Version 6: Cluster first

Simulate data

```
I <- 100
K <- 10
S <- 3

set.seed(123)

pi <- rep(0.1, 10)
#z <- sample(1:K, size = I, replace = T, prob = pi)
z <- rep(1:10, each=10)
w <- matrix(c(0.98, 0.99, 0.97,
              0.98, 0.90, 0.82,
              0.55, 0.00, 0.80,
              0.20, 0.00, 0.50,
              0.30, 0.00, 0.30,
              0.43, 0.90, 0.00,
              0.30, 0.70, 0.00,
              0.20, 0.00, 0.00,
              0.00, 0.00, 0.30,
              0.00, 0.50, 0.00),
            byrow=T,
            nrow=K, ncol=S)

colnames(w) <-  paste0("sample", 1:S)
w
```

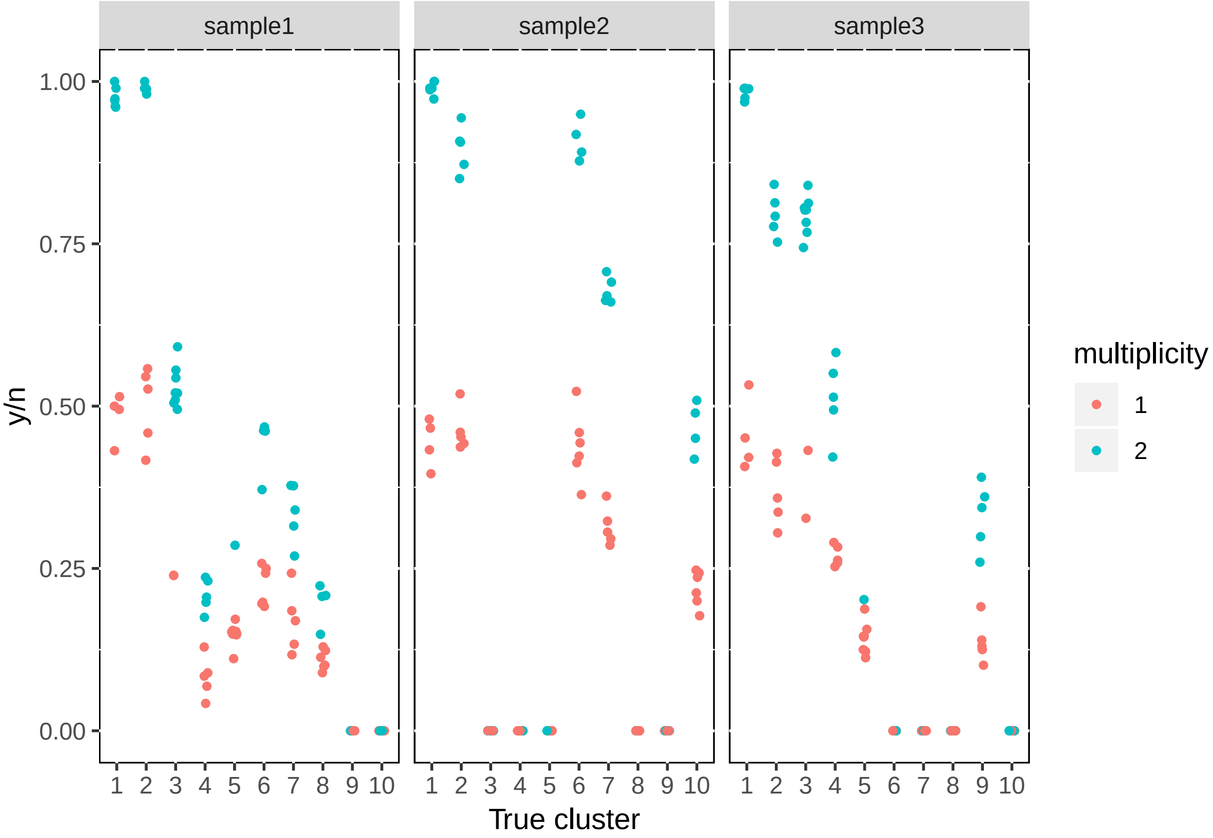
```
##       sample1 sample2 sample3
##  [1,]    0.98    0.99    0.97
##  [2,]    0.98    0.90    0.82
##  [3,]    0.55    0.00    0.80
##  [4,]    0.20    0.00    0.50
##  [5,]    0.30    0.00    0.30
##  [6,]    0.43    0.90    0.00
##  [7,]    0.30    0.70    0.00
##  [8,]    0.20    0.00    0.00
##  [9,]    0.00    0.00    0.30
## [10,]    0.00    0.50    0.00
```

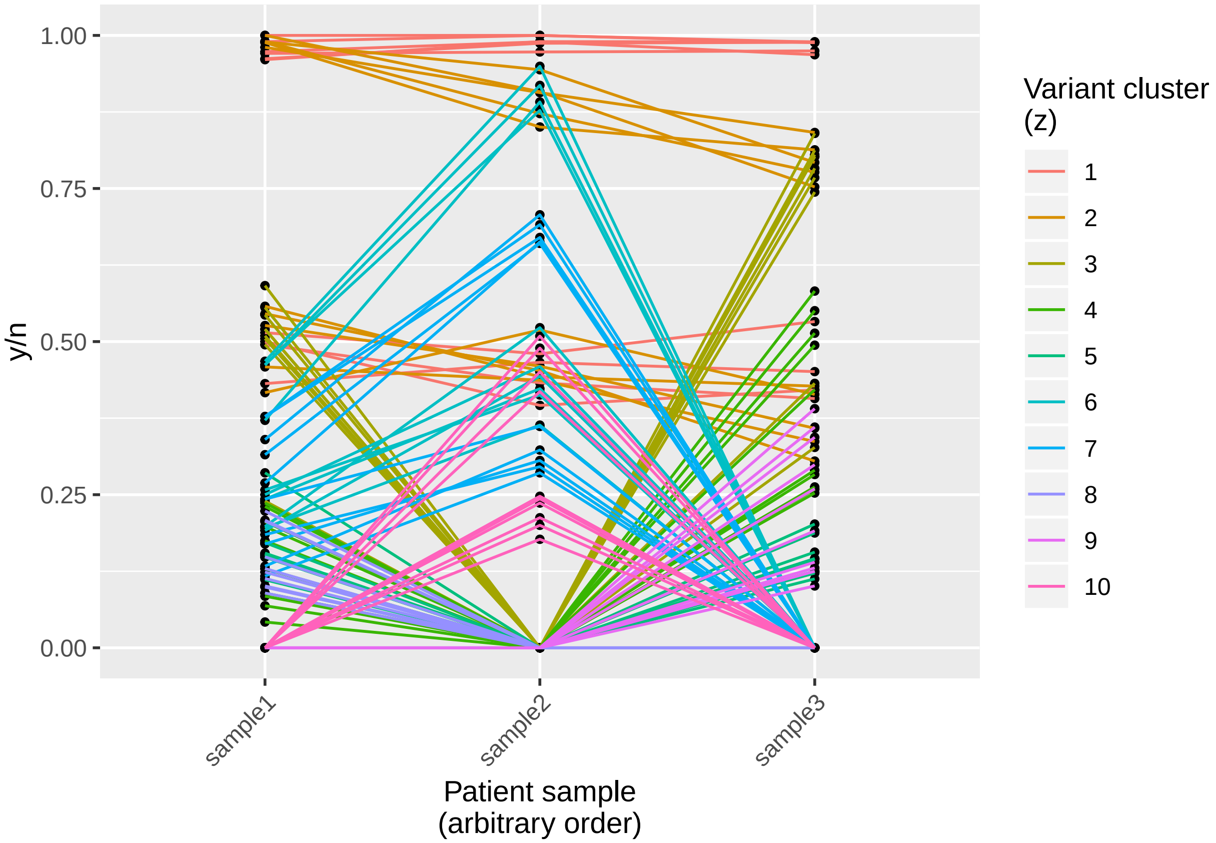
```
tcn <- matrix(2, nrow=I, ncol=S)
m <- matrix(rep(sample(1:2, size = I, replace = T), S),
            nrow=I, ncol=S)
W <- w[z, ]
calcTheta <- function(m, tcn, w) {
  (m * w) / (tcn * w + 2*(1-w))
}

theta <- calcTheta(m, tcn, W)

n <- replicate(S, rpois(I, 100))
y <- matrix(NA, nrow=I, ncol=S)
```

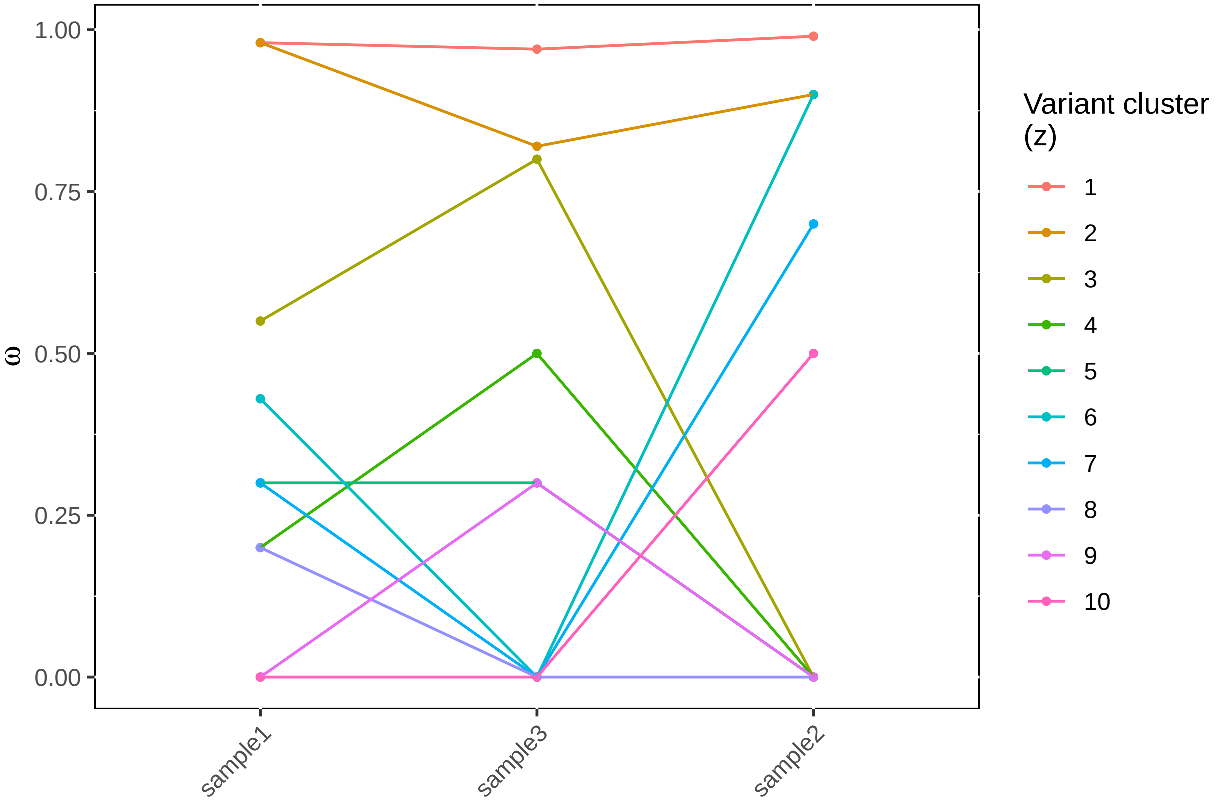
```
for (i in 1:I) {
  for (s in 1:S) {
    y[i, s] <- rbinom(1, n[i, s], theta[i,s])
  }
}

test.data <- list("I" = I, "S" = S, "K" = K,
                  "y" = y, "n" = n,
                  "m" = m, "tcn" = tcn)
```

Visualize densities of simulated data

Clustering is by ω




1.00
0.75
0.50
0.25
0.00
y/n
sample1
sample2
sample3
Patient sample
(arbitrary order)
Variant cluster
(z)
1
2
3
4
5
6
