BoolFilter Package Vignette

Levi D. McClenny, Mahdi Imani, Ulisses Braga-Neto
Dept. of Electrical and Computer Engineering
Texas A&M University - College Station, TX
levimcclenny@tamu.edu, m.imani88@tamu.edu, ulisses@tamu.edu

January 6, 2017

Contents

1 Introduction

The `BoolFilter` package provides tools for state estimation and inference of partially-observed Boolean dynamical systems. This package contains functions for simulating data, obtaining the optimal estimator in the presence of various observation noise models [2], applying the optimal MMSE filter and smoother [6], particle filter implementation of optimal filter [3], as well as means of parameter estimation and network inference [5].

`BoolFilter` was created explicitly to handle the latest tools developed for signal model of partially-observed Boolean dynamical system (POBDS). This signal model, proposed in [2], was introduced to model dynamical systems containing Boolean variables observed indirectly through a noisy measurement process. This observation noise could be a result of instrumentation, incorrect thresholding, etc.

A characteristic signal model for the partially-observed Boolean dynamical system can be represented by the equations [2] [9]

$$\begin{aligned} \mathbf{X}_k \;&=\; \mathbf{f}\left(\mathbf{X}_{k-1}\right) \;\oplus\; \mathbf{n}_k \quad \text{(state model)} \\ \mathbf{Y}_k \;&=\; \mathbf{h}\left(\mathbf{X}_k, \mathbf{v}_k\right) \quad \text{(observation model)} \end{aligned} \tag{1}$$

for $k = 1, 2, \ldots$. Here, $\mathbf{n}_k \in \{0,1\}^d$ is Boolean transition noise, "$\oplus$" indicates component-wise modulo-2 addition, and $\mathbf{f} : \{0,1\}^d \longrightarrow \{0,1\}^d$ is the network function, whereas $\mathbf{h}$ is a general function mapping the current state and observation noise $\mathbf{v}_k$ into the measurement space. The noise processes $\{\mathbf{n}_k, \mathbf{v}_k; k = 1, 2, \ldots\}$ are assumed to be "white" in the sense that the noises at distinct time points are uncorrelated random variables. It is also assumed that the noise processes are uncorrelated with each other and with the initial state $\mathbf{X}_0$.

Several tools for POBDSs have been developed in recent years such as the optimal filter and smoother based on the minimum mean square error (MMSE) criterion, called the Boolean Kalman Filter (BKF) [2] and Boolean Kalman Smoother (BKS) [6], respectively. In addition, schemes for simultaneous state and parameter estimation, optimal filter in the case of correlated noise, network inference, fault detection, and control for POBDS were introduced in [9] [10] [5] [1] [8] [7].

`BoolFilter` package is developed to make the latest developed tools for the POBDS accessible. First, the package can be installed from CRAN with the command

```
> install.packages('BoolFilter')
```

Once the package and its dependencies are fully installed, the package can be loaded into the workspace with the command

```
> library('BoolFilter')
```

2 Constructing a Boolean Network

BoolFilter relies on the package *BoolNet* for describing Boolean Networks as input for its functions. This is done with a specific means of inputting a target gene, followed by its update rule [11]. In order to create a Boolean Network, a text file must be created in the users current working directory with the transition rules in a specific format. For instance, the p53-MDM2 gene regulatory network is presented in Figure 1. The network contains 4 genes (ATM, p53, Wip1, Mdm2) and 1 input dna-dsb, which denotes the presence or absence of DNA double strand breaks. This netwrok is shown below in figure 1:

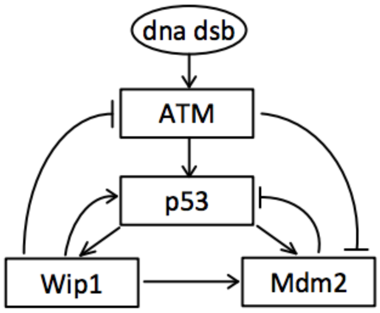

Figure 1: p53-mdm2 Network

One would create the network in a text file (named 'p53.txt', for instance, saved to the working directory) as follows:

```
targets, factors
ATM, (! Wip1 & 1)
p53, (ATM & ! Wip1 & ! Mdm2)
Wip1, p53
Mdm2, (!ATM & (p53 | Wip1)) | ( p53 & Wip1 )
```

