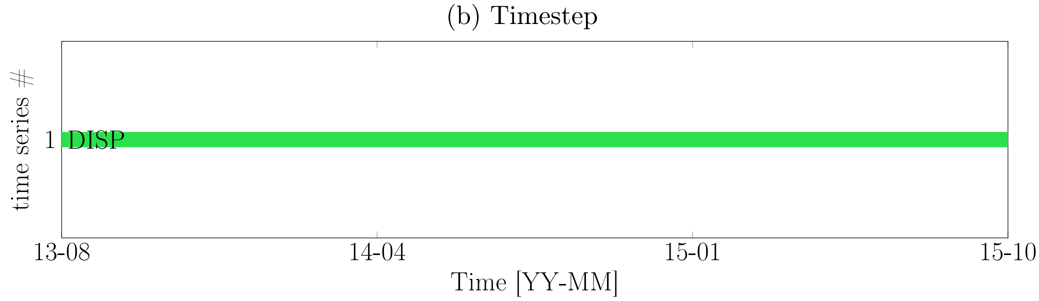
This example uses a synthetic time series that mimics displacement data measured on a bridge (Figure 25). The Figure 25a shows that data points exist between August 2013 and October 2015. The timestep in the original data is non-uniform; it varies from 1 hour to 25 hours (see Figure 25b). The most frequent time step (i.e reference time step, see §10.6.1) is 1 hour, and there is no missing data (see Figure 25c). The time series is stationary with a level, a yearly periodic, a daily periodic patterns and a residual autoregressive pattern.
In this example, we choose to resample the original data in order to have a timesteps of 6h instead of 1h.



(a) Amplitude



(b) Timestep



(c) Availability

Figure 15: Raw data in the example #1 where the reference timestep is 1h.

8.1.2 Model description

The model includes one model class, and the hidden states variables are

$$\mathbf{x} = [x^{\mathtt{LL}}, x^{\mathtt{P1},\text{yearly}}, x^{\mathtt{P2},\text{yearly}}, x^{\mathtt{P1},\text{daily}}, x^{\mathtt{P2},\text{daily}}, x^{\mathtt{AR}}].$$

The associated model parameters are

$$\boldsymbol{\theta} = [\sigma_w^{\texttt{LL}}, p^{\texttt{P},\text{yearly}}, \sigma_w^{\texttt{P},\text{yearly}}, p^{\texttt{P},\text{daily}}, \sigma_w^{\texttt{P},\text{daily}}, \phi^{\texttt{AR}}, \sigma_w^{\texttt{AR}}, \sigma_v].$$

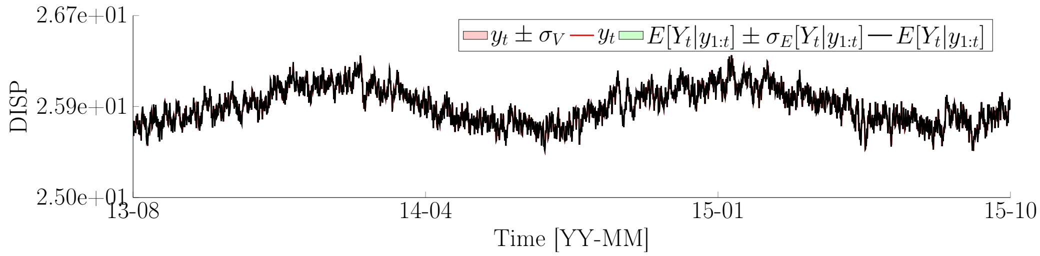
The optimized model parameters values computed using the Newton-Raphson algorithm (see 10.4) with a training period of 180 days are

$$\boldsymbol{\theta}^* = [0, 365.2422, 0, 1, 0, 0.903, 0.035, 7.8 \times 10^{-6}].$$

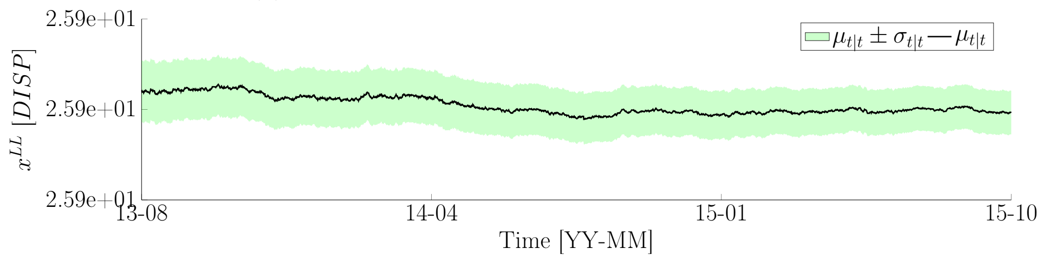
The estimated initial hidden states mean and covariance values are

$$\begin{aligned}
\boldsymbol{\mu}_0^* &= [25.9, -0.202, -0.004, -0.004, 0.054, -0.012]^{\intercal}, \text{and} \\
\boldsymbol{\Sigma}_0^* &= \text{diag}([4.21 \times 10^{-5}, 8.3 \times 10^{-5}, 8.46 \times 10^{-5}, 4.29 \times 10^{-7}, 4.29 \times 10^{-7}, 1.59 \times 10^{-3}]).
\end{aligned}$$

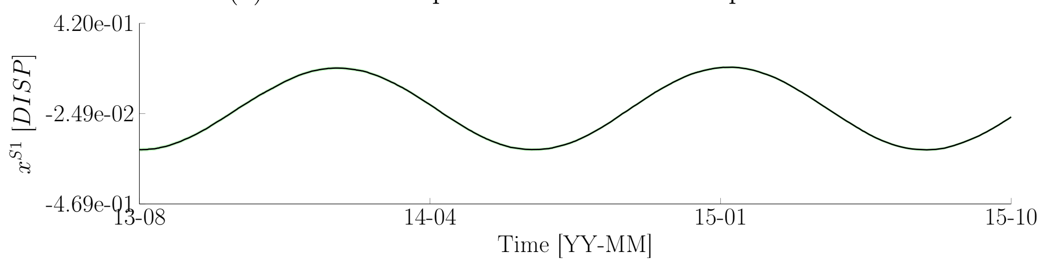
The hidden states computed using the estimated model parameters and initial hidden states are presented in Figure 16.



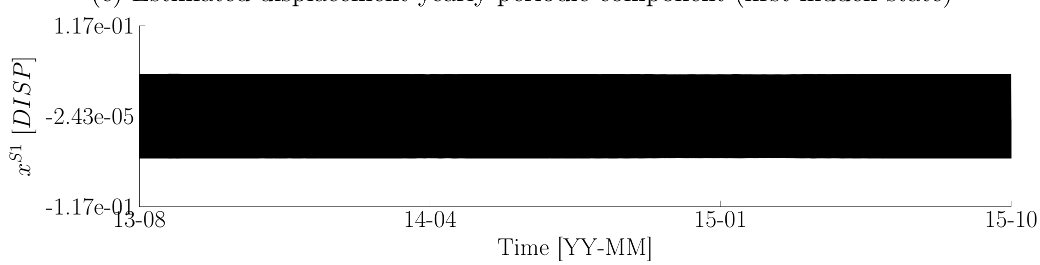
(a) Observed and estimated displacement data



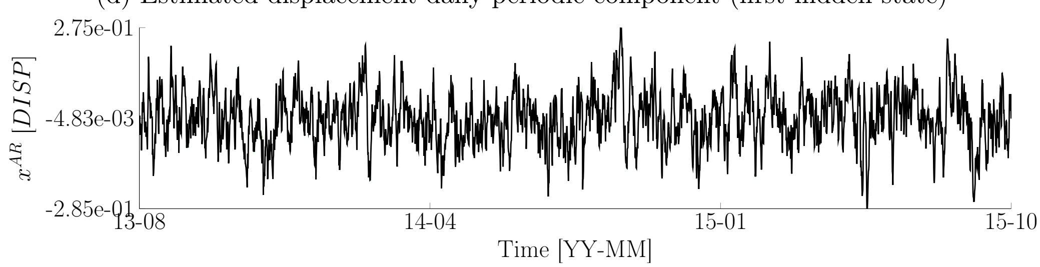
(b) Estimated displacement local level component.



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



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



(e) Estimated displacement autoregressive component

Figure 16: Estimated results using OpenBDLM with the optimized model parameters and estimated initial hidden states. The hidden states are estimated from the data presented in Figure 25a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states.

8.1.3 Run the example from the pre-existing configuration file

There is a configuration file CFG_Example_DISP_optim.m which is located in the "config_files" folder of the OpenBDLM package. CFG_Example_DISP_optim.m contains the optimized model parameters and estimated initial hidden states values (see Listing 13). There is also a data file DATA_Example_DISP_optim.mat that is located in the "data/mat" subfolder. Therefore, it is possible to run the example #1 by following the steps below while interacting with the MATLAB command line:

1. Start OpenBDLM. Type `OpenBDLM_main('CFG_Example_DISP_optim.m');`.
2. Access hidden states estimation menu. Type `3`.
3. Run the Kalman filter to estimate the hidden states. Type `1`.
4. Save and quit. Type `Q`.

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

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

%% C - Model structure

% 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,   0.903,     6 %#6
     '\sigma_w', 'AR',  '1',   '1', [0  Inf],      'N/A',  NaN, NaN,   0.035,     7 %#7
     '\sigma_v', '',    '1',   '1', [0  Inf],      'N/A',  NaN, NaN,   7.8e-06,   8 %#8
};

%% E - Initial states values
% Initial hidden states mean for model 1:
model.initX{ 1 }=[      25.9    -0.202  -0.00406        -0.00439        0.0548  -0.0129  ]';

% Initial hidden states variance for model 1:
model.initV{ 1 }=diag([         4.21E-05        8.3E-05 8.46E-05        4.29E-07        4.29E-07        0.00159 ]);

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

Listing 13: Configuration file for the example #1

8.1.4 Run the example from command line interaction

The analysis of a new dataset usually requires to start from scratch. This section explains how to run the example #1 from scratch, that is, how to load and resample the data presented in Figure 25, configure the model, estimate the model parameters, optimize the initial states, and estimate the hidden states as presented in Figure 16. This may be done by following steps below while interacting with the MATLAB command line:

1. Start OpenBDLM. Type `OpenBDLM_main;`.
2. Choose the interactive tool. Type `0`.
3. Enter the project name. Type `Example_DISP`.
4. Disregard generating synthetic data. Type `no`.
5. Load new data. Type `0`.
6. Select from the graphical user interface[10] the data file Example_DISP_DISP.csv located in the folder "/data/csv/Example_DISP" (See Figure 17). The Figure 25 that represents the raw data should popup on screen.
7. Access the resampling menu. Type `4`.
8. Resample data to obtain timesteps of 6h (0.25 day). Type `0.25`. Another set of figures should popup on the screen this time for the resampled data.
9. Save and continue. Type `7`. The same figures should popup on the screen again.
10. Select the number of model classes. Type `1`.
11. Select the model block components. Type `[11 31 31 41]`.
12. Access the training period modification menu. Type `13`.
13. Modify the training period. Type `1`.
14. Choose the starting time (day). Type `1`.
15. Choose the end time (day). Type `180`.
16. Access model parameter estimation menu. Type `1`.
17. Start Newton-Raphson algorithm. Type `1`. Once the algorithm has converged, the optimized model parameters values should be close to the values presented in Section 8.5.2. Note that it is possible to get slightly different parameter values[11].
18. Estimate the initial hidden states values. Type `2`.
19. Access hidden states estimation menu. Type `3`.
20. Estimate the hidden states using the Kalman filter. Type `1`. The estimation should be corresponding to the results presented in Figure 16.

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

[11]Keep in mind that the optimization may take several minutes. It is possible to abort the analysis here and to load the configuration file called CFG_Example_DISP_optim.m to load pre-computed values of model parameters, as presented in Section 8.5.3.

21. Access export menu. Type `17`.

22. Export the current project in a configuration file. Type `1`. A config file corresponding to the one in Listing 13 will not be created in the folder "/config_files/CFG_Example_DISP.m"

23. Save and quit OpenBDLM. Type `Q`.

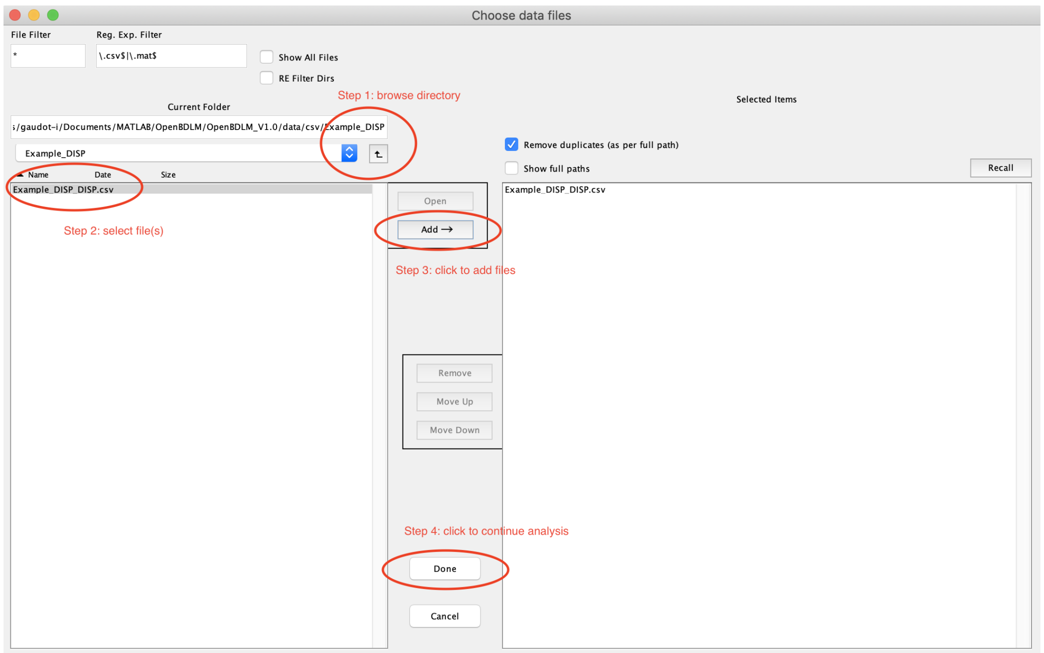


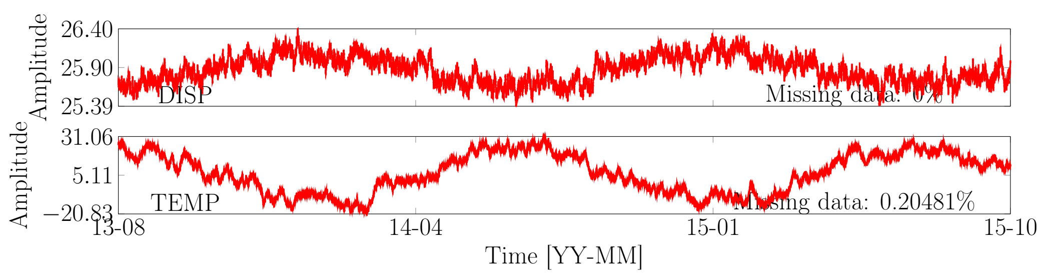
Figure 17: Interactive data loading using the graphical user interface.

