Package Tutorial for psbpHMM

Lauren Hoskovec, Ander Wilson, and Matthew Koslovsky

August 19, 2021

The R package psbpHMM provides several functions for fitting covariate-dependent infinite hidden Markov models (iHMM) via the probit stick-breaking process (PSBP). For more details on this method, see [Hoskovec et al.]. The following functions are available in the package psbpHMM:

function	brief description of usage
simData	simulate data to reproduce simulation in [Hoskovec et al.]
miHMM	fit homogeneous infinite hidden Markov model to one or more time series
mciHMM	fit covariate-dependent infinite hidden Markov model to one or more time series
getJointSimResults	summarize simulation output from jointly fit models
getIndepSimResults	summarize simulation output from independently fit models
fitDPMM	fit Dirichlet process mixture model to one or more time series
dlso_wrapper	wrapper function for dlso clustering algorithm in salso R package [Dahl. 2006]
modelAvemu	return model-averaged estimates of state-specific means from PSPB iHMM model

In this vignette, we show how to install the package, simulate data, fit three versions of our proposed approach, and summarize the MCMC output. The package contains everything needed to conduct a simulation study similar to that in [Hoskovec et al.].

Install the Package

First, we need to load the following package dependencies. Use install.packages() to install them, if needed.

```
library(Rcpp)
library(RcppArmadillo)
library(parallel)
library(gdata)
library(invgamma)
library(gtools)
library(mvtnorm)
library(matrixcalc)
library(tmvmixnorm)
library(rlist)
library(truncnorm)
library(mvnfast)
```

The package can then be installed from gitHub.

```
devtools::install_github("lvhoskovec/psbpHMM", build_vignettes = TRUE)
library(psbpHMM)
```

Simulate Data

The package contains functions to simulate data similar to the simulation study in [Hoskovec et al.], using the `simData` function. By default, `simData` generates 3-dimensional multivariate time series for 20 subjects, where each subject has 288 time points, 6 true clusters, and shared temporal patterns.

The `simData` function takes in the following parameters: `n` is the number of time series, `t.max` is the length of each time series, `p` is the dimension of the data, and `K` is the true number of hidden states shared among all time series. The parameter `trend` allows us to specify shared or distinct temporal trends among the time series, and the parameter `missingLevel` allows us to specify the proportion of missing data, which is split evenly between missing at random (MAR) and below the limit of detection (LOD). Based on these parameters, data are simulated as described in Section 3 in [Hoskovec et al.].

Here we simulate a data set with 5% missing observations.

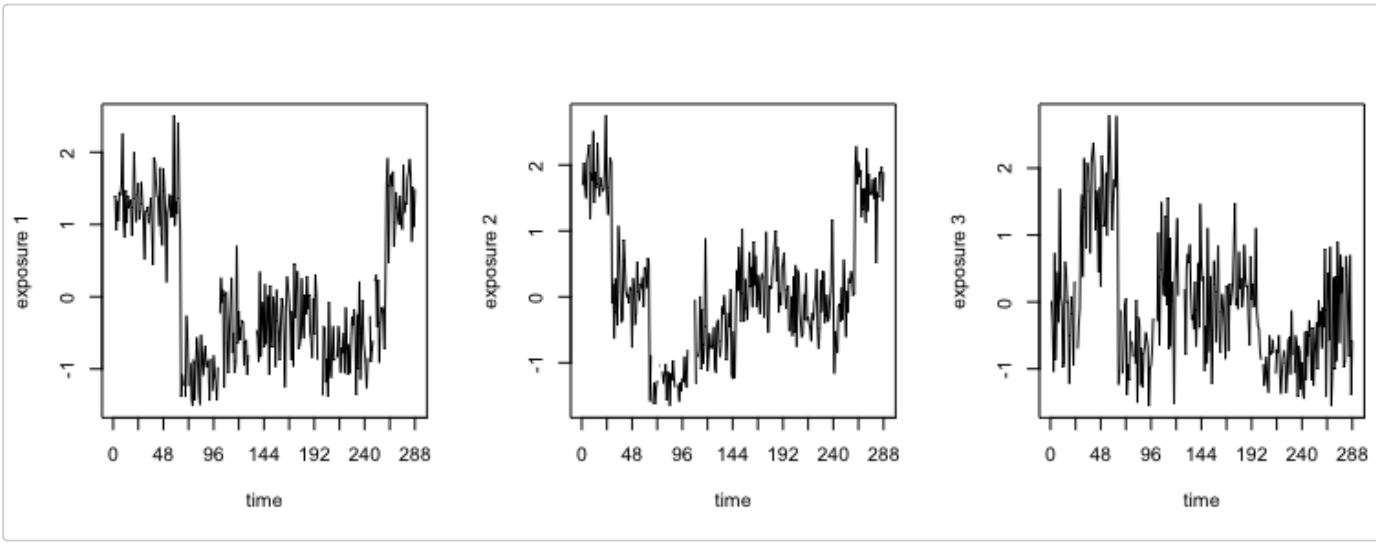
```
set.seed(42412)
n=20
p=3
t.max=288
K=6
dat = simData(n, t.max, K, p, trend = "shared", missingLevel = 0.05)
```

