Model-based Clustering for Neural Populations

The goal for this research is to do model-based clustering for neural populations, by making use of features for each counting process observation.

1 Notations

Assume we can observe neural activities for N neurons, with counting observation up to T steps. Therefore, the observation is a *N-by-T* matrix, $\mathbf{Y} \in \mathbb{Z}_{\geq 0}^{N \times T}$,with each row represents the recording from single neuron. Denote the recording for neuron i as $\mathbf{y}_i = (y_{i1}, \ldots, y_{\mathrm{iT}})'$, $i = 1, \ldots N.$, with the cluster index for neuron i as $z_i \in \{1, \ldots\}$. The number of neurons in cluster j is $n_j = \sum_{i=1}^{N} I(z_i = j)$, and $\sum_{j=1,2,\ldots} n_j = N$. The proportion/ probability in cluster z_i is ρ_{z_i}.

2 Clustering Wrapper

The model-based clustering problem can be transformed into fitting the mixture model (MM). The likelihood for each cluster depends on how we model the counting observation, but fitting strategies for MM are the same for all models. Here, I choose to fit the MM by Gibbs sampler. Depending on whether the number of cluster is finite or not, there are two versions: finite mixture model (FMM) and Dirichlet process mixture model (DPMM).

2.1 Finite Mixture Model

Assume the number of cluster is J. The full likelihood for these N neurons is

$$L = \prod_{i=1}^{N} \rho_{z_i} f\left(\mathbf{y}_i | \mathbf{\Theta}_{z_i}\right) = \prod_{j=1}^{J} \rho_j^{n_j} \left[\prod_{i:z_i=j} f\left(\mathbf{y}_i | \mathbf{\Theta}_j\right) \right]$$

, where $\mathbf{\Theta}_j$ contains all parameters in cluster j defined by the specific model. Therefore, the parameters need to update are:

(1) Cluster indicator: $\{z_i\}_{i=1}^{N}$

(2) Cluster proportion: $\rho = (\rho_1, \ldots \rho_J)'$

(3) Model parameters: $\mathbf{\Theta}_j$

The (conditional) priors for clustering-related parameters:

(1) Cluster indicator $\{z_i\}_{i=1}^N$: $P(z_i = j) = \rho_j$

(2) Cluster proportion $\rho = (\rho_1, \dots \rho_J)'$:

$$\rho \sim Dir(\delta_1, \dots \delta_J)$$

, where $\delta_1 = \ldots = \delta_J = 1$

So, the MCMC(Gibbs sampler) iteration for FMM is:

(1) Update $\{z_i\}_{i=1}^N$:

$$P\left(z_i = j \middle| \mathbf{y}_i, \{\mathbf{\Theta}_j\}_{j=1}^J\right) \propto \rho_j f(\mathbf{y}_i|\mathbf{\Theta}_j)$$

(2) Update $\rho = (\rho_1, \dots \rho_J)^{'}$:

$$\rho| \ \{\mathbf{y}_i\}_{i=1}^N, \ \{z_i\}_{i=1}^N, \{\mathbf{\Theta}_j\}_{j=1}^J \sim Dir(\delta_1 + n_1, \dots \delta_J + n_J)$$

(3) Update $\mathbf{\Theta}_j$: this is defined by the specific model. When there's no $z_i = j$, just sample $\mathbf{\Theta}_j$ from priors or by other observation-independent ways.

2.2 Dirichlet Process Mixture Model

Since calculation of posterior predictive distribution can be hard or even impossible for complicated models, instead of using the popular CRP representation of DP (Neal, 2020), I choose to use the slice sampler (Walker, 2007).

Use the "stick-breaking" construction for cluster proportion, i.e.

$$\rho_1 = \eta_1$$

$$\rho_j = (1 - \eta_1) \cdot \ldots \cdot (1 - \eta_{j-1}) \eta_j$$

$$\eta_j \sim Beta(1, \alpha)$$

In the slice sampler for DPMM, the parameters need to update are:

(1) "stick-breaking" elements: η_j

(2) Augment latent variable: $\{u_i\}_{i=1}^N$

(3) Model parameters: $\mathbf{\Theta}_j$

(4) Cluster indicator: $\{z_i\}_{i=1}^N$

So, the MCMC(Gibbs sampler) iteration for DPMM is:

(1) update η_j, for $j = 1, \ldots, z^* = max\left\{z_i\right\}_{i=1}^N$ as

$$\eta_j | \left\{z_i\right\}_{i=1}^N, \ldots \sim Beta(n_j + 1,\ N - \sum_{l=1}^{j} n_l + \alpha)$$

(2) update $\left\{u_i\right\}_{i=1}^N$:

$$u_i | \rho, \ldots \sim U(0,\ \rho_{z_i})$$

(3) update η_j, for $j = z^* + 1, \ldots,\ s^*$. s^* is the smallest value, s.t. $\sum_{j=1}^{s^*} \rho_j > 1 - \min\left\{u_1, \ldots, u_N\right\}$

$$\eta_j \sim Beta(1, \alpha)$$

(4) Update $\mathbf{\Theta}_j$: this is defined by the specific model. When there's no $z_i = j$, just sample $\mathbf{\Theta}_j$ from priors or by other observation-independent ways.

(5) Update $\left\{z_i\right\}_{i=1}^N$

$$P\left(z_i = j \middle| \mathbf{y}_i, \left\{\mathbf{\Theta}_j\right\}, \rho, \left\{u_i\right\}_{i=1}^N\right) = \frac{f\left(\mathbf{y}_i | \mathbf{\Theta}_j\right)}{\sum_{j:\rho_j > u_i} f\left(\mathbf{y}_i | \mathbf{\Theta}_j\right)}$$

3 linear Dynamical System Model

Here, I model the observations by a linear dynamical system (LDS) model.

LDS models the multi-dimensional time series using a lower dimensional latent representation of the system, which evolves over time according to linear dynamics. By specifying the linear dynamics and process noise covariance, we can also handle the interactions between different neural populations (clusters).

3.1 Model Details

Denote the latent vector in cluster j as $\mathbf{x}_t^{(j)} \in \mathbb{R}^{p_j}$. For simplicity, assume all $p_j = p$. Each observation follows a Poisson distribution:

$$\log \lambda_{\text{it}} = d_i + \mathbf{c'}_i \mathbf{x}_t^{(z_i)}$$

$$y_{\text{it}} \sim Poisson(\lambda_{\text{it}})$$

, where $\mathbf{c}_i \in \mathbb{R}^p$ and $\mathbf{x}_t^{(z_i)} \in \mathbb{R}^p$.