.

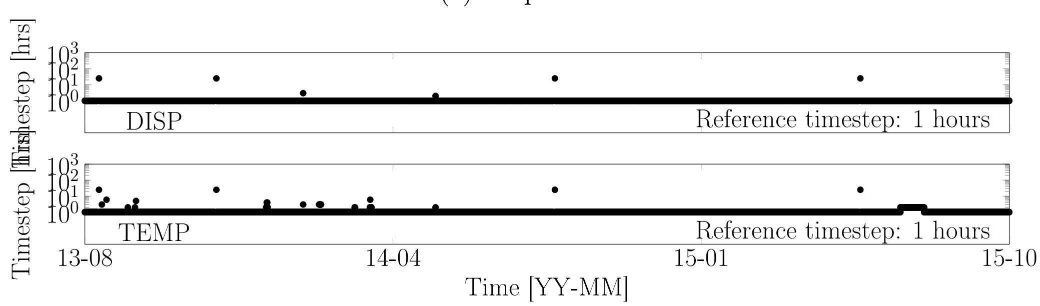
8.2 Example #2: dependence model between two time series

8.2.1 Data description

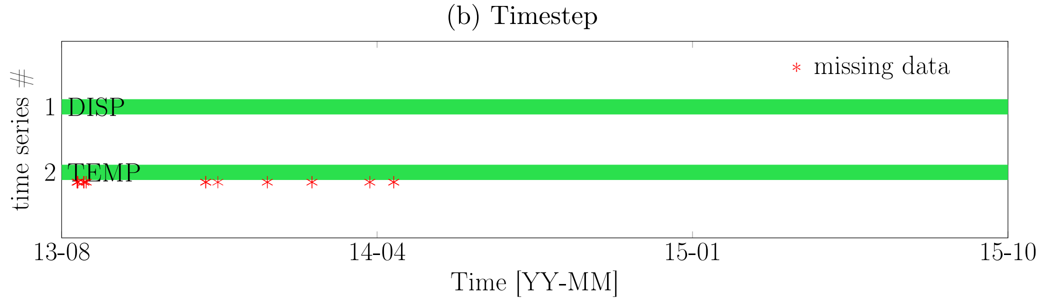
This example uses two synthetic time series that mimics the displacement and temperature data measured on a bridge. The Figure 18a shows that data points exist between August 2013 and October 2015. The timestep in the original data is non-uniform; it varies from 1 hour to 25 hours (see Figure 18b). The 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 (Section 10.6.1). There is no missing data (`NaN`) on the displacement time series, but there are missing data on the temperature time series as indicated by the red crosses on the Figure 18c. Each red cross indicates the presence of a Not a Number (`NaN`) value in the time series. After data synchronization, the time step vectors are identical on each time series (Figure 19). Both time series are stationary, and they exhibit a level, a yearly and daily periodic pattern as well as an autoregressive pattern. The periodic patterns observed on the displacement time series is due to the temperature variations observed in the temperature time series.
In this example, we choose to resample the original data in order to have timesteps of 6h instead of 1h.



(a) Amplitude

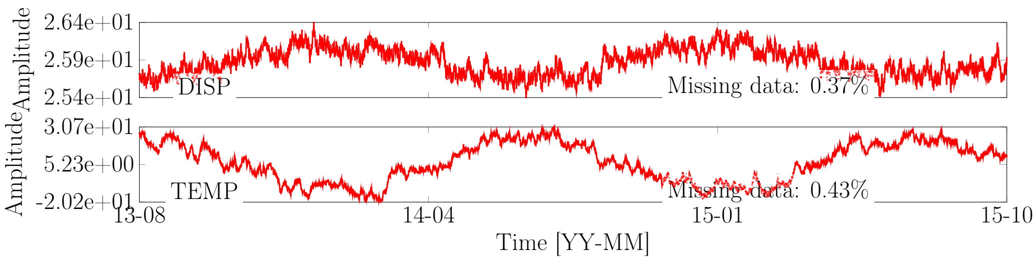


(b) Timestep

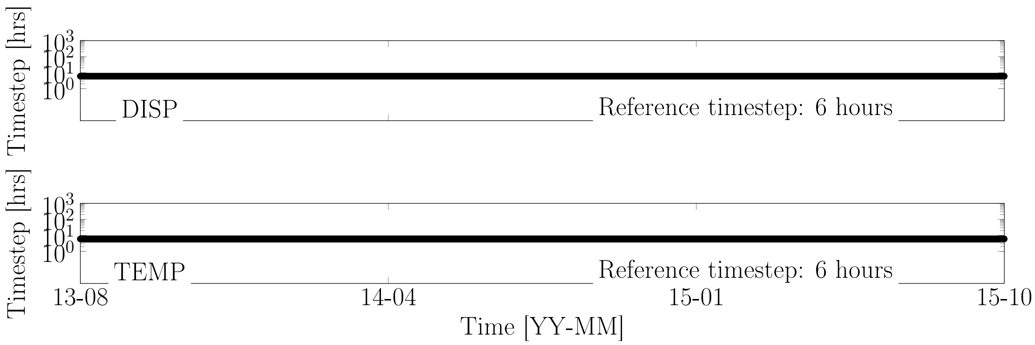


(c) Availability

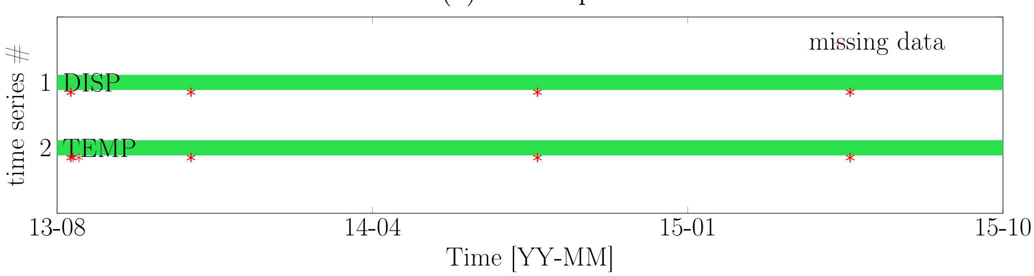
Figure 18: Raw data in the example #2 where the reference timestep is 1h.



(a) Amplitude



(b) Timestep



(c) Availability

Figure 19: Data used in example #2 after resampling to obtain a reference timestep of 6h.

8.2.2 Model description

The model includes one model class, and the hidden state variables are

$$\mathbf{x} = [x_{\mathtt{D}}^{\mathtt{LL}}, x_{\mathtt{D}}^{\mathtt{AR}}, x_{\mathtt{T}}^{\mathtt{LL}}, x_{\mathtt{T}}^{\mathtt{P1,yearly}}, x_{\mathtt{T}}^{\mathtt{P2,yearly}}, x_{\mathtt{T}}^{\mathtt{P1,daily}}, x_{T}^{\mathtt{P2,daily}}, x_{\mathtt{T}}^{\mathtt{AR}}],$$

where D and T refer to the displacement and temperature time series, respectively. The periodic patterns observed on the displacement are considered through a dependency of the displacement on the hidden state variables of the periodic and autoregressive components of the temperature time series (Section 10.7). The associated model parameters are

$$\begin{aligned}\boldsymbol{\theta} = [&\sigma_{w,\mathtt{D}}^{\mathtt{LL}}, \phi_{D}^{\mathtt{AR}}, \sigma_{w,\mathtt{D}}^{\mathtt{AR}}, \sigma_{v,\mathtt{D}},\\ &\sigma_{w,\mathtt{T}}^{\mathtt{LL}}, p_{\mathtt{T}}^{\mathtt{P,yearly}}, \sigma_{w,\mathtt{T}}^{\mathtt{P,yearly}}, p_{\mathtt{T}}^{\mathtt{P,daily}}, \sigma_{w,\mathtt{T}}^{\mathtt{P,daily}}, \phi_{\mathtt{T}}^{\mathtt{AR}}, \sigma_{w,\mathtt{T}}^{\mathtt{AR}}, \sigma_{v,\mathtt{T}}, \phi_{\mathtt{P}_y}^{\mathtt{D|T}}, \phi_{\mathtt{P}_d}^{\mathtt{D|T}}, \phi_{\mathtt{AR}}^{\mathtt{D|T}}].\end{aligned}$$

The optimized model parameters values computed using the Newton-Raphson algorithm (see 10.4) with a training period of 180 days are

$$\begin{aligned}\boldsymbol{\theta}^* &= [0, 0.90, 0.037, 1.94 \times 10^{-5},\\ &0, 365.2422, 0, 1, 0, 0.98, 0.86, 1.14 \times 10^{-4}, -0.013, 0.0706, 0.00073].\end{aligned}$$

The estimated initial hidden states mean and covariance values are

$$\begin{aligned}\boldsymbol{\mu}_0^* &= [25.9, -2.55 \times 10^{-5}, 5.53, 16.3, -0.999, 0.263, 0.669, 3.77]^\intercal, \text{and}\\ \boldsymbol{\Sigma}_0^* &= \text{diag}([4.27 \times 10^{-5}, 1.68 \times 10^{-3}, 0.715, 0.338, 0.341, 6.73 \times 10^{-5}, 6.73 \times 10^{-5}, 1.81]).\end{aligned}$$

The hidden states computed using the estimated model parameters and initial hidden states are presented in Figure 20.

8.2.3 Run the example from the pre-existing configuration file

There is a configuration file CFG_Example_DISPTEMP_optim.m which is located in the "config_files" folder of the OpenBDLM package. CFG_Example_DISPTEMP_optim.m contains the optimized model parameters and optimized initial hidden states values. There is also a data file DATA_Example_DISPTEMP_optim.mat that is located in the "data/mat" subfolder. Therefore, it is possible to run the example #2 by following the steps below while interacting with the MATLAB command line:

1. Start OpenBDLM. Type `OpenBDLM_main('CFG_Example_DISPTEMP_optim.m');`.
2. Access hidden states estimation menu. Type `3`.
3. Run the Kalman filter to estimate the hidden states. Type `1`.
4. Save and quit. Type `Q`.

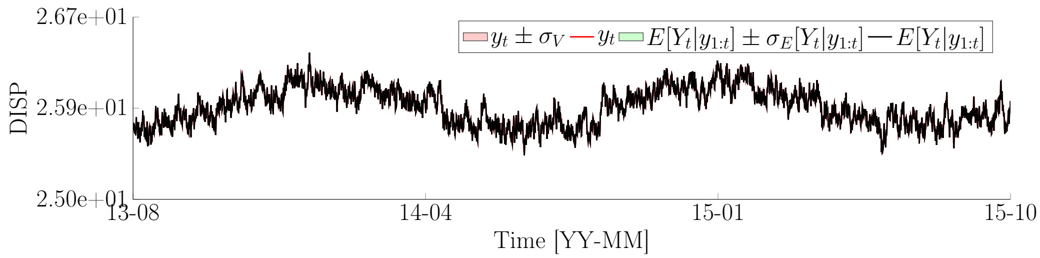
8.2.4 Run the example from command line interaction

The analysis of a new dataset usually requires to start from scratch. This section explains how to run the example #2 from scratch, that is, how to load and resample the data presented in Figure 18, configure the model, estimate the model parameters and estimate the hidden states. This may be done by following steps below while interacting with the MATLAB command line:

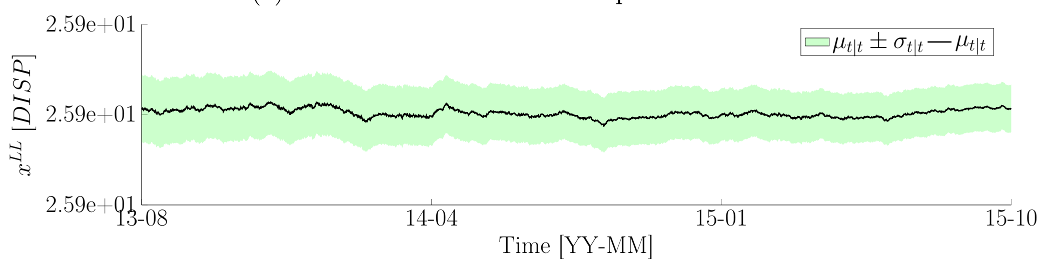
1. Start OpenBDLM. Type `OpenBDLM_main;`.
2. Choose the interactive tool. Type `0`.
3. Enter the project name. Type `Example_DISPTEMP`.
4. Disregard generating synthetic data. Type `no`.
5. Load new data. Type `0`.
6. Select from the graphical user interface the data files located in the "/data/csv/Example_DISPTEMP/" folder. The Figure 18 that represents the raw data should popup on screen.

7. Access the resampling menu. Type `4`.
8. Resample data to obtain timesteps of 6h (0.25 day). Type `0.25`. The Figure 19 should popup on the screen this time for the resampled data.
9. Save and continue. Type `7`. The same figures should popup on the screen again.
10. Select dependency for the time series #1. Type `[2]`.
11. Select dependency for the time series #2. Type `[0]`.
12. Select the number of model classes. Type `1`.
13. Select the model block components for time series #1. Type `[11 41]`.
14. Select the model block components for time series #2. Type `[11 31 31 41]`.
15. Access the training period modification menu. Type `13`.
16. Modify the training period. Type `1`.
17. Choose the starting time (day). Type `1`.
18. Choose the end time (day). Type `180`.
19. Access model parameter estimation menu. Type `1`.
20. Start the Newton-Raphson algorithm. Type `1`. Once the algorithm has converged, the optimized model parameters values should be close to the values presented in §8.2.2. Note also that it is possible to get slightly different parameter values[12].
21. Estimate the initial hidden states values. Type `2`.
22. Estimate the hidden states using the Kalman filter. Type `1`. The estimation should be similar to the results presented in Figure 20.
23. Access export menu. Type `17`.
24. Export the current project in a configuration file. Type `1`.
25. Save and quit OpenBDLM. Type `Q`.

