STAT 535
Exercise 3

Saifuddin Syed
85258144

April 5, 2018

Acknowledgement: I worked with Esten, Vaden, and Creagh on this assignment.

1 Implementing samplers over matchings

Basic sampler for permutations

Let π be the uniform distribution over S_n, the set of permutations, so for all $x \in S_n$, we have $\pi(x) \propto 1$. We will view x as a bijective function from $[n]$ to $[n]$ where $[n] = \{1, \ldots, n\}$. Given $x \in S_n$ and $i, j \in [n]$, we define $x_{(i,j)} \in S_n$ by swapping the output of i and j in x, or more formally as,

$$x_{(i,j)}(k) = \begin{cases} x(j) & k = i \\ x(i) & k = j \\ x(k) & k \in [n] \backslash \{i, j\} \end{cases}.$$

We define the proposal kernel $Q(x, y)$ for the Metropolis-Hastings algorithm as follows,

$$Q(x, y) = \begin{cases} \frac{1}{n^2} & , y = x_{(i,j)} \text{ for some } i, j \in [n] \\ 0 & \text{otherwise} \end{cases}.$$

Note that $Q(x, y) = Q(y, x)$ so it is reversible, and given $x, x' \in S_n$, one to get from x to x' within a finite number of swaps. Thus, we have Q is irreducible. We will accept this proposal with probability,

$$\alpha(x, y) = 1 \wedge \frac{\pi(y)Q(x, y)}{\pi(x)Q(y, x)} = 1$$

Thus in the Metropolis Hastings algorithm will accept every proposal.

Non-uniform case

The only difference now is that $\pi(x) \propto \gamma(x)$ for some known γ. We use the same proposal $Q(x, y)$ as the uniform case but now, we accept our proposal with probability,

$$\alpha(x, y) = 1 \wedge \frac{\pi(y)Q(x, y)}{\pi(x)Q(y, x)} = 1 \wedge \frac{\gamma(y)}{\gamma(x)}.$$

Implementation

Our implementation in `PermutationSampler.execute` is as follows.

```
@Override
public void execute(Random rand) {
  // Fill this.
   int n = this.permutation.componentSize();
   int i = rand.nextInt(n);
   int j = rand.nextInt(n);
   double log_pi_current = logDensity();
   Collections.swap(this.permutation.getConnections(), i, j);
   double log_pi_new = logDensity();

   boolean accept_proposal =
       Generators.bernoulli(rand,Math.exp(log_pi_new-log_pi_current));

   if (!accept_proposal) {
      Collections.swap(this.permutation.getConnections(), i, j);
   }
}
```

Understanding the test

Let $K_1, \ldots, K_m$ be our Kernels. We test for invariance for each kernel individually since if each K_i leaves π invariant, we can ensure that the resulting kernel $K = K_m \cdots K_1$ leaves π invariant.

However is it is possible to have each kernel K_i is not irreducible, but their product are, that is why to ensure the irreducibility, one needs to check $K = K_m \cdots K_1$ is irreducible.

1.1 Bipartite matching (non-perfect)

Let M_n be the set of matchings between the ordered sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$. We can encode $x = (x_1, \ldots, x_n)$ where $x_i \in [n] \cup \{0\}$, where $x_i = j$ implies there is a connection between a_i and b_j if $j > 0$, and $x_i = 0$ implies a_i has no connection. We also required $x_i \neq x_j$ if $x_i, x_j \neq 0$. Suppose we wish to sample form a distribution π over M_n with $\pi(x) \propto \gamma(x)$ for $x \in M_n$.

We will construct a proposal scheme for the Metropolis-Hastings algorithm as follows. Let $x \in M_n$, and $x_B = \{j \in [n] : x_i \neq j, \forall i \in [n]\}$. Suppose we sample i uniformly from $[n]$. We propose $y \in M_n$ such that

$$y_k = \begin{cases} y_k = j & k = i \\ y_k = x_k & k \in [n] \backslash \{i\} \end{cases}$$

where j is chosen uniformly from x_B if $x_i = 0$, and j is is uniformly from $x_B \cup \{0\}$ if $x_i > 0$.

Let $Q(x, y)$ be the kernel resulting from the above procedure. It is trivial to see that Q is irreducible as our scheme can add, remove and change connections. We will it is also reversible.

If $x_i = 0$ then, then a connection is made with probability $1/|x_B|$ and $y_i > 0$ and $|y_B| = |x_B| - 1$, so the same connection is removed resulting in x with probability $1/(|y_B| + 1) = 1/|x_B|$ and $Q(y, x) = Q(x, y)$.

Similarly if $x_i = j > 0$ and $y_i = k \in x_B$ with probability $1/(|x_B| + 1)$ and $j \in y_B$ with $|y_B| = |x_B|$, so given y, we will propose x with probability$1/(|y_B| + 1) = 1/(|x_B| + 1)$ and $Q(y, x) = Q(x, y)$. Finally, if $x_i = j > 0$ and $y_i = 0$, then $|y_B| = |x_B| + 1$ and given y we will propose x with probability $1/|y_B| = 1/(|x_B| + 1)$, and $Q(y, x) = Q(y, x)$.