DNA_dsb is an input to ATM. In the above Boolean network, DNA_dsb is *ON*. In order to toggle the input to *OFF*, the first line would need to be changed to

```
ATM, (! Wip1 & 0)
```

Afterward, the network can be created into a Boolean Network class by utilizing functions in the *BoolNet* package.

```
> p53net <- loadNetwork('p53.txt')
```

Again, this function is a part of *BoolNet*, and creates an object of the type that *BoolFilter* requires for its functionality. This allows for user specification of the genetic regulatory network.

For simplicity, both p53 networks come included in the *BoolFilter* package and can be loaded into the workspace using the command

```
> data(p53net_DNAdsb1) #DNA_dsb is ON
> data(p53net_DNAdsb0) #DNA_dsb is OFF
```

where "p53-dnadsb0" and "p53-dnadsb1" are referring to the p53-Mdm2 negative feedback loop regulatory network with external input 0 and 1 respectively.

In addition "cellcycle" and "melanoma" Boolean networks are available predefined in database of BoolNet and BoolFilter respectively.

BoolNet also contains functions that allow for the reconstruction of Boolean networks from time series data, generating random networks with various connectivity parameters, or loading networks from SMBL or BioTapestry. More information on the various means of generating Boolean Networks with the BoolNet package can be found in the BoolNet Vignette at https://cran.r-project.org/web/packages/BoolNet/vignettes/BoolNet_package_vignette.pdf.

3 Data Generation

Network simulation takes a manually defined network and steps through the update rules, generating a time series of the genetic expression data of user-specified length. Process noise is assumed in the data generation, where the magnitude is a user defined parameter. For the data generation in *BoolFilter*, we only considere i.i.d Bernoulli process noise. Two sets of state and measurement trajectories are created by BoolFilter.

Time series expression data can be generated using different observation models. This is useful in the sense that multiple observation noise models can potentially be found when analyzing genetic expression data. There exists a single parameter by which all the parameters for a given observation model can be input, called *obsModel*. The order in which the parameters of observation models are input to the *obsModel* parameter is outlined in the following sections.

Regardless of the observation model chosen to generate the data, the *simulateNetwork* fuction will output a list consisting of the simulated state and measurement trajectories (called *originalState* and *observation*).

3.1 Bernoulli Observation Noise

The first case will be Bernoulli observation noise. The intensity of Bernoulli observation noise is defined by parameter q (while p was the intensity of state transition process). The larger the noise parameter, the more noisy the measurements. Only one parameter value is required for Bernoulli observation noise, and *BoolFilter* will throw an error if more than one parameter is input to a Bernoulli observation model. The following command creates state and observation trajectory with the length of n.data = 100, and intensity of Bernoulli state and process noise p = 0.05 and q = 0.02 respectively.

```
> data <- simulateNetwork(p53net_DNAdsb1, n.data = 100, p = .02,
+                        obsModel = list(type = 'Bernoulli',
+                                        q = 0.05))

> #View both datasets overlayed
> plotTrajectory(data$X,
+                labels = p53net_DNAdsb1$genes,
+                dataset2 = data$Y,
+                compare = TRUE)
```

3.2 Gaussian Observation Noise

BoolFilter is capable of generating data with Gaussian noise as well. In the event the user wants to create a Gaussian observation noise process, the parameters to the *obsModel* parameter vector will be input in the following order:

```
mu0
sigma0
mu1
sigma1
```

This model assumes that the Boolean states $\mathbf{x} = 0$ and $\mathbf{x} = 1$ are observed through Gaussian distributions $\mathbf{x}_0 \sim$ *N(mu0, sigma0)* and $\mathbf{x}_1 \sim$ *N(mu1, sigma1)*, respectively, where (mu0, sigma0) and (mu1, sigma1) are pairs of the mean and standard deviation of the activated and inactivated conditions, respectively.

```
> data <- simulateNetwork(p53net_DNAdsb1, n.data = 100, p = 0.02,
+                         obsModel = list(type = 'Gaussian',
+                                         model = c(mu0 = 1,
+                                                   sigma0 = 2,
+                                                      mu1 = 5,
+                                                   sigma1 = 2)))
```

