Supplementary Material

1 Modified Example 1

As suggested by the referees, we explored a modification of Example 1, where observations with covariates $x < 3$ or $x > 6$ are placed in the second cluster. More specifically, for $i = 1, \ldots, n$, $n = 37$,

$$y_i|x_i = \begin{cases} -x_i/8 + 5 & \text{if } 3 \geq x_i \leq 6 \\ 2x_i - 12 & \text{if } x_i > 6 \text{ or } x_i < 3, \end{cases}$$
$$x_i = (0, 0.25, 0.5, \ldots, 8.75, 9).$$

The hyper–parameters are specified exactly as for Example 1: $\alpha = 1$, $a = 2$, $b = 1/4$,

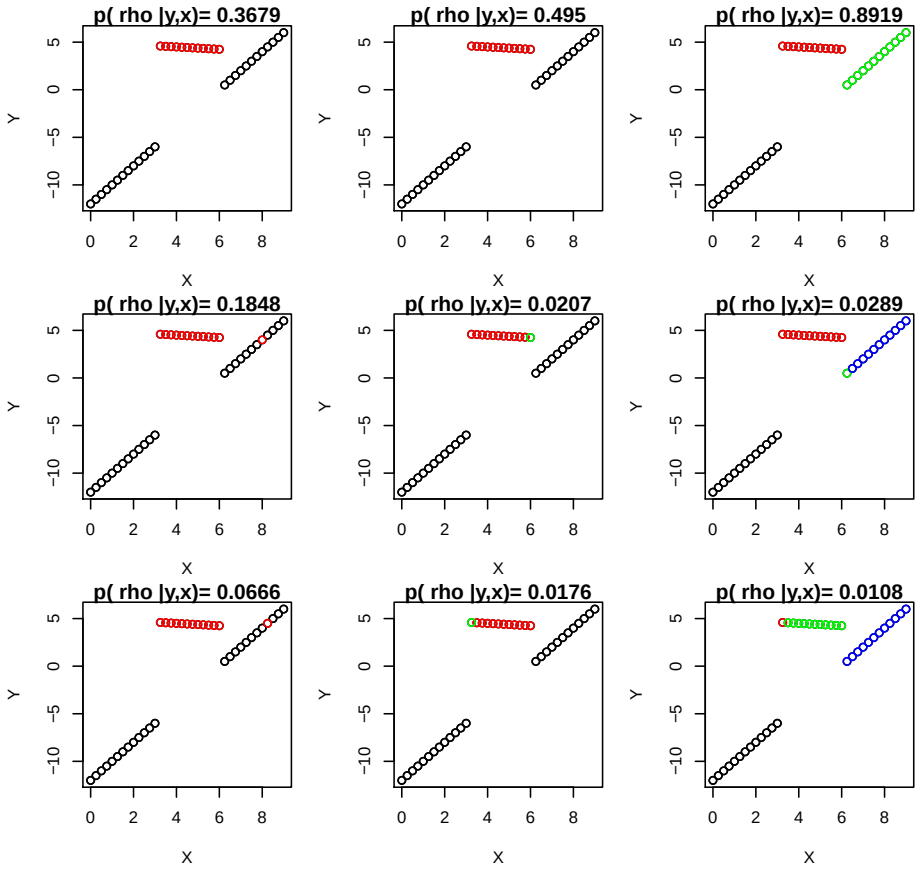
$$\beta_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ and } C = \begin{bmatrix} 1/144 & 0 \\ 0 & 1/4 \end{bmatrix}.$$

For the joint DPM model, the local model for X is $F_X = \mathrm{N}(\mu, \sigma_x^2)$ and the base measure for X is the conjugate normal inverse-gamma with additional hyperparameters $a_x = 1$, $b_x = 1$, $\mu_0 = 4.5$, $c = 1/4$.

By construction the rDPM model is not able to recover the true partition; it gives most probability (0.8919) to the partition which splits the second cluster into two clusters, one for $x < 3$ and one for $x > 6$. The DPM and jDPM models on the other hand give the highest probability to the true partition, although the probability is fairly spread out among many partitions. See Figure 1. Notice however, that prediction in Figure 2 is the best under the rDPM model, as it is only based on neighboring clusters.

2 Extensions

To illustrate our point, we have focused on curve fitting in the manuscript, but our discussion can be extended to more general regression problems. In the following subsection, we show how to extend the proposed method to univariate regression with non-continuous covariates. As is common to many methods, such as splines, extensions to multivariate regression are more complicated, but we outline the basic structure that would be required.



(a) DPM (b) joint DPM (c) restricted DPM

Figure 1: Plot of data colored by cluster membership for the three partitions with the highest estimated posterior probabilities. The plot title includes the posterior probability of the partition.

2.1 Extensions to non-continuous covariates

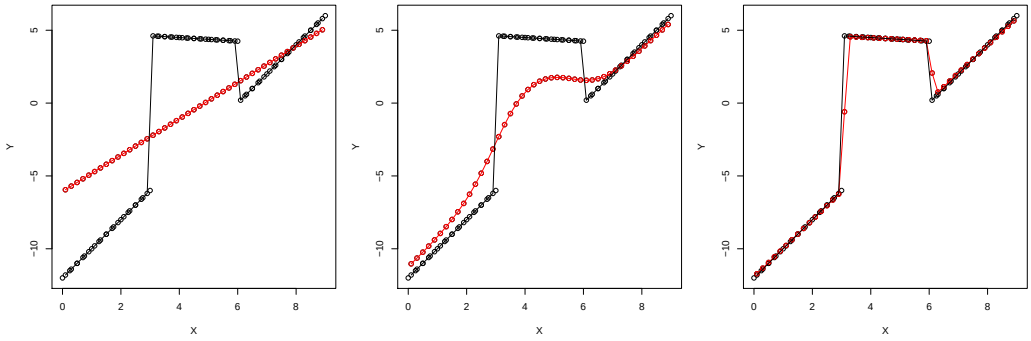
When subjects may have equal values of the covariate, a strict ordering of the covariates is no longer available, but, in most cases, a strict ordering of the unique values of the covariates is available. In particular, when the covariate is binary, ordinal, counts, or continuous with possible repeated values of the observed covariates (for example, due to rounding errors or experiment design), an ordering of the unique covariate values is sensible. We demonstrate how to handle these cases.