In these data, 2.5% of the data are MAR and 2.5% of the data are below the LOD. Hence, the fixed and known LOD is set at the 0.025 quantile of the pooled data set.

The object `dat` is a list with 6 elements: `y` is a list where each element contains an n by p matrix of simulated data for each subject with MAR observations denoted by NA and below LOD observations denoted by -Inf; `y.complete` is a list of the complete data for each subject before missing observations were removed; `z.true` is a list of the true hidden state trajectories for each subject; `K` is the true number of hidden states among all time series; `mu.true` is a K by p matrix of the true state-specific means; and `lod` is a list of the limits of detection for each subject.

We can plot the data to see what it looks like. We plot three exposures for one time series.

```
par(mfrow = c(1,3))
plot(1:288, dat$y[[1]][,1], col = dat$z.true[[1]], pch = 19, ylab = "exposure 1",
     xlab = "time", type = "l", xaxt = "n")
axis(side=1, at=seq(0,288+24,24), labels = TRUE)
plot(1:288, dat$y[[1]][,2], col = dat$z.true[[1]], pch = 19, ylab = "exposure 2",
     xlab = "time", type = "l", xaxt = "n")
axis(side=1, at=seq(0,288+24,24), labels = TRUE)
plot(1:288, dat$y[[1]][,3], col = dat$z.true[[1]], pch = 19, ylab = "exposure 3",
     xlab = "time", type = "l", xaxt = "n")
axis(side=1, at=seq(0,288+24,24), labels = TRUE)
```



Models

Statistical Approach

The package psbpHMM provides functionality to fit three types of iHMMs. Briefly, let $i = 1, \ldots, n$ denote subjects and $t = 1, \ldots, T$ denote time points, so that $\mathbf{y}_{it}$ is observed data for subject i at time t and z_{it} is the hidden state for subject i at time t.

All models have the same emission distribution for the data conditional on the hidden states. The emission distribution is

$$\begin{aligned} f(\mathbf{y}_{it}|z_{it} = k) &\equiv \mathrm{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \\ \boldsymbol{\mu}_k|\boldsymbol{\Sigma}_k &\sim \mathrm{N}\left(\boldsymbol{\mu}_0, \frac{1}{\lambda}\boldsymbol{\Sigma}_k\right) \\ \boldsymbol{\Sigma}_k &\sim \text{Inverse Wishart}(\nu, \mathbf{R}), \end{aligned}$$

where $\boldsymbol{\mu}_0$, λ, ν, and $\mathbf{R}$ are fixed hyperparameters.

The difference among the three models lies in the specification of the the transition distribution.

First (1), is a model without any covariates in the transition distribution; that is, the probability of transitioning from state j to state k is:

$$P(z_{it} = k|z_{i,t-1} = j) = \Phi(\alpha_{jk}) \prod_{l<k} \{1 - \Phi(\alpha_{jl})\}.$$

Second (2) is a model with covariate-dependent transitions. The covariate-dependent transition distribution is

$$P(z_{it} = k|z_{i,t-1} = j, \mathbf{x}_{it}) = \Phi(\alpha_{jk} + \mathbf{x}_{it}^T\boldsymbol{\beta}_k) \prod_{l<k} \{1 - \Phi(\alpha_{jl} + \mathbf{x}_{it}^T\boldsymbol{\beta}_l)\},$$

where $\mathbf{x}_{it}$ are covariates for subject i at time t.

