Distributed MCMC Inference in Dirichlet Process Mixture Models Using Julia

Or Dinari* (Ben-Gurion University, Israel),
Angel Yu* (MIT, USA),
Oren Freifeld (Ben-Gurion University, Israel), and
John W. Fisher III (MIT, USA)

May 14, 2019

*Both these authors contributed equally.

This work was partially supported by NSF award 1622501 and by the Lynn and William Frankel Center for Computer Science at BGU. Or Dinari was partially supported by Trax.

Outline

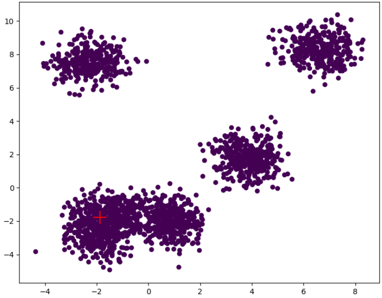
1. Motivation for Bayesian Nonparametric Mixture Models
2. Dirichlet Process Mixture Models (DPMMs)
3. Parallel MCMC Sampler for DPMMs [Chang & Fisher, NIPS '13]
4. Distributed & Parallel MCMC Sampler for DPMM [present work]
5. Results

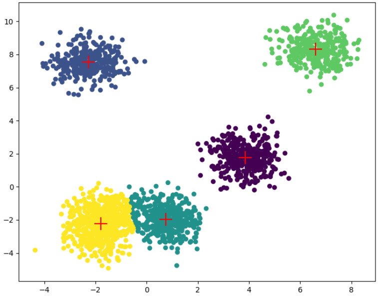
Bayesian Nonparametric Mixture Models

- Mixture models: an important approach to clustering
- Given data, how can we infer its underlying mixture model?

Bayesian Nonparametric Mixture Models

- Mixture models: an important approach to clustering
- Given data, how can we infer its underlying mixture model?





Bayesian Nonparametric Mixture Models

- Problem: how can we infer K, the number of clusters?

Bayesian Nonparametric Mixture Models

- Problem: how can we infer K, the number of clusters?
- A naive solution: try many values of K, and pick the “best”:
  - The elbow method.
  - Gap statistics.
  - Bayesian Information Criterion.

Bayesian Nonparametric Mixture Models

- Problem: how can we infer K, the number of clusters?
- A naive solution: try many values of K, and pick the "best":
  - The elbow method.
  - Gap statistics.
  - Bayesian Information Criterion.
- Problems:

Bayesian Nonparametric Mixture Models

- Problem: how can we infer K, the number of clusters?
- A naive solution: try many values of K, and pick the "best":
  - The elbow method.
  - Gap statistics.
  - Bayesian Information Criterion.
- Problems:
  - Requires performing clustering many times (one for each value of K).
  - For each of value of K: the fitting often gets stuck in a poor local maximum.

Bayesian Nonparametric Mixture Models

- A better solution: infer K together with the other parameters:

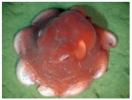
Bayesian Nonparametric Mixture Models

- A better solution: infer K together with the other parameters:

The approach:
Bayesian nonparametric mixture models

Applications

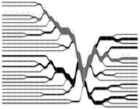
[wikipedia.org]



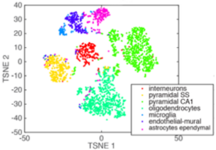
[Ed Bowlby, NOAA]

[Fox et al 2014]

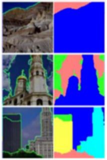
[Lloyd et al 2012; Miller et al 2010]



[Ewens 1972; Hartl, Clark 2003]

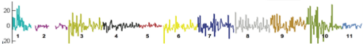


[Prabhakaran, Azizi, Carr, Pe'er 2016]

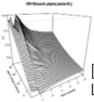


[Sudderth, Jordan 2009]

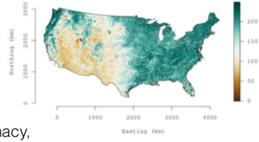
[Saria et al 2010]



[US CDC PHIL; Futoma, Hariharan, Heller 2017]

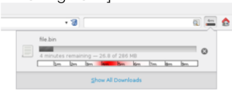


[Gramacy, Lee 2009]



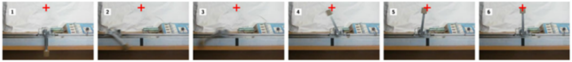
[Datta, Banerjee, Finley, Gelfand 2016]

[Kiefel, Schuler, Hennig 2014]



[Chati, Balakrishnan 2017]

[Deisenroth, Fox, Rasmussen 2015]



Slide: from Tamara Broderick's Tutorial on Bayesian Nonparametrics.

First, We Need Some Basic Concepts

In the next few slides, I will tell you a little about:

First, We Need Some Basic Concepts

In the next few slides, I will tell you a little about:

- Dirichlet Distribution (here, K is still finite and known)

First, We Need Some Basic Concepts

In the next few slides, I will tell you a little about:

- Dirichlet Distribution (here, K is still finite and known)
- Dirichlet Process ("$K = \infty$")

First, We Need Some Basic Concepts

In the next few slides, I will tell you a little about:

- Dirichlet Distribution (here, K is still finite and known)
- Dirichlet Process ("$K = \infty$")
- The Chinese Restaurant Process (one construction of DP)

First, We Need Some Basic Concepts

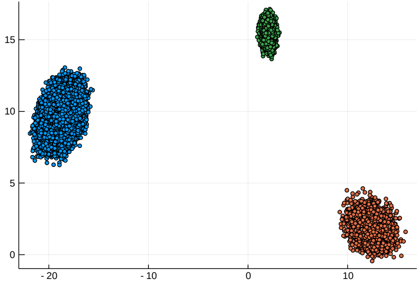
In the next few slides, I will tell you a little about:

- Dirichlet Distribution (here, K is still finite and known)
- Dirichlet Process ("$K = \infty$")
- The Chinese Restaurant Process (one construction of DP)
- Dirichlet Process Mixture Model (DPMM, [Escobar and West, 1995] [2])

Prior on components

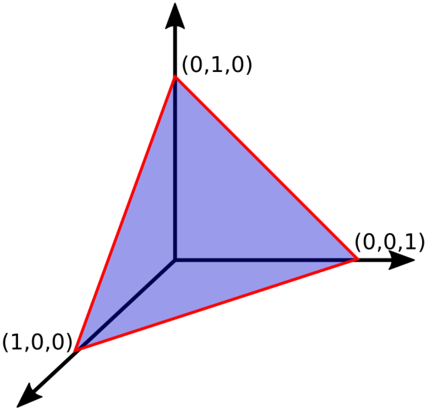
Every componenet has a weight. The weights can be:

- Known.
- Unknown and determinstic.
- Unkown and random.



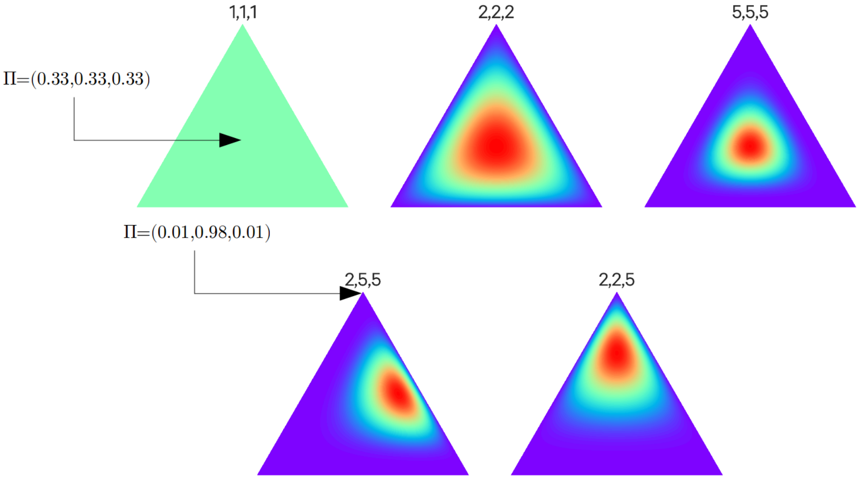
Dirichlet Distribution

$\mathrm{Dir}(\cdot)$ is a distribution over distributions.



Dirichlet Distribution

Examples for $\mathrm{Dir}(\alpha_1, \alpha_2, \alpha_3)$, $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3)$ is a point on the simplex.



Dirichlet Distribution

- $\boldsymbol{\pi} = \mathrm{Cat}(\pi_1, \pi_2, .., \pi_K)$ is a Categorical distribution.

$$\pi_j \in (0, 1), \quad \sum_{j=1}^{K} \pi_j = 1 \tag{1}$$

Dirichlet Distribution

- $\boldsymbol{\pi} = \mathrm{Cat}(\pi_1, \pi_2, .., \pi_K)$ is a Categorical distribution.

$$\pi_j \in (0,1), \quad \sum_{j=1}^{K} \pi_j = 1 \tag{1}$$

- $\boldsymbol{\pi} \sim \mathrm{Dir}(\alpha_1, \alpha_2, ..., \alpha_K)$ is the probability to draw the distribution $\boldsymbol{\pi}$.

Dirichlet Process

- The Dirichlet Process [3] generalizes the Dirichlet Distribution to the case of "$K = \infty$"

$$\mathrm{Dir}(\alpha_1, \alpha_2, \ldots)$$

Dirichlet Process

- The Dirichlet Process [3] generalizes the Dirichlet Distribution to the case of "$K = \infty$"

$$\mathrm{Dir}(\alpha_1, \alpha_2, \ldots)$$

- $G \sim \mathrm{DP}(\alpha, G_0)$:

Dirichlet Process

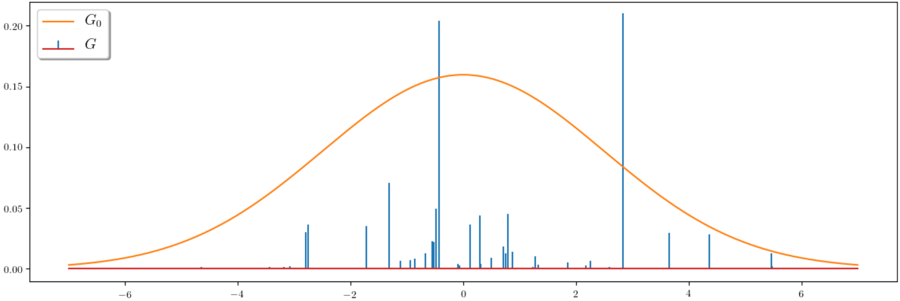
- The Dirichlet Process [3] generalizes the Dirichlet Distribution to the case of "$K = \infty$"

$$\mathrm{Dir}(\alpha_1, \alpha_2, \ldots)$$

- $G \sim \mathrm{DP}(\alpha, G_0)$:
  - G_0 - Base probability measure, either continuous or discrete.
  - α - Concentration parameter.
  - G - Random probability measure, discrete.

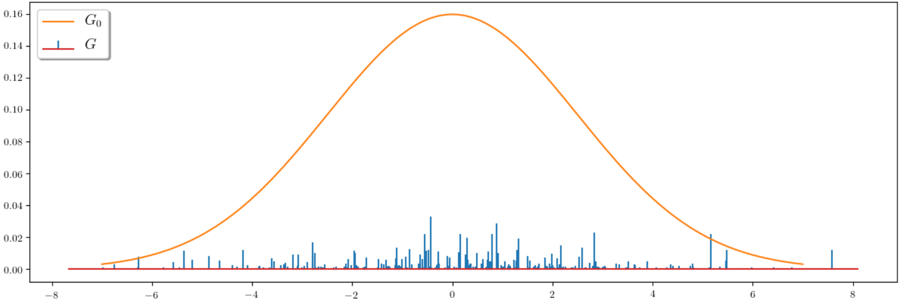
Dirichlet Process - Example

$$G_0 = \mathcal{N}(0, 2.5) \quad G \sim \mathrm{DP}(\alpha = 10, G_0) \tag{2}$$



Dirichlet Process - Example

$$G_0 = \mathcal{N}(0, 2.5) \quad G \sim \mathrm{DP}(\alpha = 100, G_0) \tag{3}$$

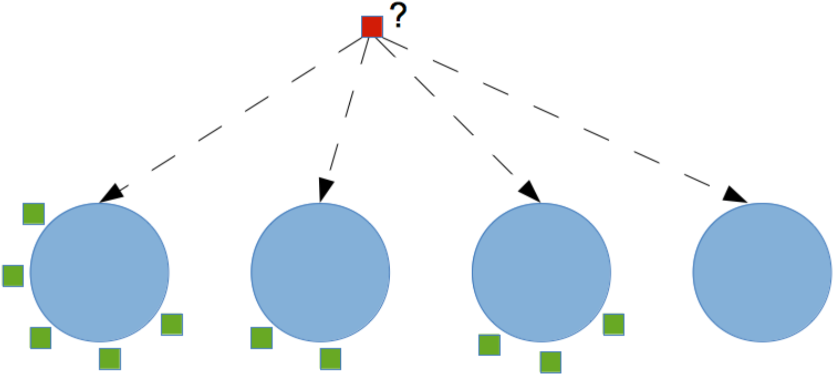


The Chinese Restaurant Process

- An intuitive way to construct a DP

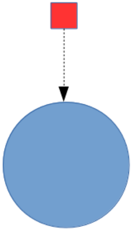
The Chinese Restaurant Process

- An intuitive way to construct a DP
- At a restaurant with an infinite amount of tables, what is the chance for a new customer to sit at an existing table, or to open a new table?



The Chinese Restaurant Process

The first customer sits at the first table with probability 1.



The Chinese Restaurant Process

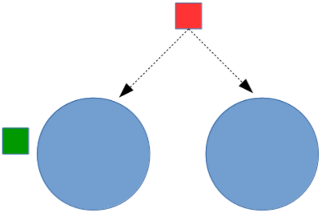
The second customer can either join an existing table with probability

$$p = \frac{|X_1|}{n - 1 + \alpha},$$

or open a new table with probability

$$p = \frac{\alpha}{n - 1 + \alpha}.$$

$|X_1|$ - Customers count at table 1.
α - Concentration parameter.
n - Customers count at the rest.