7
8
9
10


1.00
0.75
0.50
0.25
0.00
ω
sample1
sample3
sample2
Variant cluster
(z)
1
2
3
4
5
6
7
8
9
10

functions

```
runMCMC <- function(data, K, jags.file, inits, params, n.iter, thin) {
  data$K <- K
  jags.m <- jags.model(jags.file, data,
                       n.chains = 1,
                       inits = inits,
                       n.adapt = 1000)
  samps <- coda.samples(jags.m, params, n.iter=n.iter, thin=thin)
  samps
}

getParamChain <- function(samps, param) {
  chains <- do.call(rbind, samps)
  chain <- chains[, grep(param, colnames(chains))]
}

reshapeW <- function(w, S, K) {
  w.mat <- matrix(w, nrow = K)
  colnames(w.mat) <- paste0("sample", 1:S)
  w.mat
}

calcLogLik <- function(z.iter, w.iter, data) {
  W <- w.iter[z.iter, ]
  theta <- calcTheta(data$m, data$tcn, W)
  sum(dbinom(data$y, data$n, theta, log=T))
}

calcChainLogLik <- function(samps, data, K) {
  z.chain <- getParamChain(samps, "z")
  w.chain <- getParamChain(samps, "w")
  lik <- c()
  for(iter in 1:nrow(z.chain)) {
    z.iter <- z.chain[iter,]
    w.iter <- reshapeW(w.chain[iter,], data$S, K)
    lik <- c(lik, calcLogLik(z.iter, w.iter, data))
  }
  mean(lik)
}

calcBIC <- function(n, k, ll) log(n)*k - 2*ll
```

Cluster – JAGS

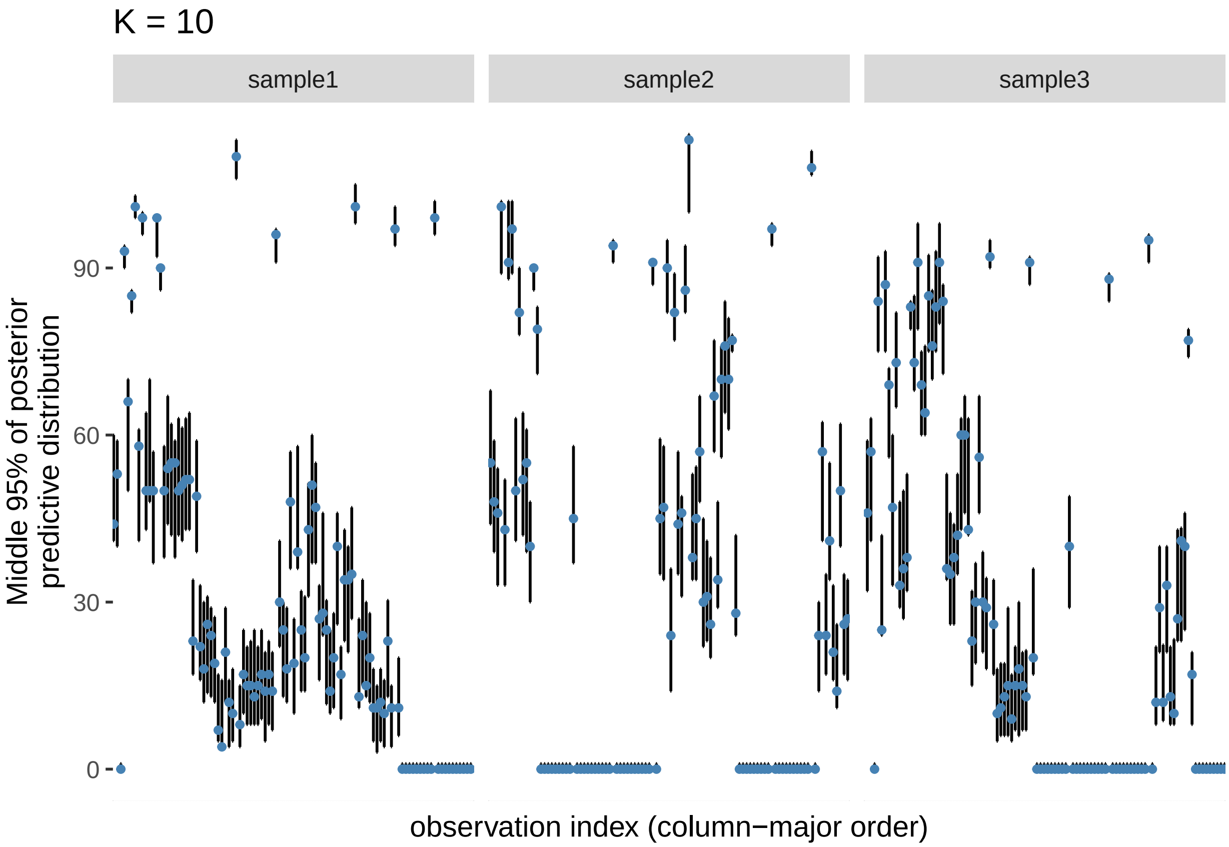
```
jags.file <- file.path(models.dir, "w.jags")
inits <- list(".RNG.name" = "base::Wichmann-Hill",
              ".RNG.seed" = 123)
test.data <- list("I" = I, "S" = S,
                  "y" = y, "n" = n,
                  "m" = m, "tcn" = tcn)
params <- c("z", "w", "ystar")
```

```
n.iter = 10000
thin = 7
K <- 10

samps <- runMCMC(test.data, K, jags.file, inits, params, n.iter, thin)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 300
##    Unobserved stochastic nodes: 431
##    Total graph size: 4196
##
## Initializing model
```