Let k_x denote the number of unique values among the observed covariates, let $n_{x,h}$ denote the number of subjects with the h^{th} unique ordered covariate, for $h = 1, \ldots, k_x$, and let $\underline{s}_{\pi_x(h)}$ denote a vector containing the labels for the $n_{x,h}$ subjects with the h^{th} unique ordered covariate.

In this setting, undesirable partitions are those which violate the constraint

$$\underline{s}_{\pi_x(1)} \leq \ldots \leq \underline{s}_{\pi_x(k_x)}, \tag{1}$$

where $\underline{s}_{\pi_x(h)} \leq \underline{s}_{\pi_x(h')}$ if $s_{\pi_x(h),i} \leq s_{\pi_x(h'),j}$, for $i = 1, \ldots, n_{x,h}$ and $j = 1, \ldots, n_{x,h'}$.



(a) DPM (b) joint DPM (c) restricted DPM

Figure 2: Plot of the curve estimate (in red) for new values of x with the true curve (in black) and the observed data (in black circles).

Again, we want to define a random partition model which both removes partitions violating (1) and maintains the DP's prior for k.

For the following proposition, we define $n^+_{x,h} = \sum_{l=1}^{h} n_{x,l}$ and $n^+_{x,0} = 0$, and similarly, $n^+_j = \sum_{l=1}^{j} n_l$ and $n^+_0 = 0$. Let

$$k_{x,h} = \sum_{j=1}^{k} I_{n^+_{x,h-1} \leq n^+_{j-1}} * I_{n^+_j \leq n^+_{x,h}}$$

for $h = 1, \ldots, k_x$, denote the number of clusters which both start and end among subjects with the h^{th} unique ordered covariate.

Proposition S.2.1 *The probability measure on the random partition defined by*

$$p^*(\rho_n|x) = \frac{\Gamma(\alpha)\Gamma(n+1)}{\Gamma(\alpha+n)} \frac{\alpha^k}{k!} \prod_{j=1}^{k} \frac{1}{n_j} * \prod_{h=1}^{k_x} k_{x,h}! * I_{\underline{s}_{\pi_x(1)} \leq \ldots \leq \underline{s}_{\pi_x(k_x)}}$$
$$* \prod_{h=1}^{k_x} \prod_{j=1}^{k} \left(\frac{(n^+_j - \max(n^+_{x,h-1}, n^+_{j-1}))!(n^+_{x,h} - n^+_j)!}{(n^+_{x,h} - \max(n^+_{x,h-1}, n^+_{j-1}))!} \right)^{I_{n^+_{x,h-1} < n^+_j < n^+_{x,h}}} \quad (2)$$

satisfies the order constraint (1) and has the same marginal for k, as that induced by the Dirichlet process.

Proof of Proposition S.2.1 For $j = 1, \ldots k$, if n_j specifies a split within subjects with the h^{th} unique ordered covariate, define $S_{j,x}$ as the set of indices of subjects among those with the h^{th} unique ordered covariate in group j, i.e

$$S_{j,x} = \{i : s_{\pi_x(h),i} = j, n^+_{x,h-1} < n^+_j < n^+_{x,h}\}.$$

The set $S_{j,x}$ may be empty if $n^+_j = n^+_{x,h}$ for some h. If multiple clusters start and end among subjects with the same covariate, the clusters are ordered according to

subject indices. Under the order constraint (1), it is straightforward to show that the partition is uniquely determined by $n_1, \ldots, n_k, k$ and the sets $S_{j,x}$. This implies that

$$p^*(n_1, S_{1,x}, \ldots, n_k, S_{k,x}, k|x) = \frac{\Gamma(\alpha)\Gamma(n+1)}{\Gamma(\alpha+n)} \frac{\alpha^k}{k!} \prod_{j=1}^{k} \frac{1}{n_j} * \prod_{h=1}^{k_x} k_{x,h}! \\ * \prod_{h=1}^{k_x} \prod_{j=1}^{k} \left(\frac{(n_j^+ - \max(n_{x,h-1}^+, n_{j-1}^+))!(n_{x,h}^+ - n_j^+)!}{(n_{x,h}^+ - \max(n_{x,h-1}^+, n_{j-1}^+))!} \right)^{I_{n_{x,h-1}^+ < n_j^+ < n_{x,h}^+}}. \quad (3)$$