This will generate a time series expression dataset with continuous entries, in which the active and inactive genes are observed through $\mathbf{x}_0 \sim$ *N(1, 2)* and $\mathbf{x}_1 \sim$ *N(5, 2)*,

3.3 Poisson Observation Noise

BoolFilter comes with a built-in Poisson observation model for modeling RNA sequencing data. Again, similar to the Gaussian observation model, the input parameters of obsModel in Poisson model are as follows:

```
s
mu
delta
```

where *s* is the sequencing depth, *mu* is the baseline expression in inactivated state, and *delta* is the differential expression.

The *delta* is a vector of the same magnitude as the number of Boolean variables where $delta_j$ denotes the effect of going from inactivated to activated state for *j*-th variable. In addition, *mu* is a scaler usually between 0 and 1 characterizing the baseline level of expression, and sequencing depth *s* is an instrument-dependent parameter. See [6] and [1], for more information about the range of different parameters.

```
> obsModel <- list(type = 'Poisson', s = 10.75, mu = 0.01, delta = c(2, 2, 2, 2))
> data <- simulateNetwork(p53net_DNAdsb1, n.data = 100, p = 0.02, obsModel)
```

3.4 Negative-Binomial Observation Noise

An additional observation model that can be used to model RNA-seq data is the Negative Binomial model. This model allows for the mean and variance to differ, unlike the Poisson distribution outlined in section 3.3. Similar to the Poisson model, the *obsModel* parameter must be a list, however it adds another extra dispersion parameter vector referred to as *phi*. The parameters should be input in the following order to obsModel:

```
s
mu
delta
phi
```

Where *s* is the sequencing depth, *mu* is the baseline expression in inactivated state, and *delta* is the differential expression, and *phi* is the inverse dispersion.
Both *delta* and *phi* must be vectors of the same magnitude as the number of Boolean variables. This allows you to specify the observation model for each gene specifically, if so desired [9].

Changing the *type* variable to 'NB' will cause the *simulateNetwork* function to generate data from the negative binomial distribution as shown below:

```
> obsModel <- list(type = 'NB',
+                     s = 10.875,
+                    mu = 0.01,
+                 delta = c(3, 3, 3, 3),
+                   phi = c(2, 2, 2, 2))
> data <- simulateNetwork(p53net_DNAdsb1, n.data = 100, p = 0.02, obsModel)
```

Data can be viewed in the following fashion, if desired:

```
> #display data without observation noise
> plotTrajectory(data$X, labels = p53net_DNAdsb1$genes)
```

4 State Estimation of Partially-Observed Boolean Dynamical Systems

4.1 Boolean Kalman Filtering

The Boolean Kalman Filtering algorithm is the optimal minimum mean-squared error (MMSE) estimator of a Partially-Observed Boolean Dynamical System [2], [9]. This algorithm estimates the state of the system recursively as new measurement arrives.

Data from various sources can be fed into the BKF algorithm. Potential sources include RNA-seq, as well as user generated data outlined in section 3. The Boolean Kalman Filter implementation present in *BoolFilter* is capable of handling all the observation noise discussed in section 3 regarding simulation - Bernoulli, Gaussian, Poisson, and Negative Binomial. The input to the BKF is handled in much the same way as the *simulateNetwork* function, and assumes the user knows the distribution and parameters for the observation model.

