OpenBDLM reference manual

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

December 5, 2018

Contents

1 What is OpenBDLM ?

OpenBDLM is a MATLAB open-source software developed to use Bayesian Dynamic Linear Models for long-term time series analysis (i.e time step in the order of one hour or higher). OpenBDLM is capable to process simultaneously any time series data to interpret, monitor and predict their long-term behavior. OpenBDLM also includes an anomaly detection tool which allows to detect abnormal behavior in a fully probabilistic framework. 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, Bayes, gaussian conditionnals

2 Installing OpenBDLM

These instructions will get you a copy of the project up and running on your local machine for direct use, testing and development purposes.

2.1 Prerequisites

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

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

2.2 Installing

1. Remove from your MATLAB path all previously OpenBDLM versions
2. Extract the ZIP file from https://github.com/CivML-PolyMtl/OpenBDLM (or clone the git repository) somewhere you can easily reach it.
3. Add "OpenBDLM-master" folder and all the subfolders to your MATLAB path:
   - using the "Set Path" dialog box in MATLAB, or
   - by running `addpath` function from the MATLAB command window

3 Getting started

3.1 Call OpenBDLM

- enter in the folder "OpenBDLM-master"
- type `[data, model, estimation, misc] = OpenBDLM_main();` , and type ↵. in the MATLAB command line.

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

```
_____________________________________________
     Starting OpenBDLM
_____________________________________________
            Structural Health Monitoring 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 `[data, model, estimation, misc] = OpenBDLM_main();`

3.2 Demo

In the MATLAB command line, type `run_DEMO;` followed by ↵ to run a demo. Some messages on the MATLAB command window show that the programs runs properly (see Listing 2).

