Cluster first, sample w in tree mh

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
```

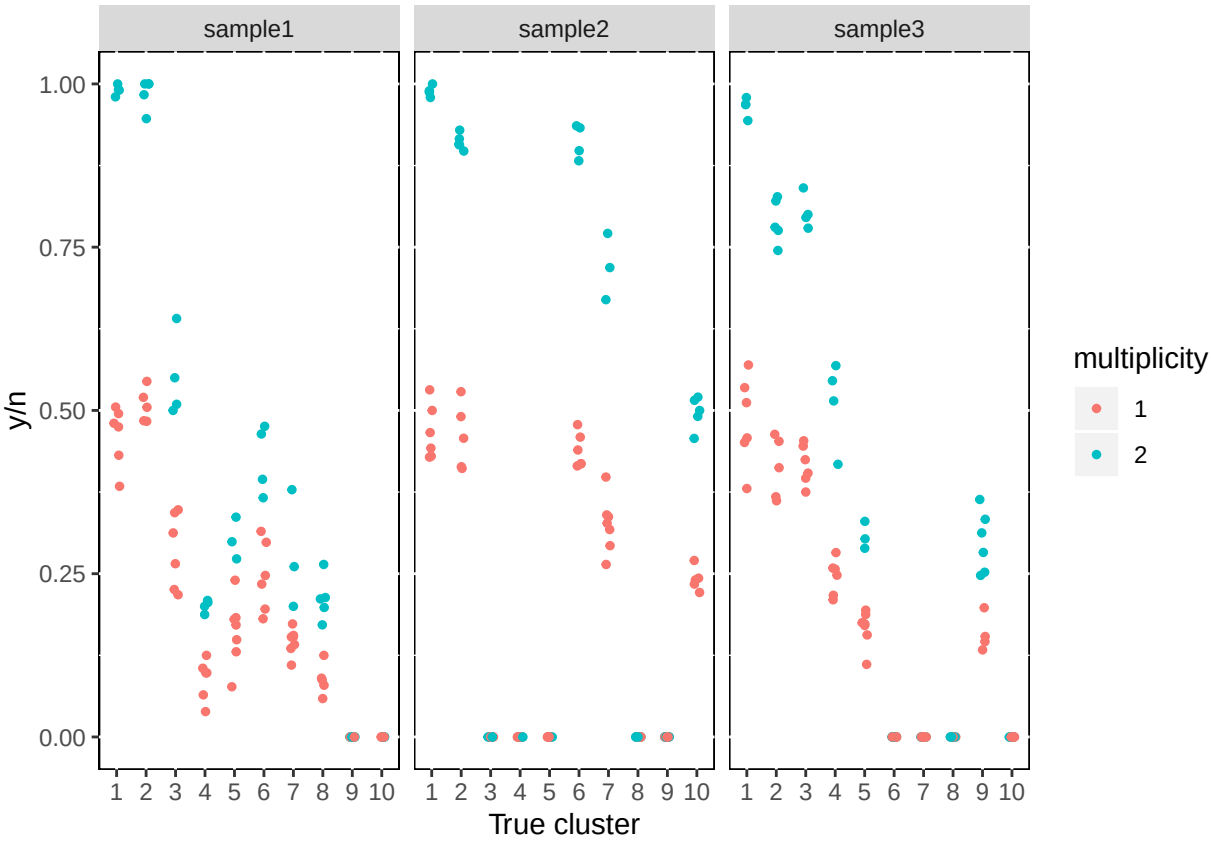
```
##       sample1 sample2 sample3
##  [1,]    0.98    0.99    0.97
##  [2,]    0.98    0.90    0.82
##  [3,]    0.55    0.00    0.80
##  [4,]    0.20    0.00    0.50
##  [5,]    0.30    0.00    0.30
##  [6,]    0.43    0.90    0.00
##  [7,]    0.30    0.70    0.00
##  [8,]    0.20    0.00    0.00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