Third (3) is a model with subject-specific effects of the covariates in the transition distribution to account for repeated sampling days for the same individual (3). Here, the transition distribution is

$$P(z_{i_st} = k|z_{i_s,t-1} = j, \mathbf{x}_{i_st}) = \Phi(\alpha_{jk} + \mathbf{x}_{i_st}^T\boldsymbol{\beta}_k + \mathbf{x}_{i_st}^T\boldsymbol{\gamma}_{ik}) \prod_{l<k} \{1 - \Phi(\alpha_{jl} + \mathbf{x}_{i_st}^T\boldsymbol{\beta}_l + \mathbf{x}_{i_st}^T\boldsymbol{\gamma}_{il})\},$$

where i_s denotes sampling day s for subject i.

The model is completed with the following hyperpriors

$$
\begin{aligned}
\alpha_{jk}|\sigma_\alpha^2 &\sim \mathrm{N}(\mu_\alpha, \sigma_\alpha^2) \text{ for } j \neq k \\
\sigma_\alpha^{-2} &\sim \mathrm{Gamma}(a_1, b_1) \\
\alpha_{jj}|m_\alpha, v_\alpha &\sim \mathrm{N}(m_\alpha, v_\alpha) \\
m_\alpha &\sim \mathrm{N}(m_0, v_0) \\
v_\alpha^{-1} &\sim \mathrm{Gamma}(a_2, b_2) \\
\boldsymbol{\beta}_k &\sim \mathrm{N}(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta) \\
\boldsymbol{\gamma}_{ik} &\sim \mathrm{N}(\boldsymbol{\mu}_\gamma, \kappa^2 \boldsymbol{\Sigma}_\gamma) \\
\kappa^{-2} &\sim \mathrm{Gamma}(\alpha_\kappa, \beta_\kappa).
\end{aligned}
$$

Function Details

Model (1) is fit using the function `miHMM` and models (2) and (3) are fit using the function `mciHMM`. The parameters in `miHMM` are a subset of those in `mciHMM`, so we describe how to use the function `mciHMM` here.

The function `mciHMM` takes many parameters. We first describe the most common ones.

To begin, `niter` is the total number of MCMC iterations and `nburn` is the number of burn-in iterations. `y` is an n-element list, where n is the number of subjects. `y` represents the emission data and may contain missing observations denoted by NA (for MAR data) or -Inf (for data below LOD). Each element of `y` is a $T \times p$ matrix, representing data for each subject, where T is the length of each time series and p is the dimension of the data. For proper implementation, `y` data need to be scaled to have mean 0 and variance 1. `X` is an n-element list of time-varying covariates for each subject. Each element of `X` is a $T \times q$ matrix where q is the dimension of the covariate data. The ordering of subjects in `X` and `y` must be the same. `rmlist` is a vector of integer labels for the repeated sampling days. Each integer corresponds a subject and an element of `y` and `X`; unique integers represent unique subjects and repeated integers represent repeated sampling days for the same subject. `missing` is logical and specifies whether `y` contains missing observations or not. If missing is true, `lod` is an n-element list of lower limits of detection for each of the p exposures and `len.imp` is the number of imputations to save. Imputations will be saved at equally spaced iterations post burn-in. `K.start` is the starting number of hidden states; the default is 12. `priors` is the list of prior hyperparameters. The prior options for this model include the following:

control parameter	model parameter	default value	description
mu0	$\boldsymbol{\mu}_0$	0	mean parameter for Normal prior on $\boldsymbol{\mu}_k$
lambda	λ	0	scale parameter for Normal prior on $\boldsymbol{\mu}_k$
nu	ν	1	degrees of freedom for Inverse Wishart prior on $\boldsymbol{\Sigma}_k$
R	\$	$\mathbf{I}_p$	matrix parameter for Inverse Wishart prior on $\boldsymbol{\Sigma}_k$
mu.alpha	μ_α	0	mean parameter for Normal priors on \alpha_{jk}, $j \neq k$
a1	a_1	1	shape parameter for gamma prior on σ_α^{-2}
b1	b_1	1	rate parameter for gamma prior on ϕ_2^{-2}
m0	m_0	0	mean parameter for Normal prior on m_α