Since (3) doesn't depend on $(S_{1,x}, \ldots S_{k,x})$, the marginal for $(n_1, \ldots, n_k, k)$ is obtained by multiplying (3) by the cardinality of $(S_{1,x}, \ldots S_{k,x})$. For $j = 1, ..., k$ such that $n_{x,h-1}^+ < n_j^+ < n_{x,h}^+$ for some h, there are

$$\begin{pmatrix} n_{x,h}^+ - \max(n_{x,h-1}^+, n_{j-1}^+) \\ n_j^+ - \max(n_{x,h-1}^+, n_{j-1}^+) \end{pmatrix}$$

ways to choose the $n_j^+ - \max(n_{x,h-1}^+, n_{j-1}^+)$ subjects with the h^{th} unique ordered covariate for group j. The cardinality is then given by the product of this number over j divided by $\prod_{h=1}^{k_x} k_{x,h}!$. This division is needed because simply taking the product does not account for ordering of clusters according to subject indices for $k_{x,h}$ clusters that both start and end among subjects with the h^{th} unique covariate. Thus,

$$p^*(n_1, \ldots, n_k, k|x) = \frac{\Gamma(\alpha)\Gamma(n+1)}{\Gamma(\alpha+n)} \frac{\alpha^k}{k!} \prod_{j=1}^{k} \frac{1}{n_j},$$

and the same arguments used in the proof of Proposition 3.1 can be applied to show the marginal prior for k is equivalent to that of the DP. $\diamond$

Since, the partition is no longer completely determined by $(n_1, \ldots, n_k, k)$, the MCMC scheme needs to be modified appropriately to handle this.

Corollary S.2.1 *The random partition model of the Dirichlet process is a special case of the covariate dependent random partition model defined by (2) when all covariates are equal.*

The proof of this proposition is straightforward. If all covariates are equal then $k_x = 1$, $k_{x,h} = k$, and $n_{x,1}^+ = n$. After plugging in these values and noticing that

$$\prod_{j}^{k-1} \frac{n_j!(n - n_j^+)!}{(n - n_{j-1}^+)!} = \frac{1}{n!} \prod_{j}^{k} n_j!,$$

(2) reduces to the random partition model of the DP.

The nice property given in Corollary S.2.1 is not satisfied by the joint DPM model. In fact, Müller and Quintana [2010] mention this as one of the undesirable features of their model.

2.2 Extensions to multivariate data

Extending the method to handle a multivariate response is quite simple. For example, if y is continuous, one only needs to replace the local normal model with a multivariate normal model. However, extending the approach to allow for multivariate covariates is tricky since there is no natural ordering in higher dimensions. Here we present the general approach for enforcing a given restriction and then discuss ideas on how to determine the restriction.

For $\rho_n \in \mathcal{P}_n$, let $I_R(\rho_n)$ indicate if ρ_n satisfies some given restriction R. Define $\{m_i\}$ to be the number of clusters of size i for $i = 1, ..., n$, let $\mathbf{k}$ and $\{\mathbf{m}_i\}$ denote the random variables with the non-bold variables indictaing the realized values, and define

$$\mathcal{P}_n(k, m_1, \ldots, m_n) = \{\rho_n \in \mathcal{P}_n | \mathbf{k} = k, \mathbf{m_1} = m_1, \ldots, \mathbf{m_n} = m_n\}$$

and

$$\mathcal{P}_n^*(k, m_1, \ldots, m_n) = \{\rho_n \in \mathcal{P}_n(k, m_1, \ldots, m_n) | I_R(\rho_n) = 1\}.$$

Proposition S.2.2 *The probability measure on the random partition defined by*

$$p^*(\rho_n|x) = \frac{\Gamma(\alpha)\Gamma(n+1)}{\Gamma(\alpha+n)} \alpha^k \prod_{i=1}^{n} \frac{1}{i^{m_i} m_i!} * \frac{1}{|\mathcal{P}_n^*(k, m_1, \ldots, m_n)|} * I_R(\rho_n) \qquad (4)$$

satisfies the constraint R and has the same marginal for k, as that induced by the Dirichlet process.

The proof that the marginal for k is the same as that of the DP is obtained by summing (4) over all $\rho_n \in \mathcal{P}_n(k, m_1, \ldots, m_n)$. The indicator function assigns zero mass to all $\rho_n \in \mathcal{P}_n(k, m_1, \ldots, m_n) \setminus \mathcal{P}_n^*(k, m_1, \ldots, m_n)$, so that the sum may be considered only over the set $\mathcal{P}_n^*(k, m_1, \ldots, m_n)$. The probability is uniform for partitions in this class. Thus, multiplying by the size of $\mathcal{P}_n^*(k, m_1, \ldots, m_n)$, gives the marginal for $k, m_1, \ldots, m_n$, which is equivalent to that of the DP.

For multivariate covariates, sensible constraints could be defined by requiring that the smallest rectangles in the covariate space (or spheres or ellipsoids) containing the covariates of subjects for each cluster do not intersect. In the univariate case, the ordering constraint (13) can also be viewed as non-intersecting 1-dim rectangles or spheres in the covariate space. The covariate random partition of (14) is obtained from (4) by noting that the size of $\mathcal{P}_n^*(k, m_1, \ldots, m_n)$ is just the number of unique ways to order the k cluster sizes, i.e. $k!/\prod_{i=1}^{n} m_i!$. In the multivariate case, this number will likely depend on the covariates, and a more general MCMC algorithm would need to be developed. In the manuscript, we focus on curve fitting and highlighting predictive properties of DP mixture models and possible improvements; however, this section provides direction of future research into a multivariate extension.