DP-Mixture Models

- Key application of DP: a prior over the parameters of a mixture model.

DP-Mixture Models

- Key application of DP: a prior over the parameters of a mixture model.
- For a mixture model with $K = \infty$, let:

$$\theta_i | G \sim G \tag{4}$$

$$x_i \sim F(\theta_i) \tag{5}$$

$$G \sim \mathrm{DP}(\alpha, G_0) \tag{6}$$

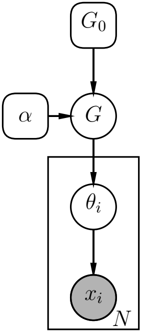
DP-Mixture Models

- Key application of DP: a prior over the parameters of a mixture model.
- For a mixture model with $K = \infty$, let:

$$\theta_i | G \sim G \tag{4}$$

$$x_i \sim F(\theta_i) \tag{5}$$

$$G \sim \mathrm{DP}(\alpha, G_0) \tag{6}$$

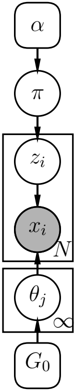


DP-Mixture Models

- In an alternative view, we can use π component weights and z points labels.

DP-Mixture Models

- In an alternative view, we can use π component weights and z points labels.



Parallel Sampler

- The CRP is usefull for understanding the DPMM.

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:
    - Slow.

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:
  - Slow.
  - Does not scale.

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:
  - Slow.
  - Does not scale.
  - Changing 1 label at a time - small moves have harder time escaping local maximum.

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:
  - Slow.
  - Does not scale.
  - Changing 1 label at a time - small moves have harder time escaping local maximum.
- Solution:

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:
  - Slow.
  - Does not scale.
  - Changing 1 label at a time - small moves have harder time escaping local maximum.
- Solution:
  [Chang & Fisher, NIPS '13]: an efficient parallel sampler which addresses these problems.

Parallel Sampler

- The CRP is usefull for understanding the DPMM.
- Problems with the CRP based sampler:
    - Slow.
    - Does not scale.
    - Changing 1 label at a time - small moves have harder time escaping local maximum.
- Solution:
  [Chang & Fisher, NIPS '13]: an efficient parallel sampler which addresses these problems.
- Remark: there also exist other efficient inference methods.

Parallel Sampler

The parallel sampler is comprised of two parts:

Parallel Sampler

The parallel sampler is comprised of two parts:

- Restricted Gibbs Sampler (K is fixed).

Parallel Sampler

The parallel sampler is comprised of two parts:

- Restricted Gibbs Sampler (K is fixed).
- Splits / Merges (changing K).

Augmented Space

- Augment the DPMM with auxiliary variables:

Augmented Space

- Augment the DPMM with auxiliary variables:

$$\bar{z}_i \in \{l, r\}, \quad \forall x_i \in \{x_1, \ldots, x_n\} \tag{7}$$

$$\bar{\pi}_j = \{\bar{\pi}_{jl}, \bar{\pi}_{jr}\}, \quad \bar{\theta}_j = \{\bar{\theta}_{jl}, \bar{\theta}_{jr}\} \tag{8}$$

Augmented Space

- Augment the DPMM with auxiliary variables:

$$\bar{z}_i \in \{l, r\}, \quad \forall x_i \in \{x_1, ..., x_n\} \tag{7}$$

$$\bar{\pi}_j = \{\bar{\pi}_{jl}, \bar{\pi}_{jr}\}, \quad \bar{\theta}_j = \{\bar{\theta}_{jl}, \bar{\theta}_{jr}\} \tag{8}$$

- Each cluster θ_j consists of 2 sub clusters $\{\bar{\theta}_{jl}, \bar{\theta}_{jr}\}$.

Augmented Space

- Augment the DPMM with auxiliary variables:

$$\bar{z}_i \in \{l, r\}, \quad \forall x_i \in \{x_1, ..., x_n\} \tag{7}$$

$$\bar{\pi}_j = \{\bar{\pi}_{jl}, \bar{\pi}_{jr}\}, \quad \bar{\theta}_j = \{\bar{\theta}_{jl}, \bar{\theta}_{jr}\} \tag{8}$$

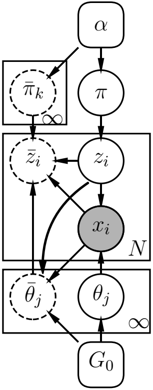
- Each cluster θ_j consists of 2 sub clusters $\{\bar{\theta}_{jl}, \bar{\theta}_{jr}\}$.
- In addition to each sample label z_i, we hold a label $\bar{z}_i$ for *left* or *right* sub cluster.

Augmented Space

- Augment the DPMM with auxiliary variables:

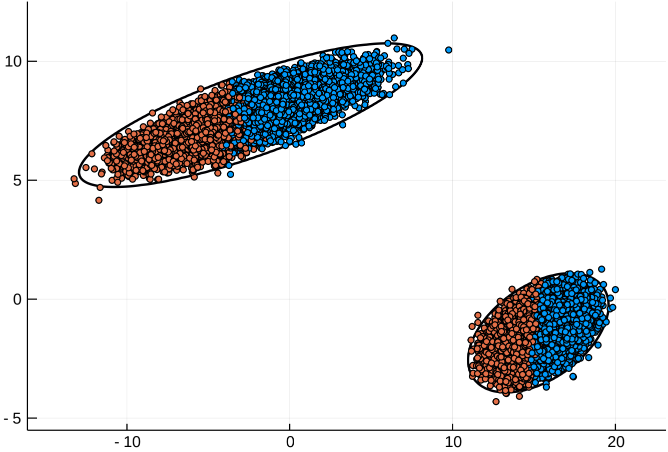
$$\bar{z}_i \in \{l, r\}, \quad \forall x_i \in \{x_1, ..., x_n\} \tag{7}$$

$$\bar{\pi}_j = \{\bar{\pi}_{jl}, \bar{\pi}_{jr}\}, \quad \bar{\theta}_j = \{\bar{\theta}_{jl}, \bar{\theta}_{jr}\} \tag{8}$$



- Each cluster θ_j consists of 2 sub clusters $\{\bar{\theta}_{jl}, \bar{\theta}_{jr}\}$.
- In addition to each sample label z_i, we hold a label $\bar{z}_i$ for *left* or *right* sub cluster.

Augmented Space



Visualization of the augmented space, 2 clusters, each has its points associated with either 'left' or 'right' sub-cluster.

Splits

- Create **new** clusters m, n by splitting an existing cluster c.

Splits

- Create **new** clusters m, n by splitting an existing cluster c.
- Examine each cluster and its sub-clusters, propose a split, and calculate the Hastings ratio for the split:

Splits

- Create **new** clusters m, n by splitting an existing cluster c.
- Examine each cluster and its sub-clusters, propose a split, and calculate the Hastings ratio for the split:

$$H_{\mathrm{split}} = \frac{\alpha \Gamma(N_{jl}) f_x(x_{\mathcal{I}_{jl}}; \lambda) \cdot \Gamma(N_{jr}) f_x(x_{\mathcal{I}_{jr}}; \lambda)}{\Gamma(N_j) f_x(x_{\mathcal{I}_j}; \lambda)} \tag{9}$$

Splits

- Create **new** clusters m, n by splitting an existing cluster c.
- Examine each cluster and its sub-clusters, propose a split, and calculate the Hastings ratio for the split:

$$H_{\mathrm{split}} = \frac{\alpha\Gamma(N_{jl})f_x(x_{\mathcal{I}_{jl}};\lambda)\cdot\Gamma(N_{jr})f_x(x_{\mathcal{I}_{jr}};\lambda)}{\Gamma(N_j)f_x(x_{\mathcal{I}_j};\lambda)} \tag{9}$$

$$\text{The accept probability } = \min[1, H_{\mathrm{split}}] \tag{10}$$

Merges

- We allow merging of two **existing** clusters m, n into a **new** cluster c.

Merges

- We allow merging of two **existing** clusters m, n into a **new** cluster c.
- Examine each pair of clusters and propose a merge, calculate the Hasting ratio for the merge:

Merges

- We allow merging of two **existing** clusters m, n into a **new** cluster c.
- Examine each pair of clusters and propose a merge, calculate the Hasting ratio for the merge:

$$H_{\mathrm{merge}} = \frac{\Gamma(N_{j_1} + N_{j_2})}{\alpha\Gamma(N_{j_1})\Gamma(N_{j_2})} \frac{p(x|\hat{z})}{p(x|z)} \times \frac{\Gamma(\alpha)}{\Gamma(\alpha + N_{j_1} + N_{j_2})}$$

$$\times \frac{\Gamma(\frac{\alpha}{2} + N_{j_1})\Gamma(\frac{\alpha}{2} + N_{j_2})}{\Gamma(\frac{\alpha}{2})\Gamma(\frac{\alpha}{2})} \tag{11}$$

Merges

- We allow merging of two **existing** clusters m, n into a **new** cluster c.
- Examine each pair of clusters and propose a merge, calculate the Hasting ratio for the merge:

$$H_{\mathrm{merge}} = \frac{\Gamma(N_{j_1} + N_{j_2})}{\alpha\Gamma(N_{j_1})\Gamma(N_{j_2})} \frac{p(x|\hat{z})}{p(x|z)} \times \frac{\Gamma(\alpha)}{\Gamma(\alpha + N_{j_1} + N_{j_2})}$$

$$\times \frac{\Gamma(\frac{\alpha}{2} + N_{j_1})\Gamma(\frac{\alpha}{2} + N_{j_2})}{\Gamma(\frac{\alpha}{2})\Gamma(\frac{\alpha}{2})} \tag{11}$$

$$\text{The accept probability } = \min[1, H_{\mathrm{merge}}] \tag{12}$$

Large Moves

Merges/Splits allows us to do large moves, changing many labels at a time, and often allowing us to escape a local maximum.

Sampler iteration

- Run an iteration of the restricted Gibbs sampler, on a fixed number of clusters:

Sampler iteration

- Run an iteration of the restricted Gibbs sampler, on a fixed number of clusters:
  - Sample variables: π, θ,z.

Sampler iteration

- Run an iteration of the restricted Gibbs sampler, on a fixed number of clusters:
  - Sample variables: π, θ,z.
  - Sample auxiliary variables: $\bar{\pi}$, $\bar{\theta}$,$\bar{z}$.

Sampler iteration

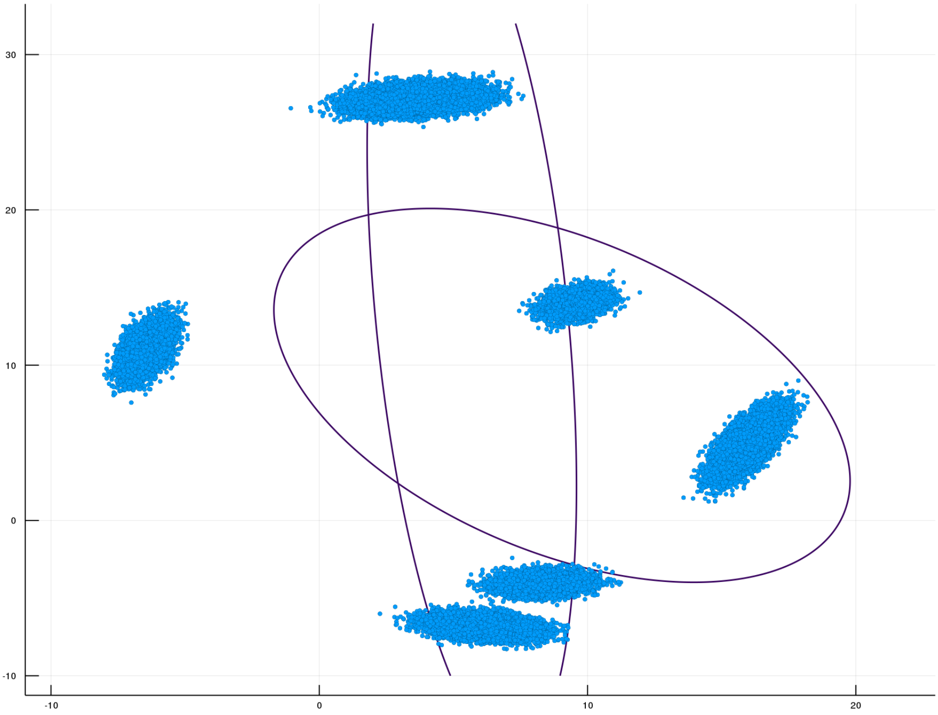
- Run an iteration of the restricted Gibbs sampler, on a fixed number of clusters:
  - Sample variables: π, θ,z.
  - Sample auxiliary variables: $\bar{\pi}$, $\bar{\theta}$,$\bar{z}$.
- Modify the number of clusters:

Sampler iteration

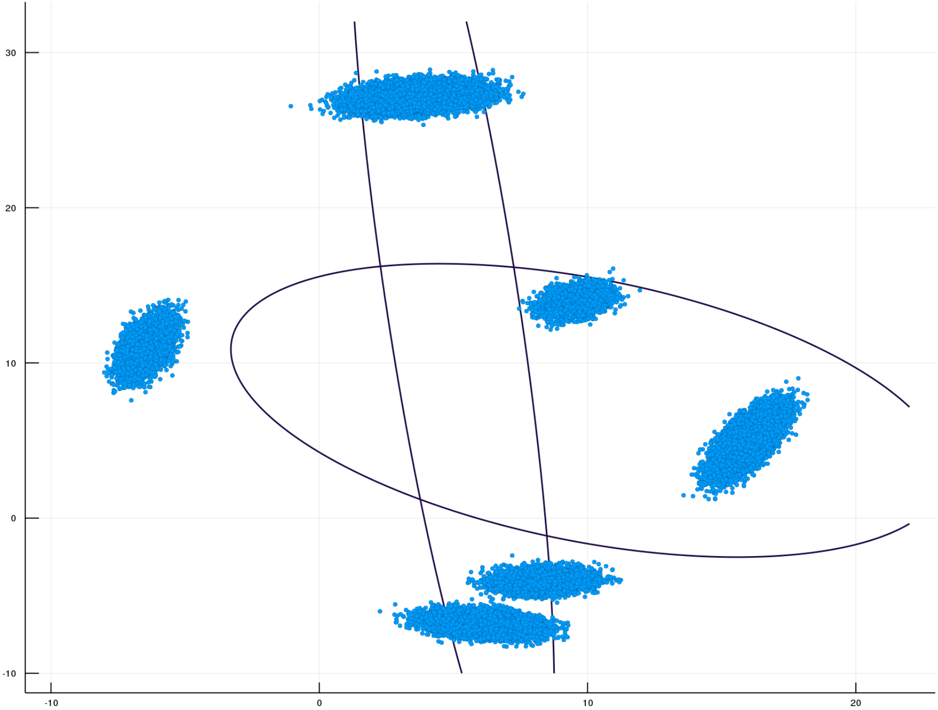
- Run an iteration of the restricted Gibbs sampler, on a fixed number of clusters:
  - Sample variables: π, θ,z.
  - Sample auxiliary variables: $\bar{\pi}$, $\bar{\theta}$,$\bar{z}$.
- Modify the number of clusters:
  - Propose and accept Splits.