```
z.chain <- getParamChain(samps, "z")
w.chain <- getParamChain(samps, "w")
```

PPD



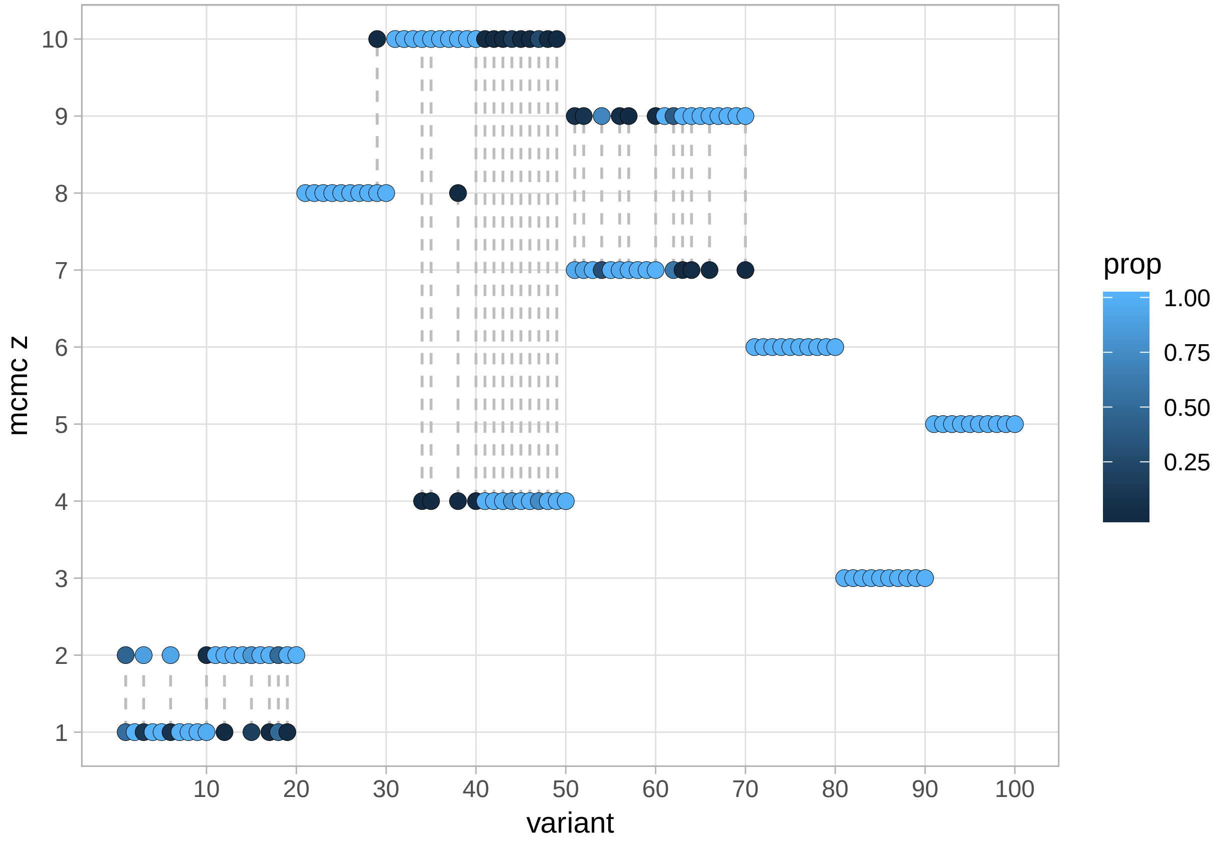
Z

```
plot.z <- function(samps, z) {
  mcmc_vals <- summary(samps)$statistics
  mcmc_z <- as.vector(mcmc_vals[substr(rownames(mcmc_vals), 1, 1) == "z", "Mean"])
```

```r
  plot(z, mcmc_z, type = "p")
  z_comp <- data.frame(z, mcmc_z)
  #plot.new()
  #grid.table(z_comp, rows=NULL)
  #z_mapping <- distinct(round(z_comp, 0))
  #z_mapping <- z_mapping[order(z_mapping$mcmc_z), ]
  #plot.new()
  #grid.table(z_mapping, rows=NULL)
}

z.chain.to.tb <- function(z.chain) {
  z.chain.tb <- z.chain %>%
    as_tibble() %>%
    mutate(iter=1:nrow(z.chain)) %>%
    gather(variant, mcmc_z, -c(iter))
  z.chain.tb <- z.chain.tb %>%
    mutate(variant = as.integer(gsub(".*\\[(.*)\\].*", "\\1", z.chain.tb$variant))) %>%
    mutate(true_z = rep(1:10, each=nrow(z.chain)*10)) %>%
    group_by(variant, mcmc_z) %>%
    mutate(count = n())
  z.chain.tb_simp <- distinct(select(z.chain.tb, -c(iter)))
  z.chain.tb_simp %>%
    group_by(variant) %>%
    mutate(prop = count/sum(count))
}

z.chain.tb <- z.chain.to.tb(z.chain)
z.chain.tb
```

```
## # A tibble: 135 x 5
## # Groups:   variant [100]
##    variant mcmc_z true_z count   prop
##      <int>  <dbl>  <int> <int>  <dbl>
##  1       1      2      1   657 0.460
##  2       1      1      1   771 0.540
##  3       2      1      1  1428 1
##  4       3      2      1  1260 0.882
##  5       3      1      1   168 0.118
##  6       4      1      1  1428 1
##  7       5      1      1  1428 1
##  8       6      2      1  1315 0.921
##  9       6      1      1   113 0.0791
## 10       7      1      1  1428 1
## # ... with 125 more rows
```

```r
z.seg.tb <- tibble(variant = numeric(),