Use is as follows:

```
> #simulate bernoulli noise on p53 network
> data <- simulateNetwork(p53net_DNAdsb1, n.data = 100, p = .02,
+                         obsModel = list(type = 'Bernoulli',
+                                            q = 0.02))
```

For this example, the p53 network was used for simplicity. Any network can be used, be it user specified, reconstructed, etc.

```
> #run BKF algorithm on simulated data
> Results <- BKF(data$Y, p53net_DNAdsb1, p=0.02,
+                          obsModel = list(type = 'Bernoulli',
+                                             q = 0.02))
```

The output, in this case labeled 'Results,' contains the estimated state, unnormalized posterior probability, as well as the conditional MSE at each time point. After performing the BKF and estimating states, the differences between the estimate and the original state trajectory can be viewed by plotting the trajectories of the genes using *plotTrajectory* and comparing them as:

```
> #plot the original and estimated trajectories on top of each other.
> plotTrajectory(data$X,
+                labels = p53net_DNAdsb1$genes,
+                dataset2 = Results$Xhat,
+                compare = TRUE)
```

4.2 Boolean Kalman Smoother

In cases when the measurements are available offline, the optimal MMSE estimator of state is called Boolean Kalman Smoother [6]. The method contains forward-backward process and can only be implemented on a batch data set.

The implementation is the same as the Boolean Kalman Filter:

```
> data <- simulateNetwork(p53net_DNAdsb1, n.genes = 100, p = 0.02,
+                         obsModel = list(type = 'Bernoulli',
+                                         q = 0.05))
> Results <- BKS(data$Y, p53net_DNAdsb1, p = 0.02,
+                         obsModel = list(type = 'Bernoulli',
+                                         q = 0.05))
```

It is again worth noting that the Boolean Kalman Filter algorithm is the optimal estimator for on-line process, whereas the Boolean Kalman Smoother is the optimal estimator for a given batch available data.

4.3 Particle Filter Implementation of BKF

The Boolean Kalman Filter is the optimal MMSE estimator of a POBDS [2]. However, the computation of optimal filter becomes impractical for a large POBDS. This opens the floor for an approximation of the optimal estimator. In order to approximate a BKF, we implement a SIR-BKF approach, outlined in [3].

This approximation approach allows us to handle the issue of high dimensional networks in which the BKF approach can prove inefficient in the category of computational intensity. One such network is the 10-gene mammalian cell cycle network outlined in [4]

Use of the SIR-BKF algorithm is outlined below, applied to the 10-gene Mammalian Cell Cycle network. First data is generated using the *simulateNetwork()* function described in section 2:

```
> data(cellcycle)
> data <- simulateNetwork(cellcycle, n.data = 100, p = 0.02,
+                         obsModel = list(type = 'Gaussian',
+                                     model = c(mu0 = 1,
+                                            sigma0 = 2,
+                                               mu1 = 5,
+                                            sigma1 = 2)))
```

Then we apply the Particle Filter algorithm to obtain an approximation of the state variables through the observation noise:

```
> Results <- SIR_BKF(data$Y, N = 1000, alpha = 0.9, cellcycle, p = 0.02,
+                           obsModel = list(type = 'Gaussian',
+                                     model = c(mu0 = 1,
+                                            sigma0 = 2,
+                                               mu1 = 5,
+                                            sigma1 = 2)))
```

The output 'Results' contains the estimated state and the conditional MSE at each time point. To compare the original and estimated state trajectories, one can use the following command:

```
> #compare the estimated and original state trajectories for selected Boolean variables
> VarPlot=c(1,2,5,7)
> plotTrajectory(data$X[VarPlot,],
+                labels = cellcycle$genes[VarPlot],
+                dataset2 = Results$Xhat[VarPlot,],
+                compare = TRUE)
```

In the above code, the index *c(1,4,5,7)* specifies the indices of Boolean variables to be plotted.

5 Multiple Model Adaptive Estimation

Suppose that the nonlinear signal model is incompletely specified. For example, the relationship between Boolean variables may be only partially known, or the statistics of the noise processes may need to be estimated. We assume that the missing information can be coded into a finite-dimensional vector parameter $\Theta = \{\theta_1, ..., \theta_M\}$. Assuming $P(\theta_i)$, for $i = 1, ..., M$, is the prior knowledge of ith model, the model selection is achieved by running a bank of BKFs running in parallel, one for each candidate model. Setting initial probability of models as their prior ($p_0^i = P(\theta_i)$, $i = 1, ..., M$), when measurement at time step k appears, the probability of each model is updated recursively as follows:

$$p_k^i = \frac{||\boldsymbol{\beta}_k^i||_1 \, p_{k-1}^i}{\sum_{j=1}^{M} ||\boldsymbol{\beta}_k^j||_1 \, p_{k-1}^j} \tag{2}$$

where $\boldsymbol{\beta}_k^i$ denotes the unnormalized PDV at time k, computed at the update step of the BKF for model θ_i.
The process continues for different measurement recursively till the maximum probability exceed a given threshold near 1 ($\max_{i=1,...,M} p_k^i > t_s$). This leads to the following selected model:

$$\theta^* = \underset{i=1,...,M}{\operatorname{argmax}} \, p_k^i \tag{3}$$

For more information about this method, refer to [5].

Two sources of unknown models can be modeled for implementation purposes:

1. Unknown network: User can define multiple networks as a possible network model to the system.

2. Unknown process noise: Different possible intensities of Bernoulli process noise can be entered (as a vector) to the algorithm.

The main algorithm is implemented as follows:

```
> #load potential networks
> data(p53net_DNAdsb0)
> data(p53net_DNAdsb1)
> net1 <- p53net_DNAdsb0
> net2 <- p53net_DNAdsb1
```

Once the networks have been identified, one can be simulated as follows:

```
> #define observation model
> observation = list(type = 'NB',
+                        s = 10.875,
+                       mu = 0.01,
+                    delta = c(2, 2, 2, 2),
+                      phi = c(3, 3, 3, 3))
> #simulate data using one of the networks and a given 'p'
> data <- simulateNetwork(net1, n.data = 100, p = 0.02, obsModel = observation)
```

The final step will be to call the MMAE function.

```
> #run MMAE to determine model selection and parameter estimation
> MMAE(data$Y, net = c("net1","net2"), p = c(0.02,0.1,0.15), threshold = 0.8,
+                                                    Prior = NA,
+                                                    obsModel = observation)
```

Note that if the network or process noise intensity is known, one can only input them as their are to the algorithm. In addition, when the prior is not input to this function, uniform prior is considered for all models.

References

[1] Arghavan Bahadorinejad and Ulisses Braga-Neto. Optimal fault detection and diagnosis in transcriptional circuits using next-generation sequencing. *IEEE/ACM Transactions on Computational Biology and Bioinformatics.*

[2] Ulisses Braga-Neto. Optimal state estimation for boolean dynamical systems. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, pages 1050–1054. IEEE, 2011.

[3] Ulisses Braga-Neto. Particle filtering approach to state estimation in boolean dynamical systems. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 81–84. IEEE, 2013.

[4] Adrien Fauré, Aurélien Naldi, Claudine Chaouiya, and Denis Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131, 2006.

[5] Mahdi Imani and Ulisses Braga-Neto. Optimal gene regulatory network inference using the boolean kalman filter and multiple model adaptive estimation. In *2015 49th Asilomar Conference on Signals, Systems and Computers*, pages 423–427. IEEE, 2015.

[6] Mahdi Imani and Ulisses Braga-Neto. Optimal state estimation for boolean dynamical systems using a boolean kalman smoother. In *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 972–976. IEEE, 2015.

[7] Mahdi Imani and Ulisses Braga-Neto. Point-based value iteration for partially-observed boolean dynamical systems with finite observation space. In *the 55th IEEE Conference on Decision and Control (CDC)*. IEEE, 2016.

[8] Mahdi Imani and Ulisses Braga-Neto. State-feedback control of partially-observed boolean dynamical systems using rna-seq time series data. In *2016 American Control Conference (ACC2016)*. IEEE, 2016.

[9] Mahdi Imani and Ulisses Braga-Neto. Maximum-likelihood adaptive filter for partially-observed boolean dynamical systems. *IEEE transaction on Signal Processing*, 65:359–371, 2017.

[10] Levi Daniel McClenny, Mahdi Imani, and Ulisses Braga-Neto. Boolean kalman filter with correlated observation noise. In *the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*. IEEE, 2017.

[11] Christoph Müssel, Martin Hopfensitz, and Hans A Kestler. Boolnet package vignette. 2015.