control parameter	model parameter	default value	description
v0	v_0	1	variance parameter for Normal prior on m_α
a2	a_2	1	shape parameter for gamma prior on v_α^{-1}
b2	b_2	1	rate parameter for gamma prior on v_α^{-1}
mu.beta	$\boldsymbol{\mu}_\beta$	$\mathbf{0}$	mean parameter for Normal prior on $\boldsymbol{\beta}_k$
Sigma.beta	$\boldsymbol{\Sigma}_\beta$	$\mathbf{I}$	variance parameter for Normal prior on $\boldsymbol{\beta}_k$
mu.gamma	$\boldsymbol{\mu}_\gamma$	$\mathbf{0}$	mean parameter for Normal prior on $\boldsymbol{\gamma}_{ik}$
Sigma.gamma	$\boldsymbol{\Sigma}_\gamma$	$\mathbf{I}$	variance parameter for Normal prior on $\boldsymbol{\beta}_k$
a.kappa	α_κ	1	shape parameter for gamma prior on κ^{-2}
b.kappa	α_κ	1	rate parameter for gamma prior on κ^{-2}

Next we describe the parameters for tuning the Metropolis-Hastings independence sampler when there are missing observations. Details can be found in the Supporting Information of [Hoskovec et al.]. The independence MH sampler uses the following the parameters

control parameter	model parameter	default value	description
tau2	τ^2	0.25	variance parameter for Normal proposal on lower triangular elements
a.tune	a_δ	10	shape parameter for inverse gamma proposal on diagonal elements
b.tune	b_δ	1	rate parameter for inverse gamma proposal on diagonal elements

There is an option to use a resolvent kernel to update the parameters of the lower triangular matrix in the decomposition of $\boldsymbol{\Sigma}_k$. The parameter `resK` is a logical parameter; if set to TRUE then the resolvent kernel will be used and `eta.star` specifies the parameter of the geometric distribution for the resolvent kernel. The default value for `eta.star` is 5.

The function also takes optional parameters used to conduct a simulation study and evaluation of hidden state estimation and imputation. `z.true` is an n-element list of true hidden state trajectories for each subject. `mu.true` is a $K_{\text{true}} \times p$ matrix of true state-specific means, where K_{true} is the true number of hidden states. `ycomplete` is a list of the same dimensions as `y` that contains the true values for the missing observations in `y`.

The parameter `holdout` is rarely used. It is the same class and dimension of `y` but specifies which observations are in the holdout data set for evaluating imputation on an incomplete data set. In `holdout`, 0 denotes the data is observed, 1 denotes MAR, and 2 denotes below LOD. The only reason to specify `holdout` is when evaluating the multiple imputation approach on an incomplete data set; that is, when `ycomplete` still contains missing observations. `holdout` specifies the additional data that was removed for validation, but not the observations that are missing in `ycomplete`.

Examples

Fit Models

We demonstrate each model here. First we fit the homogenous PSPB iHMM to the simulated data. Since our data has missing observations, we specify the tuning parameters for the MH independence sampler. Here we run the model for 100 iterations with 50 burn-in and save 25 imputations post burn-in. On your own data, be sure to run the model for a long enough chain to reach convergence. To save time, we have provided the results of the model fits in the package, which can be loaded with the data() function. We show the code for fitting the model below, and then show how to load the results.

```
fit_no_cov = miHMM(niter = 100, nburn = 50, y = dat$y, ycomplete = dat$y.complete,
      z.true = dat$z.true, lod = dat$lod, mu.true = dat$mu.true, missing = TRUE,
      tau2 = 0.25, a.tune = 10, b.tune = 1, resK = TRUE, eta.star = 5,
      len.imp = 25)
```

The function `miHMM` returns an object of type “ihmm”, which is a list with the following components:

name	description
z.save	list of estimated hidden states for each time series at each iteration
K.save	list of estimated number of hidden states for each time series at each iteration
ymar	matrix with `len.imp` rows of imputed values for MAR data
ylod	matrix with `len.imp` rows of imputed values fro data below LOD
beta.save	list of posterior estimates of beta_k, state-specific regression coefficients in PSBP, NULL in miHMM
gamma.save	list of posterior estimates of gamma_ik, state-specific subject-specific regression coefficients in PSBP, NULL in miHMM
mu.save	list of posterior estimates of mu_k, state-specific means
hamming	posterior hamming distance between true and estimated states, if z.true is given
mu.mse	mean squared error for estimated state-specific means, if mu.true is given
mu.sse	sum of squared errors for estimated state-specific means, if mu.true is given
mar.mse	mean squared error of MAR imputations, if ycomplete is given
lod.mse	mean squared error of imputations below LOD, if ycomplete is given
mar.sse	sum of squared errors of MAR imputations, if ycomplete is given
lod.sse	sum of squared errors of imputations below LOD, if ycomplete is given

name	description
mismat	list, each element is a matrix indicating types of missing data for each time series, 0 = complete, 1 = MAR, 2 = below LOD
ycomplete	complete data
MH.arate	MH acceptance rate for lower triangular elements
MH.lamrate	MH acceptance rate for diagonal elements

