PhD Written Candidacy Examination Part III: Bayesian Estimation of Models for Repeated Measures and Sequential Data

Rick Farouni

Tuesday 15th December, 2015

Examination Committee

Dr. Bob Cudeck Dr. Steve MacEachern Dr. Zhong-Lin Lu

Contents

List of Figures

Part I

Dr. Cudeck: Bayesian estimation of models for repeated measures and sequential data

Mixture models have been studied by many authors in order to address the problem of heterogeneity in a population. For example, mixture models are proposed as a way to account for different types of response styles in repeated measures studies. A very general approach of this kind is a model that specifies that a heterogeneous population is a mixture of components of K normal populations. Let $\mathbf{y}$ be a $(Jx1)$ multivariate observation. To be concrete, if there are $K = 3$ component sub-populations then a popular specification for the distribution of y is

$$g(\mathbf{y}) = \pi_1 g_1(\mathbf{y}) + \pi_2 g_2(\mathbf{y}) + \pi_3 g_3(\mathbf{y})$$

where $g_k(\mathbf{y})$ is the normal distribution with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $|\boldsymbol{\Sigma}_k$, and π_k , with $0 < \pi_k < 1$ and $\pi_1 + \pi_2 + \pi_3 = 1$, is the probability of membership in class k.
In most approaches to the mixture problem it is essential to assume that K is known and specified a priori. Unfortunately specification of K is extremely difficult even when $K \leq 3$. Furthermore, in most approaches to the mixture problem the component distributions, $g_k(\mathbf{y})$, are normals even though it is easy to imagine other distribution being more flexible.

1 Question Part A: Model Specification

Question Present a Bayesian version of the mixture model that is as general as possible in terms of the number and type of component distributions.

Dirichlet Process Mixture of Normals Model For subjects $i = 1, 2, .., T$, let $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{iD}]$ be vector of D measurements for subject i, where $D \ll T$. Here we assume that the subjects can be divided into K clusters such that each subject (i.e. observation) belongs to one particular cluster only. That is, each observation $\mathbf{y}_i$ is given a cluster label

$z_i \in \{1, \ldots, K\}$. We specify a **Dirichlet Process Mixture of Normals** model as follows

$$\begin{aligned} G \mid \alpha &\sim \mathrm{DP}(\alpha G_0) \\ (\boldsymbol{\mu}_t, \Sigma_t) \mid G &\sim G \\ \mathbf{y}_t \mid \boldsymbol{\mu}_t, \Sigma_k &\sim \mathrm{Normal}(\boldsymbol{\mu}_t, \Sigma_t) \end{aligned} \tag{1}$$

where

α *is the concentration parameter*

$G_0 = p(\boldsymbol{\mu}, \Sigma) \equiv \text{Normal-Inv-Wishart}(\boldsymbol{\mu}_0, \kappa_0, \nu_0, \Lambda_0)$ *is the centering distribution*

$\theta_k = (\boldsymbol{\mu}_k, \Sigma_k)$ *mean and covariance of observations in the kth class*

$\Theta = (\theta_1, \cdots, \theta_K)$ *collection of all K parameter vectors*

$\mathcal{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T)$ *the data consiting of all T observations*

$\mathcal{Z} = (z_1, \cdots, z_T)$ *vector of class labels for all T observations*

$\mathcal{C} = \{c_1, c_2, \ldots, c_K : c_k \subseteq \{1, 2, \ldots, T\}\}$ *clustering of the observations into K clusters*

$\mathcal{N} = \{N_1, N_2, \ldots, N_K : N_k = \sum_{i=1}^{T} \delta(z_i = k)\}$ *number of observations in each cluster*

2 Question Part B: Forward Simulation

Question Generate artificial data based on the model you specify in Part A

Figure 1 shows plots of $T = 500$ three dimensional observations generated four times under Model 1 with the following the hyperparameters:

$$\begin{aligned}
\alpha &= 0.5 \quad \textit{is the concentration parameter} \\
\boldsymbol{\mu}_0 &= [0, 0, 0] \quad \textit{prior belief about mean vector} \\
\kappa_0 &= .03 \quad \textit{number of pseudo-observations ascribed to the prior} \\
\nu_0 &= 30 \quad \textit{confidence of prior belief} \\
\Lambda_0 &= \mathbf{I}_3 \quad \textit{variation from the mean vector}
\end{aligned}$$

The four realizations of the Dirichlet process mixture model show large variation in the separation of the mixture components. An alternative way of generating data starts with pre-setting the values of the $\boldsymbol{\mu}$ and Σ component parameters to a fixed values. For example, Σ can be assigned the identity matrix and $\boldsymbol{\mu}$ can take values of the coordinates of a $D - dimensinal$ hypercube for a reasonable number of components we expect to obtain based on the value of α. Figure 2 shows simulated data, for 300 observations, where the values of the $\boldsymbol{\mu}$ components are the multiset permutations of the $(\pm 1.5, \pm 1.5, \pm 1.5)$ coordinate values.