Sampler iteration

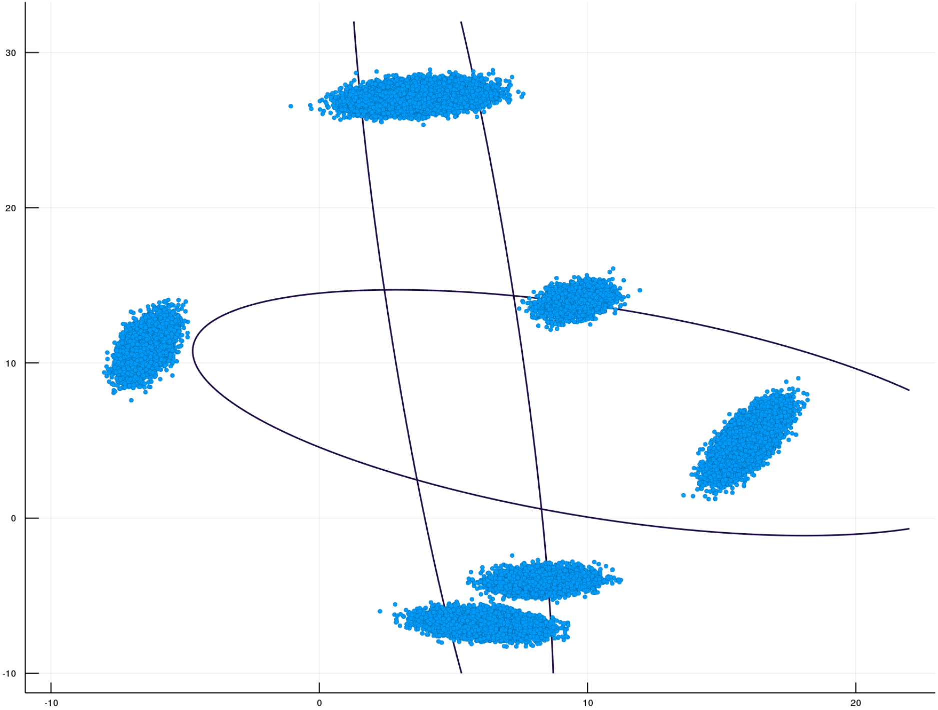
- Run an iteration of the restricted Gibbs sampler, on a fixed number of clusters:
  - Sample variables: π, θ,z.
  - Sample auxiliary variables: $\bar{\pi}$, $\bar{\theta}$,$\bar{z}$.
- Modify the number of clusters:
  - Propose and accept Splits.
  - Propose and accept Merges.