3 Acknowledgments for ADNI Data

Data collection and sharing for the application in Section 7 of this work was funded by the Alzheimer's Disease Neuroimaging Initiative (ADNI) (National Institutes of

Health Grant U01 AG024904). ADNI is funded by the National Institute on Aging, the National Institute of Biomedical Imaging and Bioengineering, and through generous contributions from the following: Abbott; Alzheimer's Association; Alzheimer's Drug Discovery Foundation; Amorfix Life Sciences Ltd.; AstraZeneca; Bayer HealthCare; BioClinica, Inc.; Biogen Idec Inc.; Bristol-Myers Squibb Company; Eisai Inc.; Elan Pharmaceuticals Inc.; Eli Lilly and Company; F. Hoffmann-La Roche Ltd and its affiliated company Genentech, Inc.; GE Healthcare; Innogenetics, N.V.; Janssen Alzheimer Immunotherapy Research & Development, LLC.; Johnson & Johnson Pharmaceutical Research & Development LLC.; Medpace, Inc.; Merck & Co., Inc.; Meso Scale Diagnostics, LLC.; Novartis Pharmaceuticals Corporation; Pfizer Inc.; Servier; Synarc Inc.; and Takeda Pharmaceutical Company. The Canadian Institutes of Health Research is providing funds to support ADNI clinical sites in Canada. Private sector contributions are facilitated by the Foundation for the National Institutes of Health (`www.fnih.org`). The grantee organization is the Northern California Institute for Research and Education, and the study is coordinated by the Alzheimer's Disease Cooperative Study at the University of California, San Diego. ADNI data are disseminated by the Laboratory for Neuro Imaging at theUniversity of California, Los Angeles. This research was also supported by NIH grants P30 AG010129, K01 AG030514, and the Dana Foundation.

4 Code for Simulated Examples

4.1 DPM

```
#DP mixture for regression
#Calls dp MCMC function to obtain posterior samples

library(mvtnorm)
library(msm)
library(MCMCpack)

#####################################################################
#SIMULATE DATA

##### EXAMPLE 1: PIECEWISE LINEAR

x=seq(0,9,.25)
n=length(x)
y=rep(0,n)
y[x<=6]=-1/8*x[x<=6]+5
y[x>6]=2*x[x>6]-12
plot(x,y)

#Hyperparameters
```

```
beta_0=matrix(c(0,0), 2,1)
C=diag(c(1/(36*4),1/4),2)
iC=diag(c(36*4,4),2)
a_y=2
b_y=1/4
alpha=1

##### EXAMPLE 2: PARABOLA

set.seed(10001)

n=50
x=runif(n, -5, 5)
y=rnorm(n,x^2, 1)
plot(x,y)

#Hyperparameters
beta_0=matrix(c(-12,0), 2,1)
C=diag(c(1/50,1/25),2)
iC=diag(c(50,25),2)
a_y=2
b_y=1*(a_y-1)
alpha=1

##### EXAMPLE 3: XSINX

set.seed(10001)

n=100
x=runif(n, -2*pi, 2*pi)
y=rnorm(n,x*sin(x), 1/4)
plot(x,y)

#Hyperparameters
beta_0=matrix(c(0,0), 2,1)
C=diag(c(1/(16*18^2),1/(16*9)),2)
iC=diag(c(16*18^2,16*9),2)
a_y=2
b_y=1/16
alpha=1

############################################################

## INPUT FOR MCMC
```

```
S=10000
burnin=1000

#Initialize chain

#Start everyone in 1 group
config_init=rep(1,n)

#compute updated parameters for y
X=matrix(c(rep(1,n),x),n,2)
C_hat=C+ t(X)%*%X
decomp=eigen(C_hat, symmetric=TRUE)
iC_hat=decomp$vectors%*%diag(1/decomp$values)%*%t(decomp$vectors)
beta_hat=iC_hat%*%(C%*%beta_0+t(X)%*%y)
a_y_hat=a_y+n/2
b_y_hat=b_y+(sum(y^2)+t(beta_0)%*%C%*%beta_0-t(beta_hat)%*%C_hat%*%beta_hat)/2

#initialize parameters for y
beta_init=beta_hat
sigma_y_init=b_y_hat/(a_y_hat-1)

#############################################################################

#Call dpm MCMC function
set.seed(101010)
output=dpm_mcmc(S, burnin, alpha,y, x, beta_0, C, a_y, b_y, config_init,