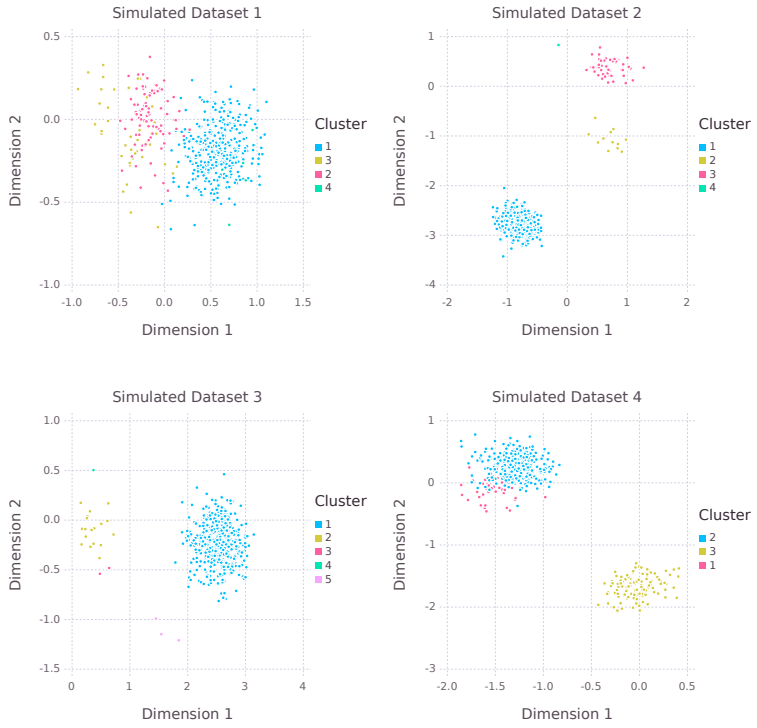

Figure 1: Four realization of the data generating model

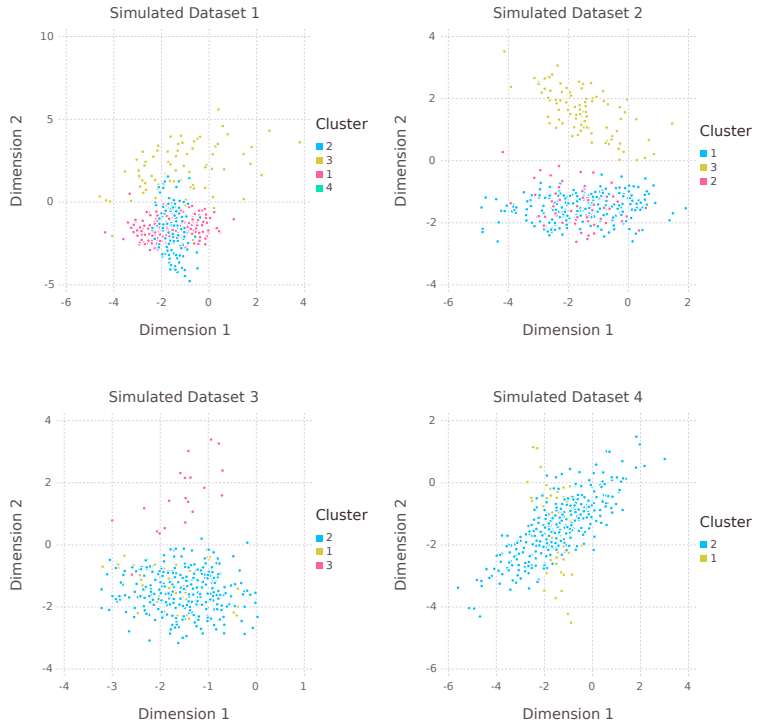

Figure 2: Four realization of the data generating model for fixed parameter values

For posterior simulation to be conducted in Part C, we have chosen Dataset 4 with four components. Figure 3 shows the dataset from different views so that its three dimensional structure can be visualized.

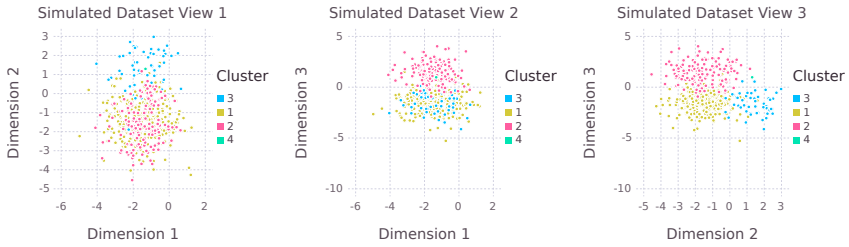

Figure 3: Three views of Dateset 4

3 Question Part C: Posterior Simulation

Question There are several ways to estimate the model. Write computer code in your language of choice that fits the model of Part (A) to the data of Part (B). Use the estimation method you believe is most appropriate and most flexible. (i) Briefly describe the method and the properties that make it attractive. Give proper references. (ii) Fit the model. (iii) Review how model fit is assessed and report estimates. (iv) Comment on how well the parameters are recovered. Attach computer code as an appendix.

Answer Numerical integration can be divided into deterministic methods and simulation methods. Deterministic (i.e. gird based) numerical integration methods such as adaptive quadrature have lower variance but breakdown in high dimensions. Simulation methods have higher variance but scale better in high dimensions. Two of the most important simulation based methods are Markov Chain Monte Carlo and **Sequential Monte Carlo**. For posterior inference on the selected dataset, we consider a Sequential Monte Carlo method known as **Sequential Importance Resampling** or the Particle Filter.

Let $\mathbf{z}_{1:T} = \{z_1, z_2, \ldots, z_T\}$ and $\mathbf{y}_{1:T} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$ denote the latent cluster labels and the set of observations up to time T. The posterior distribution of the cluster labels is

$$\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) = \frac{p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})}{p(\mathbf{y}_{1:T})} = \frac{p(\mathbf{y}_{1:T} \mid \mathbf{z}_{1:T}) p(\mathbf{z}_{1:T})}{p(\mathbf{y}_{1:T})} \tag{2}$$

As T increases, we obtain a sequence of probability densities of increasing dimension, namely $\{\pi_T(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})\}_{T \in \mathbb{N}}$. Sequential Monte Carlo methods sample sequentially from $\{\pi_T(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})\}_{T \in \mathbb{N}}$ providing an approximation of each target distribution $\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ and an estimate of its corresponding normalizing constant $Z_T = p(\mathbf{y}_{1:T})$.

The Particle Filter algorithm extends Sequential Importance Sampling method, which in turn builds upon **Importance Sampling**, which in turn is a modification of Monte Carlo simulation methods. Each of the four methods has been developed to overcome a specific type of problem. The four problems are basically concerned with the target posterior, Equation 2.