Load the object using the data() function.

```
data("fit_no_cov")
```

First, let's look at the MH acceptance rates. We want to set the tuning parameters so that the MH acceptances rates are between 0.1 and 0.4.

```
fit_no_cov$MH.arate; fit_no_cov$MH.lamrate
```

```
## [1] 0.4643436
```

```
## [1] 0.2355484
```

Looks good. We'll return to summarizing the results of the model fit later.

Next, we fit the covariate-dependent PSBP. To incorporate covariates, we simulate cyclical temporal trends via a harmonic function of time. We will assume the time points for each time series all correspond to the same time of day, though they may take place on different days. Hence, the harmonic function of time will be identical for all time series. The current version of this model only permits time-varying covariates.

```
t.max = 288
n = 20
transT = seq(1:t.max)/t.max*2*pi
X1 = cbind(sin(transT), cos(transT), sin(2*transT), cos(2*transT))
X = list()
for(i in 1:n){
  X[[i]] = X1
}
q = ncol(X[[1]])
```

Now that we have covariates for each subject, we can fit the model.

```
fit_cyclical = mciHMM(niter = 100, nburn = 50, y = dat$y, ycomplete = dat$y.complete,
                      X = X, z.true = dat$z.true, lod = dat$lod, mu.true = dat$mu.true,
                      missing = TRUE, tau2 = 0.25, a.tune = 10, b.tune = 1, resK = TRUE,
        eta.star = 5,
                      len.imp = 25)
```

The function `mciHMM` also returns an object of type "ihmm". Load it with the data() function.

```
data("fit_cyclical")
```

Last, we fit the model with subject-specific effects of covariates. To do so, we specify the parameter `rmlist` to provide categorical indicators of the repeated time series for each subject. Each integer corresponds to a unique subject. The order of the categorical variables in the vector `rmlist` must be the same as the order of the time series in the lists `X` and `y` data. The following (arbitrarily constructed) `rmlist` represents 4 unique subjects each with 5 repeated sampling days.

```
rmlist = c(1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4)
```

```
fit_ss = mciHMM(niter = 100, nburn = 50, y = dat$y, rmlist = rmlist, ycomplete =
        dat$y.complete,
             X = X, z.true = dat$z.true, lod = dat$lod, mu.true = dat$mu.true,
             missing = TRUE, tau2 = 0.25, a.tune = 10, b.tune = 1, resK = TRUE, eta.star = 5,
             len.imp = 25)
```

Load the results with the data() function.

```
data("fit_ss")
```

Summarize Results

The package provides the function `getJointSimResults` to return the evaluation criteria measures used in our simulation study in [Hoskovec et al.]. This function returns the hamming distance (hamming), which is a measure of the distance between the estimated hidden state trajectories and the true hidden state trajectories, and the MSE for state-specific means (mu.mse) averaged over the iterations post burn-in. It also returns the MSE for MAR data (mar.mse) and data below the LOD (lod.mse) average over the specified number of imputations post burn-in. The best performing method will have the smallest hamming distance, MSE for state-specific means, and MSE for imputations. Here we show the summary for each of the models we fit.

Model (1): no covariates

```
getJointSimResults(fit_no_cov)
```

```
## $hamming
## [1] 0.2480208
##
## $mu.mse
## [1] 0.09285312
##
## $mar.mse
## [1] 0.7588954
##
## $lod.mse
## [1] 1.670608
```

Model (2): cyclical function of time as covariates

```
getJointSimResults(fit_cyclical)
```

```
## $hamming
## [1] 0.2422292
##
## $mu.mse
```

```
## [1] 0.05262417
##
## $mar.mse
## [1] 0.5641672
##
## $lod.mse
## [1] 0.1327652
```

Model (3): subject-specific cyclical function of time

```
getJointSimResults(fit_ss)
```

```
## $hamming
## [1] 0.1281528
##
## $mu.mse
## [1] 0.04286519
##
## $mar.mse
## [1] 0.5392352
##
## $lod.mse
## [1] 0.2377336
```

From this abbreviated simulation study demonstration, the subject-specific cyclical model best estimated the hidden state trajectories and state-specific means with lowest Hamming distance and MSE for μ. The subject-specific cyclical model also had the lowest MSE for MAR imputations, closely followed by the cyclical model without random effects. The cyclical model without random effects had the lowest MSE for below LOD imputations, followed by the subject-specific model. The model without any covariates performed worst in all measures.