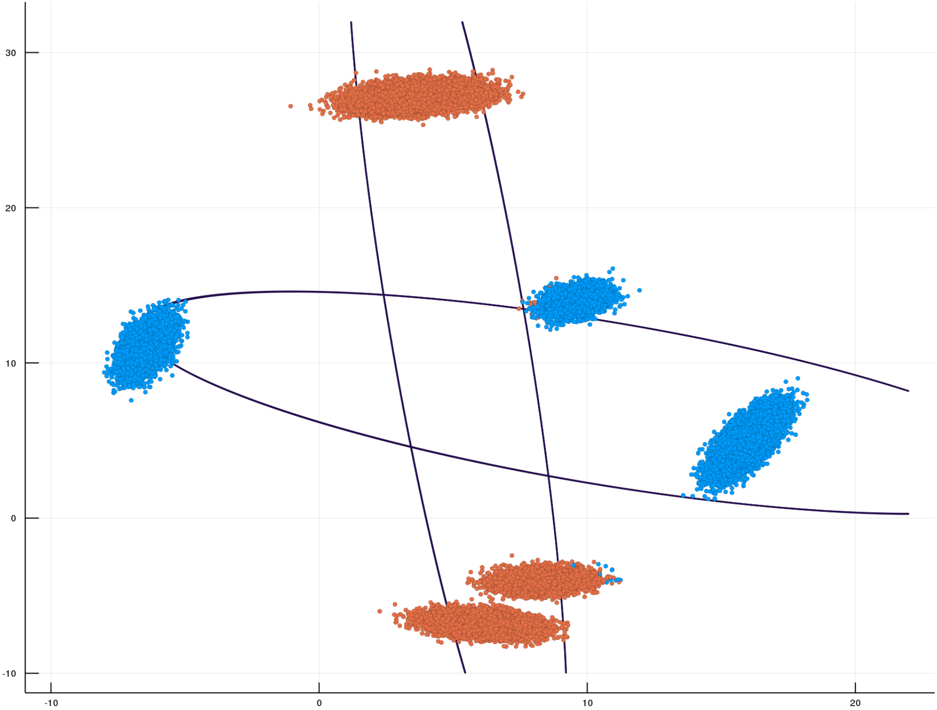
30
20
10
0
-10
-10
0
10
20



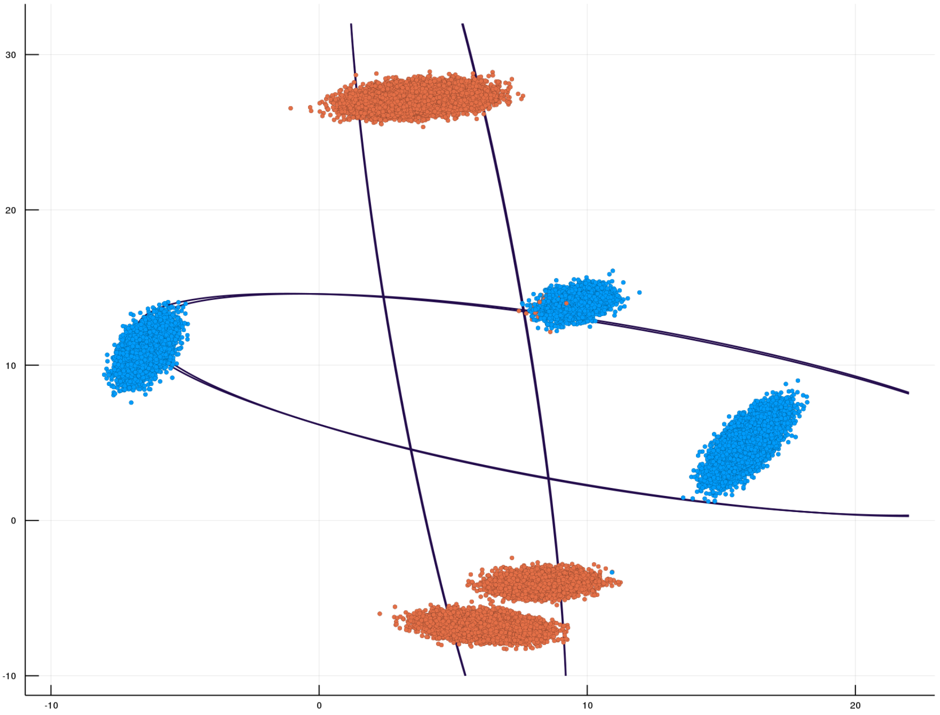
30
20
10
0
-10
-10
0
10
20



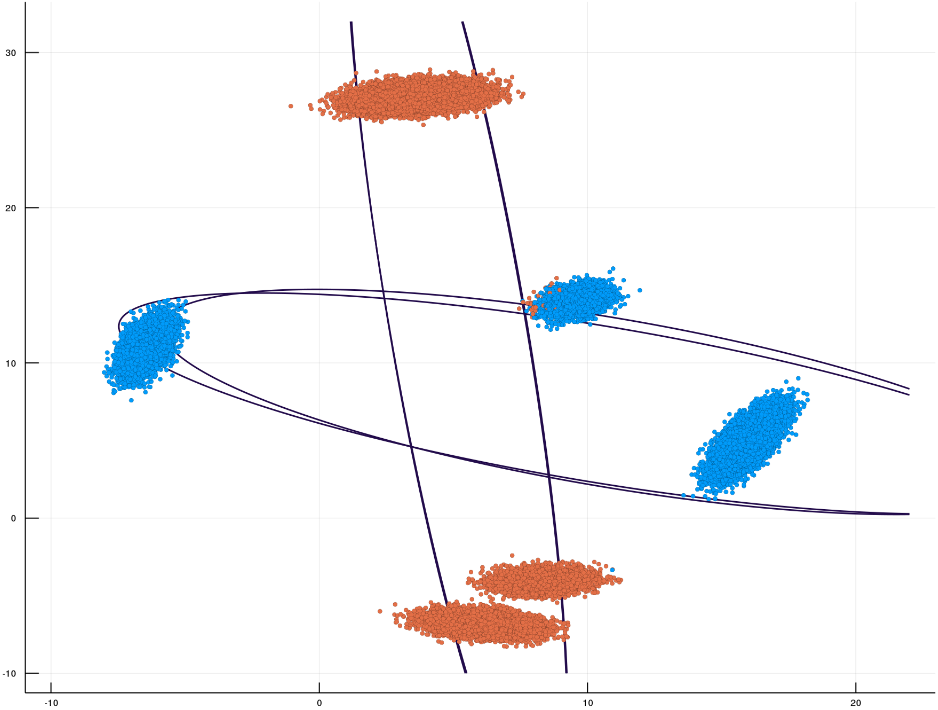
30
20
10
0
-10
-10
0
10
20



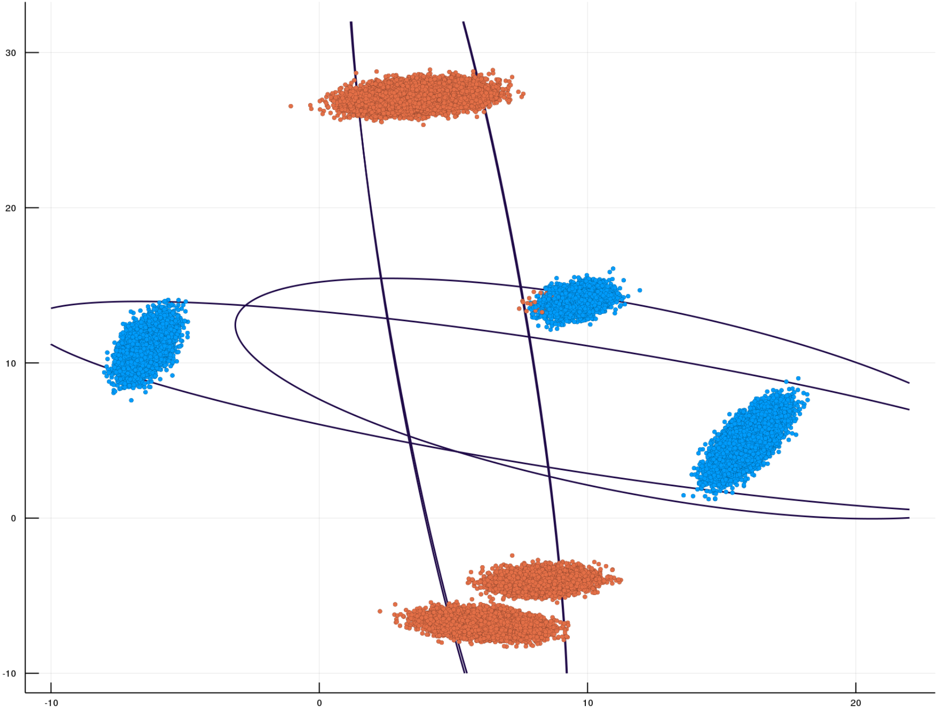
30
20
10
0
-10
-10
0
10
20



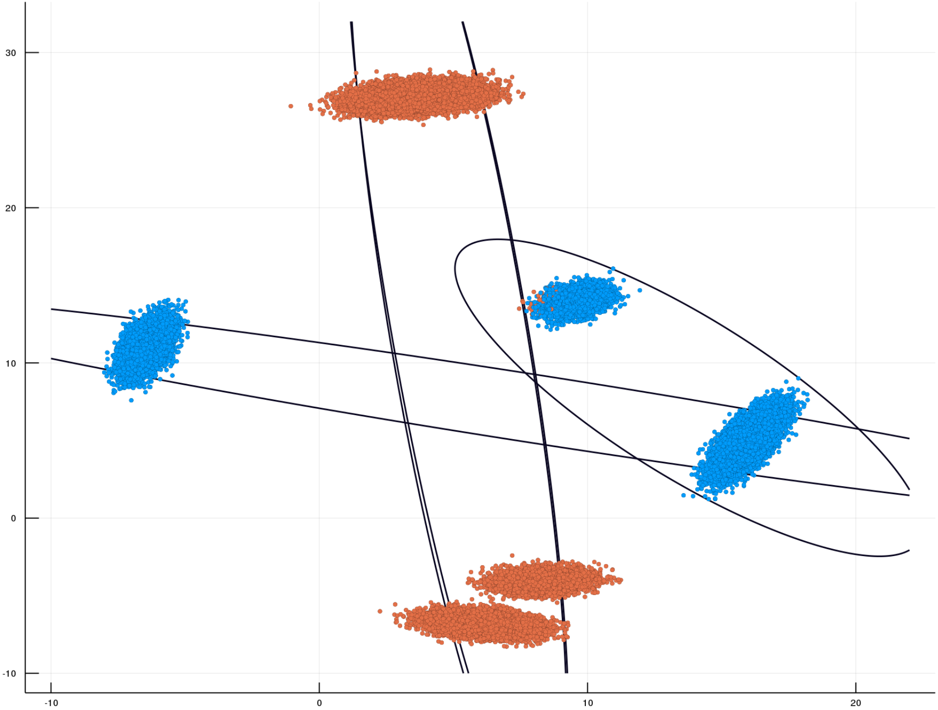
30
20
10
0
-10
-10
0
10
20

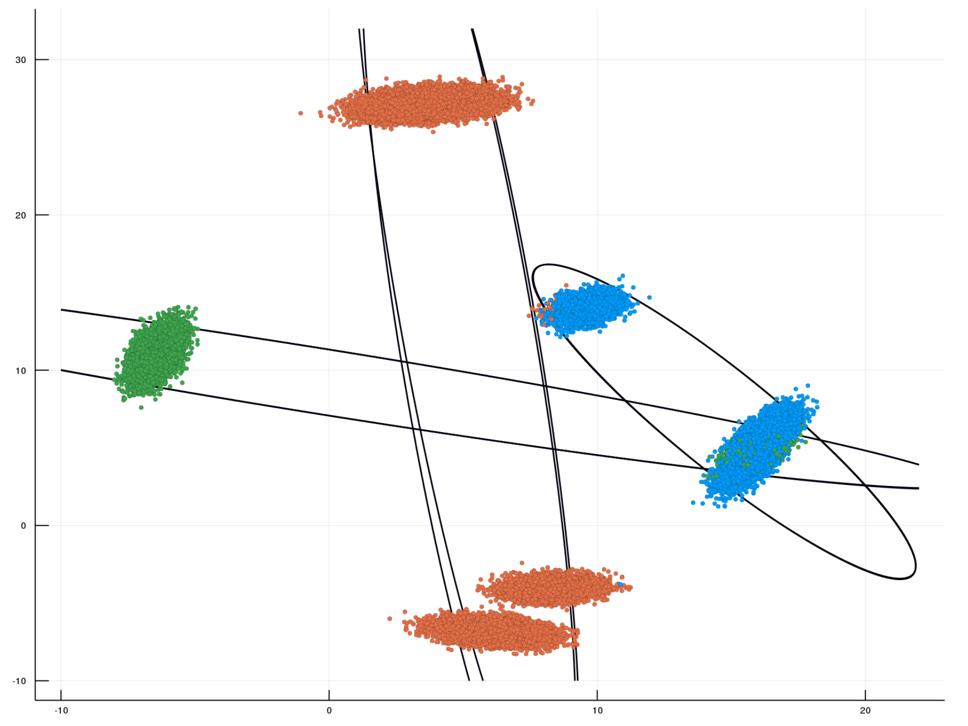


30
20
10
0
-10
-10
0
10
20

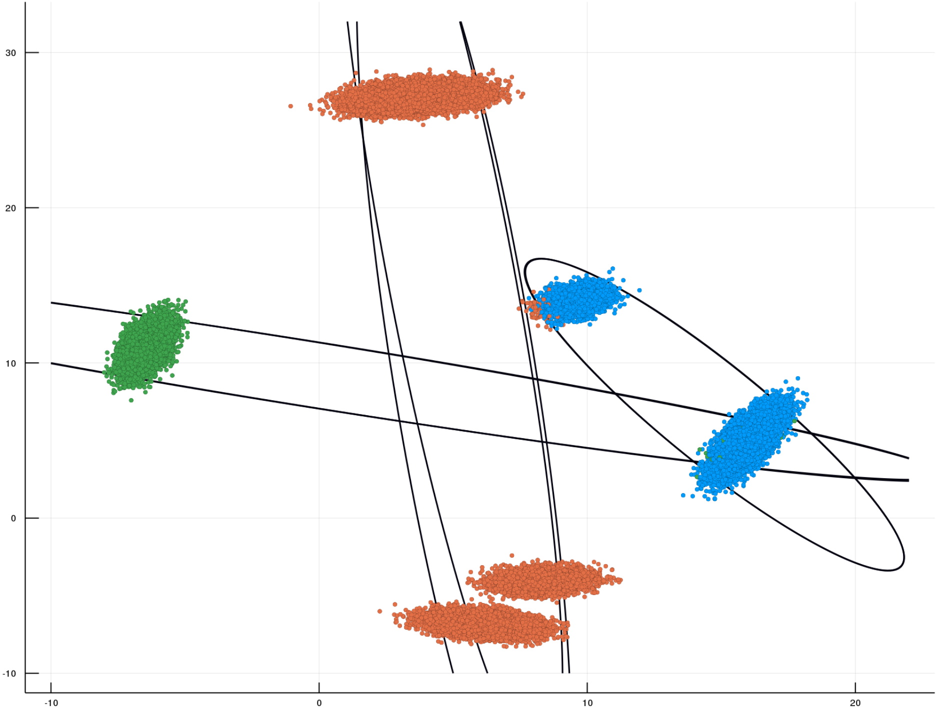


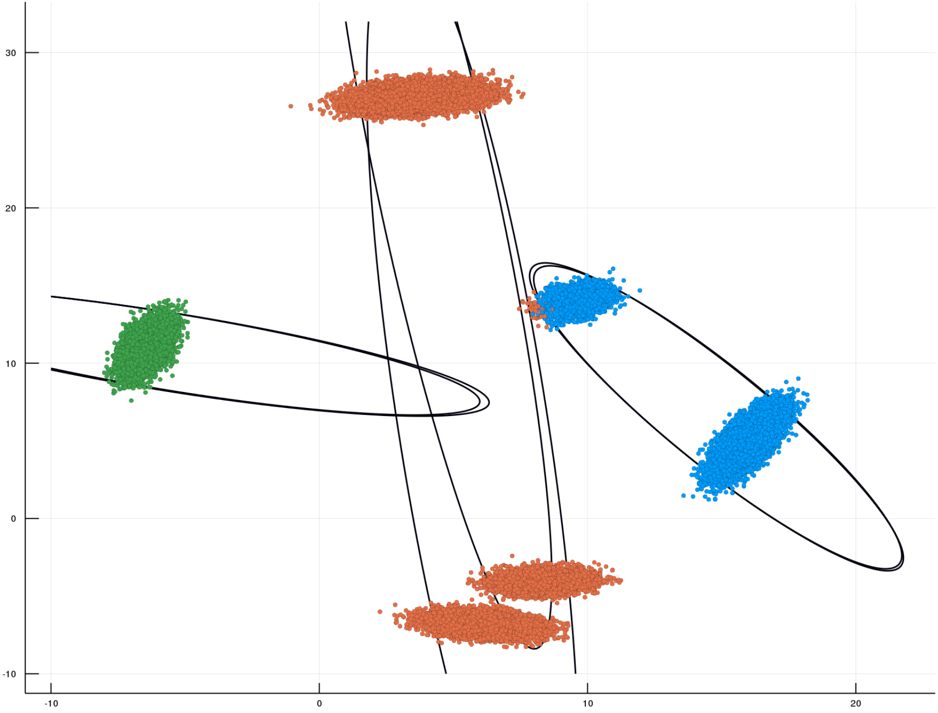
30
20
10
0
-10
-10
0
10
20


30
20
10
0
-10
-10
0
10
20

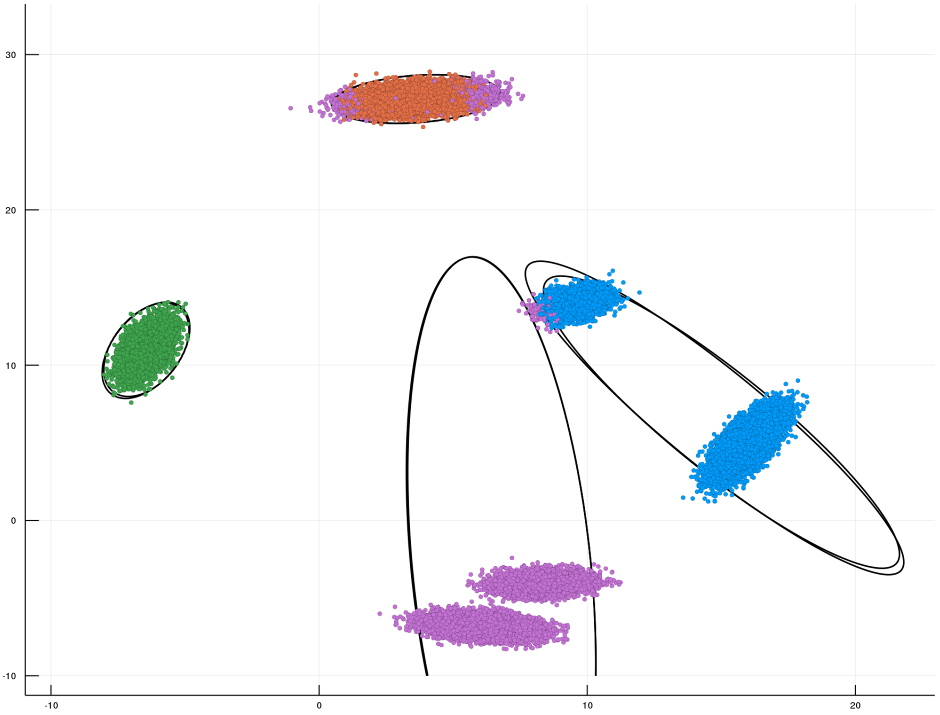


30
20
10
0
-10
-10
0
10
20

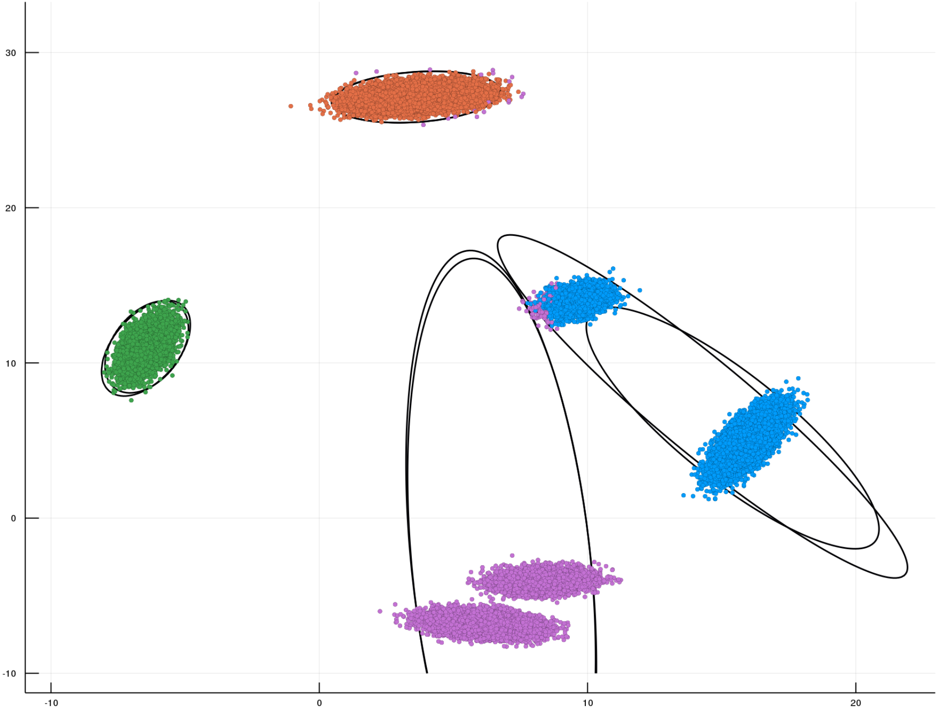

30
20
10
0
-10
-10
0
10
20

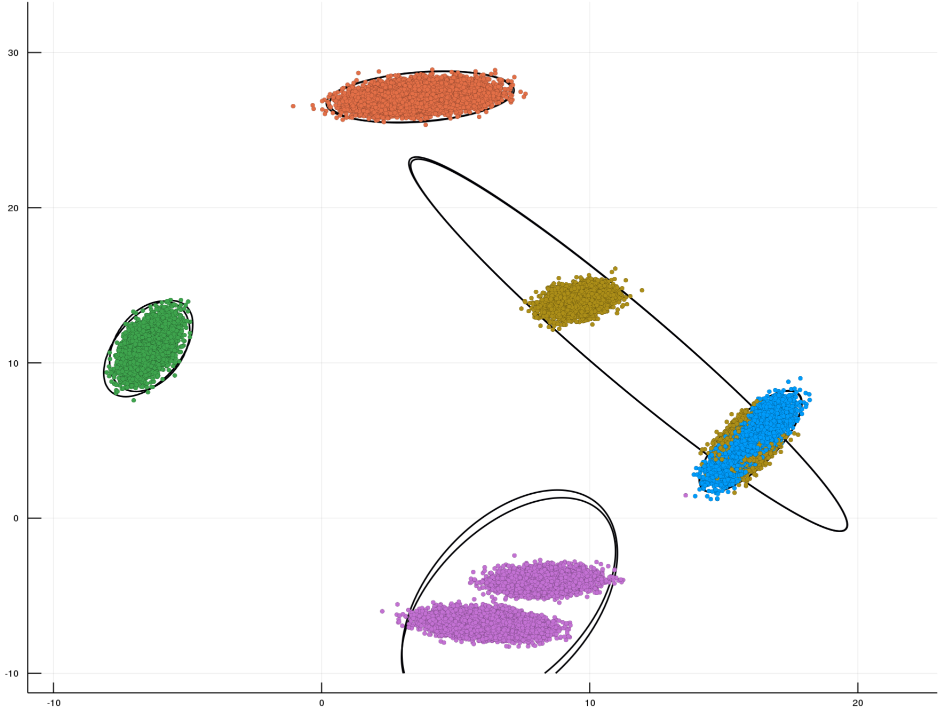


30
20
10
0
-10
-10
0
10
20

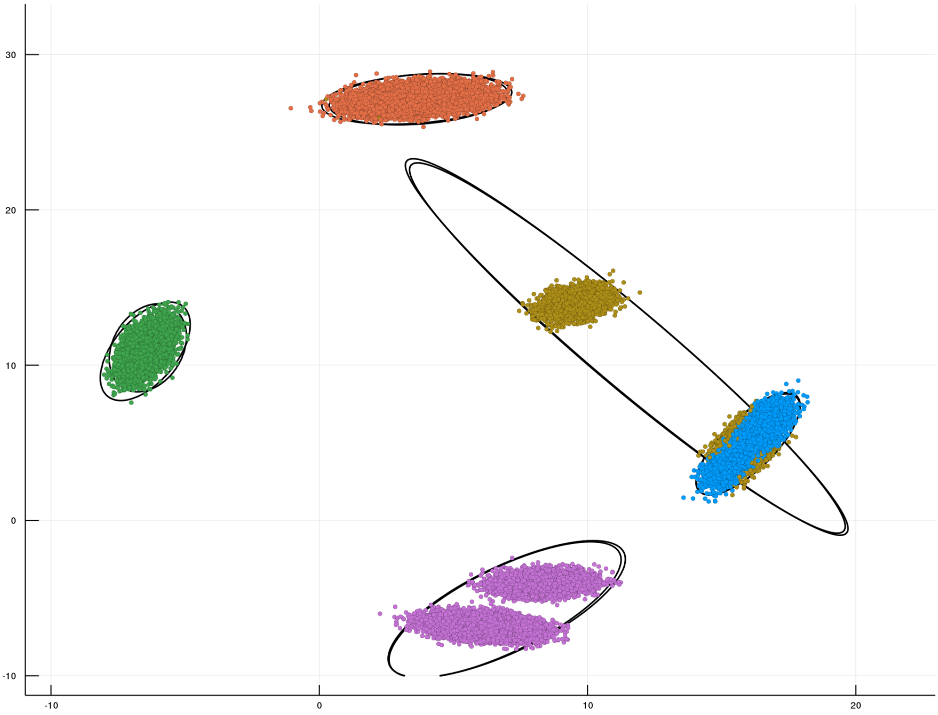


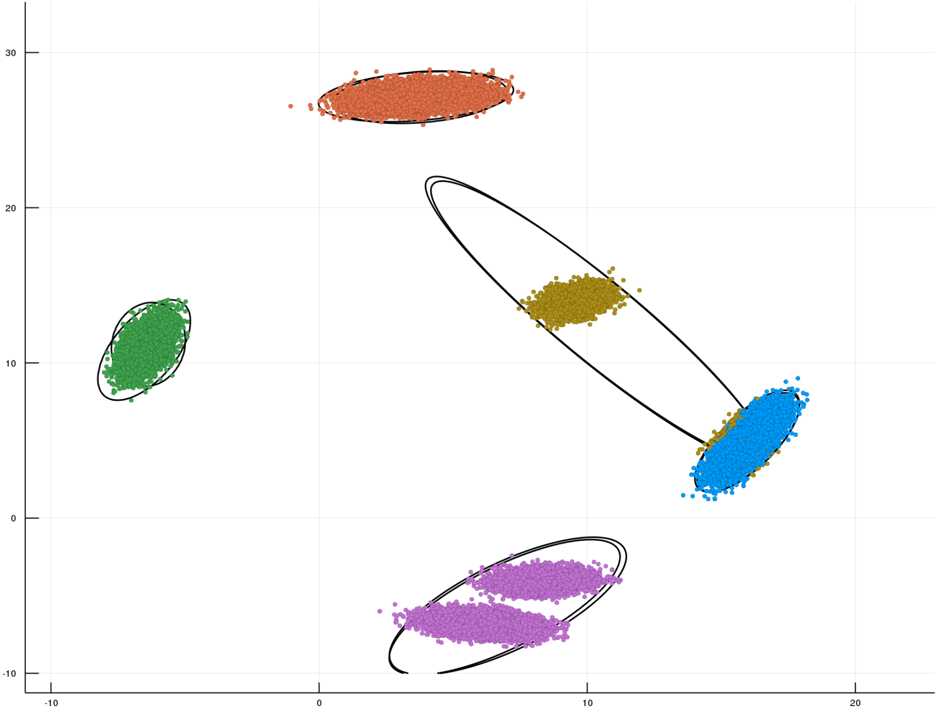
30
20
10
0
-10
-10
0
10
20


30
20
10
0
-10
-10
0
10
20

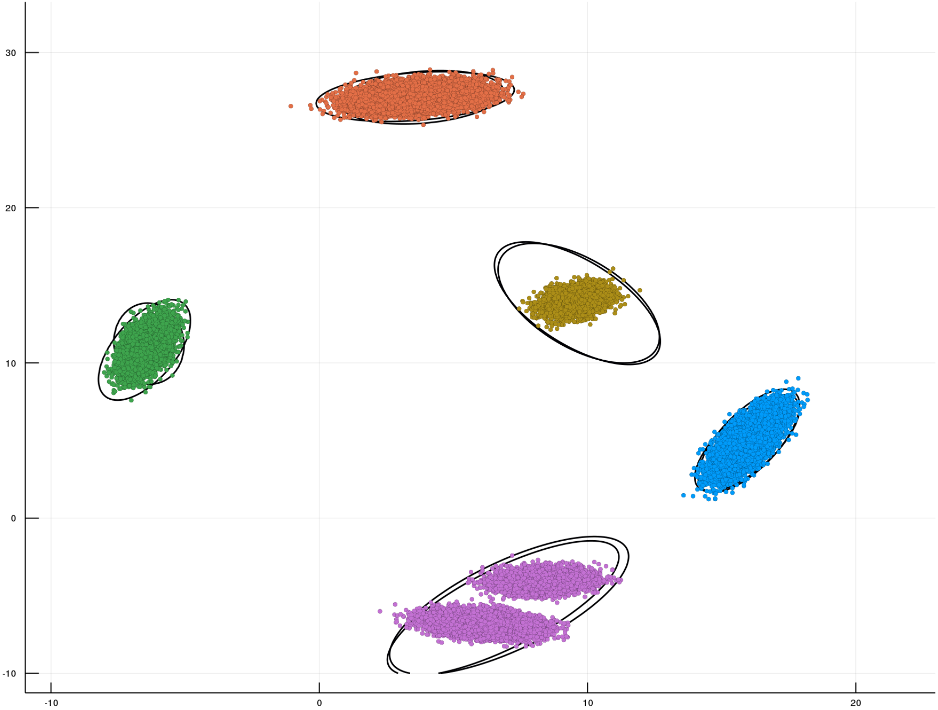


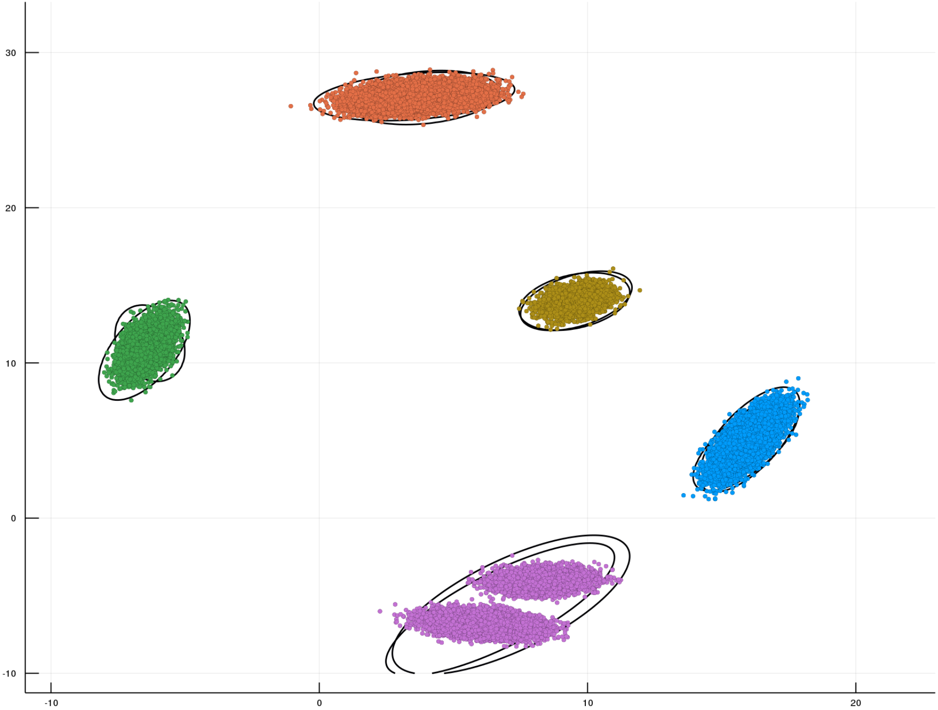
30
20
10
0
-10
-10
0
10
20

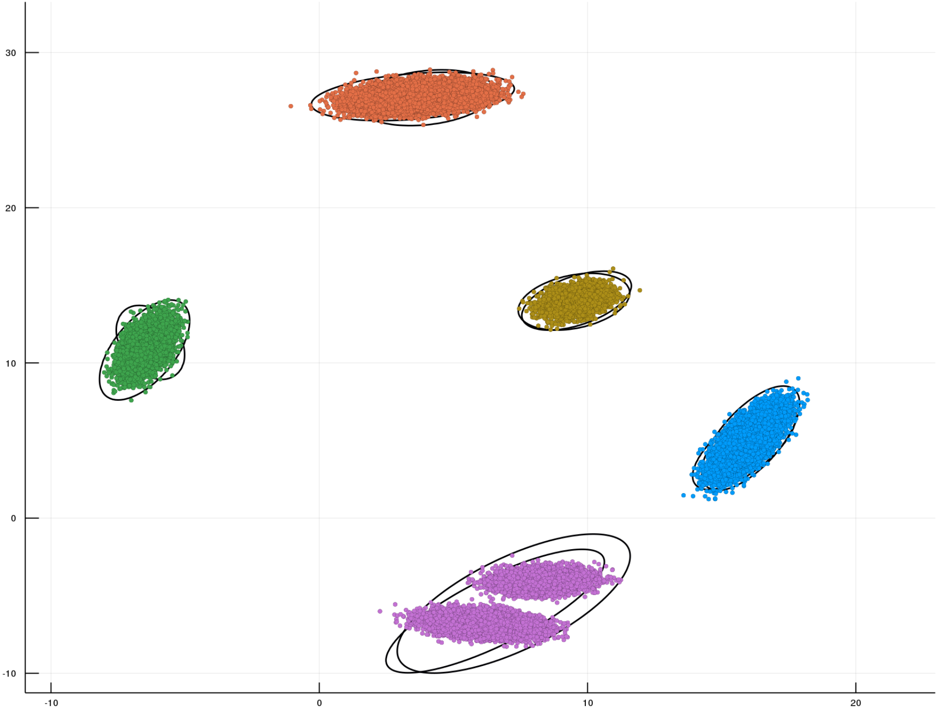

30
20
10
0
-10
-10
0
10
20

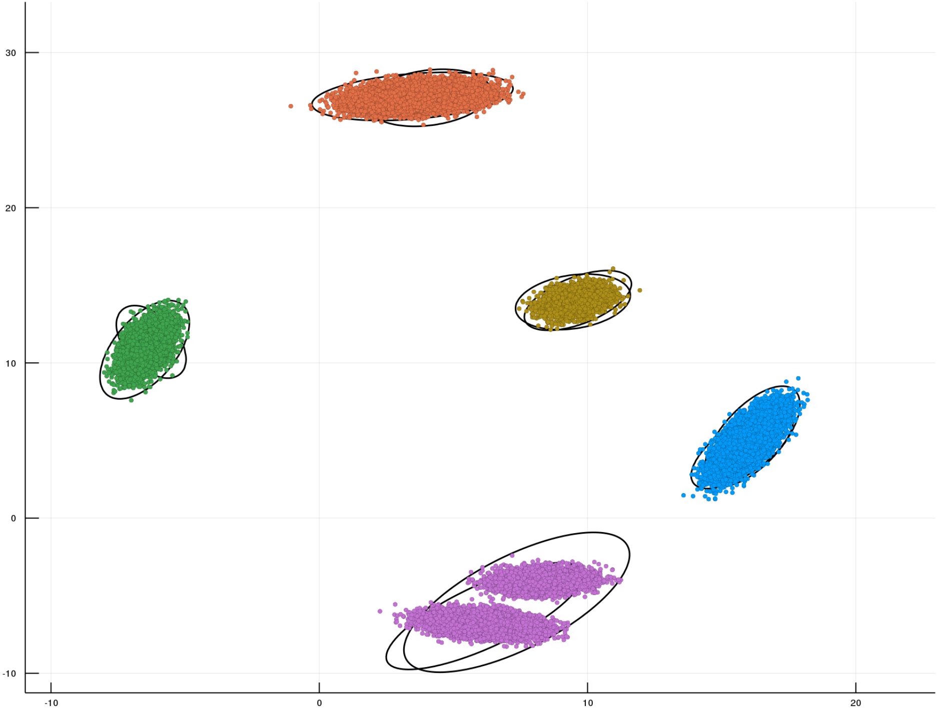


30
20
10
0
-10
-10
0
10
20

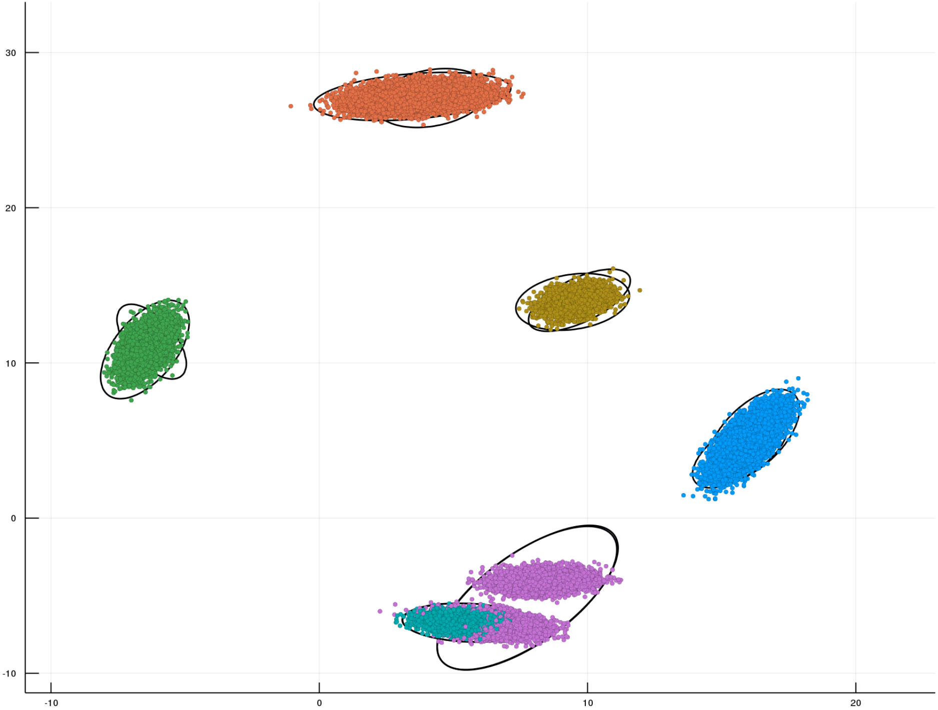

30
20
10
0
-10
-10
0
10
20


30
20
10
0
-10
-10
0
10
20

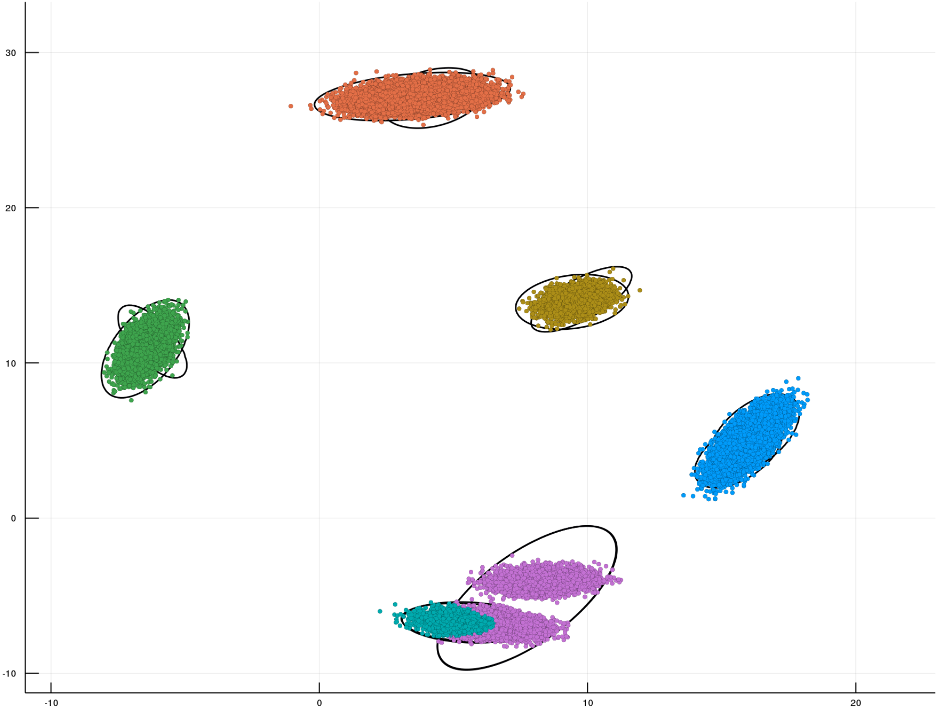

30
20
10
0
-10
-10
0