3.1 Monte Carlo Simulation

Problem 1 The target distribution is impossible to compute in closed-form

Solution We can simulate the target distribution using random samples (i.e. particles) drawn from it. More specifically, for particles, $\mathbf{z}_{1:T}^{(i)} \sim \pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ for $i = 1, ..., N$, the perfect Monte Carlo approximation is:

$$\hat{\pi}_N(d\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) = \frac{1}{N} \sum_{i=1}^{N} \delta_{\mathbf{z}_{1:T}}(i)(d\mathbf{z}_{1:T})$$

3.2 Importance Sampling

Problem 2 The target $\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ is a complex high-dimensional distribution such that sampling from it is practically impossible

Solution We choose a proposal distribution $q(.)$ we can sample from and reweigh the probability measure accordingly. More specifically, let

$$\begin{aligned}\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) &= \frac{\left[\frac{\pi(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})}{q(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})}\right] q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})}{\int \left[\frac{\pi(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})}{q(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})}\right] q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) d\mathbf{z}_{1:T}} \\ &\propto w_T(\mathbf{z}_{1:T}) q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})\end{aligned}$$

where

$$w_T(\mathbf{z}_{1:T}) = \frac{\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})}{q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})}$$

Now for particles, $\mathbf{z}_{1:T}^{(i)} \sim q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ for $i = 1, ..., N$, the Monte Carlo approximation is:

$$\hat{\pi}_N(d\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) = \sum_{i=1}^{N} W_T(\mathbf{z}_{1:T}^{(i)}) \delta_{\mathbf{z}_{1:T}}(i)(d\mathbf{z}_{1:T})$$

where the normalized importance weights are

$$W_T(\mathbf{z}_{1:T}^{(i)}) = \frac{w_T(\mathbf{z}_{1:T}^{(i)})}{\sum_{j=1}^{N} w_T(\mathbf{z}_{1:T}^{(j)})} \quad , w_T(\mathbf{z}_{1:T}^{(i)}) = \frac{\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})}{q(\mathbf{z}_{1:T}^{(i)} \mid \mathbf{y}_{1:T})}$$

3.3 Sequential Importance Sampling

Problem 3 The number of observations T is large and the computational complexity of sampling increases at least linearly with T

Solution We can use a divide-and-conquer strategy and break the problem into smaller parts. This approach to problem solving is known as *recursion*. Recursion is usually contrasted with *iteration*, where we resort to repetition until a task is completed (e.g. MCMC).

First, consider a general state space model

$$z_t \sim p(z_t \mid z_{t-1}) \quad t \geq 1$$
$$\mathbf{y}_t \sim p(\mathbf{y}_t \mid z_t) \quad t \geq 1$$

that assumes:

1. The Markov property of latent labels

$$p(z_t \mid \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1}) = p(z_t \mid z_{t-1})$$

2. The conditional independence of observations

$$p(\mathbf{y}_t \mid \mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}) = p(\mathbf{y}_t \mid z_t)$$

These two assumptions imply that

$$p(\mathbf{y}_{1:T} \mid \mathbf{z}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{y}_t \mid z_t)$$
$$p(\mathbf{z}_{1:T}) = \prod_{t=1}^{T} p(z_t \mid z_{(t-1)}) \qquad (p(z_0) = c)$$

allowing the full the posterior to be factored as such

$$\pi(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) \propto \prod_{t=1}^{T} p(\mathbf{y}_t \mid z_t) p(z_t \mid z_{(t-1)})$$

which gives us the following recursion relation:

$$\pi(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \propto p(\mathbf{y}_t \mid z_t) p(z_t \mid z_{(t-1)}) \pi(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1})$$

In sequential importance sampling, we select an importance distribution with the following recursive structure

$$q(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) = q(\mathbf{z}_t \mid \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}) q(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1})$$

The above decomposition implies we can at time t draw particles $\mathbf{z}_{1:t}^{(i)} \sim q(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ for $i = 1, ..., N$ by

1. drawing samples $\mathbf{z}_{1:t-1}^{(i)}$ from $q(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1})$

2. drawing new samples $\mathbf{z}_t^{(i)}$ from $q(\mathbf{z}_t \mid \mathbf{z}_{1:t-1}^{(i)}, \mathbf{y}_{1:t})$.

The Monte Carlo approximation of $\pi(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ at time t is:

$$\hat{\pi}_N(d\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) = \sum_{i=1}^{N} W_t(\mathbf{z}_{1:t}^{(i)}) \delta_{\mathbf{z}_{1:t}}(i)(d\mathbf{z}_{1:t})$$

as a result we get the following recursion relation for the importance weights:

$$\begin{aligned} w_t(\mathbf{z}_{1:t}) &= \frac{\pi(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})}{q(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})} \\ &\propto \frac{p(\mathbf{y}_t \mid z_t) p(z_t \mid z_{t-1}) \pi(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1})}{q(z_t \mid \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}) q(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1})} \\ &\propto \frac{p(\mathbf{y}_t \mid z_t) p(z_t \mid z_{t-1})}{q(z_t \mid \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}))} w_{t-1}(\mathbf{z}_{1:t-1}) \end{aligned}$$

So if at time $t-1$ we have a set of particles and corresponding weights $\{(W_{t-1}^{(i)}, \mathbf{z}_{1:t-1}^{(i)}) : i = 1, ..., N\}$ that represent the distribution $\pi(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1})$, we can obtain a Monte Carlo approximation to $\pi(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ at time t by

1. drawing new samples $z_t^{(i)}$ from $q(z_t \mid \mathbf{z}_{1:t-1}^{(i)}, \mathbf{y}_{1:t})$.
2. updating the weights $\frac{p(\mathbf{y}_t \mid z_t^{(i)}) p(z_t^{(i)} \mid z_{t-1}^{(i)})}{q(z_t^{(i)} \mid \mathbf{z}_{1:t-1}^{(i)}, \mathbf{y}_{1:t}))} w_{t-1}(\mathbf{z}_{1:t-1}^{(i)})$ and then normalizing them

so for particles, $z_t^{(i)} \sim q(z_t \mid \mathbf{z}_{1:t-1}^{(i)}, \mathbf{y}_{1:t})$ for $i = 1, ..., N$, the Monte Carlo approximation for the filtering distribution is:

$$\hat{\pi}_N(dz_t \mid \mathbf{y}_{1:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{z_t}(i)(dz_t)$$

such that the set of particles and corresponding weights $\{(W_t^{(i)}, z_t^{(i)}) : i = 1, ..., N\}$ represent the distribution $\pi(z_t \mid \mathbf{y}_{1:t})$ at time t.