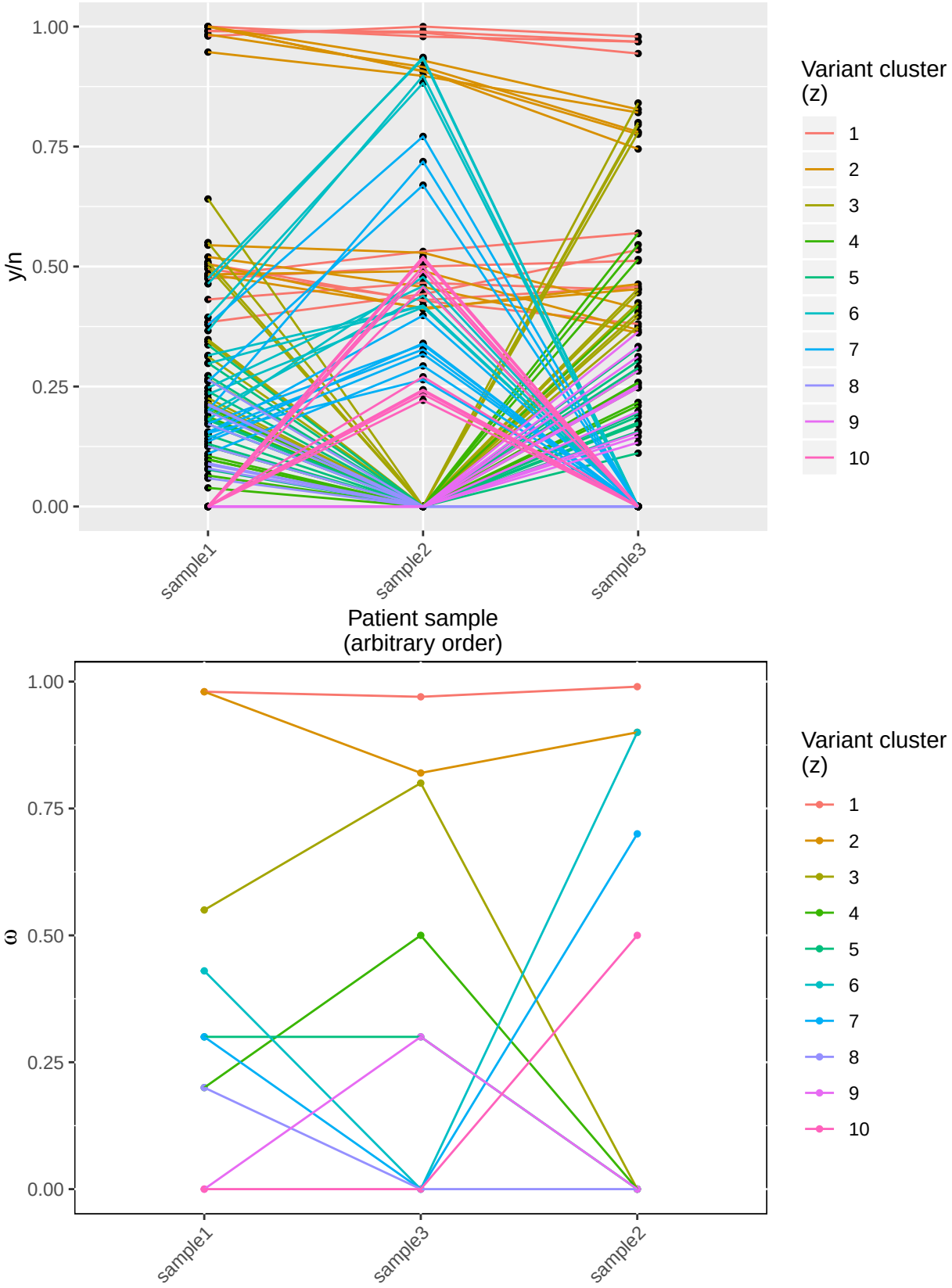
```
tcn <- matrix(2, nrow=I, ncol=S)
m <- matrix(rep(sample(1:2, size = I, replace = T), S),
            nrow=I, ncol=S)
W <- w[z, ]
calcTheta <- function(m, tcn, w) {
  (m * w) / (tcn * w + 2*(1-w))
}
```

```
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
ω
sample1
sample3
sample2