Although the loading (d_i and $\mathbf{c}_i$) is determined by neuron index i, the distribution is also cluster-dependent. That is,

$$(d_i, \mathbf{c}_i')' \sim N(\boldsymbol{\mu}_{\text{dc}}^{(z_i)}, \mathbf{\Sigma}_{\text{dc}}^{(z_i)})$$

By doing this, the loading within each cluster is also correlated.

Denote all latent states as $\mathbf{x}_t = \left(\mathbf{x}_t'^{(1)}, \mathbf{x}_t'^{(2)}, \dots\right)'$ and they evolve linearly with a Gaussian noise:

$$\mathbf{x}_1 \sim N(\mathbf{x}_0, \mathbf{Q}_0)$$

$$\mathbf{x}_{t+1}|\mathbf{x}_t \sim N(\mathbf{A}\mathbf{x}_t + \mathbf{b}, \mathbf{Q})$$

For simplicity, assume $\mathbf{Q}_0$ is known (e.g. $\mathbf{Q}_0 = \mathbf{I} \times 10^{-2}$).

If we assume process noise covariance is block diagonal (Joshua et al., 2020), we can write things as:

$$\mathbf{x}_{t+1}^{(j)}|\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(2)}, \dots \sim N(\sum_{l=1,\dots} \mathbf{A}_{j\leftarrow l}\mathbf{x}_t^{(l)} + \mathbf{b}_j, \mathbf{Q}^{(j)})$$

Notice $\{\mathbf{A}_{j\leftarrow l}\}$ forms the full transition matrix as:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{1\leftarrow 1} & \mathbf{A}_{1\leftarrow 2} & \dots \\ \mathbf{A}_{2\leftarrow 1} & \mathbf{A}_{2\leftarrow 2} & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Denote the j^{th} row block of $\mathbf{A}$ as $\mathbf{A}_j = \begin{pmatrix} \mathbf{A}_{j\leftarrow 1} & \mathbf{A}_{j\leftarrow 2} & \dots \end{pmatrix}$. Then, $\sum_{l=1,\dots} \mathbf{A}_{j\leftarrow l}\mathbf{x}_t^{(l)} + \mathbf{b}_j = \mathbf{A}_j\mathbf{x}_t + \mathbf{b}_j$.

If we further let $\mathbf{Q}$ be diagonal, with the k^{th} row of $\mathbf{x}_t$, $\mathbf{A}$, $\mathbf{b}$ denoted as x_{kt}, $\mathbf{a}_k$, b_k. The corresponding process noise variance is q_k. Then:

$$x_{k,t+1}|x_{\text{kt}} \sim N\left(\mathbf{a}_k'\mathbf{x}_t + b_k, q_k\right)$$

For completeness, Gibbs samplers for all three versions (no constraint, block diagonal and diagonal $\mathbf{Q}$) are shown below.

Since the progress noise is independent at each step, $f\left(\mathbf{y}_i|\mathbf{\Theta}_j\right) = \prod_{t=1}^{T} P(y_{\text{it}}|\mathbf{\Theta}_j)$, where $P(\cdot)$ is the Poisson density and $\mathbf{\Theta}_j$ contains all parameters in cluster j.

3.2 Conditional Priors for Parameters

The parameters need to estimate:

(1) Latent vectors: $\{\mathbf{x}_t\}_{t=1}^{T}$

(2) Initials: $\mathbf{x}_0$

(3) Linear mapping (loading) for latent vectors: $\{d_i\}_{i=1}^{N}$ and $\{\mathbf{c}_i\}_{i=1}^{N}$

(4) Mean and covariance for loading in each cluster: $\left\{\boldsymbol{\mu}_{\text{dc}}^{(j)}\right\}_j$ and $\left\{\mathbf{\Sigma}_{\text{dc}}^{(j)}\right\}_j$

(5) Linear dynamics for latent vectors: $\mathbf{A}$ and $\mathbf{b}$

(6) Process noise: $\mathbf{Q}$

The conditional priors for these parameters:

(1) Latent vectors $\{\mathbf{x}_t\}_{t=1}^T$: the conditional prior is defined by

$$\mathbf{x}_1 \sim N(\mathbf{x}_0, \mathbf{Q}_0)$$

$$\mathbf{x}_{t+1}|\mathbf{x}_t \sim N(\mathbf{A}\mathbf{x}_t + \mathbf{b}, \mathbf{Q})$$

(2) Initials $\mathbf{x}_0$: assume there are J clusters,

$$\mathbf{x}_0 \sim N(\boldsymbol{\mu}_{\mathbf{x}_{00}},\ \boldsymbol{\Sigma}_{\mathbf{x}_{00}})$$

, where $\boldsymbol{\mu}_{\mathbf{x}_{00}} = \mathbf{0}_{\mathrm{Jp}}$ and $\boldsymbol{\Sigma}_{\mathbf{x}_{00}} = \mathbf{I}_{\mathrm{Jp}}$

(3) Linear mapping (loading) for latent vectors

I just ignore that it is just multivariate GLM problem. Do it later...

(4) Mean and covariance for loading in each cluster

MV-GLM...

(5) Linear dynamics for latent vectors $\mathbf{A}$ and $\mathbf{b}$: if the number of cluster is J

MV-GLM...

(6) Process noise $\mathbf{Q}$: if the number of cluster is J

MV-GLM...

3.3 MCMC (Gibbs Sampler)

3.3.1 Update $\{\mathbf{x}_t\}_{t=1}^T$

Use Laplace approximation and make use of the block tri-diagonal Hessian.

Denote t^{th} column of mean firing rate and observation as $\widetilde{\boldsymbol{\lambda}}_t = (\lambda_{1t}, \ldots, \lambda_{\mathrm{Nt}})'$ and $\widetilde{\mathbf{y}}_t = (y_{1t}, \ldots, y_{\mathrm{Nt}})'$. The linear mapping matrix for all observations is $\mathbf{C}$, such that $\log \widetilde{\boldsymbol{\lambda}}_t = \mathbf{d}+\mathbf{C}\mathbf{x}_t$. Let $\mathbf{x} = (\mathbf{x}'_1, \ldots, \mathbf{x}'_T)'$and $f(\mathbf{x}) = \log P(\mathbf{x}|\{\mathbf{y}_i\}_{i=1}^N, \mathbf{C}, \mathbf{Q}_0, \mathbf{A}, \mathbf{b}, \mathbf{Q}, \ldots)$