We can now focus on the marginal distribution $p(z_t \mid \mathbf{y}_{1:t-1})$, which has the following recursive relations that can be obtained by the Chapman-Kolmogorov equation:

$$\begin{aligned} \pi(z_t \mid \mathbf{y}_{1:t-1}) &= \int p(z_t \mid z_{t-1}) \pi(z_{t-1} \mid \mathbf{y}_{1:t-1}) dz_{t-1} \\ \pi(z_t \mid \mathbf{y}_{1:t}) &= \frac{p(\mathbf{y}_t \mid z_t) p(z_t \mid \mathbf{y}_{1:t-1})}{\int p(\mathbf{y}_t \mid z_t) p(z_t \mid \mathbf{y}_{1:t-1}) dz_t} \end{aligned}$$

based on which, the prior and posterior can be approximated as follows:

$$\hat{\pi}_N(z_t \mid \mathbf{y}_{1:t-1}) \propto \sum_{i=1}^{N} W_t^{(i)} p(z_t \mid z_{t-1}^{(i)})$$
$$\hat{\pi}_N(z_t \mid \mathbf{y}_{1:t}) \propto \sum_{i=1}^{N} W_t^{(i)} p(\mathbf{y}_t \mid z_t) p(z_t \mid z_{t-1}^{(i)})$$

which play a part in the following relationship:

$$\begin{aligned}
\pi(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) &\propto p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}, \mathbf{y}_t) \\
&\propto p(\mathbf{y}_t \mid \mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}) p(\mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}) \\
&\propto p(\mathbf{y}_t \mid \mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}) p(z_t \mid \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1}) p(\mathbf{z}_{1:t-1} \mid \mathbf{y}_{1:t-1}) \\
&\propto p(\mathbf{y}_t \mid \mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}) p(z_t \mid \mathbf{y}_{1:t-1})
\end{aligned}$$

3.4 Sequential Importance Resampling (The Particle Filter)

Problem 4 The Degeneracy Problem: Almost all of the particles can have zero, or close to zero, weights.

Solution Multiply particles with large weights and eliminate those with small weights. More specifically, resample the collection of weights and particles $\{W_t^{(i)}, \mathbf{z}_{1:t}^{(i)}\}$ (i.e. select $\mathbf{z}_{1:t}^{(i)}$ with probability $W_t^{(i)}$) to obtain N new equally-weighted particles $\{\frac{1}{N}, \bar{\mathbf{z}}_{1:t}^{(i)}\}$.
That is, the distribution $\pi(\mathbf{z}_t \mid \mathbf{y}_{1:t})$ which is approximated by

$$\hat{\pi}_N(d\mathbf{z}_t \mid \mathbf{y}_{1:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{\mathbf{z}_t}(i)(d\mathbf{z}_t)$$

becomes in turn approximated by a resampled empirical measure

$$\bar{\pi}_N(d\mathbf{z}_t \mid \mathbf{y}_{1:t}) = \sum_{i=1}^{N} \frac{N_t^{(i)}}{N} \delta_{\mathbf{z}_t}(i)(d\mathbf{z}_t) = \sum_{i=1}^{N} \frac{1}{N} \delta_{\bar{\mathbf{z}}_t}(i)(d\mathbf{z}_t)$$

where $N_t^{(i)}$ is the number of offspring associated with each particle $\mathbf{z}_{1:t}^{(i)}$
Several resampling algorithms have been proposed, but we will discuss the **stratified sampling algorithm** that has been proposed by (Carpenter, Clifford, & Fearnhead, 1999). The main idea of the algorithm is to remove particles whose weights fall below a given threshold $\frac{1}{c}$ and resample from the remaining particles. The threshold is chosen so that the resampling is optimal for all unbiased resampling procedures in terms of minimizing variability in the weights introduced by resampling.

3.5 Particle Filter Implementation

We have implemented the Sequential Importance Resampling algorithm in *Julia*. The exact details of the implementation of the multivariate Dirichlet Process Mixture of Normals are given in(Fearnhead, 2004) and in (Wood & Black, 2008). The authors of latter paper kindly provided *Matlab* code of their implementation. The *Julia* code we wrote was based upon their Matlab code, especially the *resample startified* function that implements the resampling step. The *Julia* code can be found in the appendix.

3.6 Model Fit

Figure 4 displays the results of data clustering using Dirichlet Process Mixture of Normals model.