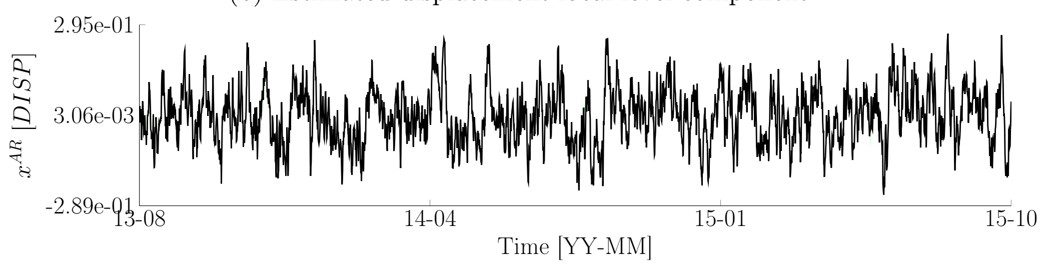
[12] Keep in mind that the optimization may take several minutes. It is possible to abort the analysis here and to load the configuration file called CFG_Example_DISPTEMP_optim.m to load pre-computed values of model parameters, as presented in Section 8.2.3.



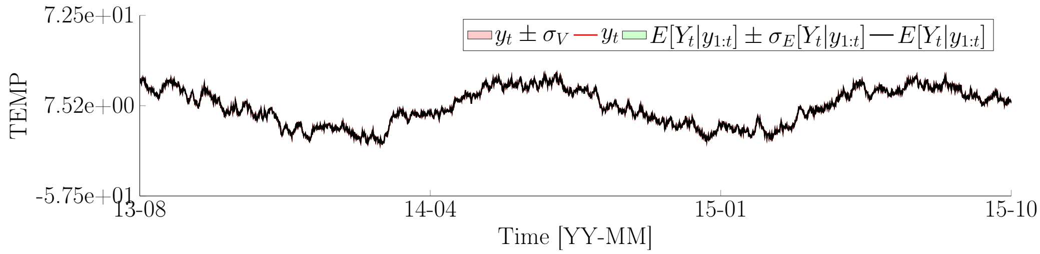
(a) Observed and estimated displacement data



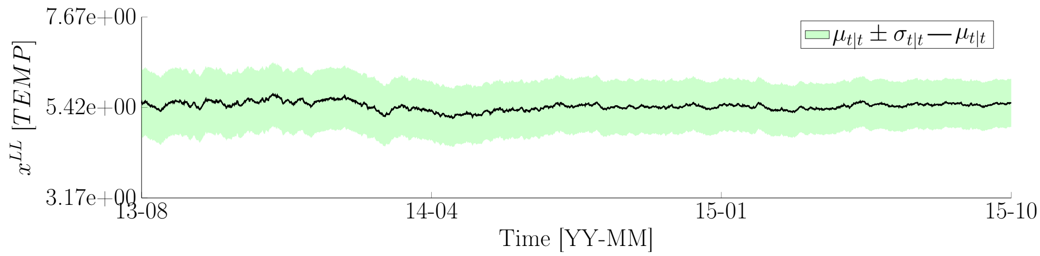
(b) Estimated displacement local level component.



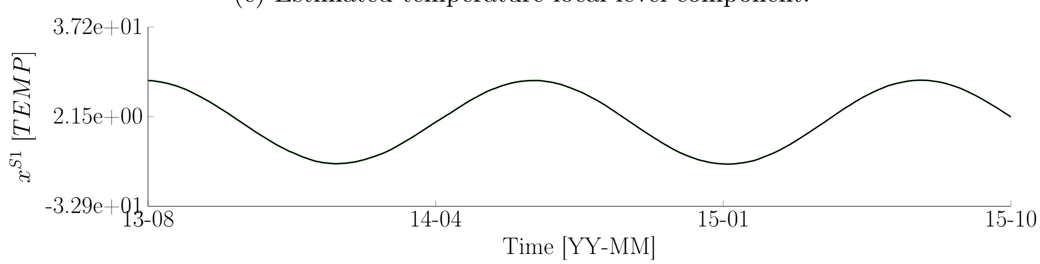
(c) Estimated displacement autoregressive component.



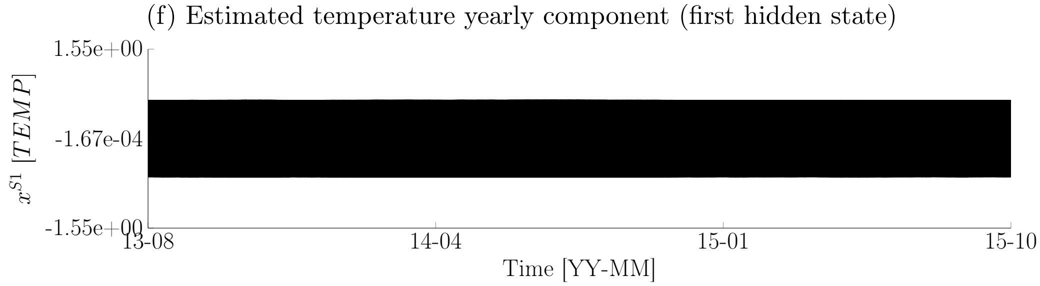
(d) Observed and estimated temperature data



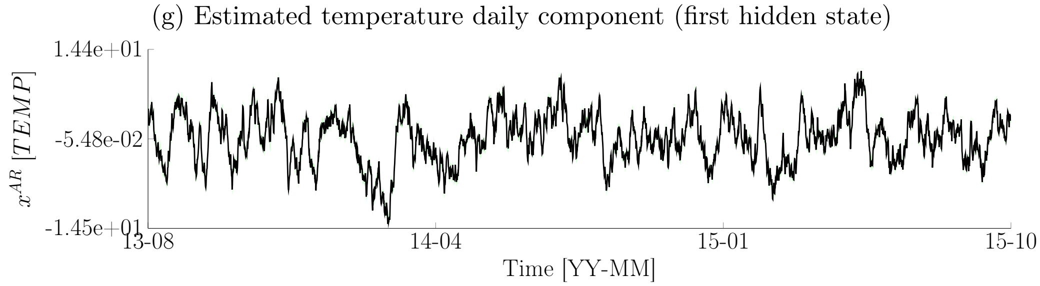
(e) Estimated temperature local level component.



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



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



(h) Estimated temperature autoregressive component

Figure 20: Estimated results using OpenBDLM with the optimized model parameters and initial hidden states. The hidden states are estimated from the data presented in Figure 19a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

8.3 Example #3: time series with anomaly

8.3.1 Data description

This example uses a synthetic time series that mimics displacement data measured on a bridge. The Figure 21a shows that data points exist between May 2008 and April 2012 with the exception of a period of several months where data is missing. The timestep is non-uniform; it varies from 1 hour to 10 days (see Figure 21b). The most frequent (i.e reference) time step is 12 hour (see Section 10.6.1). The baseline switches between a trend stationary to acceleration stationary dynamics during a specific time window to mimic a fictitious anomaly. The anomaly time window has a length of 26 days, starting on June 25, 2010. The displacement also exhibits a yearly periodic and autoregressive patterns.

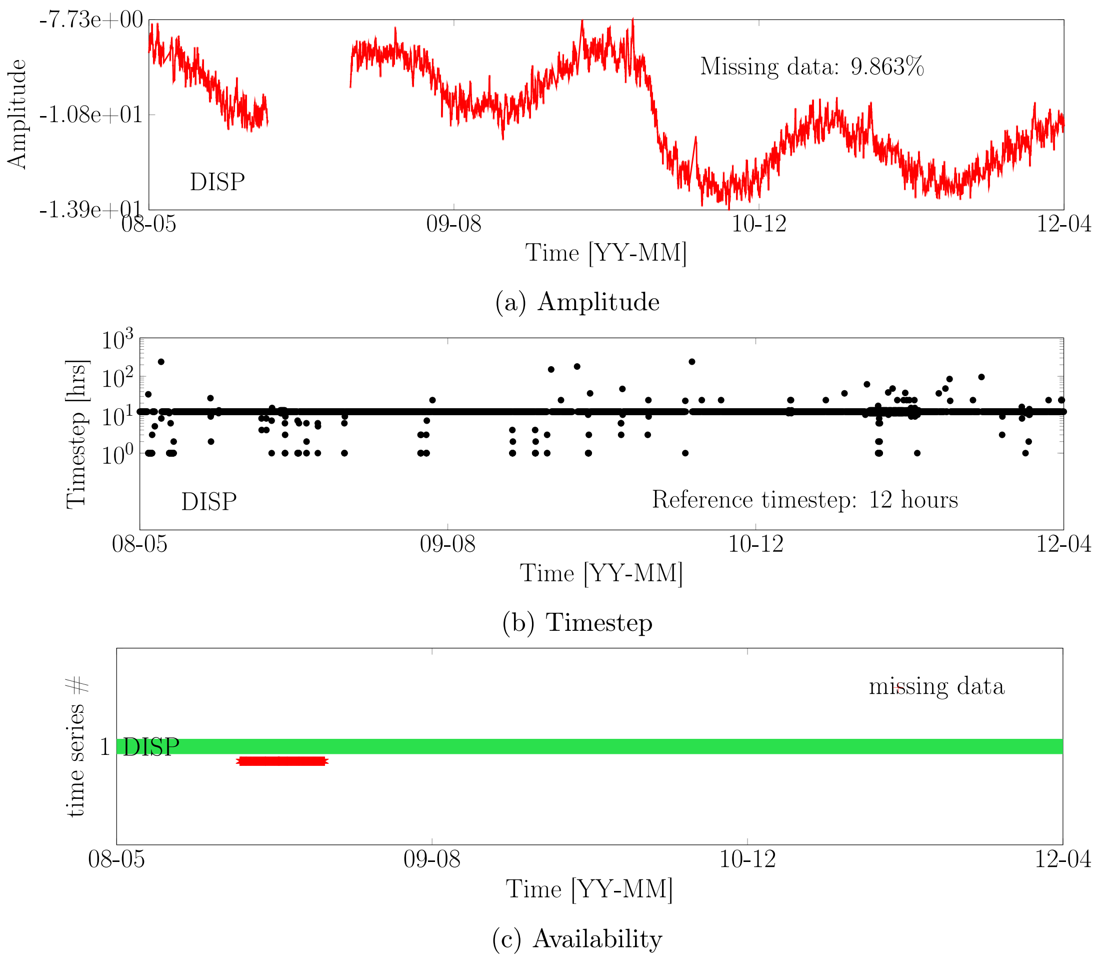


Figure 21: Data used in example #3

.

8.3.2 Model description

The model includes two model classes, and the hidden state variables are

$$\mathbf{x}^1 = [x^{\texttt{L}}, x^{\texttt{LT}}, x^{\texttt{LAc}}, x^{\texttt{P1,yearly}}, x^{\texttt{P2,yearly}}, x^{\texttt{AR}}]$$

for the model class #1, and

$$\mathbf{x}^2 = [x^{\mathtt{L}}, x^{\mathtt{LT}}, x^{\mathtt{LA}}, x^{\mathtt{P1},\text{yearly}}, x^{\mathtt{P2},\text{yearly}}, x^{\mathtt{AR}}]$$

for the model class #2. The associated model parameters are

$$\begin{aligned}\boldsymbol{\theta} = [&\sigma_w^{\mathtt{TcA},1}, p^{\mathtt{P},\text{yearly}}, \sigma_w^{\mathtt{P},\text{yearly}}, \phi^{\mathtt{AR}}, \sigma_w^{\mathtt{AR}},\\ &\sigma_w^{\mathtt{LA},2}, \sigma_w^{\mathtt{TcA},12}, \sigma_w^{\mathtt{LA},21}, \sigma_v^1, \sigma_v^2, Z^{11}, Z^{22}].\end{aligned}$$

The optimized model parameters values computed using the Newton-Raphson algorithm (see 10.4) with a training period including the entire dataset are

$$\begin{gathered}\boldsymbol{\theta}^* = [0, 365.24, 0, 0.73, 0.214,\\ 0.104, 1.79 \times 10^{-12}, 0.073.5 \times 10^{-7}, 0.013, 0.10671, 0.99998, 0.99915].\end{gathered}$$

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

$$\begin{aligned}\boldsymbol{\mu}_0^{1,*} &= [-9.36, -0.0135, -7.79 \times 10^{-9}, 1.03, 0.00033, -0.597]^{\intercal}, \text{and}\\ \boldsymbol{\Sigma}_0^{1,*} &= \text{diag}[0.128, 0.000956, 2.42, 0.00101, 0.0009, 0.32], \text{and}\\ \pi_0^{1,*} &= 0.995.\end{aligned}$$

The estimated initial hidden states mean, covariance, and model probability for model class #2 are

$$\begin{aligned}\boldsymbol{\mu}_0^{2,*} &= [-9.48, 0.11, -0.067, 1.03, 0.0003, -0.512]^{\intercal}, \text{and}\\ \boldsymbol{\Sigma}_0^{2,*} &= \text{diag}[0.61, 0.495, 0.998, 0.001, 0.0008, 0.675], \text{and}\\ \pi_0^{2,*} &= 0.00472.\end{aligned}$$

The hidden states estimated using the Kalman smoother along with the optimal model parameters and initial hidden states are presented in Figure 22.

8.3.3 Run the example from the pre-existing configuration file

There is a configuration file CFG_Example_DISP_ANOMALY_optim.m which is located in the “config_files” folder of the OpenBDLM package.
CFG_Example_DISP_ANOMALY_optim.m contains the optimized model parameters and initial hidden states values. There is also a data file
DATA_Example_DISP_ANOMALY_optim.mat that is located in the “data/mat” subfolder. Therefore, it is possible to run the example #3 by following the steps below while interacting with the MATLAB command line:

1. Start OpenBDLM. Type `OpenBDLM_main('CFG_Example_DISP_ANOMALY_optim.m');`.
2. Access hidden states estimation menu. Type `3`.
3. Run the Kalman smoother to estimate the hidden states. Type `2`.
4. Save and quit. Type `Q`.