  beta_init, sigma_y_init)

# Define variables
config=output$config
beta=output$beta
sigma_y=output$sigma_y

#############################################################################

### ANALYSIS OF PARTITIONS

# Need to reorder labels
config_reorder=matrix(0,S,n)
config_count=c(1)
config_index=c(1)
for(s in 1:S){
 #reorder the configuration
 uniq_s=unique(config[s,])
 for( h in 1:k[s]){
```

```
  config_reorder[s,config[s,]==uniq_s[h]]=h
 }
 if(s>1){
  #check if we have seen this configuration before
  before=colSums(matrix(t(config_reorder[config_index,]), nrow=n)==
    matrix(config_reorder[s,],n,length(config_index) ))==n
  if(sum(before)>0){
   config_count[before]=config_count[before]+1
  }
  else{
   config_count=c(config_count,1)
   config_index=c(config_index,s)
  }
 }
}

# Compute estimated probability of configurations
config_p=config_count/S
# Sort estimated probabilities from highest to lowest
config_sp=sort(config_p, decreasing=TRUE, index.return=TRUE)
#Number of configurations with positive estimated probability
length(config_p)

# Ordered Configurations
config_top=config[config_index[config_sp$ix],]

# Compute estimated regression line for each configurations
beta_est=list()
sigma_est=list()
X=matrix(c(rep(1,n),x),n,2)
bCb= t(beta_0)%*%C%*%beta_0
for (i in 1:length(config_p)){
 k_i=max(config_top[i,])
 betai=matrix(0,2,k_i)
 sigmai=matrix(0,1,k_i)
 for(j in 1:k_i){
  C_hat=C+ t(matrix(X[config_top[i,]==j,],ncol=2))%*%
    matrix(X[config_top[i,]==j,], ncol=2)
  decomp=eigen(C_hat, symmetric=TRUE)
  iC_hat=decomp$vectors%*%diag(1/decomp$values)%*%t(decomp$vectors)
  beta_hat=iC_hat%*%(C%*%beta_0+t(matrix(X[config_top[i,]==j,],ncol=2))
    %*%y[config_top[i,]==j])
  a_hat=a_y+sum(config_top[i,]==j)/2
  b_hat=b_y+(sum(y[config_top[i,]==j]^2)+bCb-t(beta_hat)%*%C_hat%*%beta_hat)/2
  betai[,j]=beta_hat
```

```
  sigmai[j]=b_hat/(a_hat-1)
 }
 beta_est[[i]]=betai
 sigma_est[[i]]=sigmai
}

# Plot top 3 models
par(mfrow=c(3,1), mar=c(2,2,1,1)+.1)
for(i in 1:3){
 plot(x,y, main=paste( "p( rho |y,x)=",format(config_sp$x[i],digits=4,
   scientific=F)))
 k_i=max(config_top[i,])
 for(j in 1:k_i){
  abline(beta_est[[i]][1,j], beta_est[[i]][2,j], col=j)
  points(x[config_top[i,]==j], y[config_top[i,]==j],col=j)
 }
}

#########################################################################

######## PREDICTION

# Covariates for new subjects

# Ex 1
x_new=c(0.2,3.3,5.9,6.2,6.3,7.9,8.1,8.7)

# Ex 2
x_new=seq(-4.5,4.5,1)

# Ex 3
x_new=seq(-2*pi,2*pi,pi/8)

#########COMPUTE PREDICTION

# Number of new subjects
m=length(x_new)

# Create X_new matrix
X_new=matrix(c(rep(1,m),x_new),ncol=2)

# Initialize
pred_mat=matrix(0,S,m)

for(s in 1:S){
```

```
 #need counts
 n_s=matrix(0,1,k[s])
 for(j in 1:k[s]){
  n_s[j]=sum(config[s,]==j)
 }
 #calculate prediction
 pred_s=1/(alpha+n)*(X_new%*%cbind(beta_0, beta[[s]]))%*%
   matrix(c(alpha,n_s),(k[s])+1,1)
 pred_mat[s,]=rowSums(pred_s)
}

# Final Prediction
y_pred=colSums(pred_mat)/S

# True Prediction

# Ex 1
y_true=rep(0,m)
y_true[x_new<=6]=-1/8*x_new[x_new<=6]+5
y_true[x_new>6]=2*x_new[x_new>6]-12

# Ex 2
y_true=x_new^2

# Ex 3
y_true=x_new*sin(x_new)

## PLOT PREDICTION

par(mfrow=c(1,1))
plot(c(x,x_new, x_new),c(y,y_true,y_pred))
points(x_new, y_pred, col=2)
lines(x_new,y_pred,col=2)
lines(x_new,y_true)

## Compute L2 Error
l2_err=sum(((y_true-y_pred)^2)/m)^.5
```