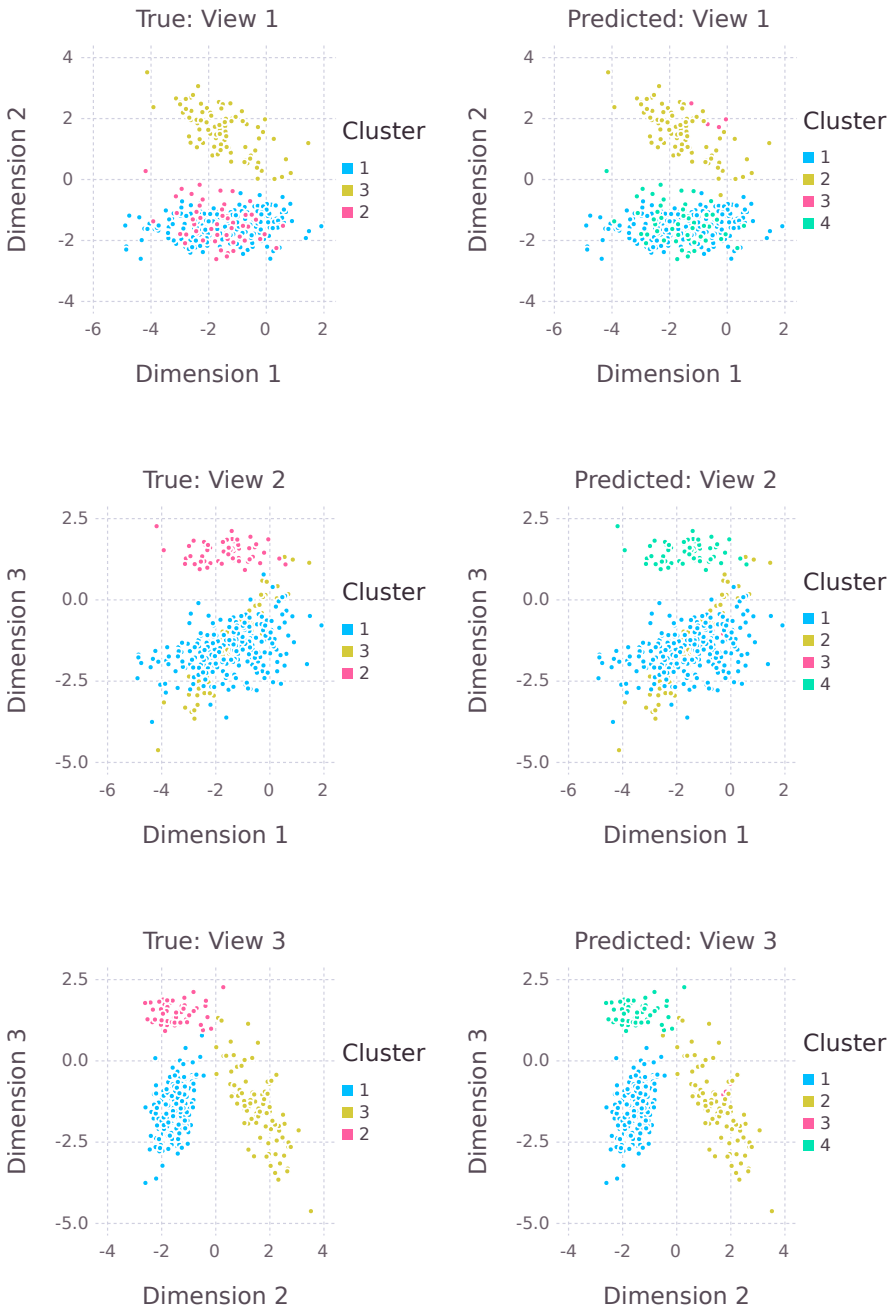

Figure 4: Latent Classes: True vs Predicted

As can be seen from the graph, the cluster labelling is shuffled. This is due to the invariance of the posterior distribution under the relabelling of the components. The estimated cluster labels were permuted in order to produce the accuracy estimates.

It can be seen that whereas the true number of clusters is three, the mean number of clusters that the posterior inference gave was four. It should be noted that the mean of the posterior distribution is just a point summary of the uncertainty that is inherent in the inference and that a more complete picture is provided by the entire distribution of the number of clusters. Moreover, the DPMN model assumes an infinite number of clusters and for finite data the expected number of clusters is a function of the concentration parameter that we set to $\alpha = 0.4$. The full implication of the model's assumptions and the effect to of the value of α on inference is covered in Chapter 1.

At any rate, we can assess the model's classification accuracy with the aid of a confusion matrix, Figure 5. The confusion matrix we obtained is actually better called a **matching matrix** since the cluster labels are treated as unobserved latent variables in the model. Now, the four rows of the matching matrix refer the true cluster labels (i,.e. 1, 2, 3) plus an additional label, Label 4. The columns refer to the predicted cluster labels (i,.e. 1, 2, 3, 4). We see that for Label 1, the hit rate is %100. That is, all observations in Cluster 1, except for a a single observation, were correctly classified and labelled. An overall measure of classification accuracy is given by the **true positive rate**, which measures the proportion of cluster labels that are correctly identified. The true positive rate we obtained for the model is 0.988.

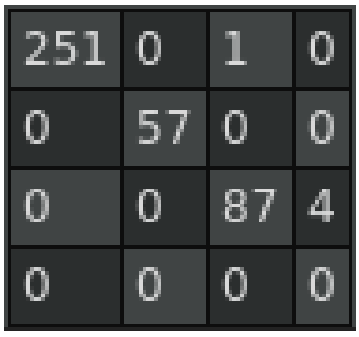

Figure 5: Confusion Matrix:
True vs Predicted Cluster Labels

One feature of the particle filter algorithm that is important to consider is the number of particles used represent the target distribution. The inaccuracy in the particle filter comes through approximating the posterior distribution by a finite number of particles, so an adequate number of particles is necessary for robust inference. We used 6000 particles to fit the model,but fewer number of particles was also attempted (e.g. 3000) with not much noticeable decrease in accuracy.