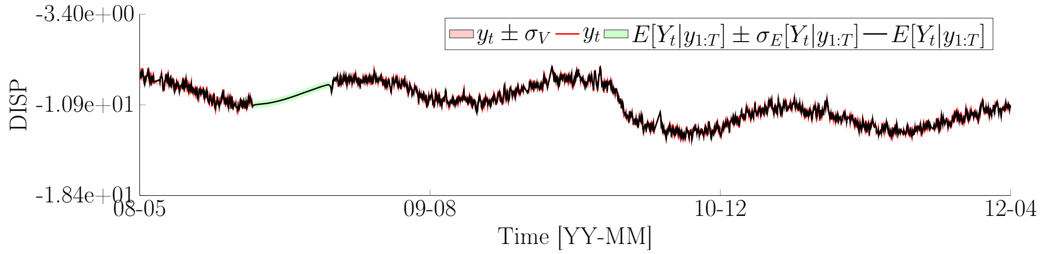
8.3.4 Run the example from command line interaction

The analysis of a new dataset usually requires to start from scratch. This section explains how to run the example #3 from scratch, that is, how to load the data presented in Figure 21, configure the model, estimate the model parameters and estimate the hidden states. This may be done by following steps below while interacting with the MATLAB command line:

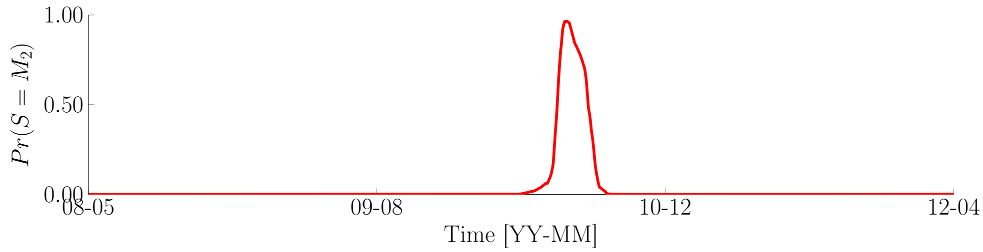
1. Start OpenBDLM. Type `OpenBDLM_main;`.
2. Choose the interactive tool. Type `0`.
3. Enter the project name. Type `Example_DISP_ANOMALY`.
4. Disregard generating synthetic data. Type `no`.
5. Load new data. Type `0`.
6. Select from the graphical user interface the data file Example_DISP_ANOMALY_DISP.csv data file located in the folder " /data/csv/Example_DISP_ANOMALY/". The Figure 21 representing the processed data should popup on screen.
7. Save and continue without pre-processing. Type `7`. The Figure 21 should popup again on screen.
8. Select the number of model classes. Type `2`.
9. Select the model block components for model class #1. Type `[23 31 41]`.
10. Select the model block components for model class #2. Type `[13 31 41]`.
11. Select the model parameter constrains. Type `[0 1 1]` [13].
12. Access model parameter estimation menu. Type `1`.
13. Start Newton-Raphson algorithm. Type `1`. Once the algorithm has converged, the optimized model parameters values should be close to the values presented in Section 8.3.2. Note that it is possible to get slightly different parameter values[14].
14. Estimate the initial hidden states values. Type `2`.
15. Access hidden states estimation menu. Type `3`.
16. Estimate the hidden states using the Kalman smoother. Type `2`. The estimation should be corresponding to the results presented in Figure 22.
17. Access export menu. Type `17`.
18. Export the current project in a configuration file. Type `1`.
19. Save and quit OpenBDLM. Type `Q`.

[13] `[0 1 1]` forces the second and third block component (i.e. 31 and 41) to share the same parameters.

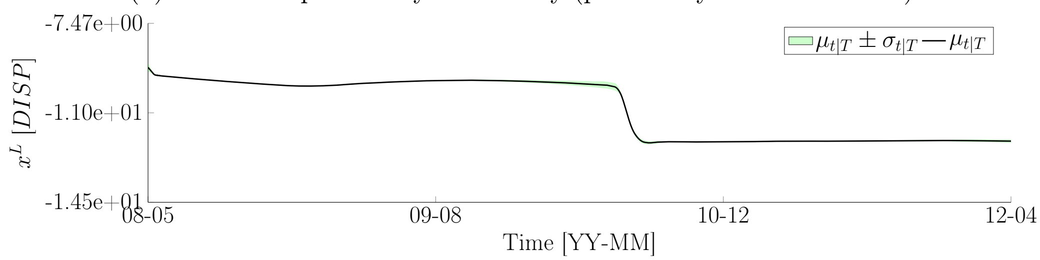
[14] Keep in mind that the optimization may take several minutes. It is possible to abort the analysis here and to load the configuration file called CFG_Example_DISP_ANOMALY_optim.m to load pre-computed values of model parameters, as presented in Section 8.3.3.



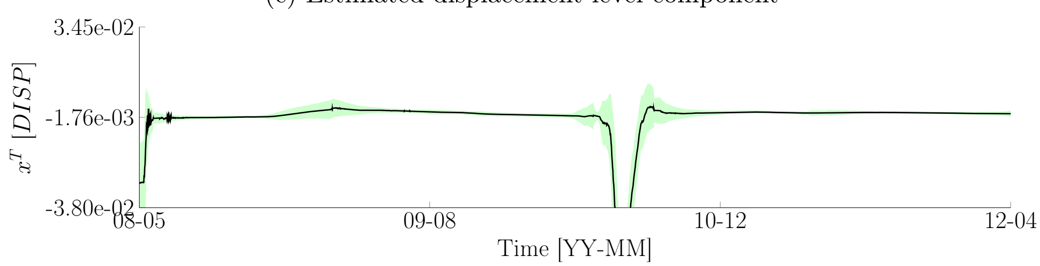
(a) Observed and estimated displacement data



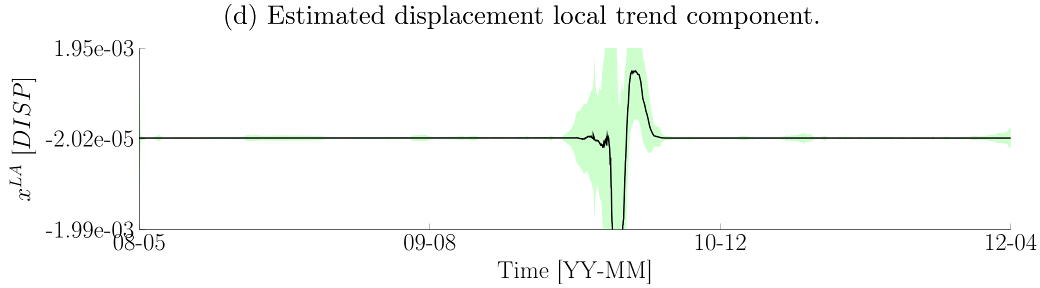
(b) Estimated probability of anomaly (probability of model class 2)



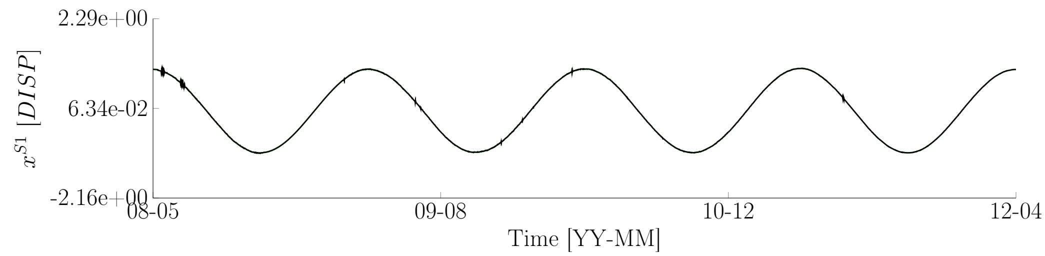
(c) Estimated displacement level component



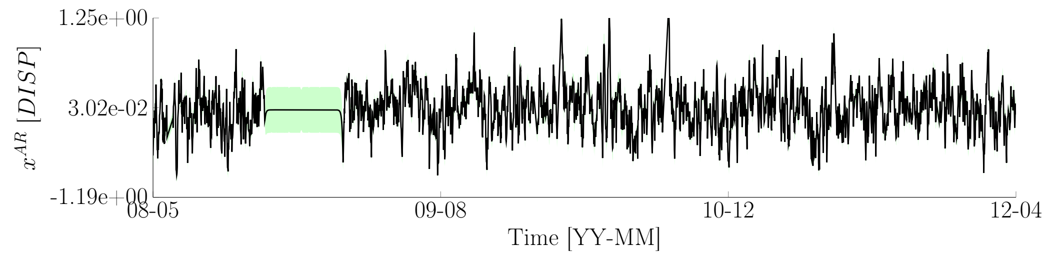
(d) Estimated displacement local trend component.



(e) Estimated displacement local acceleration component.



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



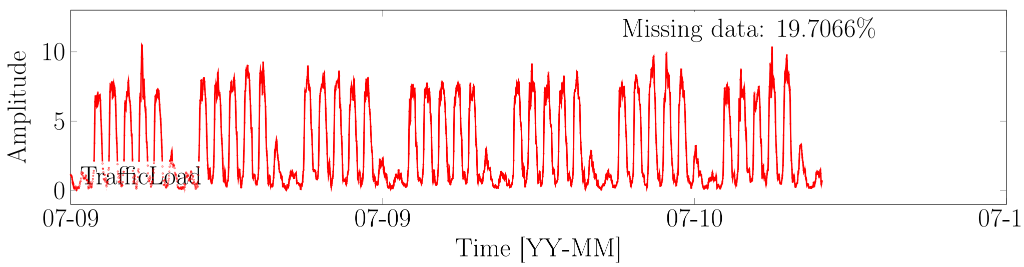
(g) Estimated displacement autoregressive component

Figure 22: Estimated results using OpenBDLM with the optimized model parameters and initial hidden states. The hidden states are estimated from the data presented in Figure 21a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

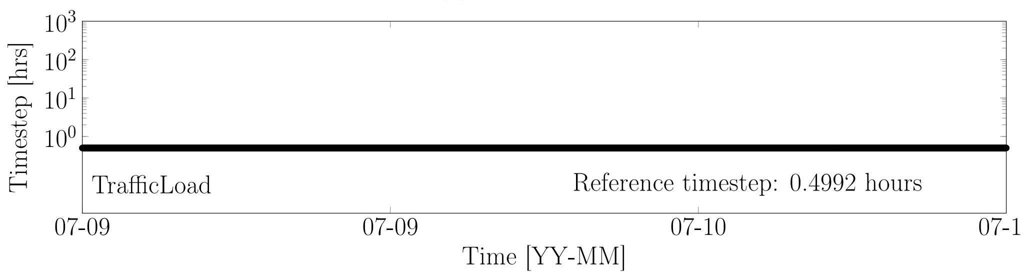
8.4 Example #4: single time series analysis with non-harmonic periodic pattern

8.4.1 Data description

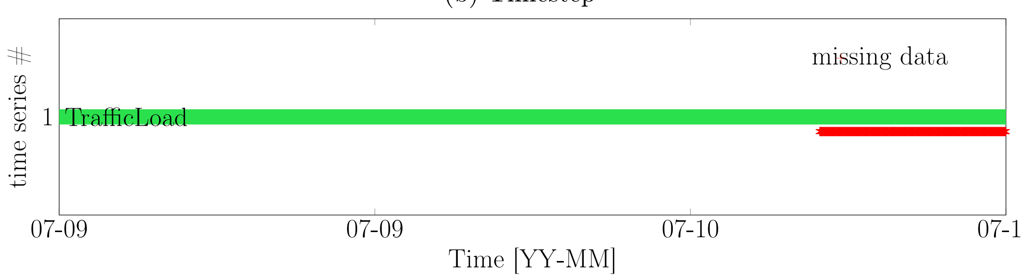
This case-study is conducted on the traffic loading data collected on the Tamar Bridge [1, 2]. The Figure 23a shows that data points exist between September 01 and October 21, 2007. Beyond that date, the timestamps are associated with `NaN` (i.e. missing data) in order to forecast the predicted traffic load for a period of two weeks beyond the available data (see Figure 23c). The timestep is 30 minutes for all data points (Figure 23a). The time series is stationnary with a level, a periodic pattern with a period of 7 days, and autoregressive pattern.



(a) Amplitude



(b) Timestep



(c) Availability

Figure 23: Data used in the example #4.

8.4.2 Model description

The model includes one model class, and the hidden states variables are

$$\mathbf{x} \quad = \quad \left[x^{\mathtt{LL}}, x^{\mathtt{KR}}_0, x^{\mathtt{KR}}_1, \cdots, x^{\mathtt{KR}}_{100}, x^{\mathtt{AR}}\right]^{\mathtt{T}}.$$

The associated model parameters are

$$\boldsymbol{\theta} = \left[\sigma_w^{\mathtt{LL}}, \ell^{\mathtt{KR}}, p^{\mathtt{KR}}, \sigma_{w,0}^{\mathtt{KR}}, \sigma_{w,1}^{\mathtt{KR}}, \phi^{\mathtt{AR}}, \sigma_w^{\mathtt{AR}}, \sigma_v\right]. \tag{1}$$

The optimized model parameters values computed using the Newton-Raphson algorithm (see 10.4) with a training period of 14 days are

$$\boldsymbol{\theta}^* = [0, 0.052, 7, 2.1 \times 10^{-5}, 0, 0.81, 0.30, 4.6 \times 10^{-5}].$$

The estimated initial hidden states mean and covariance are

$$\begin{aligned}\boldsymbol{\mu}_0^* &= [3.11, 0, -1.72, -1.87, \ldots, -0.126]^{\intercal}, \text{and}\\ \boldsymbol{\Sigma}_0^* &= \text{diag}[0.0828, 8.19, 0.459, 0.436, \ldots, 0.182].\end{aligned}$$

The hidden states computed using the estimated model parameters and initial hidden states are presented in Figure 24.

8.4.3 Run the example from the pre-existing configuration file

There is a configuration file CFG_Example_TRAFFIC_optim.m which is located in the “config_files” folder of the OpenBDLM package. CFG_Example_TRAFFIC_optim.m contains the optimized model parameters and initial hidden states values. There is also a data file DATA_Example_TRAFFIC_optim.mat that is located in the “data/mat” subfolder. Therefore, it is possible to run the example #4 by following the steps below while interacting with the MATLAB command line:

1. Start OpenBDLM. Type `OpenBDLM_main('CFG_Example_TRAFFIC_optim.m');`.
2. Access hidden states estimation menu. Type `3`.
3. Run the Kalman filter to estimate the hidden states. Type `1`.
4. Save and quit. Type `Q`.