4.2 joint DPM

```
#Joint DP mixture for regression
#Calls jdpm MCMC function to obtain posterior samples

library(mvtnorm)
library(msm)
```

```
library(MCMCpack)

#########################################################################
#SIMULATE DATA

##### EXAMPLE 1: PIECEWISE LINEAR

x=seq(0,9,.25)
n=length(x)
y=rep(0,n)
y[x<=6]=-1/8*x[x<=6]+5
y[x>6]=2*x[x>6]-12
plot(x,y)

#Hyperparameters

#Y-parameters
beta_0=matrix(c(0,0), 2,1)
C=diag(c(1/(36*4),1/4),2)
iC=diag(c(36*4,4),2)
a_y=2
b_y=1/4
alpha=1

#X-parameters
mu_0=4.5
c_x=1/4
a_x=1
b_x=1

##### EXAMPLE 2: PARABOLA

set.seed(10001)

n=50
x=runif(n, -5, 5)
y=rnorm(n,x^2, 1)
plot(x,y)

#Hyperparameters

#Y-parameters
beta_0=matrix(c(-12,0), 2,1)
C=diag(c(1/50,1/25),2)
iC=diag(c(50,25),2)
```

```
a_y=2
b_y=1*(a_y-1)
alpha=1

#X-parameters
mu_0=0
c_x=1/2
a_x=1
b_x=1

##### EXAMPLE 3: XSINX

set.seed(10001)

n=100
x=runif(n, -2*pi, 2*pi)
y=rnorm(n,x*sin(x), 1/4)
plot(x,y)

#Hyperparameters

#Y-parameters
beta_0=matrix(c(0,0), 2,1)
C=diag(c(1/(16*18^2),1/(16*9)),2)
iC=diag(c(16*18^2,16*9),2)
a_y=2
b_y=1/16
alpha=1

#X-parameters
mu_0=0
c_x=1/9
a_x=1
b_x=1

############################################################

## INPUT FOR MCMC

S=10000
burnin=1000

#Initialize chain

#Start everyone in 1 group
```

```
config_init=rep(1,n)

#compute updated parameters for y
X=matrix(c(rep(1,n),x),n,2)
C_hat=C+ t(X)%*%X
decomp=eigen(C_hat, symmetric=TRUE)
iC_hat=decomp$vectors%*%diag(1/decomp$values)%*%t(decomp$vectors)
beta_hat=iC_hat%*%(C%*%beta_0+t(X)%*%y)
a_y_hat=a_y+n/2
b_y_hat=b_y+(sum(y^2)+t(beta_0)%*%C%*%beta_0-t(beta_hat)%*%C_hat%*%beta_hat)/2

#initialize parameters for y
beta_init=beta_hat
sigma_y_init=b_y_hat/(a_y_hat-1)

#compute updated hyperparameters for x
c_x_hat=c_x+n
mu_x_hat=1/c_x_hat*(c_x*mu_0+sum(x))
a_x_hat=a_x+n/2
b_x_hat=b_x+1/2*(sum(x^2)+c_x*mu_0^2-c_x_hat*mu_x_hat^2)

#initialize parameters for x
mu_x_init=mu_x_hat
sigma_x_init=b_x_hat/(a_x_hat-1)

#########################################################################

#Call jdpm MCMC function
set.seed(101010)
output=jdpm_mcmc(S, burnin, alpha,y, x, beta_0, C, a_y, b_y, mu_0, c_x, a_x, b_x,
  config_init, beta_init, sigma_y_init, mu_x_init, sigma_x_init)

# Define variables
config=output$config
beta=output$beta
sigma_y=output$sigma_y
mu_x=output$mu_x
sigma_x=output$sigma_x

#########################################################################

### ANALYSIS OF PARTITIONS

# Need to reorder labels
config_reorder=matrix(0,S,n)
```

```
config_count=c(1)
config_index=c(1)
for(s in 1:S){
 #reorder the configuration
 uniq_s=unique(config[s,])
 for( h in 1:k[s]){
  config_reorder[s,config[s,]==uniq_s[h]]=h
 }
 if(s>1){
  #check if we have seen this configuration before
  before=colSums(matrix(t(config_reorder[config_index,]), nrow=n)==
    matrix(config_reorder[s,],n,length(config_index) ))==n
  if(sum(before)>0){
   config_count[before]=config_count[before]+1
  }
  else{
   config_count=c(config_count,1)
   config_index=c(config_index,s)
  }
 }
}

# Compute estimated probability of configurations
config_p=config_count/S
# Sort estimated probabilities from highest to lowest
config_sp=sort(config_p, decreasing=TRUE, index.return=TRUE)
# Number of configurations with positive estimated probability
length(config_p)

# Ordered Configurations
config_top=config[config_index[config_sp$ix],]

# Compute estimated regression line for each configurations
beta_est=list()
sigma_est=list()
X=matrix(c(rep(1,n),x),n,2)
bCb= t(beta_0)%*%C%*%beta_0
for (i in 1:length(config_p)){
 k_i=max(config_top[i,])
 betai=matrix(0,2,k_i)
 sigmai=matrix(0,1,k_i)
 for(j in 1:k_i){
  C_hat=C+ t(matrix(X[config_top[i,]==j,],ncol=2))%*%
    matrix(X[config_top[i,]==j,], ncol=2)
  decomp=eigen(C_hat, symmetric=TRUE)
```

```
  iC_hat=decomp$vectors%*%diag(1/decomp$values)%*%t(decomp$vectors)
  beta_hat=iC_hat%*%(C%*%beta_0+t(matrix(X[config_top[i,]==j,],ncol=2))
    %*%y[config_top[i,]==j])
  a_hat=a_y+sum(config_top[i,]==j)/2
  b_hat=b_y+(sum(y[config_top[i,]==j]^2)+bCb-t(beta_hat)%*%C_hat%*%beta_hat)/2
  betai[,j]=beta_hat
  sigmai[j]=b_hat/(a_hat-1)
 }
 beta_est[[i]]=betai
 sigma_est[[i]]=sigmai
}

# Plot top 3 models
par(mfrow=c(3,1), mar=c(2,2,1,1)+.1)
for(i in 1:3){
 plot(x,y, main=paste( "p( rho |y,x)=",format(config_sp$x[i],digits=4,
   scientific=F)))
 k_i=max(config_top[i,])
 for(j in 1:k_i){
  abline(beta_est[[i]][1,j], beta_est[[i]][2,j], col=j)
  points(x[config_top[i,]==j], y[config_top[i,]==j],col=j)
 }
}

#########################################################################

######## PREDICTION

# Covariates for new subjects

# Ex 1
x_new=c(0.2,3.3,5.9,6.2,6.3, 7.9,8.1, 8.7)

# Ex 2
x_new=seq(-4.5,4.5,1)

# Ex 3
x_new=seq(-2*pi,2*pi,pi/8)

#########COMPUTE PREDICTION

# Number of new subjects
m=length(x_new)

# Create X_new matrix
```

```
X_new=matrix(c(rep(1,m),x_new),ncol=2)

# Initialize
pred_mat=matrix(0,S,m)
weight_mat=matrix(0,S,m)

# Compute marginal for x
x_sc=(x_new-mu_0)*(c_x*a_x/((c_x+1)*b_x))^.5
marg_x=dt(x_sc,2*a_x)

for(s in 1:S){
 #need counts
 n_s=matrix(0,1,k[s])
 #calculate weight matrix
 weight_mat_s=matrix(0,m,k[s]+1)
 weight_mat_s[,1]=alpha*marg_x
 for(j in 1:k[s]){
  n_s[j]=sum(config[s,]==j)
  weight_mat_s[,j+1]=n_s[j]*dnorm(x_new, mu_x[[s]][j], sigma_x[[s]][j]^.5)
 }
 #calculate prediction
 pred_s=(X_new%*%cbind(beta_0, beta[[s]]))*weight_mat_s
 pred_mat[s,]=rowSums(pred_s)
 weight_mat[s,]=rowSums(weight_mat_s)
}

# Final Prediction
y_pred=colSums(pred_mat)/colSums(weight_mat)

# True Prediction

# Ex 1
y_true=rep(0,m)
y_true[x_new<=6]=-1/8*x_new[x_new<=6]+5
y_true[x_new>6]=2*x_new[x_new>6]-12

# Ex 2
y_true=x_new^2

# Ex 3
y_true=x_new*sin(x_new)

## PLOT PREDICTION

par(mfrow=c(1,1))
```

```
plot(c(x,x_new),c(y,y_true))
points(x_new, y_pred, col=2)
lines(x_new,y_pred,col=2)
lines(x_new,y_true)

## Compute L2 Error
l2_err=sum(((y_true-y_pred)^2)/m)^.5
```