```
     Starting OpenBDLM_V???...
     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...
     See you soon !
```

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

4 Theory

4.1 Linear gaussian state-space model

Bayesian dynamic linear models are a class of linear gaussian state-space models which can be described from the transition and the observation equations. The transition equation describes the dynamics of the system, and is formulated as

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

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

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

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

4.2 Kalman filter

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

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

where $\mathcal{L}_t$ is the marginal likelihood describing the probability of observing observations $\mathbf{y}_t$ at time t given all the observations up to time $t-1$. The standard Kalman filter expressed in Eq. 3 can process stationary, trend stationary, and acceleration stationary time series, but it is not capable of handling non-stationary time-series, which is a major limitation when it comes to anomaly detection. The generalization of the Kalman Filter for non-stationary time-series is found in the Switching Kalman filter (SKF) equations.

4.3 Switching Kalman filter

We may be interested in anomaly detection, that is, modelling and detecting the changes of behavior due to changes in the dynamics of the baseline response

of the time-series. One way to model changing dynamics is to run in parallel a collection of S linear models, each having their own system dynamics $\mathbf{A}_t$ and $\mathbf{Q}_t$. In such approach, a discrete markovian switching variable $s_t = 1, .., j, .., \mathrm{S}$ 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 S^t. Therefore, the estimation quickly becomes intractable. One solution is to merge at each time t the states sharing the same dynamics using gaussian mixture. This technique, known as the Switching Kalman filter, allows to keep the dimension of the state vector equal to S at each time t. The SKF algorithm can be divided into two successive steps, (i) the "Filter" and, (ii) the "Collapse" step. Following the notation used in Eq. 3 the first step can be expressed in its short form as:

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

where the superscripts $i(j)$ indicates that the current state at time t is $s_t = j$ given the state at time $t-1$ is $s_{t-1} = i$, and $\mathcal{L}_t^{i(j)}$ the marginal likelihood that describes the probability of observing observations $\mathbf{y}_t$ at time t given all the observations up to time $t-1$, and given the state at time $t1$ was $s_{t-1} = i$ and that it switches to $s_t = j$ at time t. The state probability $\pi_{t|t}^j$ at each time t is computed from the previous state probabilities $\boldsymbol{\pi}_{t-1|t-1}$, the likelihood $\mathcal{L}_t^{i(j)}$, and the transition probability $Z_t^{i(j)}$, such as

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

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

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

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

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

where state switching probabilities $W_{t|t-1}^{i(j)}$ are used as weighting factors for the gaussian mixture. From Eq. 7, it is clear that the SKF algorithm provides a set a 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:

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

where the state probabilities $\pi^j_{t|t}$ is used as weighting factors for the gaussian mixture.

4.4 Model identification

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 learning, or model identification.

4.4.1 Maximum log A Posteriori (MAP)

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

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

where $p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta})$ is the likelihood PDF, $p(\boldsymbol{\theta})$ is the prior PDF. The likelihood PDF is the joint prior probability of observations, that is, plausibility of the available observations $\mathbf{y}_{1:\mathtt{Tr}}$ given the parameter vector $\boldsymbol{\theta}$. Assuming that the observations are independent from each other, the joint log-likelihood function is defined as the sum of the marginal log-likelihoods, such as

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

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

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

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

4.4.2 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. 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.

4.4.3 Laplace Approximation

The MAP and MLE are point estimation methods which do not take into account the uncertainty in the parameter estimates. The estimation of the uncertainties in the model parameters estimates can be addressed using the Laplace approximation.

$$\begin{aligned} p(\boldsymbol{\theta}|\mathbf{y}_{1:\mathtt{Tr}}) &\approx \mathcal{N}\left(\boldsymbol{\theta}; \boldsymbol{\theta}^*, -\mathcal{H}(\boldsymbol{\theta}^*)^{-1}\right) \text{ for the MAP} \\ p(\mathbf{y}_{1:\mathtt{Tr}}|\boldsymbol{\theta}) &\approx \mathcal{N}\left(\boldsymbol{\theta}; \boldsymbol{\theta}^*, -\mathcal{H}(\boldsymbol{\theta}^*)^{-1}\right) \text{ for the MLE,} \end{aligned} \tag{11}$$

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

4.4.4 Parameter-wise Newton-Raphson

The parameter-wise Newton-Raphson 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 $\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta})$. The function $\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta})$ is either the log a posteriori or the log likelihood PDF. One iteration of the Newton-Raphson algorithm is

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \lambda \frac{\nabla \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})}{\nabla^2 \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})}, \tag{12}$$

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

$$\begin{cases} \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}}) & < & \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{new}}) \\ \left|\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{new}}) - \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})\right| & \leqslant & \tau \cdot \left|\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i_{\text{old}})\right| \end{cases}, \tag{13}$$

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

$$\begin{aligned} \nabla \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i) &= \frac{\partial \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta})}{\partial \theta^i} \approx \frac{\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta} + \Pi(i)\Delta\theta^i) - \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta} - \Pi(i)\Delta\theta^i)}{2\Delta\theta^i} \\ \nabla^2 \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}^i) &= \frac{\partial^2 \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta})}{\partial^2 \theta^i} \approx \frac{\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta} + \Pi(i)\Delta\theta^i) - 2\mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta}) + \mathcal{L}_{1:\mathtt{Tr}}(\boldsymbol{\theta} - \Pi(i)\Delta\theta^i)}{(\Delta\theta^i)^2}, \end{aligned} \tag{14}$$

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

4.4.5 Model parameter space transformation

There are some model parameters which are defined in a bounded interval. 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. One possibility is to transform the bounded parameters space into an unbounded parameters space, where the parameters lie in the $[-\text{Inf}, +\text{Inf}]$ interval, and to perform the learning procedure in the transformed, unbounded, space. The transformation is done using a function $g(.)$ so that,

$$\theta^{i,\text{tr}} = g(\theta^i), \quad \theta^{i,\text{tr}} \in [-\text{Inf}, +\text{Inf}]. \tag{15}$$

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

- $\theta \in [-\text{Inf}, +\text{Inf}]$, $g(\theta) = 1$, so that $\theta^{\text{tr}} = \theta$ and $\theta = \theta^{\text{tr}}$
- $\theta \in [0, +\text{Inf}]$, $g(\theta) = \ln(\theta)$, 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 $[0, +\text{Inf}]$ interval and the logarithme transformation is used. Moroever, the autoregression coefficient model parameters are real numbers that lie in the $[0, 1]$ interval, and the sigmoid transformation is used.

5 Block component

5.1 Local Level

5.2 Local Trend

5.3 Local Acceleration

5.4 Local Level Trend compatible

5.5 Local Level Acceleration compatible

5.6 Local Trend Acceleration compatible

5.7 Periodic

5.8 First order Autoregressive

6 Dependencies

7 OpenBDLM inputs and outputs

7.1 Inputs

OpenBDLM_main accepts three types of input

- **no input**

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

- **the name of configuration file given as a character vector**