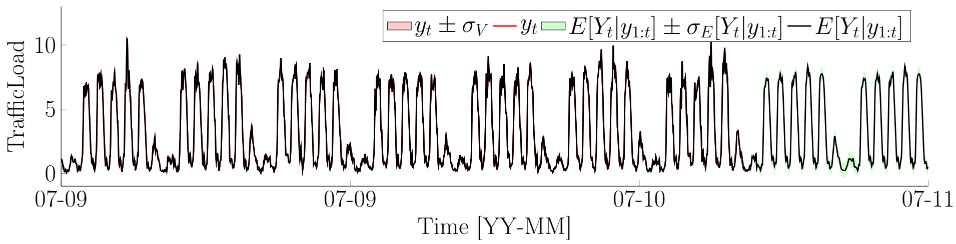
8.4.4 Run the example from command line interaction

The analysis of a new dataset usually requires to start from scratch. This section explains how to run the example #4 from scratch, that is, how to load the data presented in Figure 23, configure the model, estimate the model parameters and estimate the hidden states as presented in Figure 24. This may be done by following steps below while interacting with the MATLAB command line:

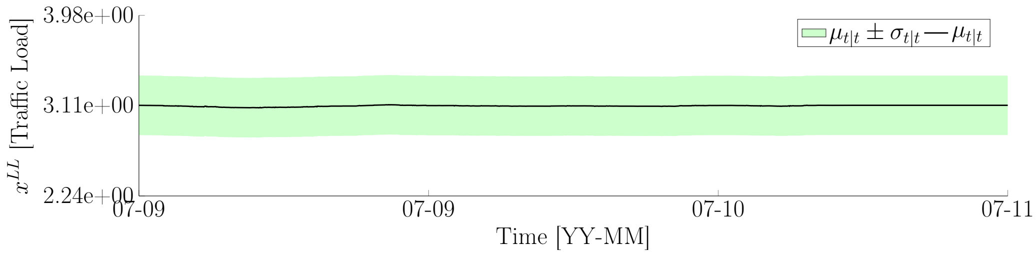
1. Start OpenBDLM. Type `OpenBDLM_main;`.
2. Choose the interactive tool. Type `0`.
3. Enter the project name. Type `Example_TRAFFIC`.
4. Disregard generating synthetic data. Type `no`.
5. Load new data. Type `0`.

6. Select from the graphical user interface the data file Example_TRAFFIC_TrafficLoad.csv located in the folder "/data/csv/Example_TRAFFIC". The Figure 23 representing the raw data should popup on screen.
7. Save and continue without pre-processing. Type `7`. The Figure 23 should popup on screen again,.
8. Select the number of model classes. Type `1`.
9. Select the model block components. Type `[11 51 41]`.
10. Access model parameters menu. Type `11`.
11. Modify a model parameter. Type `1`.
12. Modify the third model parameter (period of the kernel regression component). Type `3`.
13. Provide new value for the period (in days). Type `7`.
14. Provide new bounds for the period. Type `[NaN NaN]`.
15. Access the training period modification menu. Type `13`.
16. Modify the training period. Type `1`.
17. Choose the starting time (day). Type `1`.
18. Choose the end time (day). Type `14`.
19. Access model parameter estimation menu. Type `1`.
20. Start Newton-Raphson algorithm. Type `1`. Once the algorithm has converged, the optimized model parameters values should be close to the values presented in Section 8.4.2. Note that it is possible to get slightly different parameter values[15].
21. Estimate the initial hidden states values. Type `2`.
22. Access hidden states estimation menu. Type `3`.
23. Estimate the filtered hidden states. Type `1`. The estimation should correspond to the results presented in Figure 24.
24. Access export menu. Type `17`.
25. Export the current project in a configuration file. Type `1`.
26. Save and quit OpenBDLM. Type `Q`.

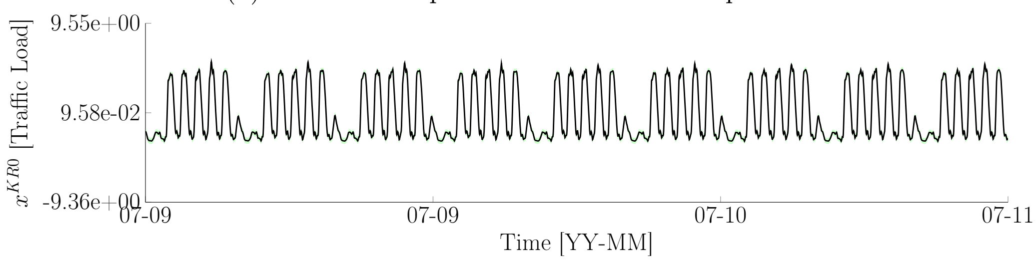
[15] Keep in mind that the optimization may take several minutes. It is possible to abort the analysis here and to load the configuration file called CFG_Example_TRAFFIC_optim.m to load pre-computed values of model parameters, as presented in Section 8.4.3.



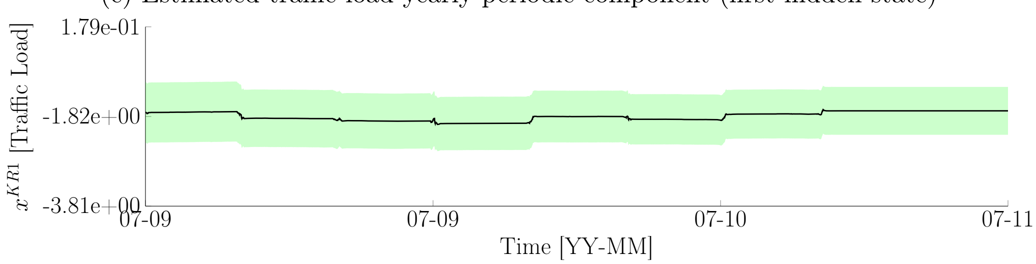
(a) Observed and estimated traffic load data



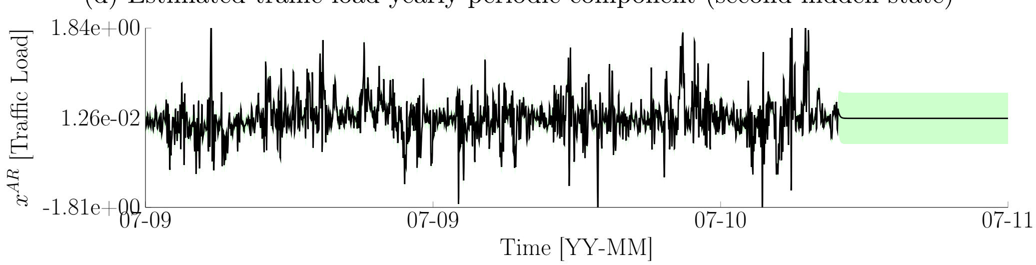
(b) Estimated displacement local level component.



(c) Estimated traffic load yearly periodic component (first hidden state)



(d) Estimated traffic load yearly periodic component (second hidden state)



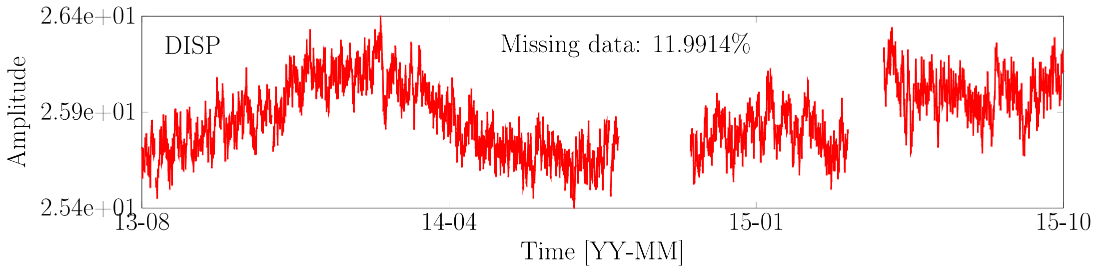
(e) Estimated traffic load autoregressive component

Figure 24: Estimated results using OpenBDLM with the optimized model parameters and optimized initial hidden states. The hidden states are estimated from the data presented in Figure 23a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states, respectively.

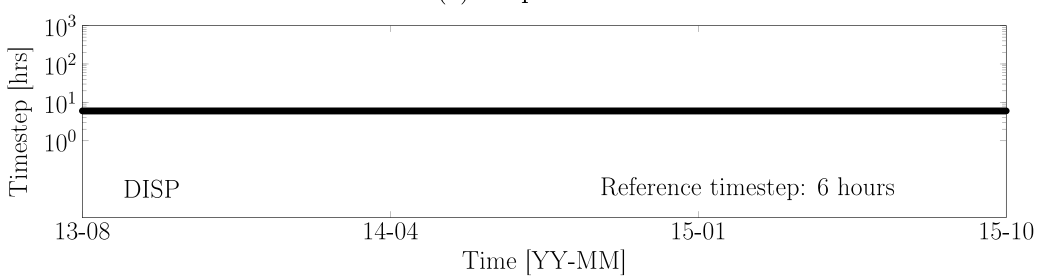
8.5 Example #5: single time series analysis with interventions

8.5.1 Data description

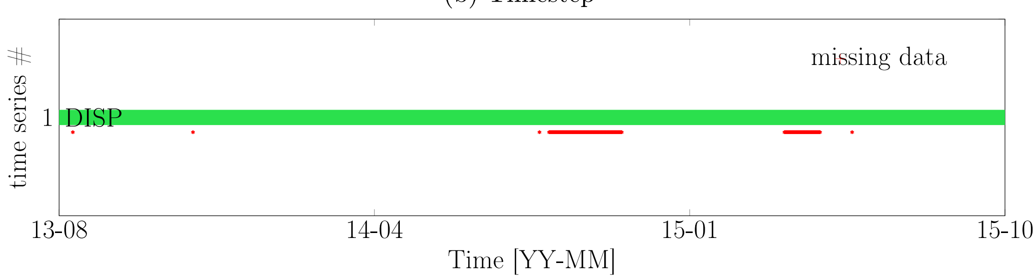
This case is based on Example #1 (see 8.1) where we removed some part of the data and introduce discrete shifts into it. This example illustrates what typically happens a sensor fails; when a sensor fails, the data start to be missing (i.e. `NaN`) and it often takes from several weeks to several months before the sensor is replaced. When the sensor is replaced, it is in most cases re-initialized at a different initial value than the previous sensor which lead to a discrete shift in the time series, as depicted in Figure 25. Here in addition to the components employed for Example #1, we also employ a Local intervention in order to estimate the magnitude of the corrections require to eliminate the discrete shifts created by sensor replacement.
Note that again, in this example, we choose to resample the original data in order to have a timesteps of 6h instead of 1h.



(a) Amplitude



(b) Timestep



(c) Availability

Figure 25: Raw data in the example #1 where the reference timestep is 1h.

8.5.2 Model description

The model includes one model class, and the hidden states variables are

$$\mathbf{x} = [x^{\mathtt{LL}}, x^{\mathtt{P1},\text{yearly}}, x^{\mathtt{P2},\text{yearly}}, x^{\mathtt{P1},\text{daily}}, x^{\mathtt{P2},\text{daily}}, x^{\mathtt{AR}}, x^{\mathtt{LI}}].$$

When using a Level intervention component, the user must provide discrete timestamps where it is required to estimate the magnitude of a discrete shift in the dataset (see §10.5.10). A user can do so by specifying in the configuration file `data.interventions=[t_1, t_2, ... , t_n]`. The configuration file for this problem is presented in the Listing 14 where the timestamps where the shift occur are `data.interventions=[735929.875, 736099.875];`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% A - Project name
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
misc.ProjectName='Example_DISP_INTERVENTION';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% B - Data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dat=load('DATA_Example_DISP_INTERVENTION.mat');
data.values=dat.values;
data.timestamps=dat.timestamps;
data.labels={'DISP'};
data.interventions=[735929.875, 736099.875];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 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
% 61: Level Intervention

% Model components
model.components.block{1}={[11 31 31 41 61 ] };