                   mcmc_z_1 = numeric(),
                   mcmc_z_2 = numeric())
for (i in 1:ncol(z.chain)) {
  z.vals <- as.integer(names(table(z.chain[,i])))
  if (length(z.vals) > 1) {
    z.seg.tb[i, ] <- c(i, z.vals[1], z.vals[2])
  } else {
    z.seg.tb[i, ] <- c(i, z.vals, z.vals)
  }
}
#z.seg.tb
z.plot <- ggplot(z.chain.tb, aes(variant, mcmc_z)) +
  ylab("mcmc z") +
  xlab("variant") +
  theme_light() +
  scale_y_continuous(breaks = 1:K, minor_breaks=NULL) +
  scale_x_continuous(breaks = seq(10,100,10), minor_breaks=NULL) +
  geom_segment(data = z.seg.tb,
               aes(x=variant, xend=variant,
                   y=mcmc_z_1, yend=mcmc_z_2),
                   color="gray", linetype=2) +
  geom_point(aes(y=mcmc_z, fill=prop),
             pch=21, size=3, stroke=0)

z.plot
```



```
ggsave(file.path(figs.dir, "zplot.pdf"), z.plot, width=14, height=6)

# mcmc_vals <- summary(samps)$statistics
# mcmc_z <- as.vector(mcmc_vals[substr(rownames(mcmc_vals), 1, 1) == "z", "Mean"])
# plot.z(samps, z)
```

```
z.map.tb <- z.chain.tb %>%
  group_by(variant) %>%
  filter(prop == max(prop))
```

```
z.map.tb
```

```
## # A tibble: 100 x 5
## # Groups:   variant [100]
##    variant mcmc_z true_z count  prop
##      <int>  <dbl>  <int> <int> <dbl>
##  1       1      1      1   771 0.540
##  2       2      1      1  1428 1
##  3       3      2      1  1260 0.882
##  4       4      1      1  1428 1
##  5       5      1      1  1428 1
##  6       6      2      1  1315 0.921
##  7       7      1      1  1428 1
##  8       8      1      1  1428 1
##  9       9      1      1  1428 1
## 10      10      1      1  1381 0.967
## # ... with 90 more rows
```

```
z.map <- z.map.tb$mcmc_z
z.map
```

```