10
20



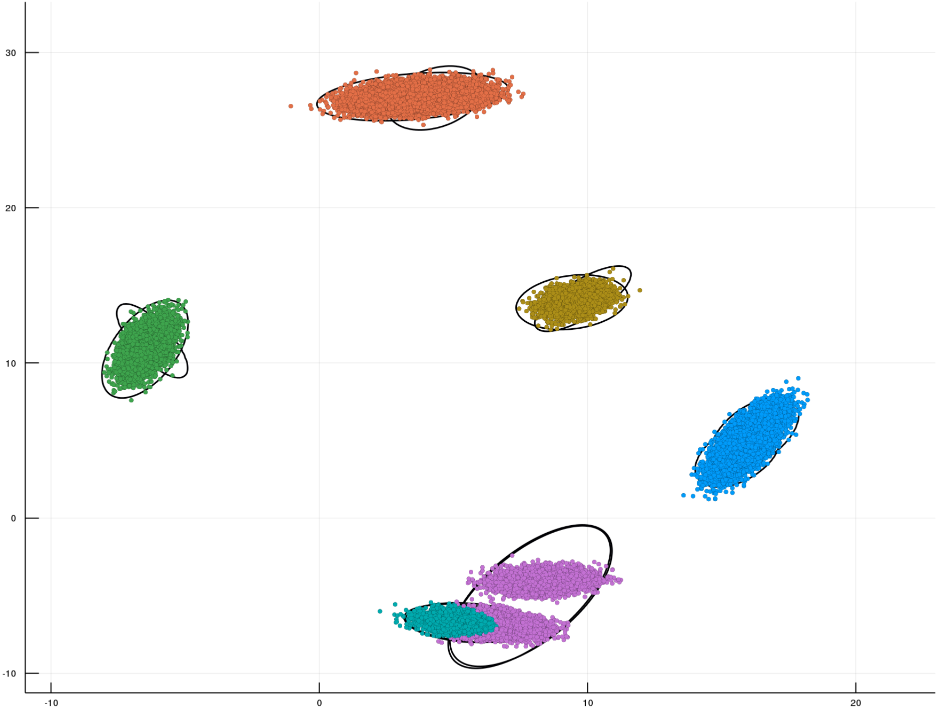
30
20
10
0
-10
-10
0
10
20

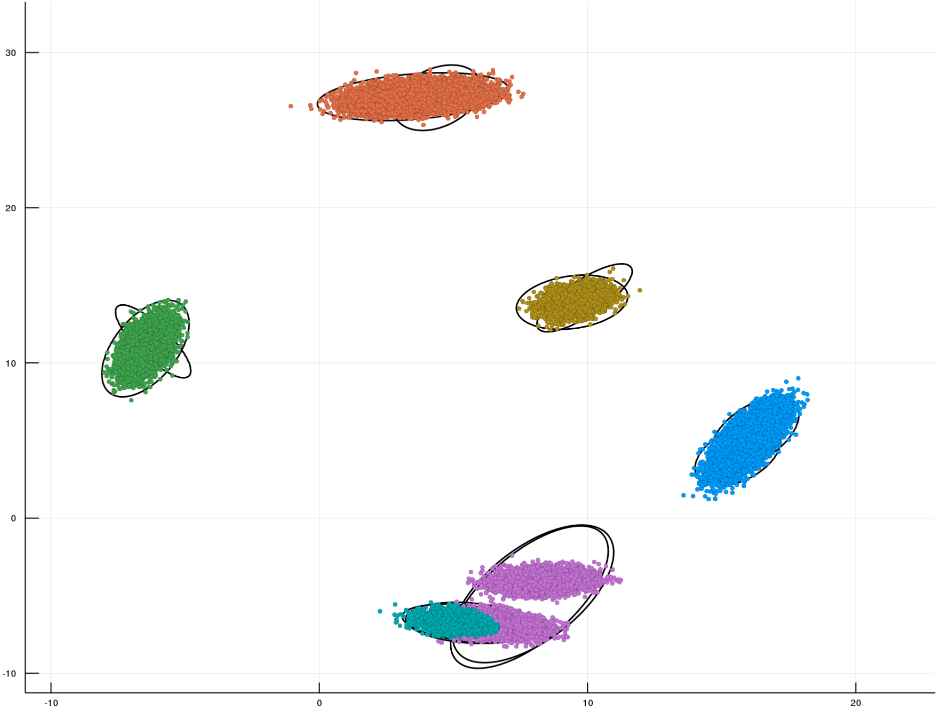


30
20
10
0
-10
-10
0
10
20

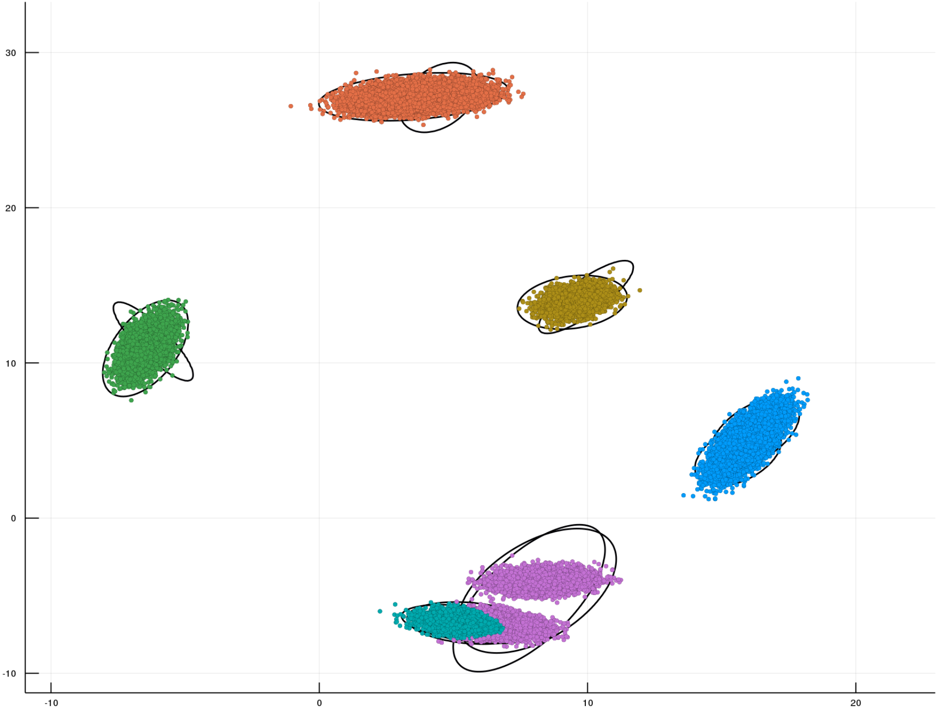


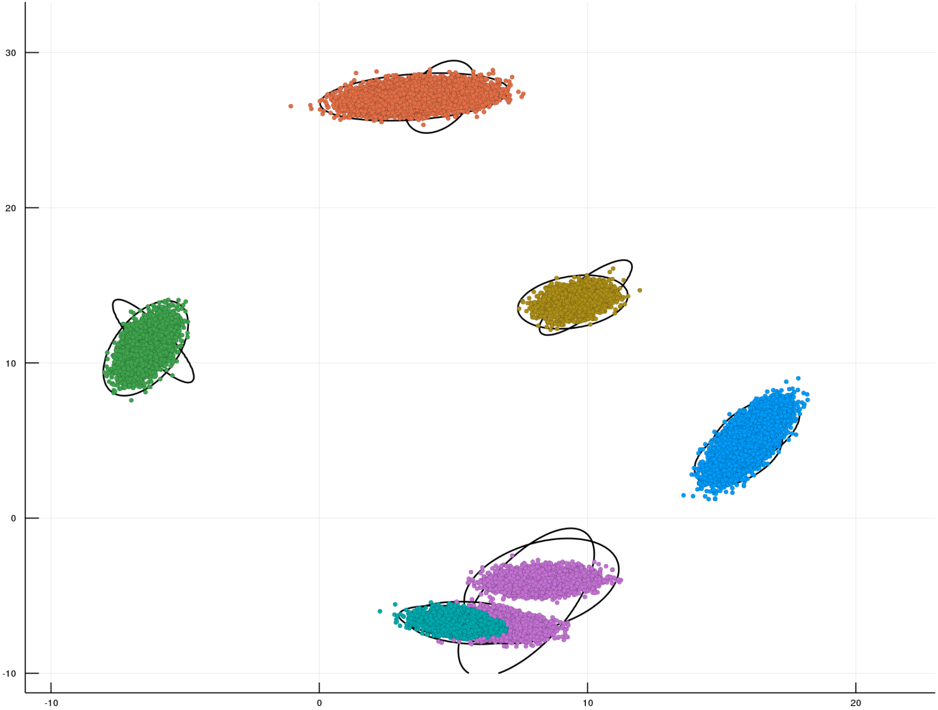
30
20
10
0
-10
-10
0
10
20

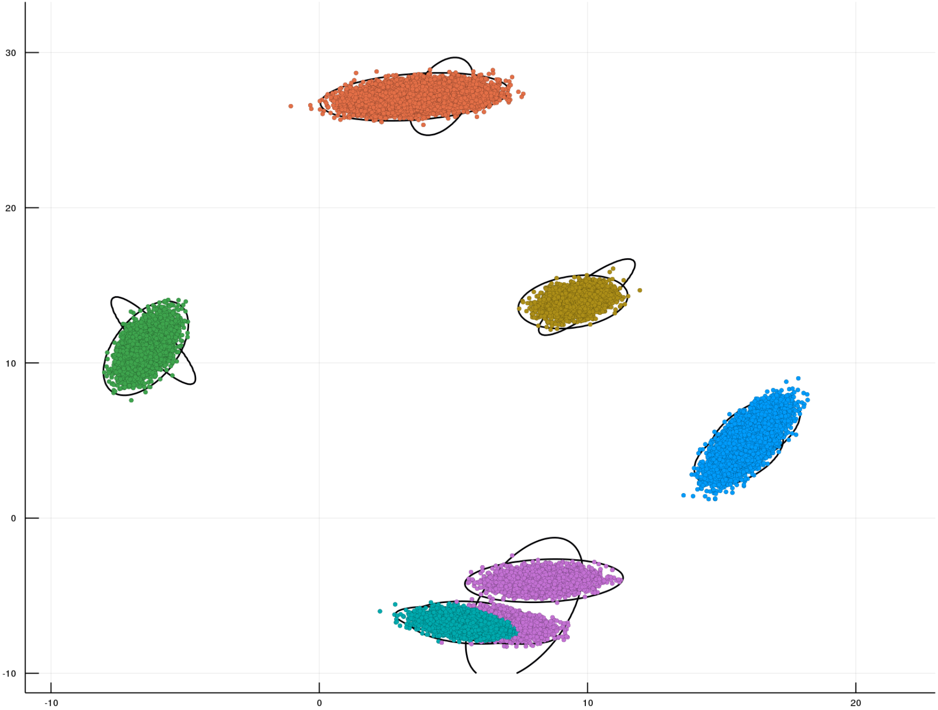

30
20
10
0
-10
-10
0
10
20

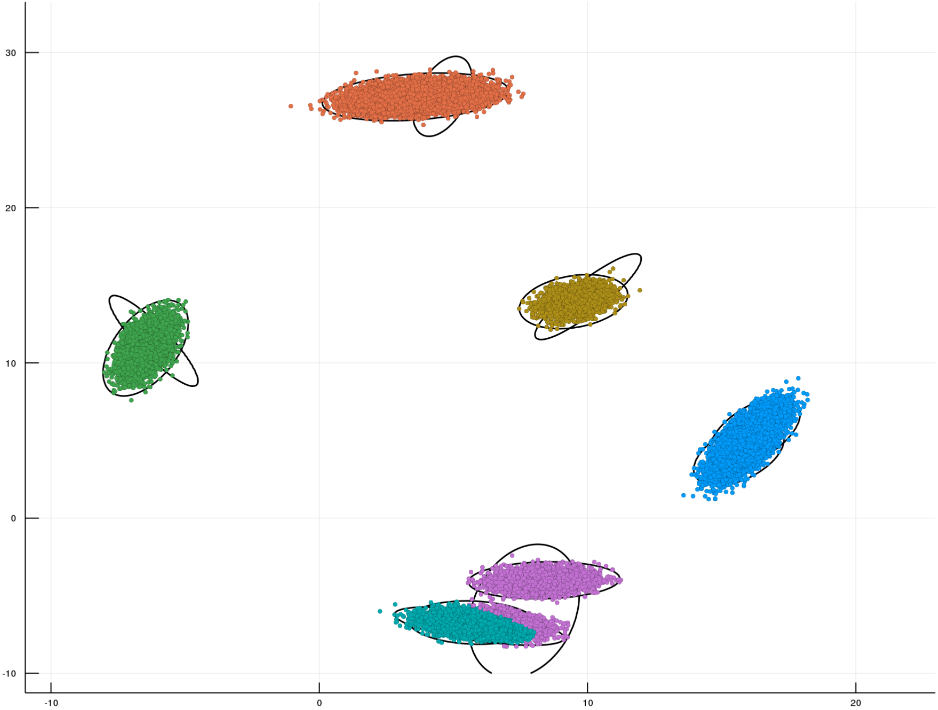


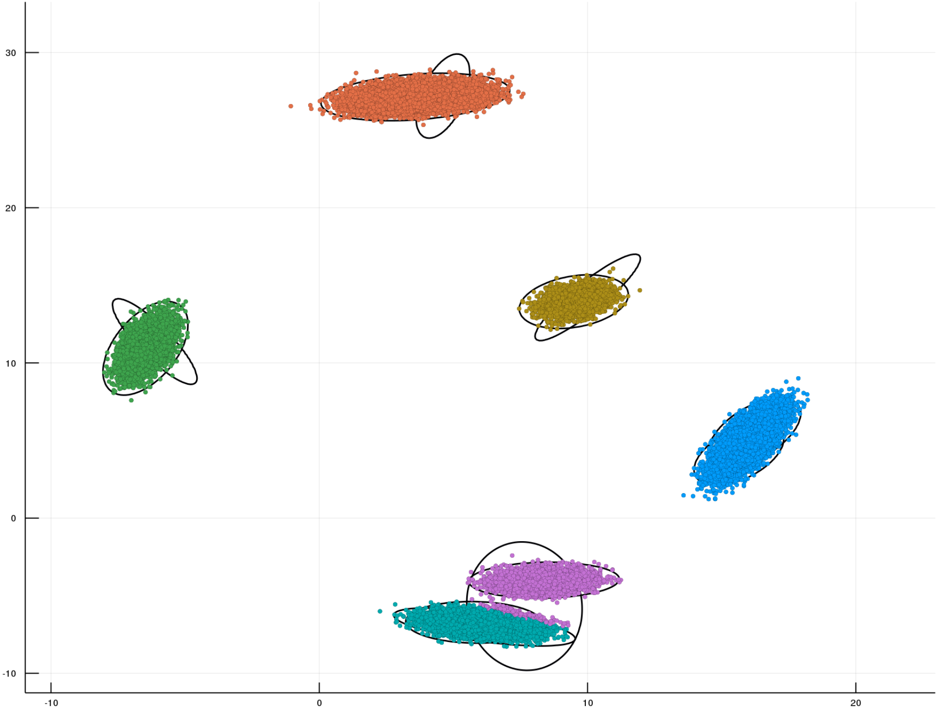
30
20
10
0
-10
-10
0
10
20


30
20
10
0
-10
-10
0
10
20

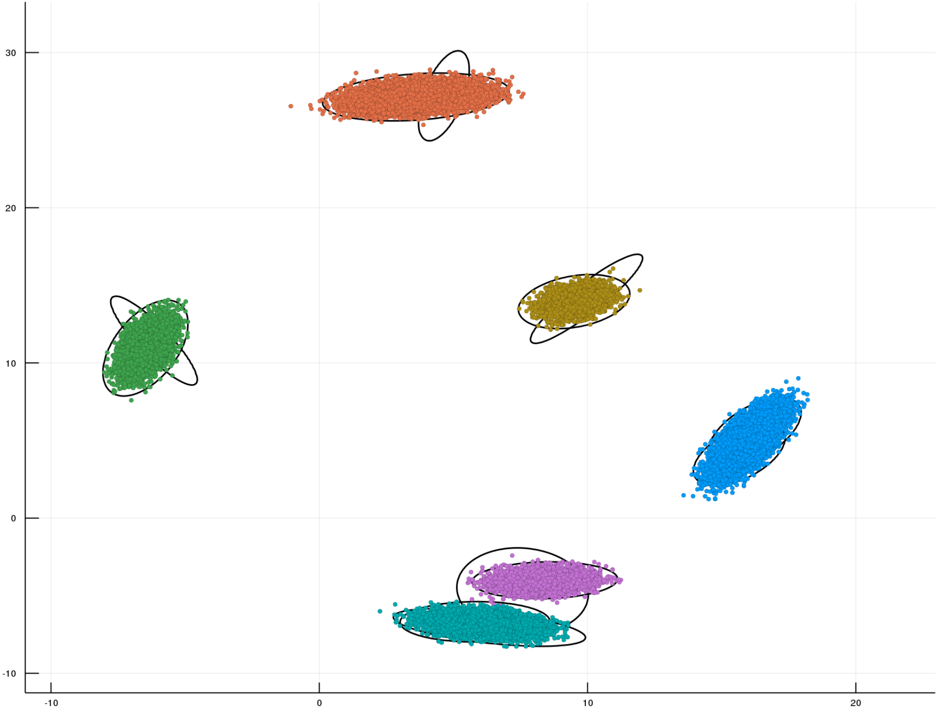

30
20
10
0
-10
-10
0
10
20

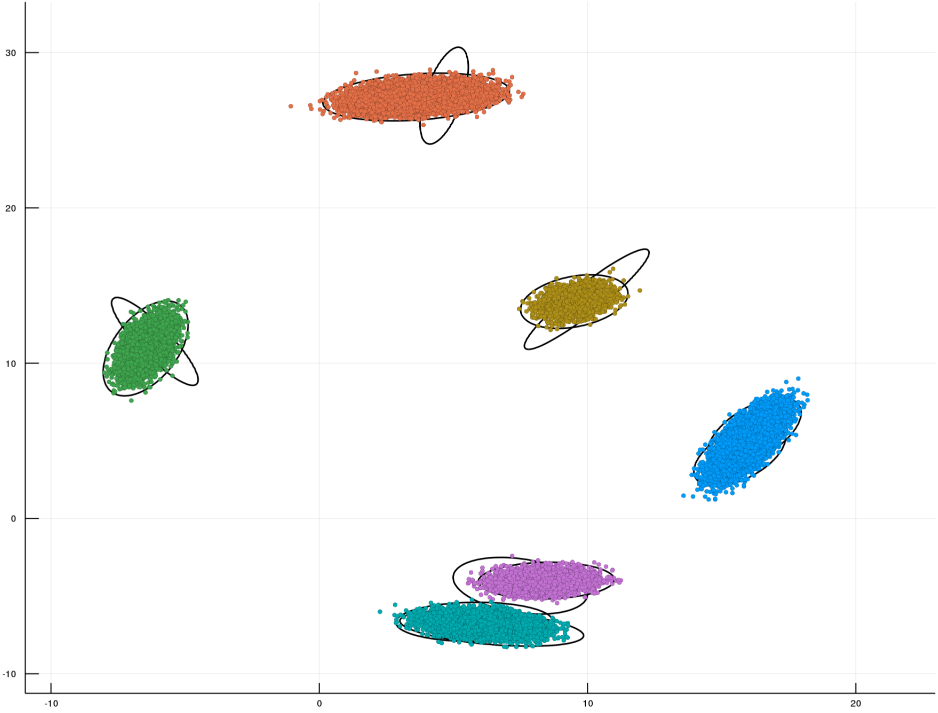

30
20
10
0
-10
-10
0
10
20

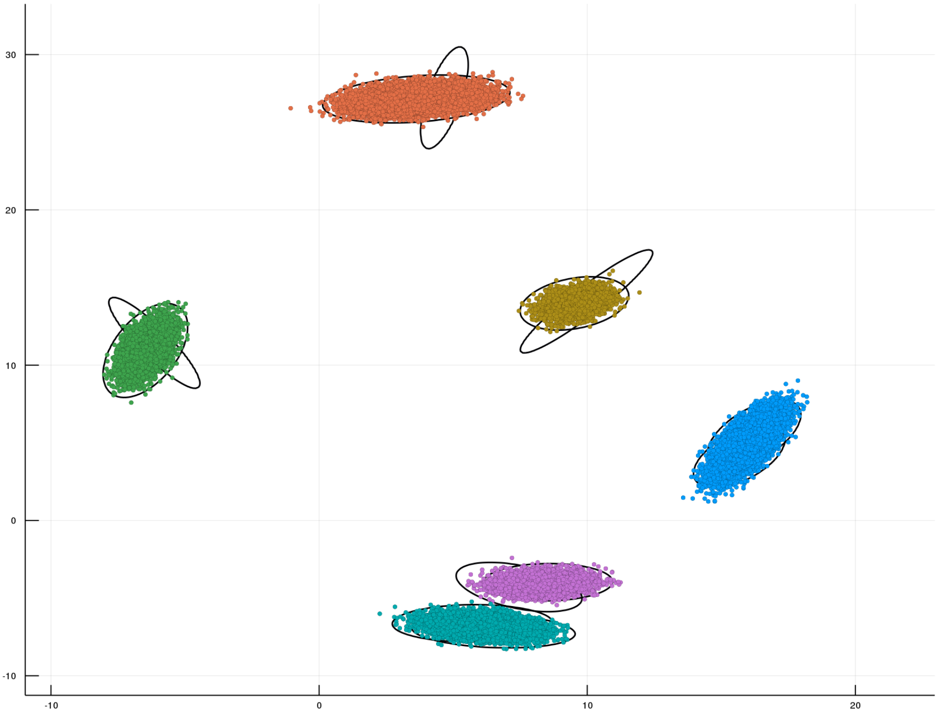

30
20
10
0
-10
-10
0
10
20



30
20
10
0
-10
-10
0
10
20


30
20
10
0
-10
-10
0
10
20


30
20
10
0
-10
-10
0
10
20



30
20
10
0
-10
-10
0
10
20

Distributed Parallel Sampler

- Chang and Fisher's sampler:

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - C++/MATLAB.

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - C++/MATLAB.
  - A shared memory model

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - $C++$/MATLAB.
  - A shared memory model
  - Highly optimized for GMM and MNMM cases but not flexible.

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - C++/MATLAB.
  - A shared memory model
  - Highly optimized for GMM and MNMM cases but not flexible.