% 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,            365.2422,      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,            0.90399,       6      %#6
     '\sigma_w',   'AR',          '1',    '1',  [0  Inf    ],    'N/A',       NaN,     NaN,            0.035428,      7      %#7
     '\mu_b',      'LI',          '1',    '1',  [NaN  NaN  ],    'N/A',       NaN,     NaN,            0,             8      %#8
     '\sigma_b',   'LI',          '1',    '1',  [NaN  NaN  ],    'N/A',       NaN,     NaN,            0.16276,       9      %#9
     '\sigma_v',   '',            '1',    '1',  [0  Inf    ],    'N/A',       NaN,     NaN,            6.193e-06,     10     %#10
};
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% E - Initial states values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial hidden states mean for model 1:
model.initX{ 1 }=[      25.9    -0.194  -0.01   -0.00411        0.0551  -0.0127 0     ]';
% Initial hidden states variance for model 1:
model.initV{ 1 }=diag([        8.26E-05        0.000143        0.000106        4.86E-07        4.86E-07        0.00167 1E-20  ]);
% Initial probability for model 1
model.initS{1}=[1     ];
```

Listing 14: Configuration file for the example #5

The model parameters associated with this model are

$$\boldsymbol{\theta} = [\sigma_w^{\mathtt{LL}}, p^{\mathtt{P},\text{yearly}}, \sigma_w^{\mathtt{P},\text{yearly}}, p^{\mathtt{P},\text{daily}}, \sigma_w^{\mathtt{P},\text{daily}}, \phi^{\mathtt{AR}}, \sigma_w^{\mathtt{AR}}, \mu_b^{\mathtt{LI}}, \sigma_b^{\mathtt{LI}}, \sigma_v].$$

The optimized model parameters values computed using the Newton-Raphson algorithm (see 10.4) with a training period of 180 days are

$$\boldsymbol{\theta}^* = [0, 365.2422, 0, 1, 0, 0.903, 0.035, 0, 0.166.19 \times 10^{-6}].$$

The estimated initial hidden states mean and covariance values are

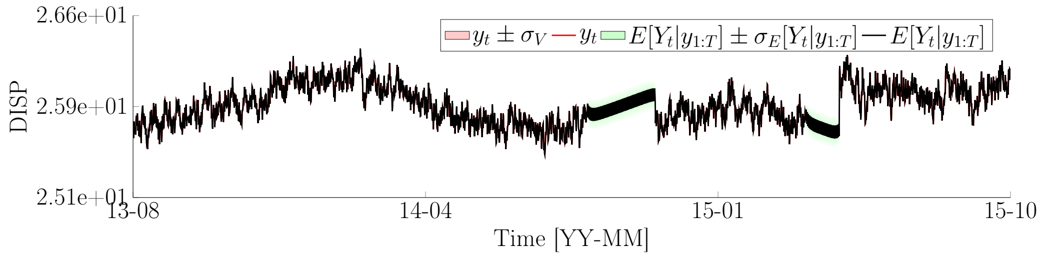
$\boldsymbol{\mu}_0^* = [25.9, -0.194, -0.01, -0.004, 0.055, -0.013, 0]^\intercal$, and
$\boldsymbol{\Sigma}_0^* = \mathrm{diag}([8.26 \times 10^{-5}, 1.4 \times 10^{-4}, 1.1 \times 10^{-4}, 4.86 \times 10^{-7}, 4.86 \times 10^{-7}, 1.67 \times 10^{-3} 1E - 20])$.

The hidden states computed using the estimated model parameters and initial hidden states are presented in Figure 26. You can see in Figure 26b that the level remains constant throughout the entire time series despite the jumps. This is because the discrete shift are estimated by the Level intervention component displayed in Figure 26e.

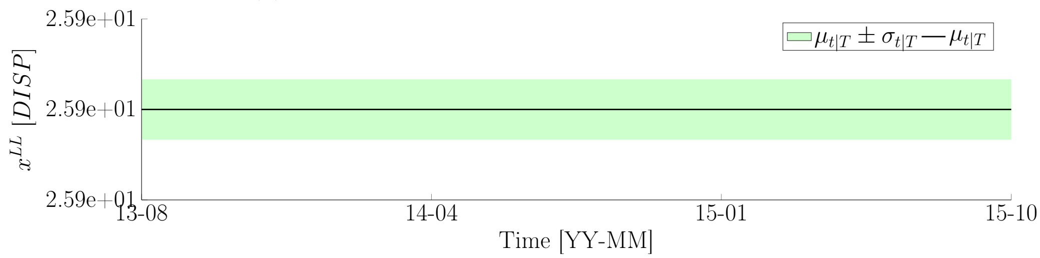
8.5.3 Run the example from the pre-existing configuration file

There is a configuration file CFG_Example_DISP_INTERVENTION_optim.m which is located in the "config_files" folder of the OpenBDLM package. CFG_Example_DISP_INTERVENTION_optim.m contains the optimized model parameters and estimated initial hidden states values (see Listing 14). There is also a data file DATA_Example_DISP_INTERVENTION_optim.mat that is located in the "data/mat" subfolder. Therefore, it is possible to run the example #1 by following the steps below while interacting with the Matlab command line:

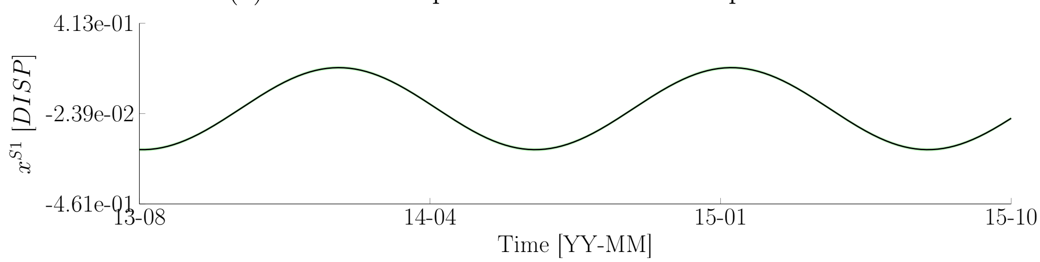
1. Start OpenBDLM. Type `OpenBDLM_main('CFG_Example_DISP_INTERVENTION_optim.m');`.
2. Access hidden states estimation menu. Type `3`.
3. Run the Kalman smoother to estimate the hidden states. Type `1`.
4. Save and quit. Type `Q`.



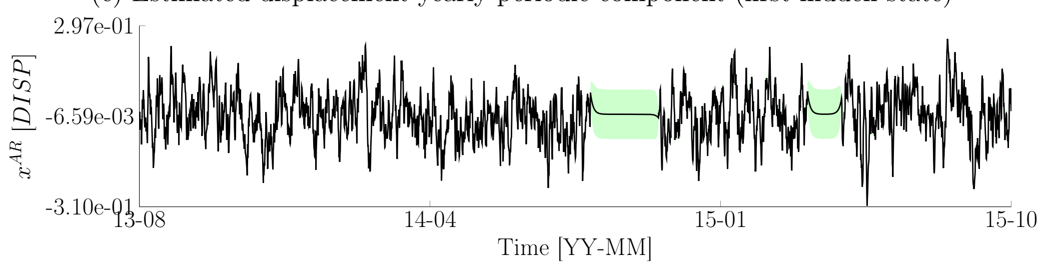
(a) Observed and estimated displacement data



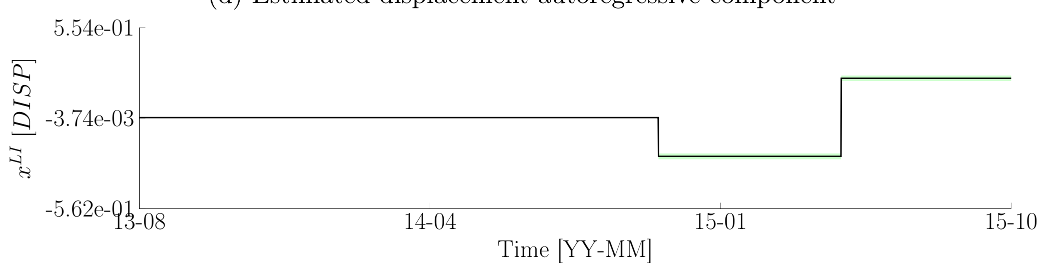
(b) Estimated displacement local level component.



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



(d) Estimated displacement autoregressive component



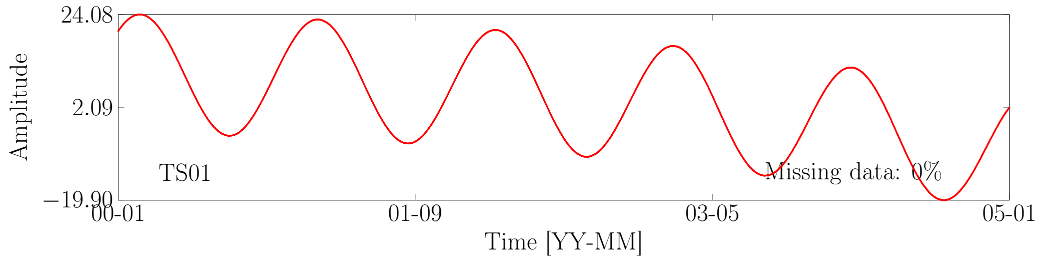
(e) Estimated displacement daily periodic component (first hidden state)

Figure 26: Estimated results using OpenBDLM with the optimized model parameters and estimated initial hidden states. The hidden states are estimated from the data presented in Figure 25a. The solid line and shaded area represent the mean and standard deviation of the estimated hidden states.

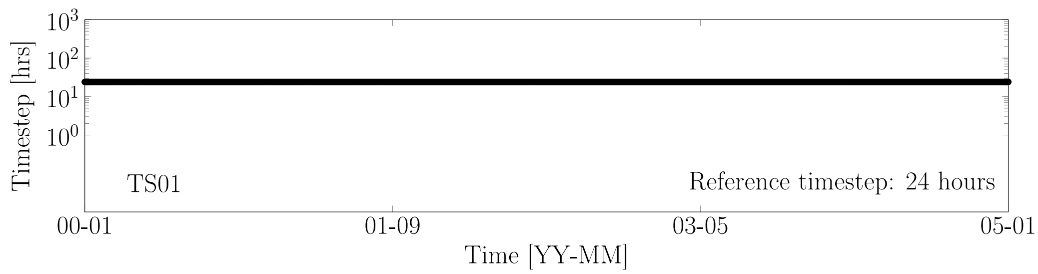
8.6 Example #6: generate and analyze synthetic data

8.6.1 Purpose

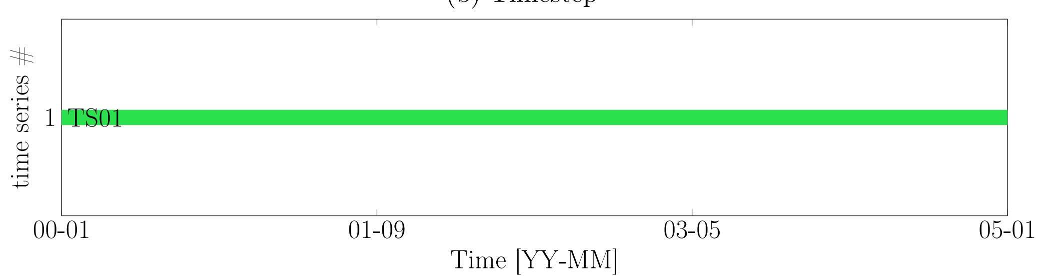
This example illustrates how to create a synthetic time series using OpenBDLM. The objective is to create a 4-years long time series with an acceleration stationary baseline, and a yearly periodic pattern as well as an autoregressive process superimposed into it. The timestep is 1 day. This example corresponds to the OpenBDLM demo presented in Section 1.3.



(a) Amplitude



(b) Timestep



(c) Availability

Figure 27: Data used in example #5

.

8.6.2 Model description

The model includes one model class, and the block components are

$$\mathbf{x} = [x^{\mathtt{L}}, x^{\mathtt{T}}, x^{\mathtt{LA}}, x^{\mathtt{P1},\text{yearly}}, x^{\mathtt{P2},\text{yearly}}, x^{\mathtt{AR}}].$$

The associated model parameters are

$$\boldsymbol{\theta} = [\sigma_w^{\mathtt{LA}}, p^{\mathtt{PD},\text{yearly}}, \sigma_w^{\mathtt{PD},\text{yearly}}, \phi^{\mathtt{AR}}, \sigma_w^{\mathtt{AR}}, \sigma_{v,\mathrm{D}}].$$

The model parameters values assigned by default from OpenBDLM are

$$\boldsymbol{\theta}^{\text{defaut}} = [1 \times 10^{-8}, 365.24, 0, 0.75, 0.01, 0.01].$$

The default initial hidden states mean, covariance, and model probability are

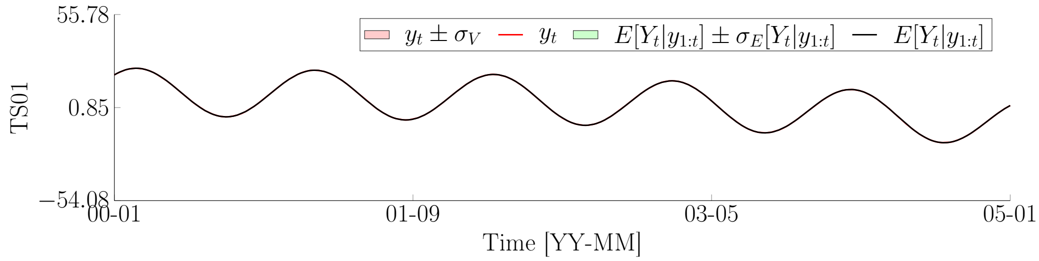
$$\begin{aligned}
\boldsymbol{\mu}_0^{\text{defaut}} &= [10, -1 \times 10^{-5}, -0.001, 10, 10, 0]^\intercal, \text{and} \\
\boldsymbol{\Sigma}_0^{\text{defaut}} &= \text{diag}[0.01, 0.01, 0.01, 0.04, 0.04, 0.01], \\
\pi_0^{1,\text{default}} &= 1.
\end{aligned}$$

The synthetic data generated from this model structure, model parameters values and initial hidden states values are presented in Figure 27. The hidden states computed using the same (i.e. true) model structure, model parameters values and initial hidden states values are presented in Figure 28.

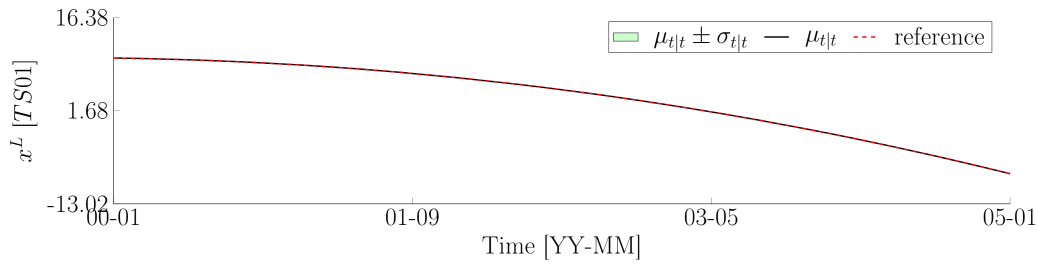
8.6.3 Run the example from command line interaction

This section explains how to run the example #6, that is, how to generate the synthetic data presented in Figure 27, and estimate the hidden states as presented in Figure 28.

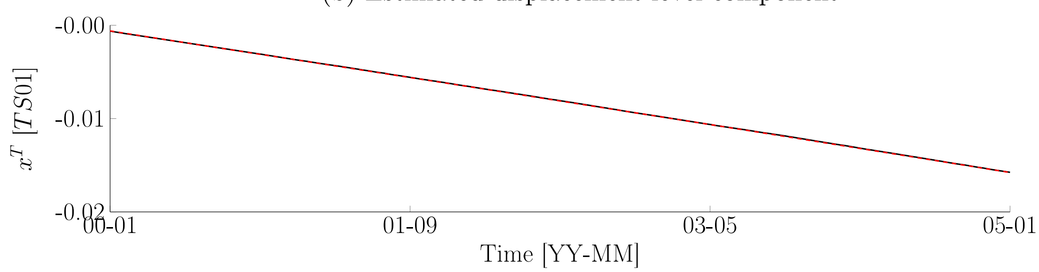
1. Start OpenBDLM. Type `OpenBDLM_main;`.
2. Choose the interactive tool. Type `0`.
3. Enter the project name. Type `Example_SYNTHETIC`.
4. Generate synthetic data. Type `yes`.
5. Provide the number of time series. Type `1`.