The first and second derivative with respect to $\mathbf{x}$, for $t = 2, \ldots, T-1$:

$$\begin{aligned}
\frac{\partial f}{\partial \mathbf{x}_1} &= \mathbf{C}'\left(\widetilde{\mathbf{y}}_1 - \widetilde{\boldsymbol{\lambda}}_1\right) - \mathbf{Q}_0^{-1}\left(\mathbf{x}_1 - \mathbf{x}_0\right) + \mathbf{A}'\mathbf{Q}^{-1}(\mathbf{x}_2 - \mathbf{A}\mathbf{x}_1 - \mathbf{b}) \\
\frac{\partial f}{\partial \mathbf{x}_t} &= \mathbf{C}'\left(\widetilde{\mathbf{y}}_t - \widetilde{\boldsymbol{\lambda}}_t\right) - \mathbf{Q}^{-1}\left(\mathbf{x}_t - \mathbf{A}\mathbf{x}_{t-1} - \mathbf{b}\right) + \mathbf{A}'\mathbf{Q}^{-1}(\mathbf{x}_{t+1} - \mathbf{A}\mathbf{x}_t - \mathbf{b}) \\
\frac{\partial f}{\partial \mathbf{x}_T} &= \mathbf{C}'\left(\widetilde{\mathbf{y}}_T - \widetilde{\boldsymbol{\lambda}}_T\right) - \mathbf{Q}^{-1}\left(\mathbf{x}_T - \mathbf{A}\mathbf{x}_{T-1} - \mathbf{b}\right) \\
\frac{\partial^2 f}{\partial \mathbf{x}_1 \partial \mathbf{x}_1'} &= -\mathbf{C}'\mathrm{Diag}\left(\widetilde{\boldsymbol{\lambda}}_1\right)\mathbf{C} - \mathbf{Q}_0^{-1} - \mathbf{A}'\mathbf{Q}^{-1}\mathbf{A} \\
\frac{\partial^2 f}{\partial \mathbf{x}_t \partial \mathbf{x}_t'} &= -\mathbf{C}'\mathrm{Diag}\left(\widetilde{\boldsymbol{\lambda}}_t\right)\mathbf{C} - \mathbf{Q}^{-1} - \mathbf{A}'\mathbf{Q}^{-1}\mathbf{A} \\
\frac{\partial^2 f}{\partial \mathbf{x}_T \partial \mathbf{x}_T'} &= -\mathbf{C}'\mathrm{Diag}\left(\widetilde{\boldsymbol{\lambda}}_T\right)\mathbf{C} - \mathbf{Q}^{-1} \\
\frac{\partial^2 f}{\partial \mathbf{x}_1 \partial \mathbf{x}_2'} &= \frac{\partial^2 f}{\partial \mathbf{x}_t \partial \mathbf{x}_{t+1}'} = \mathbf{A}'\mathbf{Q}^{-1} \qquad \frac{\partial^2 f}{\partial \mathbf{x}_t \partial \mathbf{x}_{t-1}'} = \mathbf{Q}^{-1}\mathbf{A}
\end{aligned}$$

So, the gradient is:

$$\nabla = \frac{\partial f}{\partial \mathbf{x}} = \left(\left(\frac{\partial f}{\partial \mathbf{x}_1}\right)', \ \ldots, \left(\frac{\partial f}{\partial \mathbf{x}_T}\right)'\right)'$$

And the block tri-diagonal Hessian:

$$H = \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}'} = \begin{pmatrix}
\frac{\partial^2 f}{\partial \mathbf{x}_1 \partial \mathbf{x}_1'} & \mathbf{A}'\mathbf{Q}^{-1} & 0 & \cdots & 0 \\
\mathbf{Q}^{-1}\mathbf{A} & \frac{\partial^2 f}{\partial \mathbf{x}_2 \partial \mathbf{x}_2'} & \mathbf{A}'\mathbf{Q}^{-1} & \cdots & \vdots \\
0 & \mathbf{Q}^{-1}\mathbf{A} & \frac{\partial^2 f}{\partial \mathbf{x}_3 \partial \mathbf{x}_3'} & \cdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & \cdots & \cdots & \frac{\partial^2 f}{\partial \mathbf{x}_T \partial \mathbf{x}_T'}
\end{pmatrix}$$

Use Newton-Raphson to find $\boldsymbol{\mu}_{\mathbf{x}} = \mathrm{argmax}_{\mathbf{x}}\left(f\left(\mathbf{x}\right)\right)$ and $\boldsymbol{\Sigma}_{\mathbf{x}} = -\left[\frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}'} \ |_{\mathbf{X}=\boldsymbol{\mu}_{\mathbf{X}}}\right]^{-\mathbf{1}}$, such that $\left(P(\mathbf{x}|\left\{\mathbf{y}_i\right\}_{i=1}^N, \ldots\right) \approx N\left(\boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{x}}\right)$. When using Newton-Raphson (NR), $H \backslash \nabla$ in MATLAB will make use of block tri-diagonal structure automatically.

However, NR is not robust to bad initials. At the first few iterations, simply using fitting from previous step may lead to infinite Hessian. When the initial from previous step fails, use the approximation at recursive priors, , i.e. the adaptive smoother estimates, as the initial. The adaptive smoother estimates

are from backward RTS smoother from adaptive filter, and the details about Poisson adaptive filter can be found in Eden et al., 2004.

To sample efficiently and make best use of sparse covariance, use Cholesky decomposition of $\mathbf{\Sigma}_{\mathbf{x}}^{-1} = \mathbf{R}\mathbf{R}'$: sample $\mathbf{Z} \sim N(\mathbf{R}'\mu_{\mathbf{x}}, \mathbf{I})$, then $\mathbf{x} = (\mathbf{R}')^{-1}\mathbf{Z} \sim N(\boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}})$.

3.3.2 Update $\mathbf{x}_0$

$$P(\mathbf{x}_0|\mathbf{x}_1,\ \mathbf{Q}_0 \ldots) \propto N(\mathbf{x}_1|\mathbf{x}_0,\ \mathbf{Q}_0)N(\mathbf{x}_0|\boldsymbol{\mu}_{\mathbf{x}_{00}},\ \mathbf{\Sigma}_{\mathbf{x}_{00}})$$

By conjugacy, $\mathbf{x}_0|\mathbf{x}_1,\ \mathbf{Q}_0 \ldots \sim N(\boldsymbol{\mu}_{\mathbf{x}_0},\ \mathbf{\Sigma}_{\mathbf{x}_0})$

$$\begin{aligned}\mathbf{\Sigma}_{\mathbf{x}_0} &= \left[\mathbf{\Sigma}_{\mathbf{x}_{00}}^{-1} + \mathbf{Q}_0^{-1}\right]^{-1} \\ \boldsymbol{\mu}_{\mathbf{x}_0} &= \mathbf{\Sigma}_{\mathbf{x}_0}\left(\mathbf{\Sigma}_{\mathbf{x}_{00}}^{-1}\boldsymbol{\mu}_{\mathbf{x}_{00}} + \mathbf{Q}_0^{-1}\mathbf{x}_1\right)\end{aligned}$$

3.3.3 Update $\{d_i\}_{i=1}^N$ and $\{\mathbf{c}_i\}_{i=1}^N$

To update efficiently, use Laplace approximation again. Denote

MV-GLM...