We also provide functions to post-process the results. The following code applies to all three of our models. We will show on `fit_cyclical` only. First, we can calculate the most optimal hidden state trajectory using the draws-based latent structure optimization method described by Dahl (2006). We provide a wrapper function for the `dlso` function from the `salso` R package.

```
zbest = dlso_wrapper(fit_cyclical)
```

The function `dlso_wrapper` returns a list where each element represents a subject and contains the vector of estimated hidden state trajectories where each unqiue state is identified via a unique integer from 1 to the total number of clusters estimated. For example, subject one's estimated hidden state trajectory is:

```
zbest[[1]]
```

```
##   [1]  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3
##  [26]  3  3  3  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [51]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  1  1  1  1  1  1  1  1  1  1
##  [76]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
## [101]  1  1  1  1  5  8  5 13  1  1  1 11  8 11  5  1  5 13 11  5  5  5  1  1  1
## [126]  1 11  1  1 11 11  5  5  5  5 11  5  5  1  8  8  4 11 11  5  1  4  4  4  7
## [151]  4  4  7  7  4  7  7  4 12  7  4  4  4  4  4  4  4  7  4  4  4  4  4  5  7
## [176]  7  4  4  7 10  4  4  7  4  7  7  7  4  4 12  7  7  7  7  4  2  4  4  4  6
## [201]  6  6  1  6  1  4  6  6  6  6  1  6  1  6  4  6  1  6  6  9  9  6  6  6  6
## [226]  1  6  6  9  6  6  9  6  6  6  9  6  6  6  6  6  6  1  6  6  6  6  6  9  6
```

```
## [251]  4  9  6  6  6  6  6  6  6  6  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3
## [276]  3  3  3  3  3  3  3  3  3  3  3  3  3
```

Next, given the most optimal hidden state trajectories, we can calculate model-averaged estimates of the state-specific means with `modelAvemu`. This function must take in the exposure data as a matrix so we transform the list into the correct matrix here.

```
ymatrix=NULL
for(i in 1:n){
  ymatrix = rbind(ymatrix, dat$y[[i]])
}
mubest = modelAveMu(fit = fit_cyclical, zbest1=zbest, ymatrix = ymatrix)
```

We provide code to visualize the model-averaged exposure means. Install the packages ggplot2 and viridisLite if they are not already installed.

```
library(ggplot2)
library(viridisLite)
```

The following code plots the model-averaged state-specific means including the posterior mean and 95% credible interval for each exposure.