6. Provide the date corresponding of the first data sample. Type `2000-01-01`.
7. Provide the date corresponding of the last data sample. Type `2005-01-01`.
8. Provide the timestep in day. Type `1`.
9. Select the number of model classes. Type `1`.
10. Select the model block components. Type `[13 31 41]`. The Figure 27 should popup on screen.
11. Access hidden states estimation menu. Type `3`.
12. Estimate the filtered hidden states. Type `1`. The estimation should correspond to the results presented in Figure 28.
13. Save and quit OpenBDLM. Type `Q`.



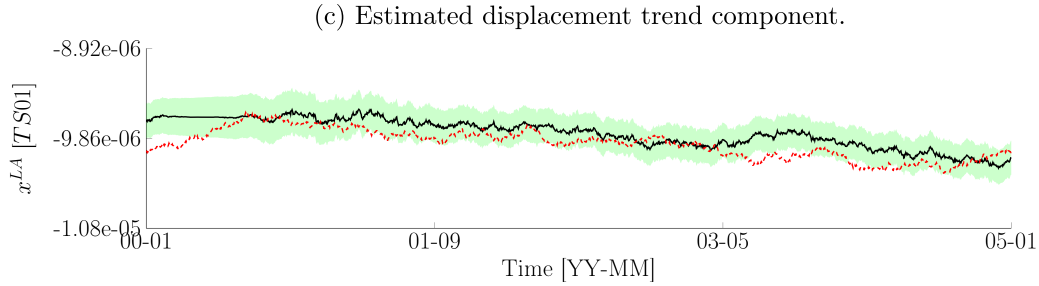
(a) Observed and estimated displacement data



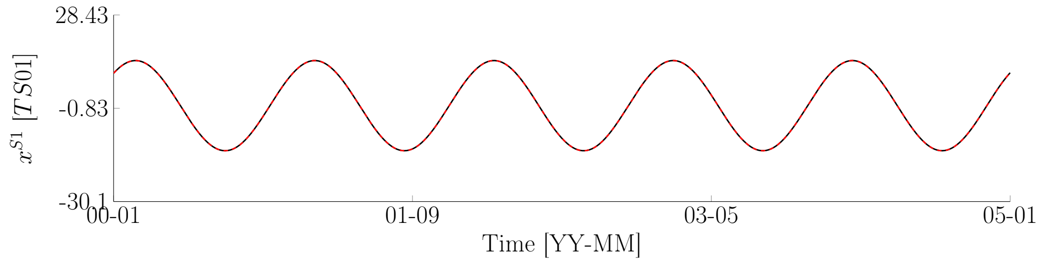
(b) Estimated displacement level component



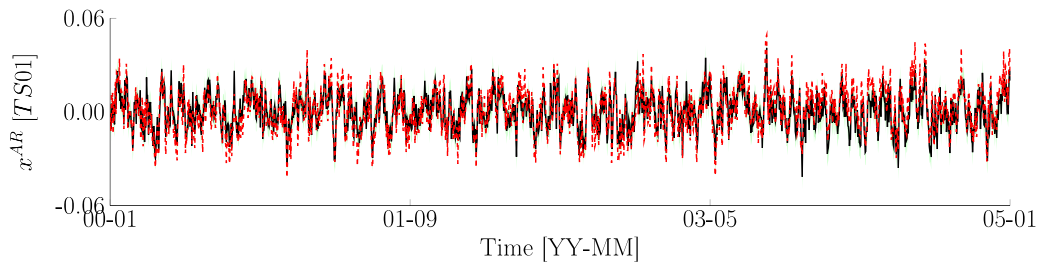
(c) Estimated displacement trend component.



(d) Estimated displacement local acceleration component.



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



(f) Estimated displacement autoregressive component

Figure 28: Estimated results using OpenBDLM with the default model parameters and optimized initial hidden states. The hidden states are estimated from the data presented in Figure 27a. 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.

9 FAQ Troubleshooting

- **The state estimation crashes. What can I do ?**
  There are two well-known issues that make the state estimation to crash.

  - numerical instabilities of the Kalman computation due to missing data and/or non-uniform time step vector. Possible solution: switch to UD computation (see Section 4.6). An alternative consists in removing missing data in the original data, and/or make the timestep vector uniform using the pre-processing tools (see Section 4.2).
  - pinv error. Possible solution: in the `KalmanFilter.m` function, change the tolerance value of the built-in MATLAB function `pinv.m`. See https://www.mathworks.com/help/matlab/ref/pinv.html for more details.

- **The model parameter estimation crashes. What can I do ?**
  This is likely due to the fact that the state estimation crashes. There are two well-known issues that make the state estimation to crash.

  - numerical instabilities of the Kalman computation due to missing data and/or non-uniform time step vector. Possible solution: switch to UD computation (see Section 4.6). An alternative consists in removing missing data in the original data, and/or make the timestep vector uniform using the pre-processing tools (see Section 4.2).
  - pinv error. Possible solution: in the `KalmanFilter.m` function, change the tolerance value of the built-in MATLAB function `pinv.m`. See https://www.mathworks.com/help/matlab/ref/pinv.html for more details.

- **The model parameter estimation is really slow. What can I do ?**
  Estimating model parameter is usually slow process. There are some tips to speed-up the procedure:

  - shorten the training period (see `misc.options.trainingPeriod`).
  - decrease the number of data points by averaging (see Section 4.2).
  - perform parallel computation (see `misc.options.isParallel`). Note that parallel computation requires the MATLAB *Parallel Computing Toolbox*.
  - fix the model parameter values that are known in order to reduce the total number of model parameters to learn (i.e. set the model parameters bound to `[NaN, NaN]`, see Section 4.5).
  - When using the regime switching, constrain model parameters between each other (if applicable) to reduce the total number of model parameters to learn.
  - abort the process and start again with different starting values of model parameters.

- **How to choose the right model structure for my data ?**
  Currently, one has to inspect the data to propose candidates model configurations. Different candidate models can be compared based on the log-likelihood values calculated for test sets not employed to train the model. Moreover, the presence of non-stationarity in the autoregressive hidden states may indicate that the model is

incorrect or, at least, incomplete (note that this may also be due to inadequate model parameters values).

- **I cannot compile the figures exported in .tex files.**
  By default, you need to employ the Lulatex compiler. Lulatex is employed here because of its capacity to compile large figures. If your figure contains few data points (e.g. <50000) you can use the standard Latex compiler by commenting the line `\RequirePackage{luatex85}` in the preamble of the .tex file you want to compile.

- **The default value for model parameters and initial hidden states do not satisfy me. How can I change them ?**
  It is possible to change the default values from the function `buildModel.m`.

- **What is the procedure to save the optimized values for model parameters ?**
  The optimized model parameter values are automatically saved inside a project. Note however that the associated configuration file remains unchanged. It is possible to export the model parameters values in a configuration file using OpenBDLM export menu (type `17` from the main menu).

- **Can I change the model parameters values and properties inside a project ?**
  Yes, type `11` from the main menu.

- **Can I change the initial hidden states values inside a project ?**
  Yes, type `12` from the main menu.

- **Is there a way to keep track of the analysis when OpenBDLM runs in batch mode ?**
  Yes, this is the purpose of the `LOG_*.txt` files which are saved in the "log_files" folder. Each time an analysis is performed (interactive or batch mode), a log file is created that records information about the analysis.

- **How can I delete projects ?**
  From the OpenBDLM main menu, type `D` and then select the indexes of the projects to delete.

- **How can I clean my OpenBDLM working directory ?**
  Type `Clean` and then press Enter key ↵. This function will take care of deleting all the files related to previous analysis. Make sure that you have a copy of the files you want to keep before deleting them.

- **Where does the HTML documentation come from ?**
  The HTML documentation is generated using matlab2html. matlab2html can be downloaded from https://www.artefact.tk/software/matlab/m2html/. If you want to update the documentation: (1) Download the matlab2html function and add it in your MATLAB path (2) Make a copy of the OpenBDLM master directory, (3) Move to the directory which allows to have the copy of the OpenBDLM master directory in the current directory (move one step back in the arborescence), (4) From the MATLAB command line, type `m2html('mfiles','OpenBDLM_V1.0', 'htmldir','doc', ...`