3.3.4 Update $\left\{\mu_{\mathbf{dc}}^{(j)}\right\}_j$ and $\left\{\mathbf{\Sigma}_{\mathbf{dc}}^{(j)}\right\}_j$

MV-GLM...

3.3.5 Update A and b

MV-GLM...

3.3.6 Update Q

MV-GLM...

4 Simulations

Old results

Currently, the latent x_t estimation is not accurate enough. This may come from bad simulation/ slow convergence or other sources. See details in section 5

There are two set of simulation examples. The first example is generated from LDS model directly, while in the second example the latents are generated directly without explicit specifying linear dynamics. In all of the following results, the covariance of process noise $\mathbf{Q}$ is assumed to be block diagonal. For reference, see the code for non-constrained and diagonal $\mathbf{Q}$. But these two legacy versions assuming the loading is independent on cluster assignments. If we need to use any of one of them, make sure to replace the updates for loading to be cluster-related. The fitting results for all these three are similar, because the underlying true $\mathbf{Q}$ is diagonal.

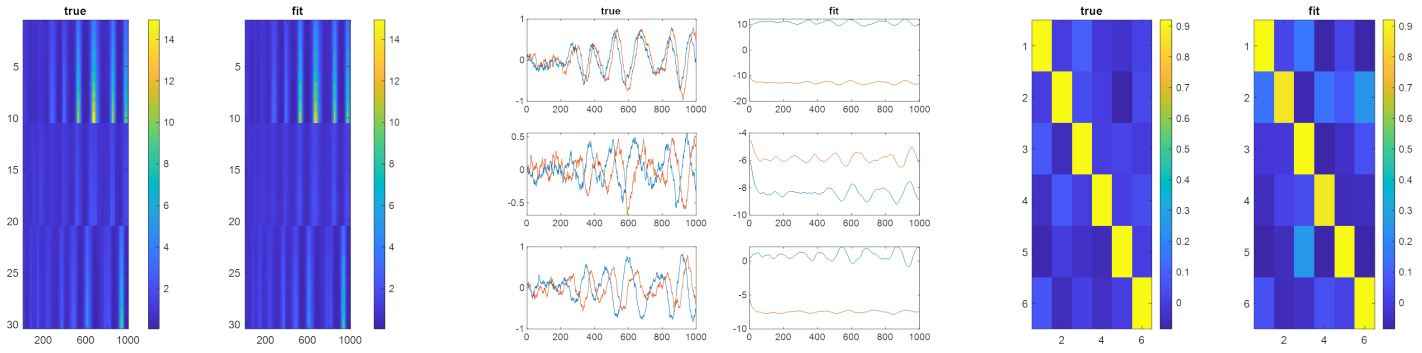
Before doing the clustering, I first assume the cluster labels are known to see the fitting performance. All the fitting results with labels are shown in means from iteration 50 to 100 (**The iteration number is not enough to achieve stationary distribution**).

4.1 Simulation 1: Generate from LDS Directly

In this simulation, there are 3 clusters with 10 neurons in each cluster. The dimension of latents in each cluster is 2. In the linear dynamics, the bias term $\mathbf{b}$ is zero. There's no within-population interaction but has some weak between-population interactions. In other words, the linear dynamics matrix $\mathbf{A}$ is roughly diagonal. The details of simulation can be found in the first section of the simulation 1.

4.1.1 Labeled Data: No Clustering

The code for fitting is here, and some results are in Figure 1.



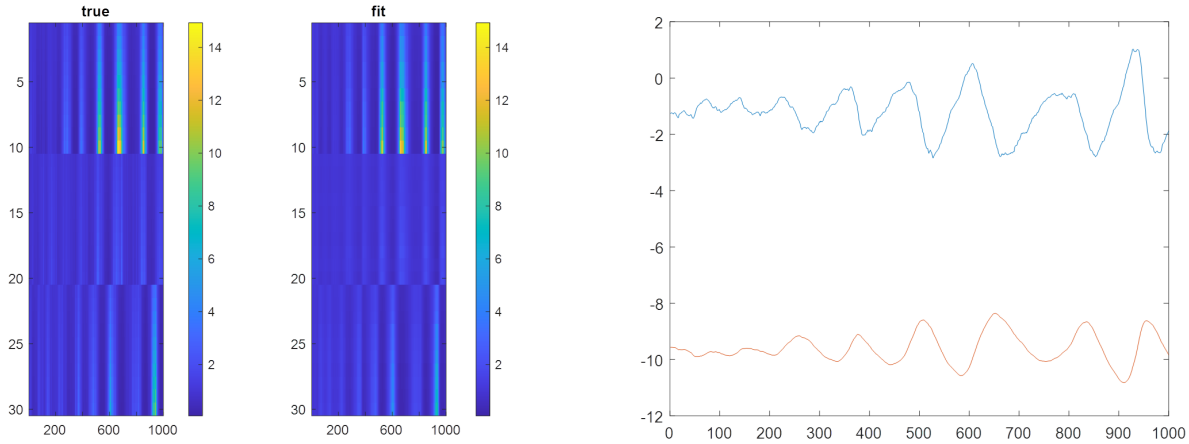
(a) overall mean firing rate (b) latents (c) dynamics $\mathbf{A}$

Figure 1: LDS sample with labels

The fitting for overall mean firing rate is perfect and the dynamics pattern is mostly recovered. The fitting of latents is OK (**may not be accurate enough**

) up to scale. In other words, the model captures the oscillating pattern of latents.

But the latent patterns of these three clusters seems quite similar. That means the differences in the mean firing rates may be purely explained by the loading, i.e. $\mathbf{d}$ and $\mathbf{C}$. To see that, I set the number of cluster to be 1 and the results are shown in Figure 2.



(a) overall mean firing rate

(b) latents

Figure 2: LDS sample with labels, one cluster

Again, the fitting for overall mean firing rate is perfect. This is why it's important to make the loading, $\mathbf{d}$ and $\mathbf{C}$, also be cluster-dependent (loading within clusters are correlated). If the loading only depends on neuron index and will not change for different clustering assignments, it's impossible to do clustering (at least in this case), since the loading is enough to capture all the patterns.

4.1.2 Unlabeled Data: Clustering

To give the full path of clustering, I show results in GIFs. Click to see the code for FMM and DPMM.

There are three results:

(1) Fit by FMM with $J = 3$, starting from random cluster assignments: GIF result 1

(2) Fit by FMM with $J = 3$, starting from all neurons in single cluster: GIF result 2

(3) Fit by DPMM ($\alpha = 10$), starting from each neuron forms its own cluster: GIF result 3

I didn't show the DPMM starting from single cluster. Since in my current implementation, the cluster generation is not efficient. In other words, the newly generated cluster will usually not be sampled and the algorithm usually gets stuck in 1 or 2 clusters.