- We extend that work, aiming for:

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - C++/MATLAB.
  - A shared memory model
  - Highly optimized for GMM and MNMM cases but not flexible.
- We extend that work, aiming for:
  - a multi-core multi-machine implementation.

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - $C{+}{+}$/MATLAB.
  - A shared memory model
  - Highly optimized for GMM and MNMM cases but not flexible.
- We extend that work, aiming for:
  - a multi-core multi-machine implementation.
  - Flexible in both the prior and the setting.

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - C++/MATLAB.
  - A shared memory model
  - Highly optimized for GMM and MNMM cases but not flexible.
- We extend that work, aiming for:
  - a multi-core multi-machine implementation.
  - Flexible in both the prior and the setting.
  - Easy to use and configure.

Distributed Parallel Sampler

- Chang and Fisher's sampler:
    - Single machine multiprocess sampler
    - $C{+}{+}$/MATLAB.
    - A shared memory model
    - Highly optimized for GMM and MNMM cases but not flexible.
- We extend that work, aiming for:
    - a multi-core multi-machine implementation.
    - Flexible in both the prior and the setting.
    - Easy to use and configure.
- The proposed implementation is done in Julia.

Distributed Parallel Sampler

- Chang and Fisher's sampler:
  - Single machine multiprocess sampler
  - $C{+}{+}$/MATLAB.
  - A shared memory model
  - Highly optimized for GMM and MNMM cases but not flexible.
- We extend that work, aiming for:
  - a multi-core multi-machine implementation.
  - Flexible in both the prior and the setting.
  - Easy to use and configure.
- The proposed implementation is done in Julia.

Why Julia?

- We do not suggest that Julia is better/worse than any other language.

Why Julia?

- We do not suggest that Julia is better/worse than any other language.
- Rather, we offer our perspective as ML researchers (as opposed to HPC/SW researchers).

Why Julia?

- We do not suggest that Julia is better/worse than any other language.
- Rather, we offer our perspective as ML researchers (as opposed to HPC/SW researchers).
- Julia:

Why Julia?

- We do not suggest that Julia is better/worse than any other language.
- Rather, we offer our perspective as ML researchers (as opposed to HPC/SW researchers).
- Julia:
    - Easy to use.

Why Julia?

- We do not suggest that Julia is better/worse than any other language.
- Rather, we offer our perspective as ML researchers (as opposed to HPC/SW researchers).
- Julia:
  - Easy to use.
  - Highly optimized, **often (but not always)** on par with **C** [1].

Why Julia?