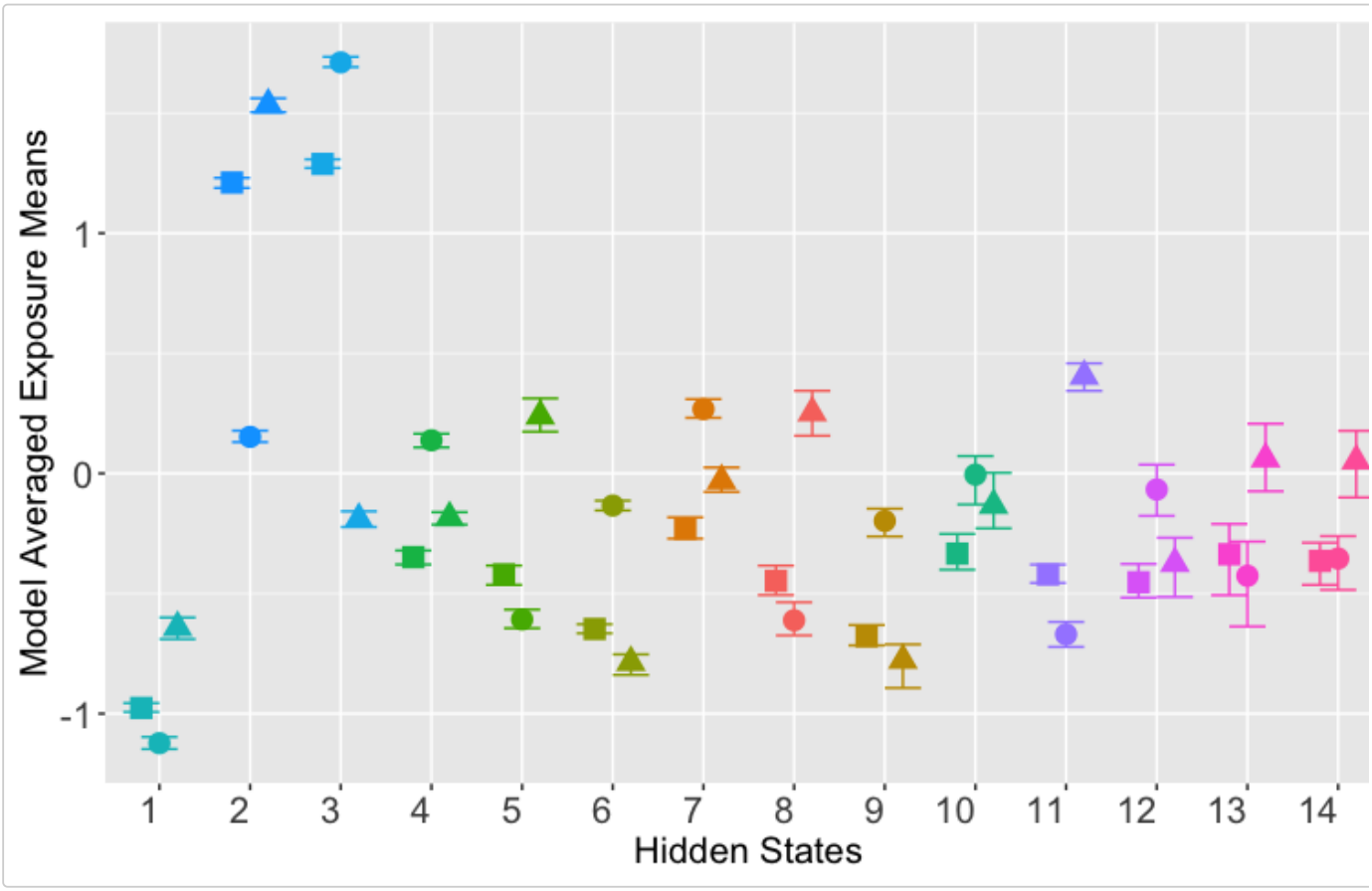
```
mmean <- matrix(unlist(mubest$mu_ma), ncol = p, byrow = TRUE)
Kmax = max(unlist(zbest))

for(k in 1:Kmax){
  whoBest <- which(unlist(zbest)==k) # the time points assigned to k in the best clustering
  ybest = as.numeric(ymatrix[whoBest, ])
}

mlwr <- matrix(unlist(mubest$mu_lwr), ncol = p, byrow = TRUE)
mupr <- matrix(unlist(mubest$mu_upr), ncol = p, byrow = TRUE)
col = viridis(n=Kmax)

mdat1 = data.frame(x = (1:Kmax)-0.2, y = mmean[,1], lwr = mlwr[,1], upr = mupr[,1], col = col)
mdat2 = data.frame(x = (1:Kmax), y = mmean[,2], lwr = mlwr[,2], upr = mupr[,2], col = col)
mdat3 = data.frame(x = (1:Kmax)+0.2, y = mmean[,3], lwr = mlwr[,3], upr = mupr[,3], col = col)

ggplot() + theme(legend.position = "none") +
  scale_x_discrete(name = "Hidden States", limits = factor(1:Kmax), breaks = 1:Kmax) +
  scale_y_continuous(name = "Model Averaged Exposure Means") +
  geom_point(data = mdat1, mapping = aes(x = x, y = y, colour = col), size = 4, shape = 15) +
  geom_errorbar(data = mdat1, mapping = aes(x = x, ymin = lwr, ymax = upr, colour = col), width
        = .4) +
  geom_point(data = mdat2, mapping = aes(x = x, y = y, colour = col), size = 4, shape = 16) +
  geom_errorbar(data = mdat2, mapping = aes(x = x, ymin = lwr, ymax = upr, colour = col), width
        = .4) +
  geom_point(data = mdat3, mapping = aes(x = x, y = y, colour = col), size = 4, shape = 17) +
  geom_errorbar(data = mdat3, mapping = aes(x = x, ymin = lwr, ymax = upr, colour = col), width
        = .4) +
  theme(text = element_text(size = 15),
        axis.text.x = element_text(size = 15, angle = 0, hjust = 1),
        axis.text.y = element_text(size = 15, angle = 0, hjust = 1),
        plot.title = element_text(size = 15))
```