```
[data, model, estimation, misc]=OpenBDLM_main('CFG_DEMO.m');
```

- **a cell array of character vectors**

```
[data, model, estimation, misc]=OpenBDLM_main({'''CFG_DEMO.m''','''Q'''});
```

7.2 Outputs

7.2.1 data, model, estimation, misc

OpenBDLM_main returns four variables as MATLAB structures:

- **data**, which stores the time series used for the analysis.
- **model**, which stores all the information about the model used for the analysis

- **estimation**, which stores the hidden states estimation computed using the current data and model.
- **misc**, which stores internal variables

7.2.2 DATA_, CFG_, PROJ_ and LOG_ files

OpenBDLM reads and/or create four types of files:

- **data file** named with a prefix **DATA_** are MAT binary files that store the time series data (see Section 10.1.2). These files are located in the "data/mat" subfolder.
- **configuration file** named with the prefix **CFG_** are MATLAB scripts used to initialize and export a project (see Section 9). These files are located in the "config_files" subfolder.
- **project file** named with the prefix **PROJ_** are MAT binary files that stores a full project for further analysis. These files are located in the "saved_projects" subfolder.
- **log file** 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.

8 OpenBDLM running modes

8.1 Interactive mode

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

8.2 Batch mode

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

9 Configuration file

9.1 Purpose

The configuration files are MATLAB scripts that are used to initialize a project. The name of a configuration file can be given as a input to the function

OpenBDLM_main.m, as mentionned in Section 7.1. The configuration file must follow a specific organization. It must be divided into 6 sections: Project name, Data, Model structure, Model parameters, Initial states values and Options. The first three sections Project name, Data, Model structure are required. The last three sections Model parameters, Initial states values and Options are optional.

9.1.1 Project name

This section of the configuration file aims at giving the name of the project as a vector of characters stored in the field **ProjectName** of the MATLAB structure **misc**.

9.1.2 Data

This section of the configuration file aims at loading the data from a DATA_ file located in "/data/mat" subfolder. The file must follow the format described in Section 10.1.2. The timestamps values, the amplitude values and the labels values must be stored in the fields **timestamps**, **values**, and **labels** of the MATLAB structure named **data**.

9.1.3 Model structure

This part of the configuration file aims at defining 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** aims at defining the block components associated with each time-series.

  The field **block** stores a $1 \times \texttt{C}$ cell array, where $\texttt{C} = \{1, 2\}$ is the number of model classes. Each cell array is a $1 \times \texttt{M}$ cell array of array, where M is the number of time series. Each block component is associated with a reference number:

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

– 51: Kernel regression

- **model.component.const** aims at constraining model parameters between block components of different model classes.

  The field **const** stores a $1 \times \texttt{C}$ cell array, where $\texttt{C} = \{1, 2\}$ is the total number of model classes. It is defined only if $\texttt{C} = 2$. The first cell is empty, and the second cell is a $1 \times \texttt{M}$ cell array of array, where M 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** aims at defining the dependencies between the time-series.

  The field **ic** stores a $1 \times \texttt{M}$ cell array of $1 \times \texttt{M} - 1$ array, where M is the number of time series. Each time-series depend on the time-series corresponding to the indexes given in the M arrays. If the array is empty, the time-series is independent.

9.1.4 Model parameters

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

- the column 1 must be a character vector that gives the name of the model parameters (e.g. 'sigma_w').
- the column 2 must be a character vector that gives the reference name of the block associated with the parameter (e.g 'LL').
- the column 3 must be a character vector that gives the index corresponding to the model class associated with the parameter (e.g either '1'or '2').
- the column 4 must be a character vector that gives the index corresponding to the observation associated with the parameter (e.g either '3').
- the column 5 must be a 1×2 array that gives the bound of the parameter(e.g [NaN, NaN], [0, Inf], [0, 1]). The bounds are used to transform bounded model parameters from bounded to unbounded space during the optimization process.
- the column 6 must be character vector that gives the type of the prior used during the optimization process (e.g either 'N/A' or ' normal'). 'N/A' indicates that no prior is used.

- the column 7 must be a real number that gives the mean of the prior when a prior of type 'normal' is used, otherwise it must be set to NaN
- the column 8 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
- the column 9 must be a real number that gives the value of the model parameters
- the column 10 must be an integer that gives the reference number of the model parameters. The model parameters with the same reference number are constrained to each other.

9.1.5 Initial states values

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

- **model.initX** is a $1 \times \mathtt{C}$ cell array of array, where $\mathtt{C} = \{1, 2\}$ is the total number of model classes. Each array is a $\mathtt{L} \times 1$ array of real number that stores the initial mean values associated with each hidden states components, where `L` is the total number of hidden states components associated with the model.
- **model.initV** a $1 \times \mathtt{C}$ cell array of array, where $\mathtt{C} = \{1, 2\}$ is the total number of model classes. Each array is a $\mathtt{L} \times \mathtt{L}$ array of real number that stores the initial variance and covariances values associated with each hidden states components.
- **model.initS** a $1 \times \mathtt{C}$ cell array of array, where $\mathtt{C} = \{1, 2\}$ is the total number of model classes. Each array is a 1×1 array of real number that gives the initial probability for the model class.