4.3 restricted DPM

```
#restricted DP mixture for regression
#Calls rdpm MCMC function to obtain posterior samples

library(mvtnorm)
library(msm)
library(MCMCpack)

#########################################################################
#SIMULATE DATA

##### EXAMPLE 1: PIECEWISE LINEAR

x=seq(0,9,.25)
n=length(x)
y=rep(0,n)
y[x<=6]=-1/8*x[x<=6]+5
y[x>6]=2*x[x>6]-12
plot(x,y)

#Hyperparameters
beta_0=matrix(c(0,0), 2,1)
C=diag(c(1/(36*4),1/4),2)
iC=diag(c(36*4,4),2)
a_y=2
b_y=1/4
alpha=1


##### EXAMPLE 2: PARABOLA

set.seed(10001)

n=50
x=runif(n, -5, 5)
y=rnorm(n,x^2, 1)
```

```
plot(x,y)

#Hyperparameters
beta_0=matrix(c(-12,0), 2,1)
C=diag(c(1/50,1/25),2)
iC=diag(c(50,25),2)
a_y=2
b_y=1*(a_y-1)
alpha=1


##### EXAMPLE 3: XSINX

set.seed(10001)

n=100
x=runif(n, -2*pi, 2*pi)
y=rnorm(n,x*sin(x), 1/4)
plot(x,y)

#Hyperparameters
beta_0=matrix(c(0,0), 2,1)
C=diag(c(1/(16*18^2),1/(16*9)),2)
iC=diag(c(16*18^2,16*9),2)
a_y=2
b_y=1/16
alpha=1

############################################################

## INPUT FOR MCMC

S=10000
burnin=1000

#Initialize chain

#Start everyone in 1 group
d_init=rep(1,n)

#########################################################################

# Call rdpm MCMC function
set.seed(101010)
output=rdpm_mcmc(S, burnin, alpha,y, x, beta_0, C, a_y, b_y, d_init)
```

```
# Define variables
d=output$d
k=output$k
lp_n_k=output$lp_n_k
n_k=output$n_k

#########################################################################

### ANALYSIS OF PARTITIONS

# Compute probability of partitions and estimated regression lines
# within cluster for each partition

#Find unique configurations
sort_mod=sort(lp_n_k, decreasing=TRUE,index.return=TRUE)
rank_mod=sort_mod$ix
uniq_p=unique(sort_mod$x)
n_uniq_p=length(uniq_p)

#Empty matrix of estimated probabilities
prob_p=matrix(0,n_uniq_p,1)

#Empty lists for regression lines
beta_est=list()
sigma_est=list()

#Empty matrices for k and cumulative cluster sizes
k_top=matrix(0,n_uniq_p,1)
cum_n_top=list()

#Index of ith unique configuration among sorted sampled configurations
#For first configuration, index is one
rind_i=1

#Sort data according to x
sort_x=sort(x, index.return=TRUE)
x_order=sort_x$x
ind_order=sort_x$ix
y_order=y[ind_order]
X_order=matrix(c(rep(1,n),x_order),n,2)

for( i in 1:n_uniq_p){
 #Compute probability
 prob_p[i]=sum(lp_n_k==uniq_p[i])/(S)
```

```
 #Find number of corresponding clusters and cluster sizes
 ind_i=rank_mod[rind_i]
 k_i=k[ind_i]
 k_top[i]=k_i
 n_i=n_k[[ind_i]]
 cum_n=cumsum(c(0,n_i))
 cum_n_top[[i]]=cum_n
 #Compute regression lines
 betai=matrix(0,2,k_i)
 sigmai=matrix(0,1,k_i)
 for(j in 1:k_i){
  C_hat=C+ t(matrix(X_order[(cum_n[j]+1):cum_n[j+1],],ncol=2))%*%
    matrix(X_order[(cum_n[j]+1):cum_n[j+1],], ncol=2)
  decomp=eigen(C_hat, symmetric=TRUE)
  iC_hat=decomp$vectors%*%diag(1/decomp$values)%*%t(decomp$vectors)
  beta_hat=iC_hat%*%(C%*%beta_0+t(matrix(X_order[(cum_n[j]+1):cum_n[j+1],],ncol=2))
    %*%y_order[(cum_n[j]+1):cum_n[j+1]])
  a_hat=a_y+n_i[j]/2
  b_hat=b_y+(sum(y_order[(cum_n[j]+1):cum_n[j+1]]^2)+t(beta_0)%*%C%*%beta_0
    -t(beta_hat)%*%C_hat%*%beta_hat)/2
  betai[,j]=beta_hat
  sigmai[j]=b_hat/(a_hat-1)
 }
 beta_est[[i]]=betai
 sigma_est[[i]]=sigmai
 #Update index
 rind_i=rind_i+sum(lp_n_k==uniq_p[i])
}

#Number of unique configurations
n_uniq_p

#Partitions are currently sorted by log of non-normalized posterior
#Sort by estimated probability
prob_ps=sort(prob_p, decreasing=TRUE, index.return=TRUE)

# Plot top 3 models
par(mfrow=c(3,1), mar=c(2,2,1,1)+.1)
for(i in 1:3){
 plot(x,y, main=paste( "p( rho |y,x)=",format(prob_ps$x[i],digits=4,
   scientific=F)))
 cum_n=cum_n_top[[prob_ps$ix[i]]]
 for(j in 1:k_top[prob_ps$ix[i]]){
  abline(beta_est[[prob_ps$ix[i]]][1,j], beta_est[[prob_ps$ix[i]]][2,j], col=j)
  points(x_order[(cum_n[j]+1):cum_n[j+1]], y_order[(cum_n[j]+1):cum_n[j+1]],col=j)