This is a big problem... That means if we occasionally combine two clusters into one, we may never correct the mistake. Further, when the data grows, the number of cluster will be hard to grow. Fix that later.

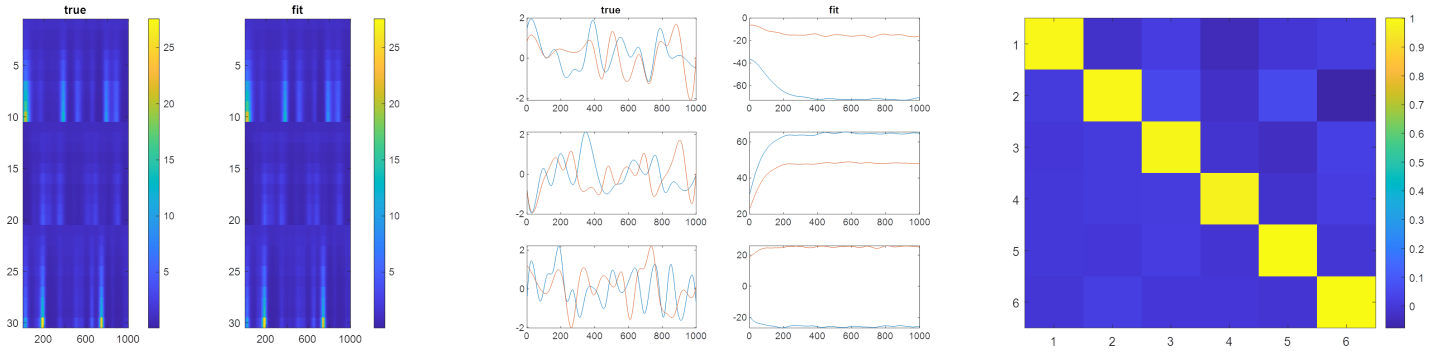
4.2 Simulation 2: Generate Latents, without Specifying Linear Dynamics

In this simulation, there are again 3 clusters with 2 latents in each. Besides set 10 neurons in each cluster, I further simulate 50 neurons in each cluster to see performance in a larger scale. Now, the latents are generated directly without specifying the underlying linear dynamics of latents. However, each latent is generated independently, so the linear dynamics matrix **A** should be roughly diagonal. The details of simulation can be found in the code.

4.2.1 Labeled Data: No Clustering

As in simulation 1, the data is fitted by the true cluster assignment (3 clusters) and forcing all neurons belong to single cluster. The code can be found here.

(1) Smaller dataset (10 neurons each):
The results with true cluster assignment is shown in Figure 3 And results

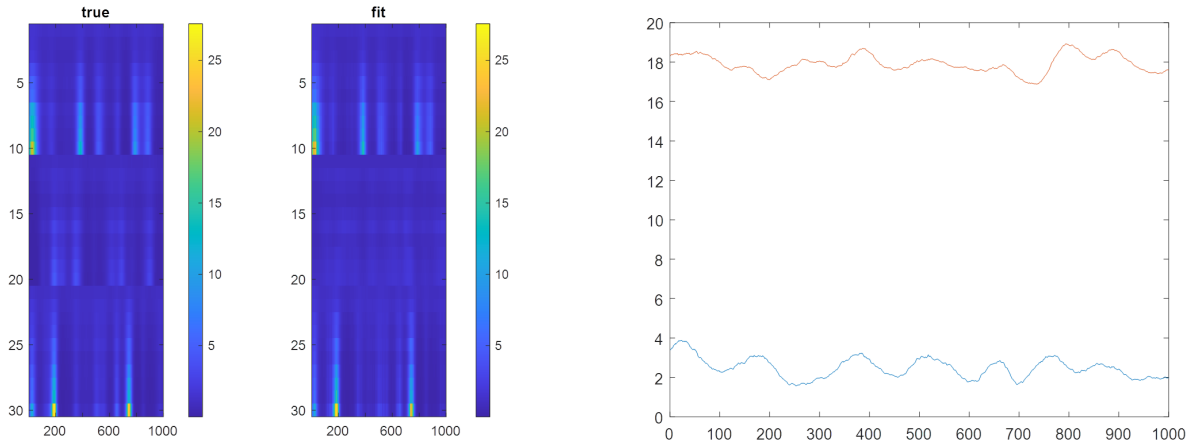


(a) overall mean firing rate (b) latents (c) dynamics **A**

Figure 3: unspecified dynamics with labels, 10 neurons each

by forcing all neurons to be in single cluster (Figure 4)

Again, the fittings for overall mean firing rate are perfect in both cases and the dynamics is roughly diagonal. The oscillating pattern of fitted

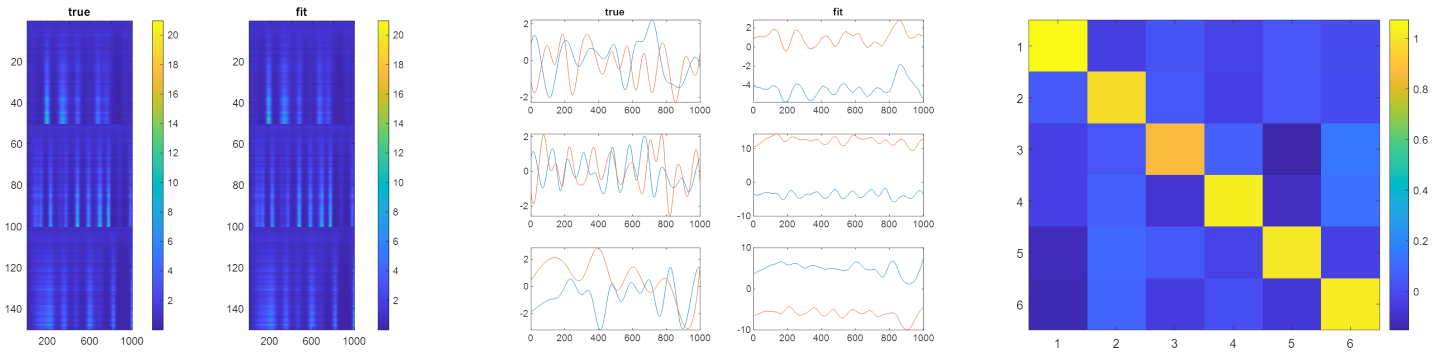


(a) overall mean firing rate (b) latents

Figure 4: unspecified dynamics with labels, one cluster, 10 neurons each

latents with true cluster index is not significant, because of plot scaling. The pattern is easier to see in the second fitting.