##   [1]  1  1  2  1  1  2  1  1  1  1  2  2  2  2  2  2  2  2  2  2  8  8  8
##  [24]  8  8  8  8  8  8  8 10 10 10 10 10 10 10 10 10 10  4  4  4  4  4  4
##  [47]  4  4  4  4  7  7  7  9  7  7  7  7  7  7  9  7  9  9  9  9  9  9  9
##  [70]  9  6  6  6  6  6  6  6  6  6  6  3  3  3  3  3  3  3  3  3  3  5  5
##  [93]  5  5  5  5  5  5  5  5
```

```
z.map.ind <- which(apply(z.chain, 1, function(x) all(x == z.map)))
w.chain.map <- w.chain[z.map.ind, ]
w.map.tb <- w.chain.map %>%
  as_tibble() %>%
  mutate(iter=1:nrow(w.chain.map)) %>%
  gather(ind, mcmc_w, -c(iter)) %>%
  group_by(ind) %>%
  summarize(mean_w = mean(mcmc_w))
w.map.tb <- w.map.tb %>%
  mutate(cluster = as.integer(gsub(".*\\[(.*)\\,.*", "\\1", w.map.tb$ind))) %>%
  mutate(sample = as.integer(gsub(".*\\,(.*)\\]", "\\1", w.map.tb$ind))) %>%
  arrange(cluster)
#w.map.tb
w.map <- matrix(data=w.map.tb$mean_w, nrow=10, ncol=3, byrow=TRUE)
w.map
```

```
##              [,1]        [,2]         [,3]
##  [1,] 0.973637944 0.988597020 0.9820710869
##  [2,] 0.986416825 0.898308330 0.7872189454
##  [3,] 0.001522582 0.001459451 0.3172062772
##  [4,] 0.298899414 0.001986741 0.2629240648
##  [5,] 0.001516133 0.456360105 0.0009553129
##  [6,] 0.204992711 0.001354114 0.0010398326
##  [7,] 0.446984100 0.906262831 0.0012807779
##  [8,] 0.526633332 0.001433847 0.7948906517
##  [9,] 0.333174387 0.666250526 0.0011254626
## [10,] 0.195689330 0.001694563 0.5222219276
```

```r
init.admat <- function(w, zero.thresh) {
  cluster.sample.presence <- apply(w, 1, function(x) which(x>zero.thresh))
  K <- nrow(w)
  S <- ncol(w)
  all.samples <- 1:S
  admat <- matrix(data=0, nrow=(1+K), ncol=K) # rows=from is root + 1:K, cols=to is 1:K

  # fill in restraints
  # can go from root to anyone, skip and start at nrow=2 (cluster 1)
  for (from in 2:(K+1)) {
    for (to in 1:K) {

      # can't go to self
      if ((from-1) == to) {