```

```
 }
}

#########################################################################

######## PREDICTION

# Covariates for new subjects

# Ex 1
x_new=c(0.2,3.3,5.9,6.2,6.3, 7.9,8.1, 8.7)

# Ex 2
x_new=seq(-4.5,4.5,1)

# Ex 3
x_new=seq(-2*pi,2*pi,pi/8)

#########COMPUTE PREDICTION

m=length(x_new)

#find position of new subjects
sort_x_old=sort(x, index.return=TRUE)
x_order_old=sort_x_old$x
pos_new=rowSums(t(matrix(x_order_old, nrow=n, ncol=m))
  <matrix(x_new, nrow=m, ncol=n))

ind_order_old=sort_x_old$ix
X_order_old=matrix(c(rep(1,n),x_order_old),n,2)
y_order_old=y[ind_order_old]

#list containing weights
weight_list=list()

#list containing predicted y
y_hat_list=list()

#Needs output from analysis of partitions to compute
#(in particular, prob_p, cum_n_top, beta_est)
for (i in 1:m){
 X_new_mat=matrix(c(1, x_new[i]),ncol=2)
 weight_list[[i]]=list()
 y_hat_list[[i]]=list()
 ni=0
```

```
for(l in 1:length(prob_p)){

 #check if new subject is bigger or smaller than any seen
 if(pos_new[i]==0||pos_new[i]==n){
  cum_group_ind=0
  if(pos_new[i]==0){
   cum_group_ind=2
  }
  else{
   cum_group_ind=k_top[l]+1
  }

  #calculate weights for old group
  n_lj=cum_n_top[[l]][cum_group_ind]-cum_n_top[[l]][cum_group_ind-1]
  new_weight=prob_p[l]*(n_lj/(n_lj+1))
  weight_list[[i]][[ni+1]]=new_weight

  #calculate prediction for old group
  y_hat=X_new_mat%*%beta_est[[l]][,cum_group_ind-1]
  y_hat_list[[i]][[ni+1]]=y_hat

  #calculate weights for new group
  new_weight=prob_p[l]*(alpha/(k_top[l]+1))
  weight_list[[i]][[ni+2]]=new_weight

  #calculate prediction for old group
  y_hat=X_new_mat%*%beta_0
  y_hat_list[[i]][[ni+2]]=y_hat
  ni=ni+2
 }
 else{

  #is the new subject between groups?
  if(sum(pos_new[i]==cum_n_top[[l]])>0){
   l_group_ind=which(pos_new[i]==cum_n_top[[l]])
   r_group_ind=l_group_ind+1

   #calculate weights for old group on the left
   n_lj=cum_n_top[[l]][l_group_ind]-cum_n_top[[l]][l_group_ind-1]
   new_weight=prob_p[l]*(n_lj/(n_lj+1))
   weight_list[[i]][[ni+1]]=new_weight

   #calculate prediction for old group on the left
   y_hat=X_new_mat%*%beta_est[[l]][,l_group_ind-1]
   y_hat_list[[i]][[ni+1]]=y_hat
```

```
    #calculate weights for old group on the right
    n_lj=cum_n_top[[l]][r_group_ind]-cum_n_top[[l]][r_group_ind-1]
    new_weight=prob_p[l]*(n_lj/(n_lj+1))
    weight_list[[i]][[ni+2]]=new_weight

    #calculate prediction for old group on the right
    y_hat=X_new_mat%*%beta_est[[l]][,r_group_ind-1]
    y_hat_list[[i]][[ni+2]]=y_hat

    #calculate weights for new group
    new_weight=prob_p[l]*(alpha/(k_top[l]+1))
    weight_list[[i]][[ni+3]]=new_weight

    #calculate prediction for new group
    y_hat=X_new_mat%*%beta_0
    y_hat_list[[i]][[ni+3]]=y_hat
    ni=ni+3
   }
   else{
    group_ind=sum(pos_new[i]>cum_n_top[[l]])+1

    #calculate weights for old group
    n_lj=cum_n_top[[l]][group_ind]-cum_n_top[[l]][group_ind-1]
    new_weight=prob_p[l]*(n_lj/(n_lj+1))
    weight_list[[i]][[ni+1]]=new_weight

    #calculate prediction for old group
    y_hat=X_new_mat%*%beta_est[[l]][,group_ind-1]
    y_hat_list[[i]][[ni+1]]=y_hat
    ni=ni+1
   }
  }
 }
}

#calculate final prediction

norm_weights=list()
y_hat=list()
y_pred=rep(0,m)

for(i in 1:m){
 norm_weights[[i]]=unlist(weight_list[[i]])/sum(unlist(weight_list[[i]]))
 y_hat[[i]]=unlist(y_hat_list[[i]])
```

```
 y_pred[i]=sum(y_hat[[i]]*norm_weights[[i]])
}

# True Prediction

# Ex 1
y_true=rep(0,m)
y_true[x_new<=6]=-1/8*x_new[x_new<=6]+5
y_true[x_new>6]=2*x_new[x_new>6]-12

# Ex 2
y_true=x_new^2

# Ex 3
y_true=x_new*sin(x_new)

## PLOT PREDICTION

par(mfrow=c(1,1))
plot(c(x,x_new,x_new),c(y,y_true,y_pred))
points(x_new, y_pred, col=2)
lines(x_new,y_pred,col=2)
lines(x_new,y_true)

## Compute L2 Error
l2_err=sum(((y_true-y_pred)^2)/m)^.5
```