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

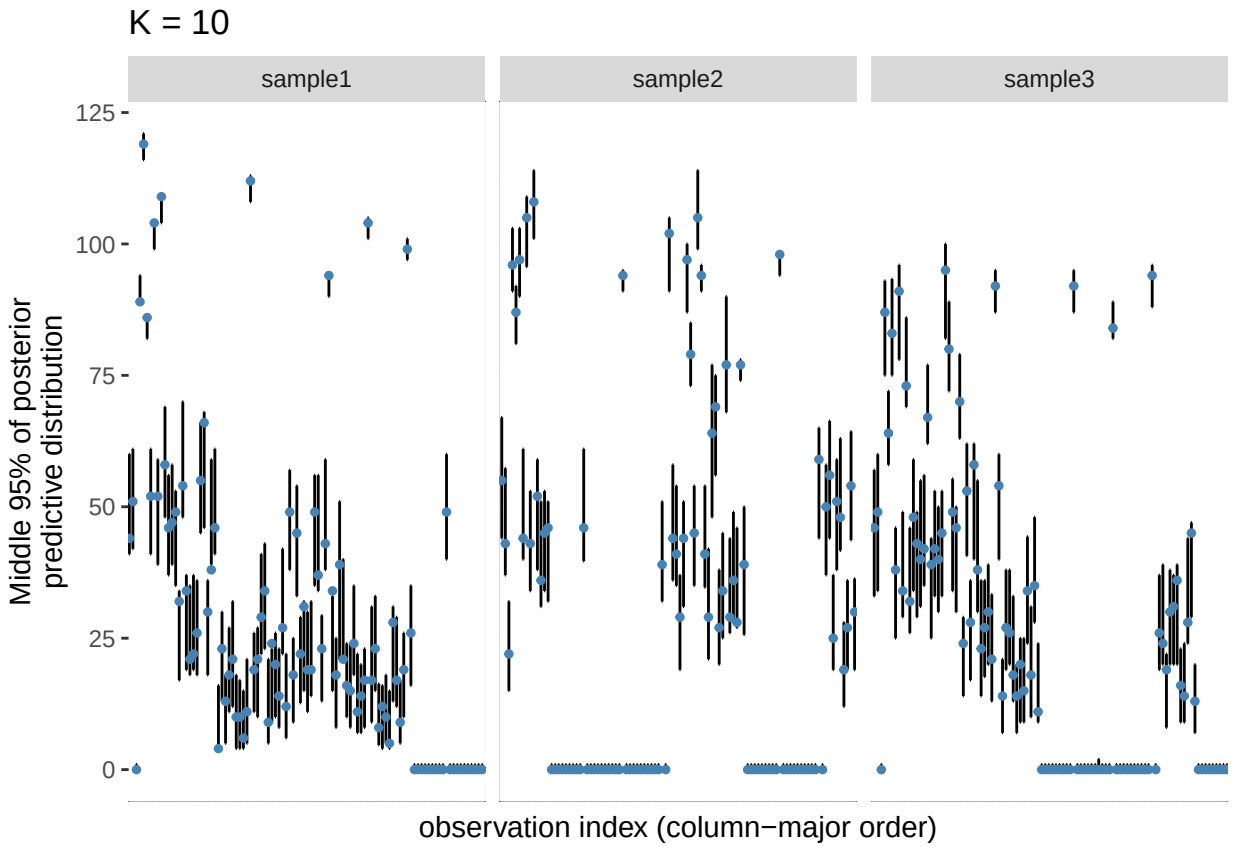
```
jags.file <- file.path(models.dir, "w.jags")
inits <- list(".RNG.name" = "base::Wichmann-Hill",
              ".RNG.seed" = 123)
test.data <- list("I" = I, "S" = S,
                  "y" = y, "n" = n,
                  "m" = m, "tcn" = tcn)
params <- c("z", "w", "ystar")
```