Therefore to sample from π we accept our proposal with acceptance ratio,

$$\alpha(x, y) = 1 \wedge \frac{\pi(y)Q(x, y)}{\pi(x)Q(y, x)} = 1 \wedge \frac{\gamma(y)}{\gamma(x)}.$$

We implemented this in `BipartiteMatching.xtend` as follows:

```
@Override
  public void execute(Random rand) {
     int n = this.matching.componentSize();
     int i = rand.nextInt(n);
     int m = this.matching.free2().size();
     double log_pi_current = logDensity();
     int connection_i = this.matching.getConnections().get(i);

     if (connection_i == BipartiteMatching.FREE) {
        int j = rand.nextInt(m);
        this.matching.getConnections().set(i, matching.free2().get(j));
     }
     else {
        int j = rand.nextInt(m+1);
        if (j == m) {
           this.matching.getConnections().set(i, BipartiteMatching.FREE);
        }
        else {
           this.matching.getConnections().set(i, matching.free2().get(j));
        }
     }
```

```
    double log_pi_new = logDensity();
    boolean accept_proposal =
       Generators.bernoulli(rand,Math.exp(log_pi_new-log_pi_current));

    if (!accept_proposal) {
      this.matching.getConnections().set(i, connection_i);
      }
  }
```

2 A statistical model involving a combinatorial space

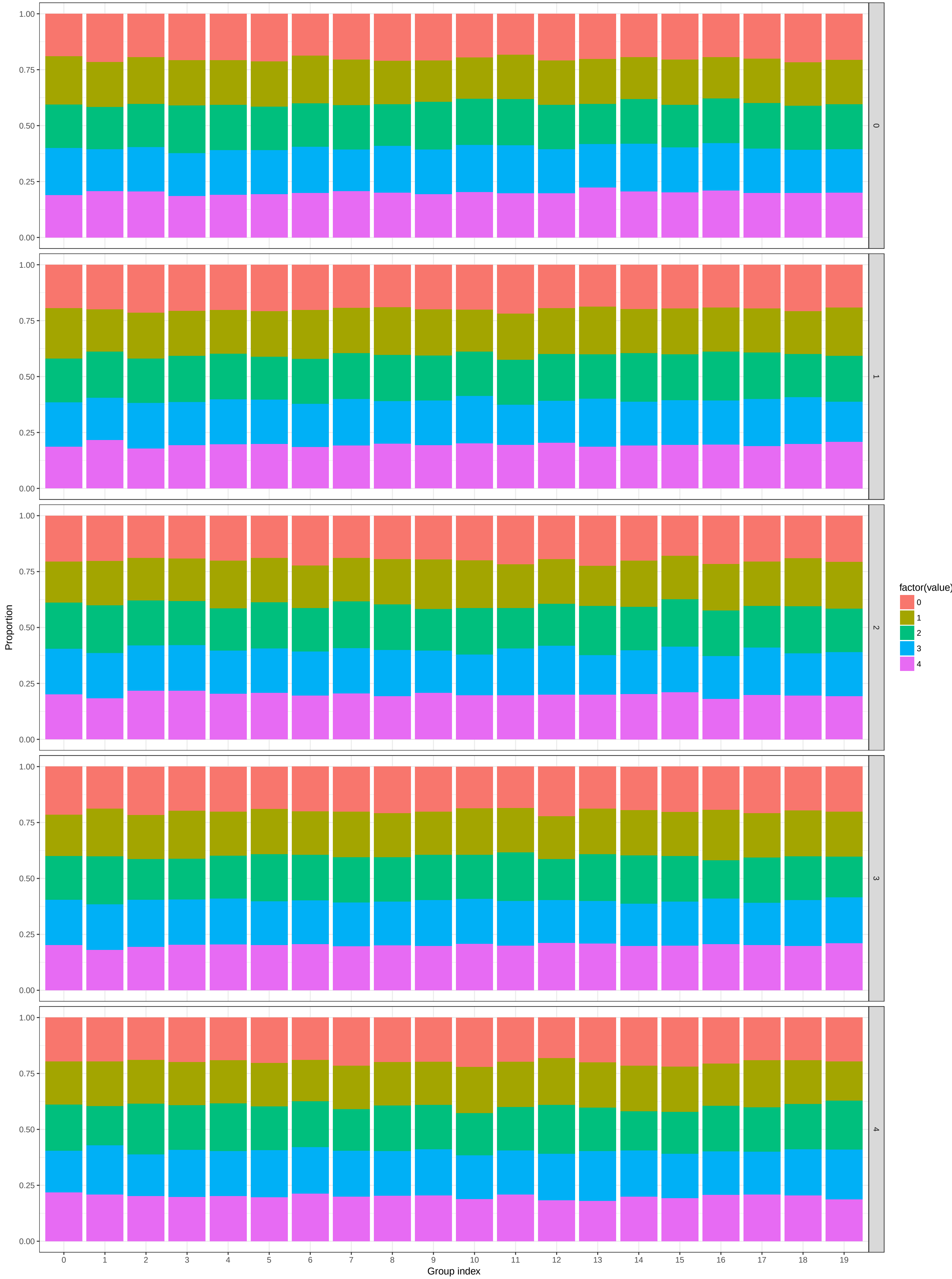
Implementation

The implementation of the model into `PermutedClustering.bl` is as follows:

```
  laws {

//    Initialize means
   means.get(0) ~ ContinuousUniform(0,1)
   for (int j : 1 ..< groupSize){
      means.get(j) | RealVar mean_last = means.get(j-1) ~
         ContinuousUniform(mean_last, mean_last + 1)
   }

//    Initialize Variances
    for (int j : 0 ..< groupSize){
      variances.get(j) ~ Exponential(10)
   }

//    Initialize permutaitons
   for (int i : 0 ..< nGroups){
      permutations.get(i) ~ UniformPermutation() }

   for (int i : 0 ..< nGroups){
      for (int j : 0 ..< groupSize){
         observations.getRealVar(i,j) | means, variances,
            int permutation = permutations.get(i).getConnections().get(j)
         ~ Normal(means.get(permutation), variances.get(permutation))
      }
   }
  }
```

Run your model on synthetic data

We include the plot of the posterior statistics on inferred permutations below:


Proportion
Group index
factor(value)
0
1
2
3
4
0.00
0.25
0.50
0.75
1.00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19