        admat[from, to] <- NA
        #print(c(from, to, "self"))
        next
      }
      # hierarchy restraints
      from.samples <- cluster.sample.presence[[from-1]]
      to.samples <- cluster.sample.presence[[to]]

      ## no restraints if same sample presence
      if (setequal(from.samples, to.samples)) {
        #print(c(from, to, "same"))
        next
      }

      ## restraint if # from.samples < # to.samples
      if(length(from.samples) < length(to.samples)) {
        #print(c(from, to, "from set is smaller than to set"))
        admat[from, to] <- NA
        next
      }

      ## no restraints if to.samples is subset of from.samples
      if (all(to.samples %in% from.samples)) {
        #print(c(from, to, "subset"))
        next
      } else {
        #print(c(from, to, "not subset"))
        admat[from, to] <- NA
      }
    }
  }
  admat
}

# test admat restraints
zero.thresh <- 0.01
test.admat <- init.admat(w, zero.thresh)

test.admat
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    0    0    0    0    0    0    0    0     0
##  [2,]   NA    0    0    0    0    0    0    0    0     0
##  [3,]    0   NA    0    0    0    0    0    0    0     0
##  [4,]   NA   NA   NA    0    0   NA   NA    0    0    NA
##  [5,]   NA   NA    0   NA    0   NA   NA    0    0    NA
##  [6,]   NA   NA    0    0   NA   NA   NA    0    0    NA
##  [7,]   NA   NA   NA   NA   NA   NA    0    0   NA     0
##  [8,]   NA   NA   NA   NA   NA    0   NA    0   NA     0
##  [9,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [10,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [11,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
```

```
rand.admat <- function(admat) {
  for(col in 1:ncol(admat)) {
    ind.0 <- which(admat[,col] == 0) # possible positions (0's)