- We do not suggest that Julia is better/worse than any other language.
- Rather, we offer our perspective as ML researchers (as opposed to HPC/SW researchers).
- Julia:
    - Easy to use.
    - Highly optimized, **often (but not always)** on par with **C** [1].
    - Short development time, similar to Python/Matlab.

Why Julia?

- We do not suggest that Julia is better/worse than any other language.
- Rather, we offer our perspective as ML researchers (as opposed to HPC/SW researchers).
- Julia:
  - Easy to use.
  - Highly optimized, **often (but not always)** on par with **C** [1].
  - Short development time, similar to Python/Matlab.
  - **Easy** to distribute: the overhead, in terms of the **programmer's time**, for distributed computing is minimal.

Distributed Parallel Sampler

- Distribute the **Data** and **Labels** across all nodes and processes.

Distributed Parallel Sampler

- Distribute the **Data** and **Labels** across all nodes and processes.
- Master/Slaves architecture.

Distributed Parallel Sampler

- Distribute the **Data** and **Labels** across all nodes and processes.
- Master/Slaves architecture.
- Extensive use of sufficient statistics.

Distributed Parallel Sampler

- Distribute the **Data** and **Labels** across all nodes and processes.
- Master/Slaves architecture.
- Extensive use of sufficient statistics.
- Minimize intra-machine communication.

Distributed Parallel Sampler

- Distribute the **Data** and **Labels** across all nodes and processes.
- Master/Slaves architecture.
- Extensive use of sufficient statistics.
- Minimize intra-machine communication.
- At no point of time, a node can see the data which belongs to other nodes.

Master Node

- Samples the components parameters and weights.

Master Node

- Samples the components parameters and weights.
- Distribute the parameters across all nodes.

Master Node

- Samples the components parameters and weights.
- Distribute the parameters across all nodes.
- Aggregates the Sufficient statistics from all the nodes.

Master Node

- Samples the components parameters and weights.
- Distribute the parameters across all nodes.
- Aggregates the Sufficient statistics from all the nodes.
- Decides on Splits/Merges.

Master Node

- Samples the components parameters and weights.
- Distribute the parameters across all nodes.
- Aggregates the Sufficient statistics from all the nodes.
- Decides on Splits/Merges.
- Distribute the decision across all nodes.

Slave Node

- Only one process communicates with the master node.

Slave Node

- Only one process communicates with the master node.
- Receives component parameters from the master node.

Slave Node

- Only one process communicates with the master node.
- Receives component parameters from the master node.
- Sample it's Data labels.

Slave Node

- Only one process communicates with the master node.
- Receives component parameters from the master node.
- Sample it's Data labels.
- Calculate sufficient statistics and send them to the master node..

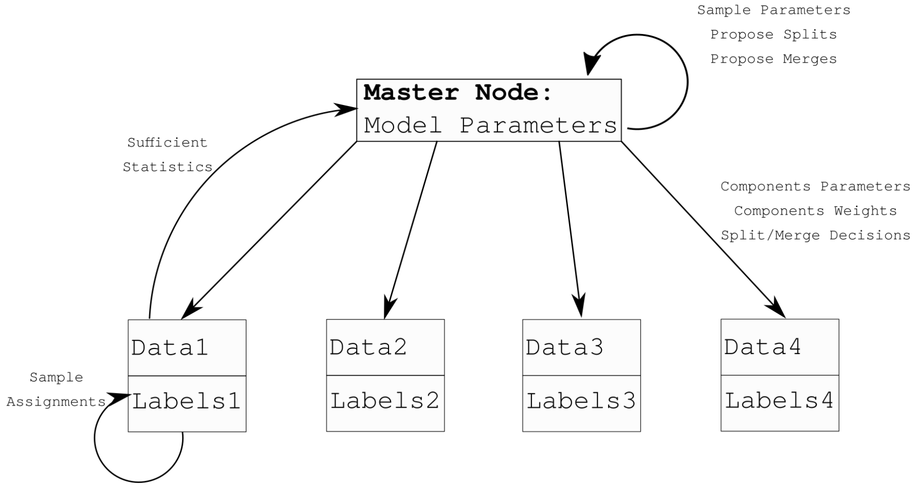
Slave Node

- Only one process communicates with the master node.
- Receives component parameters from the master node.
- Sample it's Data labels.
- Calculate sufficient statistics and send them to the master node..
- Receives Splits/Merges decisions from the master node.

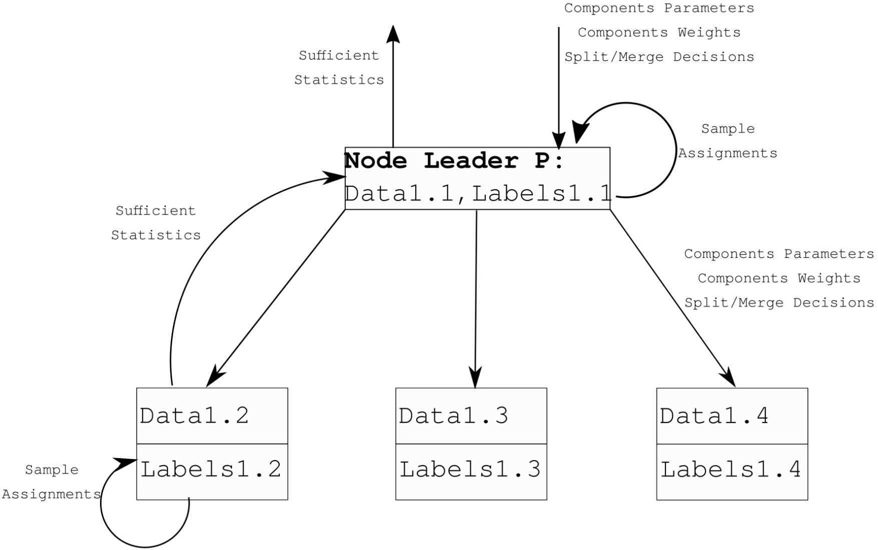
Slave Node

- Only one process communicates with the master node.
- Receives component parameters from the master node.
- Sample it's Data labels.
- Calculate sufficient statistics and send them to the master node..
- Receives Splits/Merges decisions from the master node.
- Execute Split/Merge decisions.

Architecture - Cluster



Architecture - Node



Implementation Key Ingredients

- Abstract data structures `distribution_hyper_params`, `sufficient_statistics`, `distibution_sample` defines a prior, implementing a new prior require all 3 (and the required functions).

Implementation Key Ingredients

- Abstract data structures `distribution_hyper_params`, `sufficient_statistics`, `distibution_sample` defines a prior, implementing a new prior require all 3 (and the required functions).
- `dp-parallel-sampling.jl` is the wrapper for the model, it supplies the API for running, loading/saving checkpoints, statistics.

Implementation Key Ingredients

- Abstract data structures `distribution_hyper_params`, `sufficient_statistics`, `distibution_sample` defines a prior, implementing a new prior require all 3 (and the required functions).
- `dp-parallel-sampling.jl` is the wrapper for the model, it supplies the API for running, loading/saving checkpoints, statistics.
- `global_params.jl` Defines the parameters for running the model.

Implementation Key Ingredients

- Abstract data structures `distribution_hyper_params`, `sufficient_statistics`, `distibution_sample` defines a prior, implementing a new prior require all 3 (and the required functions).
- `dp-parallel-sampling.jl` is the wrapper for the model, it supplies the API for running, loading/saving checkpoints, statistics.
- `global_params.jl` Defines the parameters for running the model.

```julia
abstract type distribution_hyper_params end
abstract type sufficient_statistics end
abstract type distibution_sample end
```

```julia
include("distributions/niw.jl")

random_seed = nothing

#Data Loading specifics
data_path = "/path/to/data/"
data_prefix = "data"

#Model Parameters
iterations = 32
hard_clustering = false
model_save_interval = 1000
initial_clusters = 1
total_dim = 2
α = 1.0

hyper_params = niw_hyperparams(1.0,
    zeros(total_dim),
    total_dim+3.0,
    Matrix{Float64}(I, total_dim, total_dim)*1.0)
```

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.
- `julia -p 8 --machine-file machinelist`

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.
- `julia -p 8 --machine-file machinelist`

```
8* lab1105l
8* lab1105j
8* lab1105f
```

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.
- `julia -p 8 --machine-file machinelist`

```
8* lab1105l
8* lab1105j
8* lab1105f
```

- Will start Julia with 8 processes on each node. All will be available to the user with same ease as a single machine multi-process.

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.
- `julia -p 8 --machine-file machinelist`

```
8* lab1105l
8* lab1105j
8* lab1105f
```

- Will start Julia with 8 processes on each node. All will be available to the user with same ease as a single machine multi-process.
- For each node we will choose one process as the 'Node Leader' process.

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.
- `julia -p 8 --machine-file machinelist`

```
1  8* lab1105l
2  8* lab1105j
3  8* lab1105f
```

- Will start Julia with 8 processes on each node. All will be available to the user with same ease as a single machine multi-process.
- For each node we will choose one process as the 'Node Leader' process.
- Note that the Master node 'Master' process, and its 'Node Leader' process are separated.

Distributing the model

- Using Julia's "Distributed" and "DistributedArrays" packages.
- `julia -p 8 --machine-file machinelist`

```
8* lab1105l
8* lab1105j
8* lab1105f
```

- Will start Julia with 8 processes on each node. All will be available to the user with same ease as a single machine multi-process.
- For each node we will choose one process as the 'Node Leader' process.
- Note that the Master node 'Master' process, and its 'Node Leader' process are separated.
- 'Node Leader' can be turned off if required.

Results

For low-dimensional Gaussians: the previous method still wins

Cores $\times$ Machines	C++ [Chang & Fisher, NIPS '13]	Julia [this work]
1$\times$1	55.87	132.88
2$\times$1	35.48	78.28
4$\times$1	16.45	42.48
8$\times$1	**10.21**	32.95
8$\times$2	–	17.56
8$\times$3	–	16.73
8$\times$4	–	12.93

Table 1: Time (in [sec]) for running 100 DP-GMM iterations with $d = 2, N = 10^6, K = 6$.

Results

For high-dimensional Gaussians: the proposed method wins even when using only a single machine

Cores $\times$ Machines	C++ [Chang & Fisher, NIPS '13]	Julia [this work]
1×1	1637.52	416.40
2×1	720.29	232.62
4×1	480.50	139.86
8×1	262.41	94.64
8×2	–	53.01
8×3	–	39.30
8×4	–	**35.68**

Table 2: Time (in [sec]) for running 100 DP-GMM iterations of $d = 30, N = 10^6, K = 6$.

Conclusion based on our Perspective as ML Researchers

- We don't claim that Julia is faster/better than X.

Conclusion based on our Perspective as ML Researchers

- We don't claim that Julia is faster/better than X.
- Distributed implementations in Julia, ours included, offers a practical and monetary value due to the ease of development and abstraction level.
- We have extended the existing model, creating a fast, scalable, easy to use tool for DP-MM.
- The code will be available next month at: `https://github.com/dinarior/dpmm_subclusters.jl`

The Chinese Restaurant Process

Choosing a table for a new customer:

$$x_i | x_{-i} \sim CRP(\alpha, G_0) = \begin{cases} X_j & \frac{|X_{-i,j}|}{n-1+\alpha} \\ X_{K+1} \sim G_0 & \frac{\alpha}{n-1+\alpha} \end{cases} \quad (13)$$

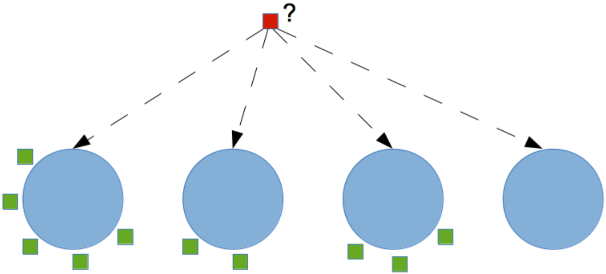
x_{-i} - All customers at the restaurant, excluding customer i

$|X_{-i,j}|$ - Customers count at table 1, excluding customer i.

α - Concentration parameter.

n - Customers count at the rest.

G_0 - Base probability measure.



DP-MM Inference - CRP Sampler

- Inference based on the CRP construction of the DP.

DP-MM Inference - CRP Sampler

- Inference based on the CRP construction of the DP.
- For points $\mathbf{x} = \{x_1, ...x_n\}$, labels $\mathbf{z} = \{z_1, ...z_n\}$, mixture components θ and α, G_0 DP hyperparams we define the sampler:

DP-MM Inference - CRP Sampler

- Inference based on the CRP construction of the DP.
- For points $\mathbf{x} = \{x_1, ...x_n\}$, labels $\mathbf{z} = \{z_1, ...z_n\}$, mixture components θ and α, G_0 DP hyperparams we define the sampler:
- Sample labels z for all points using:

$$z_i \sim DP - MM(\alpha, G_0) = \begin{cases} z_i = j & n_{-i,j} \cdot F_\theta(x_i|\theta_j) \\ z_i = K+1 & \alpha \cdot F_\theta(x_i|\theta_{K+1}) \end{cases} \tag{14}$$

DP-MM Inference - CRP Sampler

- Inference based on the CRP construction of the DP.
- For points $\mathbf{x} = \{x_1, ...x_n\}$, labels $\mathbf{z} = \{z_1, ...z_n\}$, mixture components θ and α, G_0 DP hyperparams we define the sampler:
- Sample labels z for all points using:

$$z_i \sim DP - MM(\alpha, G_0) = \begin{cases} z_i = j & n_{-i,j} \cdot F_\theta(x_i|\theta_j) \\ z_i = K + 1 & \alpha \cdot F_\theta(x_i|\theta_{K+1}) \end{cases} \tag{14}$$

- Sample mixture components parameters conditioned on the current state of the model:

$$\theta_k | x,\ z,\ G_0 \tag{15}$$

References I

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah.
Julia: A fresh approach to numerical computing.
SIAM Review, 59(1):65–98, 2017.

[2] M. D. Escobar and M. West.
Bayesian density estimation and inference using mixtures.
Journal of the american statistical association, 90(430):577–588, 1995.

[3] T. S. Ferguson.
A Bayesian analysis of some nonparametric problems.
The Annals of Statistics, 1973.