(2) Larger dataset (50 neurons each):
The results with true cluster assignment is shown in Figure 5 And results

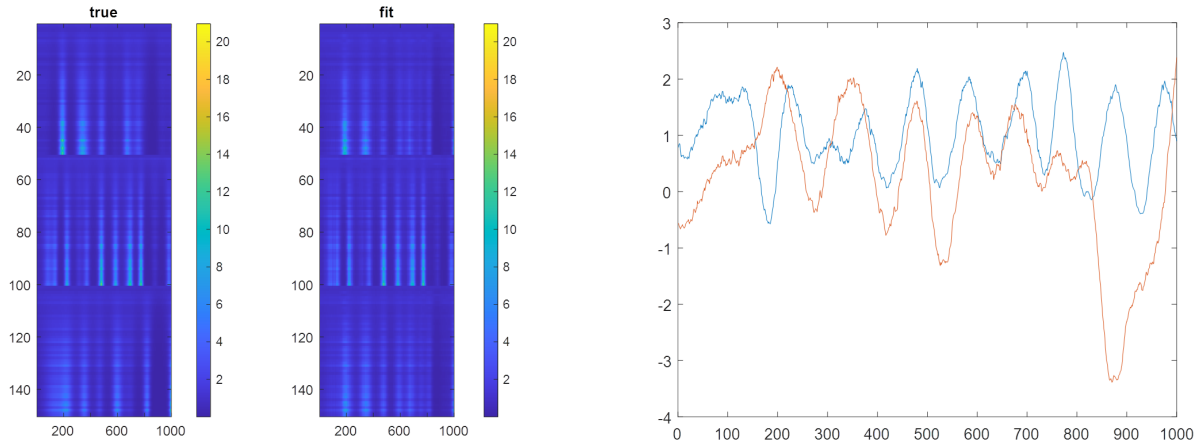


(a) overall mean firing rate (b) latents (c) dynamics **A**

Figure 5: unspecified dynamics with labels, 50 neurons each

by forcing all neurons to be in single cluster (Figure 6)

Now we can see the pattern more clearly.



(a) overall mean firing rate

(b) latents

Figure 6: unspecified dynamics with labels, one cluster, 50 neurons each

4.2.2 Unlabeled Data: Clustering

As previous, in each example, I show results from (1) FMM starting from random cluster assignment ($J = 3$), (2) FMM starting from single cluster and (3) DPMM starting from multiple clusters. The DPMM starting from single cluster need to be fixed later. These are code for FMM and DPMM.

(1) Smaller dataset (10 neurons each):

   (a) Fit by FMM with $J = 3$, start from random cluster assignments: GIF result 4

   (b) Fit by FMM with $J = 3$, start from single cluster: GIF result 5

   (c) Fit by DP ($\alpha = 5$), start from assuming each neuron forms its own cluster: GIF result 6

(2) Larger dataset (50 neurons each):

   (a) Fit by FMM with $J = 3$, start from random cluster assignments: GIF result 7

   (b) Fit by FMM with $J = 3$, start from single cluster: GIF result 8

   (c) Fit by DP ($\alpha = 1$), start from random assignment for 10 clusters: GIF result 9

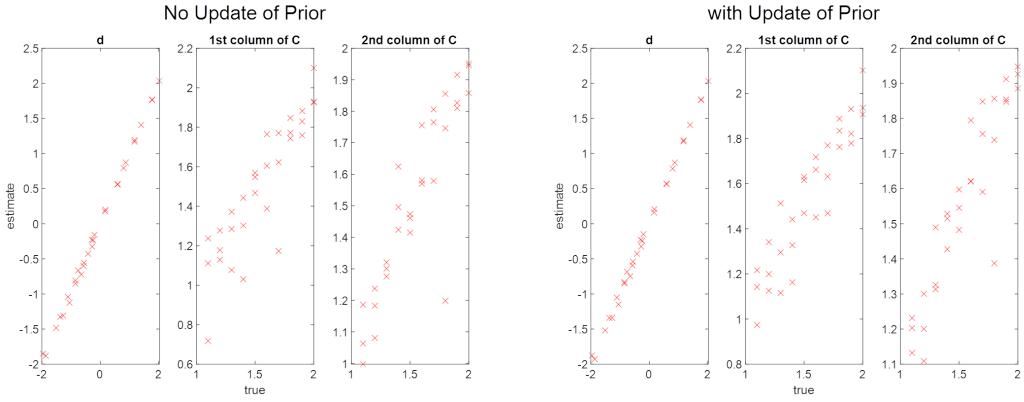
The performance is not bad. The neuron at top of each cluster cannot be clustered correctly by its nature: the signals are too weak to be clearly clustered.

5 PROBLEMS

This section shows what cause the inaccuracy in the latent estimation.

5.1 Estimation of d and C alone

At first, as guessed by Ian, this may be caused by bad estimation of loading (**d** and **C**). So I first turn off all others except for loading (related) parameters and set them to be true values. There are 2 versions: (1) update priors as in subsection 3.3.4 and (2) no update of priors, and just set the prior for each as the standard normal. For the estimation with prior update, the results are mean from iteration 50 to 100 (Figure 7).



(a) (b)

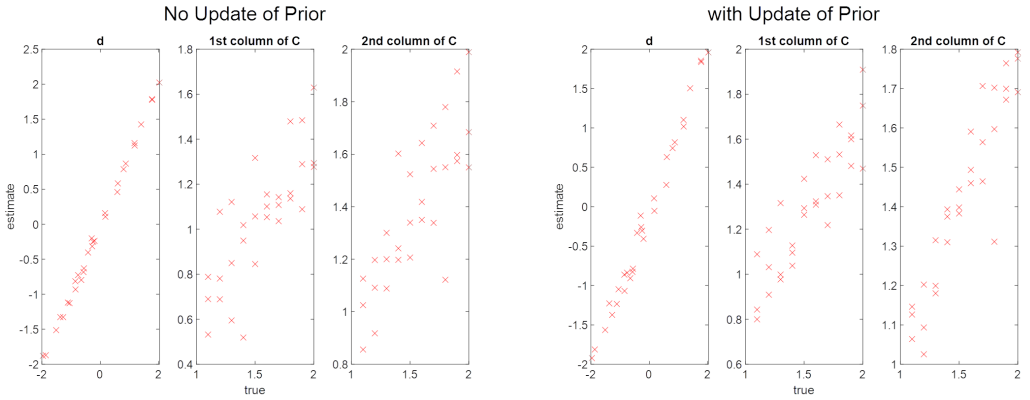
Figure 7: estimation of loading

The mixing of convergence for loading priors (in subsection 3.3.4) are fine. The update of priors don't influence results a lot, but this will help clustering a lot. It seems estimation of loading is fine. Let's see what happens when estimation of latents $\mathbf{x}_t$ is on at the same time.