    rand.ind <- sample(ind.0, size=1)
    admat[rand.ind,col] <- 1
  }
  admat
}
test.admat.2 <- rand.admat(test.admat)
test.admat.2
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    1    0    1    0    0    0    0    1     0
##  [2,]   NA    0    0    0    1    0    0    0    0     1
##  [3,]    0   NA    0    0    0    1    0    0    0     0
##  [4,]   NA   NA   NA    0    0   NA   NA    0    0    NA
##  [5,]   NA   NA    1   NA    0   NA   NA    0    0    NA
##  [6,]   NA   NA    0    0   NA   NA   NA    0    0    NA
##  [7,]   NA   NA   NA   NA   NA   NA    1    1   NA     0
##  [8,]   NA   NA   NA   NA   NA    0   NA    0   NA     0
##  [9,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [10,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [11,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
```

```
mutate.admat <- function(admat) {
  # choose a column to mutate
  K <- ncol(admat)
  rand.k <- sample(1:K, size=1)

  # mutate
  possiblePos <- which(!is.na(admat[,rand.k]))
  ind.1 <- which(admat[,rand.k] == 1)
  new.1 <- sample(possiblePos[-ind.1], size=1)
  admat[ind.1,rand.k] <- 0
  admat[new.1,rand.k] <- 1
  admat
}

test.admat.3 <- mutate.admat(test.admat.2)
test.admat.3
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    1    0    1    0    0    0    0    1     0
##  [2,]   NA    0    0    0    1    0    0    0    0     1
##  [3,]    0   NA    0    0    0    1    0    0    0     0
##  [4,]   NA   NA   NA    0    0   NA   NA    0    0    NA
##  [5,]   NA   NA    1   NA    0   NA   NA    1    0    NA
##  [6,]   NA   NA    0    0   NA   NA   NA    0    0    NA
##  [7,]   NA   NA   NA   NA   NA   NA    1    0   NA     0
##  [8,]   NA   NA   NA   NA   NA    0   NA    0   NA     0
##  [9,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [10,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [11,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
```