Part II

Appendix: Julia and R Code

.1 Forward Simulation Code in Julia

```
function sample_from_G₀!(μ,Σ,zᵢ)
  #sample from the centering distribution
  push!(Σ,rand(InverseWishart(ν₀, Λ₀))) # append to vector Σ
  push!(μ,rand(MultivariateNormal(μ₀,Σ[zᵢ]/κ₀))) # append to vector μ
  return μ,Σ
end

function polya_urn(T,α,D)
  Z=[1] # assign first observation to first cluster
  # draw a set of parameters for the first observation
  Σ=[rand(InverseWishart(ν₀, Λ₀)) for j=1:1]
  μ=[rand(MultivariateNormal(μ₀,Σ[1]/κ₀)) for j=1:1]
  for t=2:T
    K=maximum(unique(Z)) # number of clusters
    C=[find(Z.==k) for k=1:K] # determine the clustering of  indices
    N=[size(C[k],1) for k=1:K] # compute the cardinality of each cluster
    pᵢ=float64([N,α]/ (α+t-1)) # compute all K+1 mixing proportions
    zᵢ=rand(Categorical(pᵢ),1)[1] #sample cluster label for observation t
    push!(Z,zᵢ) # append new label
    if zᵢ> K # if new cluster is created, draw parameters from G₀
      μ,Σ = sample_from_G₀!(μ,Σ,zᵢ)
    end
  end
  return Z,μ,Σ
end

function generate_data_DPMG(T,α,D)
  #generate data for the Dirichlet Process Mixture of Gaussians model
    Z,μ,Σ=polya_urn(T,α,D) # generate parameters and labels
    K=maximum(unique(Z)) # determine the number of clusters
    C=[find(Z.==k) for k=1:K] # determine the clustering indices
    N=[size(C[k],1) for k=1:K] # compute the cardinality of each cluster
    y=[zeros(D,k) for k in N] #intialize data vector
    for k=1:K
        y[k]=rand(MultivariateNormal(μ[k],Σ[k]),N[k])
    end
    # concatenate cluster label vector with data array
    data=vcat(vcat(hcat(y...)),Z[vcat(C...)]')
    data=data[:, randperm(T)]  #permute data
    return data,μ,Σ
end
```


Figure 6: Julia code for generating data from a DPM model: Helping functions

```
using Gadfly, Distributions, DataFrames
α=0.5 # concentration parameter of DP
T=500 # number of observations
D=3 # dimension of observation
μ₀ = [0.  for i=1:D] # location: prior belief about mean vector
Λ₀ =1.00*eye(D) #inverse scale matrix: variation from the mean vector
ν₀= D+10 # degrees of freedom: confidence of prior belief
κ₀ = .04 # number of pseudo-observations ascribed to the prior
#intialize  create 4 simulated datasets
datasets={}; means={};covariances={} #intialize
for index=1:4
    data,μ,Σ=generate_data_DPMG(T,α,D)
    push!(datasets,data)
    push!(means,μ)
    push!(covariances,Σ)
end
p1=Dict() #plotting scripts
for i=1:4
    p1["$i"]=plot(x=datasets[i][1,:],y=datasets[i][2,:],
                  color=int16(vec(datasets[i][D+1,:])),
                  Guide.xlabel("Dimension 1"),
                  Guide.ylabel("Dimension 2"),
                  Guide.title("Simulated Dataset $i"),
                  Guide.colorkey("Cluster"),
                  Scale.color_discrete(),
                  Theme(default_point_size=.5mm))
end
draw(PDF("SimulatedData.pdf", 8inch, 8inch),
    vstack(hstack(p1["1"],p1["2"]),hstack(p1["3"],p1["4"])))
p2=Dict();ind={[1,2],[1,3],[2,3]}; j=1 # index of dataset
for i=1:3
    p2["$i"]=plot(x=datasets[j][ind[i][1],:],y=datasets[j][ind[i][2],:],
                  color=int16(vec(datasets[j][D+1,:])),
                  Guide.xlabel("Dimension $(ind[i][1])"),
                  Guide.ylabel("Dimension $(ind[i][2])"),
                  Guide.title("Simulated Dataset View $i"),
                  Guide.colorkey("Cluster"),
                  Scale.color_discrete(),
                  Theme(default_point_size=.5mm))
end
draw(PDF("SimulatedDataset.pdf", 9inch, 3inch),#write to file
```

Figure 7: Julia code for generating data from a DPM model: plotting

.2 Posterior Simulation Code in Julia: Particle Filter

```
function update_posterior(n,ȳ =0,ȳȳᵀ=0)
 # Gelman's Bayesian Data Analysis 3rd Edition Page 73
 # the fucntion first updates the parameters of the posterior distribution
# (μ,Σ) ~ Normal-InverseWishart(μₙ,κₙ,Λₙ,νₙ)
 # n is the number of observations in the cluster
 # ȳ is the mean of these observations,
 # (ȳȳᵀ - n*ȳ*ȳ') is sum of squares of these observations.
  μₙ = κ₀/(κ₀+n)*μ₀ + n/(κ₀+n)*ȳ
  κₙ = κ₀+n
  νₙ = ν₀+n
  Λₙ = Λ₀ +κ₀*n/(κ₀+n)*(ȳ-μ₀)*(ȳ-μ₀)' + (ȳȳᵀ - n*ȳ*ȳ')
  #the parameters of the posterior predictive distribution of
  # new observation ȳ ~tᵥ(μ,Σ)
  μ=μₙ
  Σ = Λₙ*(κₙ+1)/(κₙ*(νₙ-D+1))
  ν =νₙ-D+1
  return μ, Σ, ν
end

function log_posteriorpredictive(y,μ,Σ,ν,logdet_Σ=0, inv_Σ=0)
  # evaluate the the posterior predictive distribution of
  # new observation ȳ ~tᵥ(μ,Σ)
  # Compute logdet_Σ and inv_Σ if they have not been computed before
  if logdet_Σ ==0
      logdet_Σ = logdet(Σ)
      inv_Σ = inv(Σ)
  end
  logposterior = log_Γ[ν+D]-(log_Γ[ν] + D*p_log[ν]/2 + D*log_π/2)-
        logdet_Σ/2-((ν+D)/2)*log(1+(1/(ν))*(y-μ)'*inv_Σ*(y-μ))
  return logposterior[1]
end
```