Another feature of the hidden states that may be of interest is how many time points were assigned to each state. We can get this information with the table() function in base R.

```
table(unlist(zbest))
```

```
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14
## 1154 1004  899  524  380  778  494   78  135   31  205   28   23   27
```

Additional details on how to make inference on the hidden states can be found in Section 4 in [Hoskovec et al.].

Additional Models

In our simulation study, we compare our proposed joint approach to the models fit independently to each time series. Fit the cyclical model independently to each time series by

```
fit_indep_cyclical = mclapply(1:20, FUN = function(i){
  mciHMM(niter = 100, nburn = 50, y = dat$y[[i]], ycomplete = dat$y.complete[[i]],
                      X = list(X[[i]]), z.true = dat$z.true[[i]], lod = list(dat$lod[[i]]),
                      mu.true = dat$mu.true, missing = TRUE,
                      tau2 = 0.25, a.tune = 10, b.tune = 1, resK = TRUE, eta.star = 5,
                      len.imp = 25)
})
```

```
data("fit_indep_cyclical")
```

Similarly, fit the model with no covaraites independently to each time series by

```
fit_indep_nocov = mclapply(1:20, FUN = function(i){
  miHMM(niter = 100, nburn = 50, y = dat$y[[i]], ycomplete = dat$y.complete[[i]],
                      z.true = dat$z.true[[i]], lod = list(dat$lod[[i]]),
                      mu.true = dat$mu.true, missing = TRUE,
                      tau2 = 0.25, a.tune = 10, b.tune = 1, resK = TRUE, eta.star = 5,
                      len.imp = 25)
})


data("fit_indep_nocov")
```

To summarize the results from the independently fit models, use the function `getIndepSimResults`. This function averages the hamming distance and MSE for state-specific means and imputations over each of the `n` time series. Here, you must also include the emission data as a parameter.

```
getIndepSimRsults(fit1 = fit_indep_cyclical, y = dat$y)


## $hamming
## [1] 0.216559
##
## $mu.mse
## [1] 0.120665
##
## $mar.mse
## [1] 0.9345209
##
## $lod.mse
## [1] 2.21403


getIndepSimRsults(fit1 = fit_indep_nocov, y = dat$y)


## $hamming
## [1] 0.4521493
##
## $mu.mse
## [1] 0.2617047
##
## $mar.mse
## [1] 1.230829
##
## $lod.mse
## [1] 5.101946
```

Finally, we include in our package a function to fit a truncated Dirichlet process mixture model (DPMM) jointly to multiple time series. Here, `K.start` specifies the maximum number of hidden states allowed. We set `K.start = 50`.

```
fit_dpmm = fitDPMM(niter = 100, nburn = 50, y = dat$y, ycomplete = dat$y.complete,
             K.start = 50, z.true = dat$z.true, lod = dat$lod,
             mu.true = dat$mu.true, missing = TRUE,
             tau2 = 0.25, a.tune = 10, b.tune = 1, resK = TRUE, eta.star = 3, len.imp = 25)
```

```
data("fit_dpmm")
```

Get the results from the joint DPMM using `getJointSimResults`.

```
getJointSimResults(fit_dpmm)
```

```
## $hamming
## [1] 0.4777097
##
## $mu.mse
## [1] 0.3576407
##
## $mar.mse
## [1] 1.987636
##
## $lod.mse
## [1] 7.855185
```

References

Hoskovec, L., Koslovsky, M.D., Koehler, K., Peel, J.L., Volckens, J., Good, N., Wilson, A. Infinite Hidden Markov Models for Multiple Multivariate Time Series. In preparation.

Dahl, D. (2006). Model-Based Clustering for Expression Data via a Dirichlet Process Mixture Model. Bayesian Inference for Gene Expression and Proteomics, 201-218.

Contact Information

For questions or comments, please contact the package maintainer Lauren Hoskovec at lvheck@rams.colostate.edu.