```
'recursive','on', 'graph', 'on', ...
'ignoredDir', {'ExternalPackages', 'doc', 'data', 'config_files', ...
```

`'figures', 'saved_projects', 'log_files', 'version_control', ...`
`'demo', 'results', 'logo'}, 'global', 'on')`. 5) (for Mac OS X and Linux users only) in each subfolder of the "doc" folder, type `dot graph.dot −Tpng > graph.png` from the terminal command line to generate the dependency graphs. Dot tools can be downloaded from https://graphviz.gitlab.io/download/.

- **How to cite OpenBDLM ?**
  OpenBDLM, an Open-Source Software for Structural Health Monitoring using Bayesian Dynamic Linear Models
  Gaudot, I., Nguyen, L.H., Khazaeli S.and Goulet, J.-A.
  In the proceedings from the 13th International Conference on Applications of Statistics and Probability in Civil Engineering (ICASP13), May 2019
  [PDF] [EndNote] [BibTex] [3]

10 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. 5, pp8, 2019
[PDF] [EndNote] [BibTex] [DOI link] [2]

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. 26, Issue 3, pp.e2136, 2019
[PDF] [EndNote] [BibTex] [DOI link] [4]

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] [5]

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] [6]

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] [1]

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] [7]

10.1 Linear gaussian state-space model

OpenBDLM builds on Bayesian dynamic linear models (BDLMs). Bayesian dynamic linear models [8] 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

$$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{w}_t, \quad \begin{cases} \mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \\ \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t), \end{cases} \tag{2}$$

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 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

$$\mathbf{y}_t = \mathbf{C}_t\mathbf{x}_t + \mathbf{v}_t, \quad \{ \ \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t), \tag{3}$$

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 10.7 and [7]). One particularity of BDLMs is their capacity to update the current estimated state with the current observations, thus allowing to perform online state estimation for non-stationary time series.

10.2 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

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

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 $t-1$ [9]. 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. 4 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 (see §8.3). The generalization of the Kalman Filter for non-stationary time series is found in the Switching Kalman filter (SKF) equations.

10.3 Switching Kalman filter

We may be interested in anomaly detection, that is, modelling and detecting the changes of regimes 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 $\mathtt{S}$ at each time t [10]. The SKF algorithm can be divided into two successive steps, (i) the "Filter" and, (ii) the "Collapse" step. Following the notation used in Eq. 4 the first step can be expressed in its short form as

$$(\boldsymbol{\mu}^{i(j)}_{t|t}, \boldsymbol{\Sigma}^{i(j)}_{t|t}, \mathcal{L}^{i(j)}_t) = \text{Filter}(\boldsymbol{\mu}^{i}_{t-1|t-1}, \boldsymbol{\Sigma}^{i}_{t-1|t-1}, \mathbf{y}_t, \mathbf{A}^j_t, \mathbf{Q}^{i(j)}_t, \mathbf{C}^j_t, \mathbf{R}^j_t), \tag{5}$$

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 $\mathcal{L}^{i(j)}_t$ 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 $\pi^j_{t|t}$ 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

$$\pi_{t|t}^{j} = \sum_{i=1}^{\mathtt{S}} \frac{\mathcal{L}_t^{i(j)} \pi_{t-1|t-1}^{i} Z_t^{i(j)}}{c}, \tag{6}$$

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

$$W_{t-1|t}^{i(j)} = \frac{\mathcal{L}_t^{i(j)} \pi_{t-1|t-1}^{i} Z_t^{i(j)}}{c\pi_{t|t}^{j}}. \tag{7}$$

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

$$(\boldsymbol{\mu}_{t|t}^{j}, \boldsymbol{\Sigma}_{t|t}^{j}) = \text{Collapse}(\boldsymbol{\mu}_{t|t}^{i(j)}, \boldsymbol{\Sigma}_{t|t}^{i(j)}, W_{t-1|t}^{i(j)}), \tag{8}$$

where state switching probabilities $W_{t|t-1}^{i(j)}$ are used as weighting factors for the gaussian mixture. From Eq. 8, 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

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}) = \text{Merge}(\boldsymbol{\mu}_{t|t}^{j}, \boldsymbol{\Sigma}_{t|t}^{j}, \pi_{t|t}^{j}), \tag{9}$$

where the state probabilities $\pi_{t|t}^{j}$ is used as weighting factors for the gaussian mixture [5].

10.4 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.

10.4.1 Maximum log A Posteriori (MAP)

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

$$\ln p(\boldsymbol{\theta}|\mathbf{y}_{1:\mathtt{Tr}}) \propto \ln p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}), \tag{10}$$

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 density of observations, that is, plausibility of the available observations $\mathbf{y}_{1:\mathtt{Tr}}$ given the parameter vector $\boldsymbol{\theta}$. Assuming that the observations errors are independent from each other, the joint log-likelihood function is defined as the sum of the marginal log-likelihoods, such as

$$\ln p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta}) = \sum_{t=1}^{\mathtt{Tr}} \ln p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \boldsymbol{\theta}) = \sum_{t=1}^{\mathtt{Tr}} \ln \left[\sum_{j=1}^{\mathtt{S}} \sum_{i=1}^{\mathtt{S}} \mathcal{L}_t^{i(j)} \pi_{t-1|t-1}^{i} Z_t^{i(j)} \right], \tag{11}$$

where $\mathcal{L}_t^{i(j)}$ and $\pi^i_{t-1|t-1}$ are computed at each time t from the Switching Kalman Filter; `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.

10.4.2 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 [11]. 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.

10.4.3 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 [11] 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 $\boldsymbol{\theta}^*$.

10.4.4 Gradient-based optimization

The gradient-based optimizations techniques are iterative approaches which can be used to find the model parameters that correspond to the maximum of a target PDF, hereafter noted $\mathcal{T}(\boldsymbol{\theta})$. The function $\mathcal{T}(\boldsymbol{\theta})$ is either the log a posteriori or the log likelihood PDF computed from the data. One iteration of gradient based algorithm is

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \eta\nabla\mathcal{T}(\boldsymbol{\theta}_{\text{old}}), \tag{12}$$

where η is the learning rate, and ∇ the first derivative.

Parameter-wise Newton-Raphson The parameter-wise Newton-Raphson [11] 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 underscripts $1:\mathtt{Tr}$ indicate that the target function is evaluated using a *training dataset* of length `Tr`. The Newton-Raphson algorithm adaptively sets the learning rate using the second derivative and a factor noted λ. One *iteration* of the Newton-Raphson algorithm is

$$\theta^i_{\text{new}} = \theta^i_{\text{old}} - \lambda\frac{\nabla\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})}{\nabla^2\mathcal{T}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})}, \tag{13}$$

where i is the index of the parameter being learned, ∇ 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

$$\left\{\begin{array}{ccc} \mathcal{T}_{1:\texttt{Tr}}(\boldsymbol{\theta}^i_{\text{old}}) & < & \mathcal{T}_{1:\texttt{Tr}}(\boldsymbol{\theta}^i_{\text{new}}) \\ \left|\mathcal{T}_{1:\texttt{Tr}}(\boldsymbol{\theta}^i_{\text{new}}) - \mathcal{T}_{1:\texttt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})\right| & \leqslant & \tau \cdot \left|\mathcal{T}_{1:\texttt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})\right| \end{array}\right. , \tag{14}$$

where τ is a termination tolerance. The Newton-Raphson algorithm stops when each model parameters has reached the convergence.

Stochastic gradient In the stochastic gradient technique, the target function and its derivatives are approximated at each iteration using a *mini-batch* of data of length $\texttt{Tb} \ll \texttt{Tr}$. Therefore, the target function is noted $\mathcal{T}_{1:\texttt{Tb}}(\boldsymbol{\theta})$. At each iteration, the beginning of the mini-batch is selected randomly. One *epoch* consists in one pass over the full training dataset (i.e. all the training data have been seen once). Therefore, one epoch is made of many iterations. Note that more than one model parameters is usually updated during one epoch. Several epochs are needed to reach convergence. The convergence is reached when the following condition between two successive epochs is satisfied

$$\mathcal{T}^{\text{epoch}}(\boldsymbol{\theta}) > \tau \cdot \mathcal{T}^{\text{epoch-1}}(\boldsymbol{\theta}) \tag{15}$$

where $0 \leqslant \tau \leqslant 1$ is a termination tolerance. Classical implementation of stochastic gradient algorithm includes momentum approach (e.g MMT optimizer) or adaptive learning rate (e.g. Adam optimizer) to increase the performance [12].

Approximation of the derivatives In many cases, the derivatives of $\mathcal{T}(\boldsymbol{\theta})$ cannot be computed analytically. Therefore, the derivatives are approximated numerically using the central differentiation scheme, such as

$$\begin{aligned} \nabla\mathcal{T}(\boldsymbol{\theta}^i) &= \frac{\partial\mathcal{T}(\boldsymbol{\theta})}{\partial\theta^i} \approx \frac{\mathcal{T}(\boldsymbol{\theta} + \mathbb{I}(i)\Delta\theta^i) - \mathcal{T}(\boldsymbol{\theta} - \mathbb{I}(i)\Delta\theta^i)}{2\Delta\theta^i} \\ \nabla^2\mathcal{T}(\boldsymbol{\theta}^i) &= \frac{\partial^2\mathcal{T}(\boldsymbol{\theta})}{\partial^2\theta^i} \approx \frac{\mathcal{T}(\boldsymbol{\theta} + \mathbb{I}(i)\Delta\theta^i) - 2\mathcal{T}(\boldsymbol{\theta}) + \mathcal{T}(\boldsymbol{\theta} - \mathbb{I}(i)\Delta\theta^i)}{(\Delta\theta^i)^2}, \end{aligned} \tag{16}$$

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

10.4.5 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 space into an unbounded one, where the parameters lie in the interval $[-\infty, +\infty]$. The transformation is done using a function $g(.)$ so that,

$$\theta^{\text{tr}} = g(\theta), \quad \theta^{\text{tr}} \in [-\infty, +\infty]. \tag{17}$$

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.

10.5 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.

10.5.1 Local level (baseline)

The local level block component describes the local mean of a stationary time series (no trend and no acceleration) [7]. The local level describes irreversible changes.
Number of hidden states: 1
Hidden states vector:

$$\mathbf{x}^{\text{LL}} = [x^{\text{LL}}]$$

Transition matrix:

$$\mathbf{A}^{\text{LL}} = [1]$$

Observation matrix:

$$\mathbf{C}^{\text{LL}} = [1]$$

Process noise covariance matrix:

$$\mathbf{Q}^{\text{LL}} = [(\sigma_w^{\text{LL}})^2]$$

Model parameters:

$$\boldsymbol{\theta}^{\text{LL}} = [\sigma_w^{\text{LL}}]$$

σ_w^{LL} is the process noise standard deviation which can be learned from the data.

10.5.2 Local trend (baseline)

The local trend block component describes the local mean of a trend-stationary time series (trend and no acceleration) [7]. The local trend describes irreversible changes.
Number of hidden states: 2
Hidden states vector:

$$\mathbf{x}^{\mathtt{LT}} = [x^{\mathtt{L}}, x^{\mathtt{LT}}]^{\intercal}$$

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.

10.5.3 Local acceleration (baseline)

The local acceleration block component describes the local mean of a acceleration-stationary time series [7]. It describes irreversible changes.
Number of hidden states: 3
Hidden states vector:

$$\mathbf{x}^{\mathtt{LA}} = [x^{\mathtt{L}}, x^{\mathtt{T}}, x^{\mathtt{LA}}]^{\intercal}$$

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:

$$\mathbf{Q}^{\mathtt{LA}} = (\sigma_w^{\mathtt{LA}})^2 \begin{bmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}$$

Model parameters:

$$\boldsymbol{\theta}^{\mathtt{LA}} = [\sigma_w^{\mathtt{LA}}]$$

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

10.5.4 Local level compatible trend (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 [5]. 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:

$$\mathbf{x}^{\texttt{LcT}} = [x^{\texttt{LL}}, x^{\texttt{LTc}} = 0]^{\intercal}$$

Transition matrix:

$$\mathbf{A}^{\texttt{LcT}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Observation matrix:

$$\mathbf{C}^{\texttt{LcT}} = [1, 0]$$

Process noise covariance matrix:

$$\mathbf{Q}^{\texttt{LcT}} = (\sigma_w^{\texttt{LcT}})^2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Model parameters:

$$\boldsymbol{\theta}^{\texttt{LcT}} = [\sigma_w^{\texttt{LcT}}]$$

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

10.5.5 Local level compatible acceleration (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 [5]. 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:

$$\mathbf{x}^{\texttt{LcA}} = [x^{\texttt{LL}}, x^{\texttt{LTc}} = 0, x^{\texttt{LAc}} = 0]^{\intercal}$$

Transition matrix:

$$\mathbf{A}^{\texttt{LcA}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Observation matrix:

$$\mathbf{C}^{\texttt{LcA}} = [1, 0, 0]$$

Process noise covariance matrix:

$$\mathbf{Q}^{\texttt{LcA}} = (\sigma_w^{\texttt{LcA}})^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Model parameters:

$$\boldsymbol{\theta}^{\texttt{LcA}} = [\sigma_w^{\texttt{LcA}}]$$

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

10.5.6 Local trend compatible acceleration (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 [5]. 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:

$$\mathbf{x}^{\texttt{TcA}} = [x^{\texttt{L}}, x^{\texttt{LT}}, x^{\texttt{LAc}} = 0]^{\intercal}$$

Transition matrix:

$$\mathbf{A}^{\texttt{TcA}} = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Observation matrix:

$$\mathbf{C}^{\texttt{TcA}} = [1, 0, 0]$$

Process noise covariance matrix:

$$\mathbf{Q}^{\texttt{TcA}} = (\sigma_w^{\texttt{TcA}})^2 \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} & 0 \\ \frac{\Delta t^2}{2} & \Delta t & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Model parameters:

$$\boldsymbol{\theta}^{\texttt{TcA}} = [\sigma_w^{\texttt{TcA}}]$$

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

10.5.7 Periodic (Fourier form)

The periodic (Fourier form) block component describes a periodic pattern in the time series using Fourier form [8, 7]. 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:

$$\mathbf{x}^{\mathtt{P}} = [x^{\mathtt{P}_1}, x^{\mathtt{P}_2}]^{\intercal}$$

Transition matrix:

$$\mathbf{A}^{\mathtt{P}} = \begin{bmatrix} \cos\omega & \sin\omega \\ -\sin\omega & \cos\omega \end{bmatrix}$$

Observation matrix:

$$\mathbf{C}^{\mathtt{P}} = [1, 0]$$

Process noise covariance matrix:

$$\mathbf{Q}^{\mathtt{P}} = (\sigma_w^{\mathtt{P}})^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Model parameters:

$$\boldsymbol{\theta}^{\mathtt{P}} = [\sigma_w^{\mathtt{P}}, p^{\mathtt{P}}]$$

$\sigma_w^{\mathtt{P}}$ 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.

10.5.8 Periodic (Kernel regression form)

The periodic (Kernel regression form) block component describes a periodic pattern in the time series using periodic kernel regression [2]. 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:

$$\mathbf{Q}^{\mathtt{KR}} = \begin{bmatrix} (\sigma_{w,0}^{\mathtt{KR}})^2 & \mathbf{0} \\ \mathbf{0} & (\sigma_{w,1}^{\mathtt{KR}})^2 \cdot \mathbf{I}_{\mathtt{L}^{\mathtt{KR}}} \end{bmatrix}$$

Observation matrix:

$$\mathbf{C}^{\texttt{KR}} = [1, 0, \ldots, 0]$$

Model parameters:

$$\boldsymbol{\theta}^{\texttt{KR}} = [p^{\texttt{KR}}, \ell^{\texttt{KR}}, \sigma_{w,0}^{\texttt{KR}}, \sigma_{w,1}^{\texttt{KR}}]$$

In the transition matrix, $\tilde{\boldsymbol{k}}^{\texttt{KR}}(t, \mathbf{t}^{\texttt{KR}})$ corresponds to the normalized kernel, $k(t, \mathbf{t}^{\texttt{KR}})/\sum_t k(t, \mathbf{t}^{\texttt{KR}})$. $\tilde{\boldsymbol{k}}^{\texttt{KR}}(t, \mathbf{t}^{\texttt{KR}})$ is parameterized by the kernel width $\ell^{\texttt{KR}}$, its period $p^{\texttt{KR}}$, and a vector of $\texttt{L}^{\texttt{KR}}$ timestamps $\mathbf{t}^{\texttt{KR}} = [t_1^{\texttt{KR}}, \cdots, t_{\texttt{L}^{\texttt{KR}}}^{\texttt{KR}}]$ where each timestamp $t_i^{\texttt{KR}}$ is associated with a hidden control point value $x_i^{\texttt{KR}}$. $\sigma_{w,1}^{\texttt{KR}}$ controls the process noise variance of the hidden control points between successive time steps and $\sigma_{w,0}^{\texttt{KR}}$ controls the time-independent process noise in the hidden predicted pattern. $p^{\texttt{KR}}$ and $\ell^{\texttt{KR}}$ give the period and correlation length of the kernel.

10.5.9 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) [7]. It describes reversible changes.
Number of hidden states: 1
Hidden states vector:

$$\mathbf{x}^{\texttt{AR}} = [x^{\texttt{AR}}]$$

Transition matrix:

$$\mathbf{A}^{\texttt{AR}} = [\phi^{\texttt{AR}}]$$

Observation matrix:

$$\mathbf{C}^{\texttt{AR}} = [1]$$

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.

10.5.10 Local Intervention

The local intervention block component describes the discrete shifts occurring in a time series. Shifts typically happens a sensor fails; when a sensor fails, the data start to be missing (i.e. `NaN`) and it often takes from several weeks to several months before the sensor is replaced. When the sensor is replaced, it is in most cases re-initialized at a different initial value than the previous sensor which lead to a discrete shift in the time series, as depicted in Figure 25. When using a Level intervention component, the user must provide discrete timestamps where it is required to estimate the magnitude of a discrete shift in the dataset.

A user can do so by specifying in the configuration file
`data.interventions=[t_{1}, t_{2}, ... , t_{n}]`, where `n` is the number of interventions.
The local intervention describes irreversible changes.
Number of hidden states: 1
Hidden states vector:

$$\mathbf{x}^{\mathtt{LI}} = [x^{\mathtt{LI}}]$$

Transition matrix:

$$\mathbf{A}^{\mathtt{LI}} = [1]$$

Observation matrix:

$$\mathbf{C}^{\mathtt{LI}} = [1]$$

Process noise covariance matrix:

$$\mathbf{Q}^{\mathtt{LI}} = [(\sigma_w^{\mathtt{LI}})^2]$$

Model parameters:

$$\boldsymbol{\theta}^{\mathtt{LI}} = [\mu_b^{\mathtt{LI}}, \sigma_b^{\mathtt{LI}}]$$

$\sigma_w^{\mathtt{LI}}$ is the standard deviation describing the uncertainty associated with the magnitude of the shift and $\mu_b^{\mathtt{LI}}$ is its expected magnitude; Both parameters can be kept to their default values or be learned from the data.
For the Local intervention component, the Equation 2 is modified to include an additional intervention term b_t

$$\mathbf{x}_t^{\mathtt{LI}} = \mathbf{A}_t^{\mathtt{LI}}\mathbf{x}_{t-1}^{\mathtt{LI}} + w_t^{\mathtt{LI}} + b_t^{\mathtt{LI}}, \quad \begin{cases} w_t^{\mathtt{LI}} \sim \mathcal{N}(0, \mathbf{Q}_t^{\mathtt{LI}}) \\ b_t^{\mathtt{LI}} \sim \mathcal{N}(\mu_b^{\mathtt{LI}}, \sigma_b^{\mathtt{LI}}) \end{cases} \tag{18}$$

The local Intervention component allows estimating the shifts $\mathbf{x}_{t|t}^{\mathtt{LI}}$ caused by the interventions for which the discrete timestamps are specified in `data.interventions`.

10.6 Handling non-uniform time vector and missing data

10.6.1 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, OpenBDLM employs an approximate method which is based on a reference time step Δt^{ref} [7]. 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. As an approximation, the model error standard deviations σ_w in $\mathbf{Q}_t$ are scaled 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 $\mathbf{A}^{\mathtt{AR}}$ contains the autoregressive coefficients $\phi^{\mathtt{AR}}$ 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

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

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. Note that this procedure is an approximation.

10.6.2 Missing data (NaN)

The presence of missing data (`NaN`) for specific timestamps prevents the completion of the Kalman update-step [7]. However, the prediction step using the current transition model can be done. Therefore, BDLM automatically fills gaps when data are missing using the transition model in the prediction step.

10.7 Dependencies between time series

The dependencies between time series are handled by adding regression coefficients $\phi^{i|j}$ in the observation matrix (See §8.2). 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 $\phi_k^{i|j}$ 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).

Acknowledgements

The authors would like to acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), Hydro Québec (HQ), Hydro Québec Research Institute (IREQ), and the Institute For Data Valorization (IVADO) for financial supports.
The authors would like to thank (in alphabetical order) Saeid Amiri, Barghob Deka, Zachary Hamida, Shervin Khazaeli for reviewing this document, and for their help in improving the software.
The authors would like to thank Benjamin Miquel and Patrice Côté from Hydro-Québec, and Mathieu Lacoste from Ministère des Transports du Québec for their help in the project.

References

[1] 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.

[2] Luong Ha Nguyen, Ianis Gaudot, Shervin Khazaeli, and James-A. Goulet. A kernel-based method for modeling non-harmonic periodic phenomena in bayesian dynamic linear models. *Frontiers in Built Environment*, 5(0):8, 2019.

[3] Ianis Gaudot, Luong Ha Nguyen, Shervin Khazaeli, and James-A. Goulet. OpenBDLM, an open-source software for structural health monitoring using bayesian dynamic linear models. In *13thProceedings from the 13th International Conference on Applications of Statistics and Probability in Civil Engineering*, May 2019.

[4] 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*, 26(3):e2309, 2019. e2309 stc.2309.

[5] 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.

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

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

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

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

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

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

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.