Figure 8: Particle Filter: Function Definitions 1

```
function resample_stratified(z,w,N)
   #Algorithm 2:Carpenter, J., Clifford, P., & Fearnhead, P. (1999).
   #Improved particle filter for nonlinear problems
    #no putative particle is resampled more than once.
    (D, M) = size(z)
    z_resampled = zeros(Int32,D,N)
    w_resampled = zeros(N)
    selected = zeros(M)
    #permute
    permuted_indx = randperm(M)
    w = w[permuted_indx]
    z = z[:,permuted_indx]
    cdf = cumsum(w)
    cdf[end]=1
    N!=1 ? p =linspace(rand(Uniform())/N,1,N):p =1
    j=1
    for i=1:N
            while j<M && cdf[j]<p[i]
            j=j+1
        end
            selected[j] = selected[j]+1
    end
    k=1
    for i=1:M
            if(selected[i]>0)
                for j=1:selected[i]
                    z_resampled[:,k] = z[:,i]
                    w_resampled[k]= w[i]
                k=k+1
            end
        end
    end
    w_resampled = w_resampled./sum(w_resampled)
    return z_resampled#returns N samples
end
```

Figure 9: Particle Filter: Function Definitions 2

```
function compute_putative_weights(y,t,alpha=0.4)
    # Given N particles for the first t-1 observations,
    # there are M(≥2N ) possible particles for the first t observations.
    # particle n generates  K₊(n)+1 distinct putative particles
    # iα and iω are the starting and ending indexes of the K₊(n)+1 put prtcls
    # For any given particle z 1:n , with k i distinct clusters,
    # there are k i + 1 possible allocations for the (n + 1)st ob-servation
    M = sum(K₊[:,t-1])+N # M =number of putative particles to generate.
    Wᴾ = ones(M)
    iα= 1 #starting index
    iω= K₊[1,t-1]+1 #ending index
    Zᴾ = zeros(Int32,2,M)
    for n = 1:N
        Zᴾ[1,iα:iω] = n
        Zᴾ[2,iα:iω]= 1:iω-iα+1  #num_options i.e. K₊(1)+1
        #calculate the probability of each new putative particle
        m_k = cluster_counts[1:K₊[n,t-1],n,t-1]
        # multinomial conditional prior distribution
        prior = [m_k; alpha]./(alpha + t -1)
        # update the weights so that the particles
        # (and weights) now represent the predictive distribution
        Wᴾ[iα:iω] = W[n,t-1]*prior
        posterior_predictive_p = zeros(size(prior))
        for pnp_id = 1: (iω-iα)
            (μ, Σ, ν)=update_posterior(cluster_counts[pnp_id,n,t-1],
                                       Ȳ[:,pnp_id,n,t-1],SS[:,:,pnp_id,n,t-1])
            posterior_predictive_p[pnp_id] = exp(
                      log_posteriorpredictive(y,μ,Σ,ν,logdetΣ[pnp_id,n,t-1],
                                       invΣ[:,:,pnp_id,n,t-1]))
        end
        (μ, Σ, ν)=update_posterior(0,y)
        posterior_predictive_p[end] = exp(log_posteriorpredictive(y,μ,Σ,ν))
        Wᴾ[iα:iω] = Wᴾ[iα:iω].*posterior_predictive_p
        iα= iω+1
        if n!=N
            iω= iα+K₊[n+1,t-1]
        end
    end
    Wᴾ = Wᴾ ./ sum(Wᴾ)
  return Wᴾ, Zᴾ, M
end
```

Figure 10: Particle Filter: Function Definitions 3

```
function find_optimal_c(Q,N)
    Q=sort(Q,rev=true)
    c = 0
    k = 0
    M = length(Q)
    k_old = -Inf
    while k_old !=k
        k_old = k
            c = (N-k)/sum(Q[k+1:end])
            k = k+ sum(Q[k+1:M]*c .> 1)
    end
    return 1/c
end
#####################################################################
using Distributions,MLBase,Gadfly
dataset=readdlm("data.txt")'
Y=dataset[1:3,:]
const T = size(Y,2) # number of samples or time points
const D = size(Y,1) # dimension of  data
const N = 6000 # numuber of particles
#hyperparameters
const κ₀ = .05
const ν₀= D+2
const Λ₀ =0.04*eye(D)
const μ₀ = [0.  for i=1:D] #μ₀ = repmat([0],2,1)
# precomputed values
const max_class_index = T
const max_class_number = 20
const log_Γ = lgamma((1:max_class_index)/2)
const log_π = log(pi)
const p_log = log(1:max_class_index)
const class_type = Int16
#parameters θ={μₖ,Σₖ}∞
Ȳ = zeros(D,max_class_number,N,T) # means
SS = zeros(D,D,max_class_number,N,T) # sum of squares
cluster_counts = zeros(Int32,max_class_number,N,T) #
logdetΣ = zeros(max_class_number,N,T)
invΣ = zeros(D,D,max_class_number,N,T)
K₊ = ones(class_type,N,T) # number of distinct clusters
Z = zeros(class_type,N,T,2) # cluster labels (i.e. particles) where zₜ∈ {1,..,k}
W = zeros(N,T) #weights
```

Figure 11: Particle Filter: Function Definitions 4

```
# intialize
y = Y[:,1]
yy' = y * y'
(μ, Σ, ν)=update_posterior(1,y,yy')
for i=1:N
    Ỹ[:,1,i,1] = y
    SS[:,:,1,i,1] = yy'
    cluster_counts[1,i,1] = 1
    logdetΣ[1,i,1] = logdet(Σ)
    invΣ[:,:,1,i,1] =inv(Σ)
end
Z[:,1,1] = 1
W[:,1]= 1/N
cp =1  # cp is the set of current particles

for t = 2:T
    println("Observation Number = ",t)
    y = Y[:,t]
    yy' = y*y'
    (W^p,Z^p, M)=compute_putative_weights(y,t) # M= \Σ^N(k_i+1).
    # The M putative weights and particles give a discrete
    # approxmation of P(z_(1:t)|y_(1:t))
    #plot_particles(W^p,vec(slicedim(Z^p,1,2)), M,t)
    #calculate the weights of all possible putative Z at the next time-step.
    survival_threshold= find_optimal_c(W^p,N) # compute survival_threshold weight
    pass_inds = W^p.> survival_threshold #indices of surviving particles
    num_pass = sum(pass_inds) # number of surviving particles
    if cp == 1
        np = 2
    else
        np = 1
    end
    Z[1:num_pass,1:t-1,np] = Z[vec(Z^p[1,pass_inds]),1:t-1,cp]
    Z[1:num_pass,t,np] = Z^p[2,pass_inds]
    #weight of the propagated particles is kept unchanged
    W[1:num_pass,t]= W^p[pass_inds]
    # weight of resampled particles is set to the survival_threshold weight
    W[num_pass+1:end,t]= survival_threshold
    #resample the M possible particles to produce N particles for first t obs
    selected_Z^p = resample_stratified(Z^p[:,!pass_inds],W^p[!pass_inds],N-num_pass)
    max_K+ = maximum(K+[:,t-1])+1  # number of possible allocat for next cluster
```