9.1.6 Options

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

- options for the data pre-processing
  - **misc.options.NaNThreshold** real number that gives, in percent, the amount of missing data allowed at each time slice. Default: 100.
  - **misc.options.Tolerance** real number that gives the gives the duration (in number of days) after which two timestamps are not considered equal. Default: 10^{-6}.

- options for the optimization
  - **misc.options.trainingPeriod** 1×2 array of real number that defines the training period, given in number of days since the first timestamp. Default: [1 Inf].
  - **misc.options.isParallel** logical that triggers or not the parallel computation for approximating the gradient in the optimization procedure. Note that parallel computation require the MATLAB Parallel Computing Toolbox. Default: true.
  - **misc.options.maxIterations** integer that gives the maximum number of iterations for the optimization procedure. Default: 100.
  - **misc.options.maxTime** real number that gives, in minutes, the maximum amount of time to spend for the optimization procedure. Default: 60.
  - **misc.options.isMAP** logical that triggers or not the Maximum A Posteriori (MAP) estimation of the model parameters during the optimization procedure. MAP estimation includes prior information about the model parameters. Default: false.
  - **misc.options.isPredCap** logical. if isPredCap = true, the Prediction Capacity (i.e. the log-likelihood over a test dataset) is used to drive the optimization process, otherwise the log-likelihood over the full dataset is used. Default: false.
  - **misc.options.isLaplaceApprox** logical. if isLaplaceApprox = true the full Laplace approximation around the optimized model parameters values is computed. Default: false.
  - **misc.options.isMute** logical. is isMute = true, no message are output during the optimization procedure. Default: false.
- options for the estimation
  - **misc.options.MaxSizeEstimation** real number that gives the maximum size, in Mb, for which the hidden states estimations are saved in the PROJ_ file at the end of the analysis. Default: 100.
  - **misc.options.MethodStateEstimation** vector of character. It must be either 'kalman ' or 'UD '. it gives the method used for the estimation of the hidden states. Default: 'kalman '
- options for the graphical outputs
  - **misc.options.FigurePosition** 1×4 array of real number that gives the location and size of the drawable area, specified as a vector of the form [left bottom width height] in the current units of MATLAB. Default: [1001001300270]

- **misc.options.isSecondaryPlot** logical. if isSecondaryPlot= true, a closeup over two weeks is plotted at the right of each figure. Default: false.
- **misc.options.Subsample** integer that controls the number of points to plot in the figure. The number of points to plot is divided by a factor given by the values of Subsample. Default: 1.
- **misc.options.Linewidth** real number that controls the width of the line plotted in the figure. Default: 1.
- **misc.options.ndivx** integer that controls the number of labels for abscissa x-axis in each figure. Default: 4.
- **misc.options.ndivy** integer that controls the number of labels for ordinate y-axis in each figure. Default: 3.
- **misc.options.Xaxis_lag** real number that gives in number of days the amount of time the x-axis is shifted on each figure. Default: 0.
- **misc.options.isExportTEX** logical. if isExportTEX=true, the figure are exported in LaTeX format. Default: false.
- **misc.options.isExportPNG** logical. if isExportPNG=true, the figure are exported in PNG format. Default: false.
- **misc.options.isExportPDF** logical. if isExportPDF=true, the figure are exported in PDF format. Default: false.

10 Data loading

10.1 Input data format

OpenBDLM supports two types of input data format depending whether the time-series are synchronous, or not. The time-series are synchronous if they all share the same timestamps. Conversely, time-series are asynchronous if they do not all share the same timestamps.

10.1.1 Input data formatting for asynchronous time series

Comma Separated Values files One Comma Separated Values (.csv) file must be provided for each time series. The file is a two columns file that must be organized as shown in Listing 3 The first line of the file is the header. In the header, the first field must contain the label of the time-series given as a quoted delimited string, as 'name'. The second field is the date of the first timestamp given as a quoted delimited string, formatted as 'YYYY-DD-MM-HH-MM-SS'. For the remaining lines, the first field is the date given as a serial date number in number of days, given as a real number, and the second field is the magnitude of the physical quantity measured, given as a real number. The missing data must be indicated as NaN number. The .csv files must be stored in the "OpenBDLM-master/data/csv" subfolder.

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

Listing 3: CSV file example

10.1.2 Input data formatting for synchronous time series

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

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

MATLAB binary .mat files must be stored in the "OpenBDLM-master/data/mat" subfolder.

Comma Separated Values files See Paragraph 10.1.1 for .csv files formatting.

10.2 Output data format

Output data formatting is MATLAB binary .MAT file (see Paragraph 10.1.2) and CSV files (see Paragraph 10.1.1).

10.3 Data loading functions

10.3.1 List of functions

- [data, misc, dataFilename]=DataLoader(misc) Create a data file
- [dataOrig, misc]=readMultipleCSVFiles(misc) Read data from multiple CSV files
- [dat,label]=readSingleCSVFile(FileToRead, varargin) Read a single CSV file
- [misc, dataFilename] = saveDataBinary(data, misc, varargin) Save data in a binary Matlab .mat file
- [misc] = saveDataCSV(data, misc, varargin) Save time series data in separate CSV files

10.3.2 Dependency graph

11 Data editing and pre-processing

Data editing prepares the data for analysis. Data editing includes time series selection, data analysis time period selection, missing data removal, and data resampling. The time synchronization is performed automatically through the data editing process, when multiple time series are processed simultaneously.