```
getChildrenWSum <- function(admat, w, row) {
  curr.row <- admat[row,]
  children <- which(curr.row == 1)
  children.w <- w[children,]
  if(length(children) > 1) {
    return(colSums(children.w))
  } else {
    return(children.w)
  }
}

score.admat <- function(admat, w) {

  S <- ncol(w)
  # root, MCF=1.0
  children.w.sum <- getChildrenWSum(admat, w, 1)
  score <- log(sum(1 >= children.w.sum) / S)

  for(i in 2:nrow(admat)) {
    if(all(is.na(admat[i,]))) next #leaf

    curr.child.sum <- getChildrenWSum(admat, w, i)
    curr.parent.mcf <- w[i-1, ]
    curr.score <- log(sum(curr.parent.mcf >= curr.child.sum) / S)
    score <- score + curr.score
  }
  score
}

true.admat <- test.admat
true.admat[1,1] <- true.admat[2,2] <- true.admat[3,3] <-  true.admat[3,6] <-
  true.admat[4,4] <- true.admat[4,5] <- true.admat[5,8] <- true.admat[6,9] <-
  true.admat[7,7] <- true.admat[8,10] <- 1
true.admat
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    0    0    0    0    0    0    0    0     0
##  [2,]   NA    1    0    0    0    0    0    0    0     0
##  [3,]    0   NA    1    0    0    1    0    0    0     0
##  [4,]   NA   NA   NA    1    1   NA   NA    0    0    NA
##  [5,]   NA   NA    0   NA    0   NA   NA    1    0    NA
##  [6,]   NA   NA    0    0   NA   NA   NA    0    1    NA
##  [7,]   NA   NA   NA   NA   NA   NA    1    0   NA     0
##  [8,]   NA   NA   NA   NA   NA    0   NA    0   NA     1
##  [9,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [10,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
## [11,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
```

```
score.admat(true.admat, w)
```

```
## [1] 0
```