Figure 12: Particle Filter: Main Body 1

```
    W[num_pass+1:end,t]= survival_threshold
    #resample the M possible particles to produce N particles for first t obs
    selected_Zᵖ = resample_stratified(Zᵖ[:,!pass_inds],Wᵖ[!pass_inds],N-num_pass)
    max_K₊ = maximum(K₊[:,t-1])+1  # number of possible allocat for next cluster
    for npind = 1:N
        if npind ≤  num_pass
            originating_particle_id = Zᵖ[1,pass_inds][npind]
            class_id_y = Zᵖ[2,pass_inds][npind]
        else
            originating_particle_id = selected_Zᵖ[1,npind-num_pass]
            class_id_y = selected_Zᵖ[2,npind-num_pass]
            Z[npind,1:t,np] = [Z[originating_particle_id,1:t-1,cp] ,class_id_y]
        end
        originating_particle_K₊ = K₊[originating_particle_id,t-1]
        new_count = cluster_counts[class_id_y,originating_particle_id,t-1]+1
        if new_count == 1
            K₊[npind,t] = originating_particle_K₊+1
        else
            K₊[npind,t] = originating_particle_K₊
        end
        cluster_counts[1:max_K₊,npind,t]= cluster_counts[1:max_K₊,
                                            originating_particle_id,t-1]
        cluster_counts[class_id_y,npind,t] = new_count
        old_mean = Ȳ[:,class_id_y,originating_particle_id,t-1]
        Ȳ[:,1:max_K₊,npind,t] = Ȳ[:,1:max_K₊,originating_particle_id,t-1]
        Ȳ[:,class_id_y,npind,t] =  old_mean + (1/new_count)*(y - old_mean)
        SS[:,:,1:max_K₊,npind,t]= SS[:,:,1:max_K₊,originating_particle_id,t-1]
        SS[:,:,class_id_y,npind,t] = SS[:,:,class_id_y,
                                            originating_particle_id,t-1] + yyᵀ
        (μ,Σ,ν)=update_posterior(new_count,Ȳ[:,class_id_y,npind,t],
                                            SS[:,:,class_id_y,npind,t])
        logdetΣ[1:max_K₊,npind,t] = logdetΣ[1:max_K₊,
                                            originating_particle_id,t-1]
        logdetΣ[class_id_y,npind,t] = logdet(Σ)
        invΣ[:,:,1:max_K₊,npind,t] = invΣ[:,:,1:max_K₊,
                                            originating_particle_id,t-1]
        invΣ[:,:,class_id_y,npind,t]=inv(Σ)
    end
    cp = np
end
```

Figure 13: Particle Filter: Main Body 2

```
Ztrue=int16(dataset[4,:])
label_weights=Dict()
for t = 1:T
label_index=[find(Z[:,t,2].==i) for i=1:maximum(Z[:,t,2])]
label_weights["$t"]=[sum(W[:,t][label_index[i]]) for i=1:maximum(Z[:,t,2])]
end
Zpred=zeros(Int16,T)
for t = 1:T
  Zpred[t] =  findfirst(label_weights["$t"].==maximum(label_weights["$t"]))
end
##########################################################################
p1=Dict();ind={[1,2],[1,3],[2,3]} # index of dataset
for i=1:3
    p1["$i"]=plot(x=Y[ind[i][1],:],y=Y[ind[i][2],:],
                  color=Ztrue,
                  Guide.xlabel("Dimension $(ind[i][1])"),
                  Guide.ylabel("Dimension $(ind[i][2])"),
                  Guide.title("True: View $i"),
                  Guide.colorkey("Cluster"),
                  Scale.color_discrete(),
                  Theme(default_point_size=.5mm))
end
p2=Dict()
for i=1:3
    p2["$i"]=plot(x=Y[ind[i][1],:],y=Y[ind[i][2],:],
                  color=Zpred,
                  Guide.xlabel("Dimension $(ind[i][1])"),
                  Guide.ylabel("Dimension $(ind[i][2])"),
                  Guide.title("Predicted: View $i"),
                  Guide.colorkey("Cluster"),
                  Scale.color_discrete(),
                  Theme(default_point_size=.5mm))
end
draw(PDF("Estimation2.pdf", 6inch, 9inch),#write to file
        hstack(vstack(p1["1"],p1["2"],p1["3"]),vstack(p2["1"],p2["2"],p2["3"])))
##########################################
C = confusmat(9, Ztrue, Zpred)
rocnum=roc(Ztrue, Zpred)
true_positive_rate(rocnum)
precision(rocnum)
```

Figure 14: Particle Filter: Plotting

References

Carpenter, J., Clifford, P., & Fearnhead, P. (1999, feb). Improved particle filter for nonlinear problems. *IEE Proceedings - Radar, Sonar and Navigation*, *146*(1), 2.

Fearnhead, P. (2004). Particle filters for mixture models with an unknown number of components. *Statistics and Computing*, *14*(1), 11–21.

Wood, F., & Black, M. J. (2008, aug). A nonparametric Bayesian alternative to spike sorting. *Journal of neuroscience methods*, *173*(1), 1–12.