5.2 Estimation of d, C and $\mathbf{x}_t$

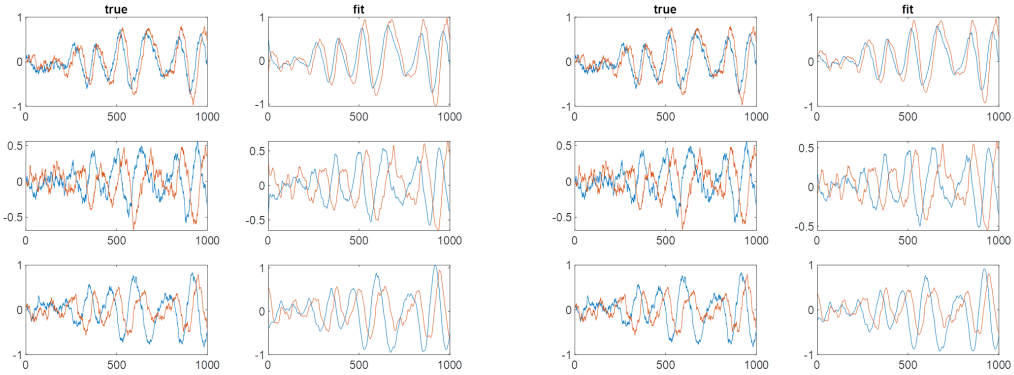
When the estimation of $\mathbf{x}_t$ is on, we need more iterations. The results are mean from iteration 500 to 1000 (Figure 8).

The estimation of loading:



(a) (b)

Figure 8: estimation of loading, with latents on



(a) no update of prior (b) with update of prior

Figure 9: estimation of loading

And the estimation of corresponding latents (Figure 9):

Things are still fine. The remaining parts may influence latent estimations are dynamics ($\mathbf{b}$, $\mathbf{A}$) and prior process noise $\mathbf{Q}$.

I've checked update of (1) dynamics alone, (2) $\mathbf{Q}$ and (3) $\mathbf{Q}$ and $\mathbf{x}_t$. All of them are fine. **So the problem must come from estimations of dynamics and $\mathbf{x}_t$.**

5.3 Estimation of b, A and $\mathbf{x}_t$

I first estimate dynamics by assuming block diagonal $\mathbf{Q}$ (as in subsection 3.3.5). When turning on (1) latents $\mathbf{x}_t$ and (2) dynamics ($\mathbf{b}$ and $\mathbf{A}$), the trace of $\mathbf{b}$ and $\mathbf{A}$ are (Figure 10):

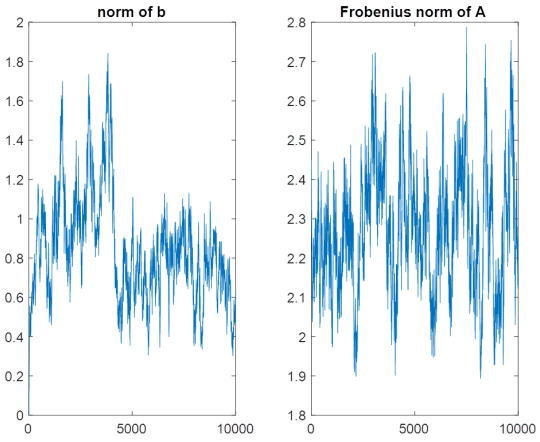
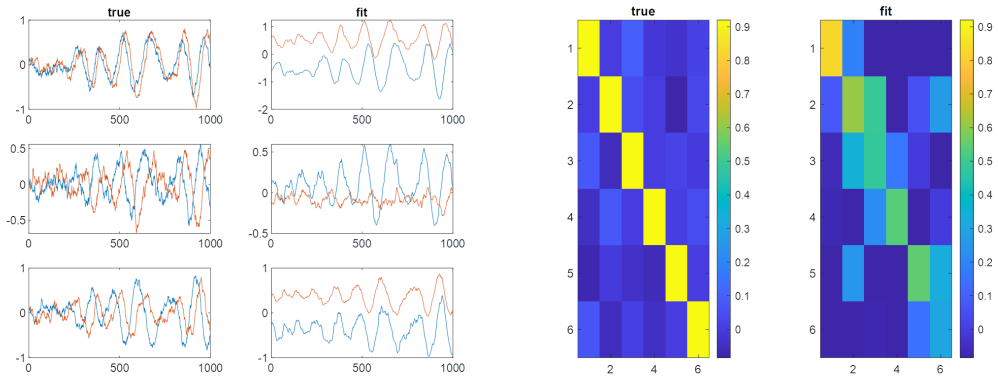


Figure 10: trace of dynamics, block diagonal

The mixing is not good, and we may haven't achieve stationary distribution yet. Anyway, I average results from iteration 5000 to 10000, and the results are shown in Figure 11).



(a) latent (b) dynamics

Figure 11: estimation of latents and dynamics, block diagonal

OK, things are not perfect now. What if I don't assume block diagonal **Q**? For the non-constraint version, the trace plot of **b** and **A** are in Figure 12.

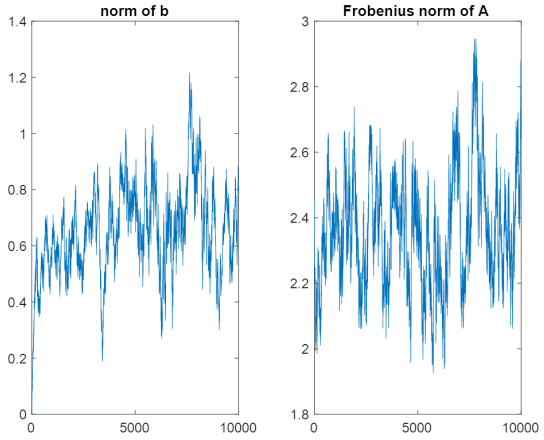
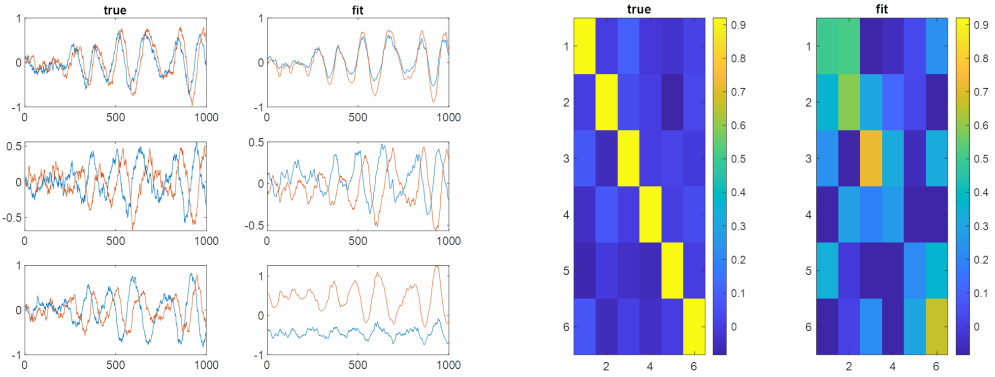


Figure 12: trace of dynamics, no constraint

Again, the average from iteration 5000 to 10000 is in Figure 13).