```
— 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 4: OpenBDLM data editing menu

11.1 Selection of time-series

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

11.2 Selection data analysis time period

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

11.3 Removing missing data

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

11.4 Data resampling

Data resampling changes the sampling rate of the time-series according to a given timestep provided by the user. If the requested timestep is higher than the original data timestep, NaN values are added. Conversely, if the requested timestep is lower than the original data timestep, OpenBDLM averages the data amplitude values within non-overlapping fixed time windows, each having the duration of the requested timestep. The first time window starts at the first timestamp, and the new timestamps are assigned at the times corresponding to the middle of each time window.

11.5 Time synchronization options

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

11.6 Data editing functions

11.6.1 List of functions

- [data, misc]=chooseTimeSeries(data, misc) Request the user to select some time series
- [timesteps]=computeTimeSteps(timestamps) Compute timestep vector from timestamps vector
- displayData(data, misc) Display on screen the content of the data in memory
- [FileInfo] = displayDataBinary(misc, varargin) Display the list of DATA_*.mat files
- [data, misc, dataFilename]=editData(data, misc, varargin) Control script to edit dataset (selection, resampling, etc..)
- [data, misc]=mergeTimeStampVectors(dataOrig, misc, varargin) Create a single time vector from a set of time series
- [data_resample, misc]=resampleData(data, misc, varargin) Resample dataset according to a given timestep.
- [data, misc]=selectTimePeriod(data, misc) Select data between two dates

11.6.2 Dependency graph

12 Model configuration

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

12.1 Dependencies between time-series

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

12.2 Block components

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

- 11: Local level
- 12: Local level plus local trend
- 13: Local level plus local trend plus local acceleration
- 21: Local level compatible with local trend
- 22: Local level compatible with local acceleration
- 23: Local trend compatible with local acceleration
- 31: Periodic

- 41: First-order autoregressive
- 51: Kernel regression

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

12.3 Parameter constrains

The field **model.component.const** stores a $1 \times \texttt{C}$ cell array, where $\texttt{C} = \{1, 2\}$ is the total number of model classes. It is defined only if $\texttt{C} = 2$. The first cell is empty, and the second cell is a $1 \times \texttt{M}$ cell array of array, where `M` 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.

12.4 Number of model class

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

13 Examples

13.1 Example 1

13.2 Example 2

13.3 Example 3

13.4 Example 4

14 Troubleshooting