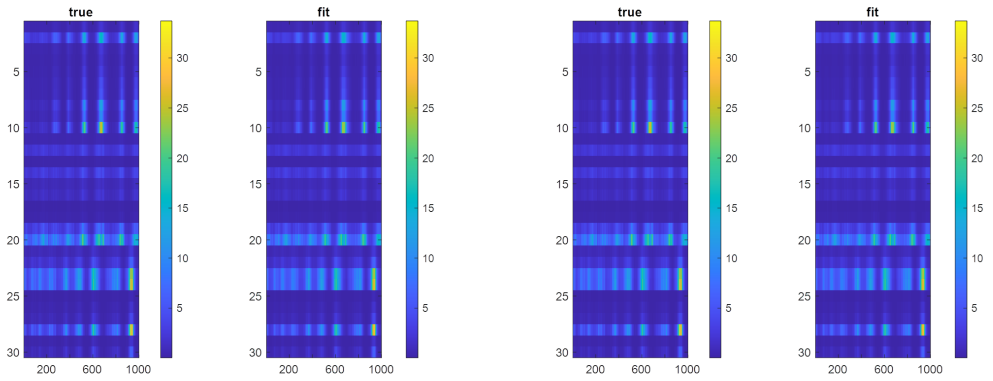


(a) latent (b) dynamics

Figure 13: estimation of latents and dynamics, no constraint

Based on trace plot and latent, things look a bit better. But it's still not good enough. The dynamics is even worse, although the latent is better.

Another Observation: Even with these inaccurate latent, the overall mean firing rate still looks perfect (with true loading **d** and **C**), because of small loading. See Figure 14



(a) block diagonal

(b) no constraint

Figure 14: overall mean firing rate, block diagonal and no constraint

In PLDS, they use EM, so there's no noise. But here I tackle things by MCMC: if the simulation is not very identifiable, then there will be a lot of noise by its nature.

Anyway, the clustering is based on the overall mean firing rate pattern. Although the estimation of latent may not be accurate enough, this should not influence the clustering result.

Maybe the next step is to test the current algorithm by a more identifiable simulation. For example, use a larger loading to make the latent effect become more significant.

5.4 Simulation 1 Revisit

The results above remind me that the 100 iterations is not enough to achieve stationary distribution. I need to check the sampling trace. Here, I rerun the MCMC for simulation 1, but use the no-constraint **Q** version. Since no constraint version is better than block-diagonal one.

First, I show the trace of sampling (Frobenius) norm for **d**, **C**, **b** and **A** as in Figure 15.

It seems we may have not achieved stationary yet. And I show the average from iteration 8000 to 10000 as in Figure 16

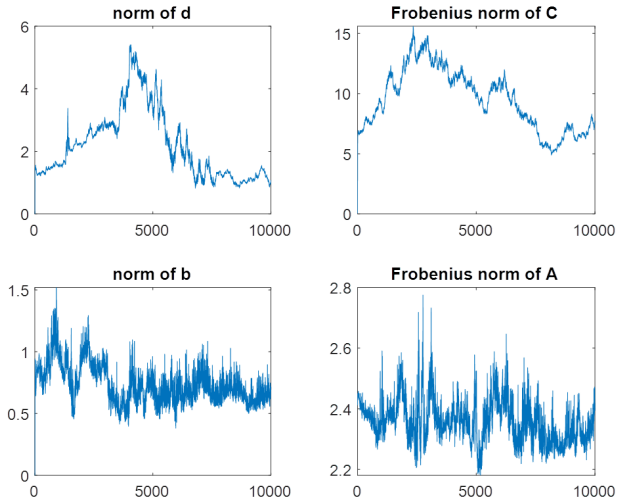
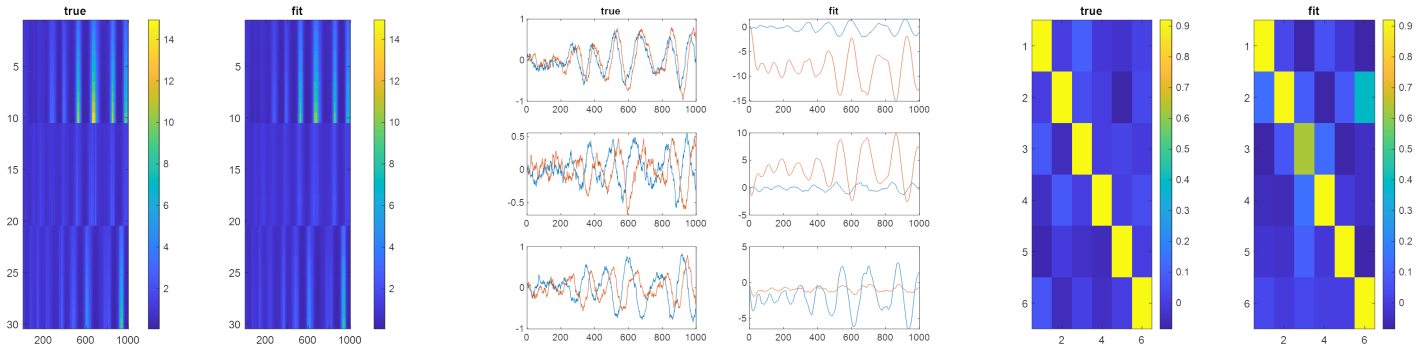


Figure 15: (Frobenius) norm for $\mathbf{d}$, $\mathbf{C}$, $\mathbf{b}$ and $\mathbf{A}$



(a) overall mean firing rate (b) latents (c) dynamics $\mathbf{A}$

Figure 16: LDS sample with labels

Things look much better! That means we need to improve the mixing of $\mathbf{b}$ and $\mathbf{A}$ later. (Also see if the mixing will be better for a more identifiable simulation)

6 TODO

(1) **Do MV-GLM**

(2) Check sampling trace and mixing.

(3) Find a more efficient way to generate new cluster parameters, otherwise the newly generated ones will always be rejected.

(4) improve DPMM. In current brute force implementation, the number of potential clusters can even go beyond number of neurons (N). There are several improvements, e.g. Kalli et al., 2011, Ge et al., 2015 and Hastie et al, 2015. Check them later.

(5) After all of them are resolved, switch to GP version