```
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: 4208
##
## Initializing model
```

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

mcmc_vals <- summary(samps)$statistics
mcmc_w <- mcmc_vals[substr(rownames(mcmc_vals), 1, 1) == "w", "Mean"]
mcmc_w <- matrix(mcmc_w, nrow=K)
colnames(mcmc_w) <- paste0("sample", 1:S)
```

PPD



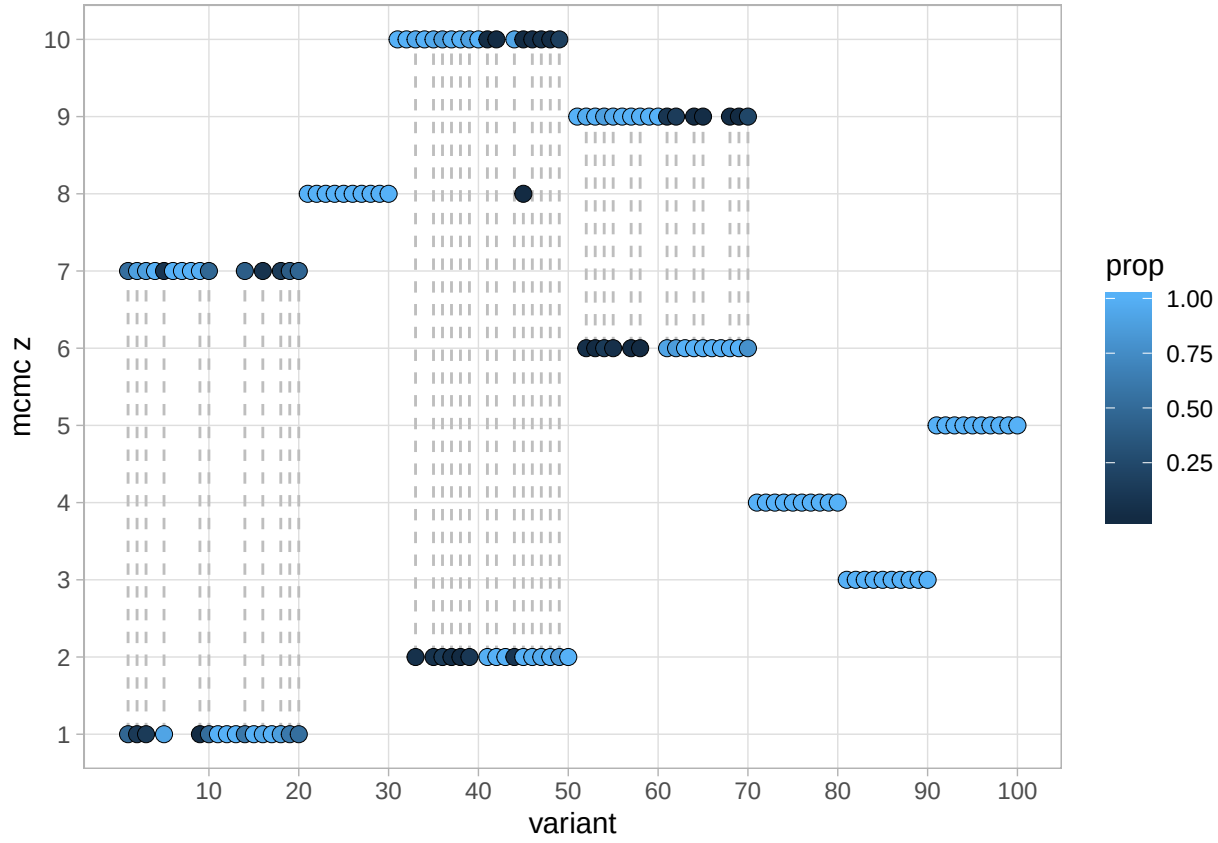
Z

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

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()) %>%
    ungroup() %>%
    mutate(iter = NULL)
  z.chain.tb_simp <- distinct(z.chain.tb)
  z.chain.tb_simp <- z.chain.tb_simp %>%
    group_by(variant) %>%
    mutate(prop = count/sum(count))

  z.chain.tb_simp
}

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

```
## # A tibble: 139 x 5
## # Groups:   variant [100]
##    variant mcmc_z true_z count   prop
##      <int>  <dbl>  <int> <int>  <dbl>
##  1       1      7      1   760 0.532
##  2       1      1      1   668 0.468
##  3       2      1      1   167 0.117
##  4       2      7      1  1261 0.883
##  5       3      7      1  1240 0.868
##  6       3      1      1   188 0.132
##  7       4      7      1  1428 1
##  8       5      1      1  1301 0.911
##  9       5      7      1   127 0.0889
## 10       6      7      1  1428 1
## # ... with 129 more rows
```

```
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
```




```
#gsave(file.path(figs.dir, "z_plot.pdf"), z.plot, width=14, height=6)
```

```
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      7      1   760 0.532
##  2       2      7      1  1261 0.883
##  3       3      7      1  1240 0.868
##  4       4      7      1  1428 1
##  5       5      1      1  1301 0.911
##  6       6      7      1  1428 1
##  7       7      7      1  1428 1
##  8       8      7      1  1428 1
##  9       9      7      1  1406 0.985
## 10      10      1      1   750 0.525
## # ... with 90 more rows
```

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

```
##   [1]  7  7  7  7  1  7  7  7  7  1  1  1  1  1  1  1  1  1  1  1  8  8  8  8  8
##  [26]  8  8  8  8  8 10 10 10 10 10 10 10 10 10 10  2  2  2 10  2  2  2  2  2  2
##  [51]  9  9  9  9  9  9  9  9  9  9  6  6  6  6  6  6  6  6  6  6  4  4  4  4  4
##  [76]  4  4  4  4  4  3  3  3  3  3  3  3  3  3  3  5  5  5  5  5  5  5  5  5  5
```

```
z.map.ind <- which(apply(z.chain, 1, function(x) all(x == z.map)))
```

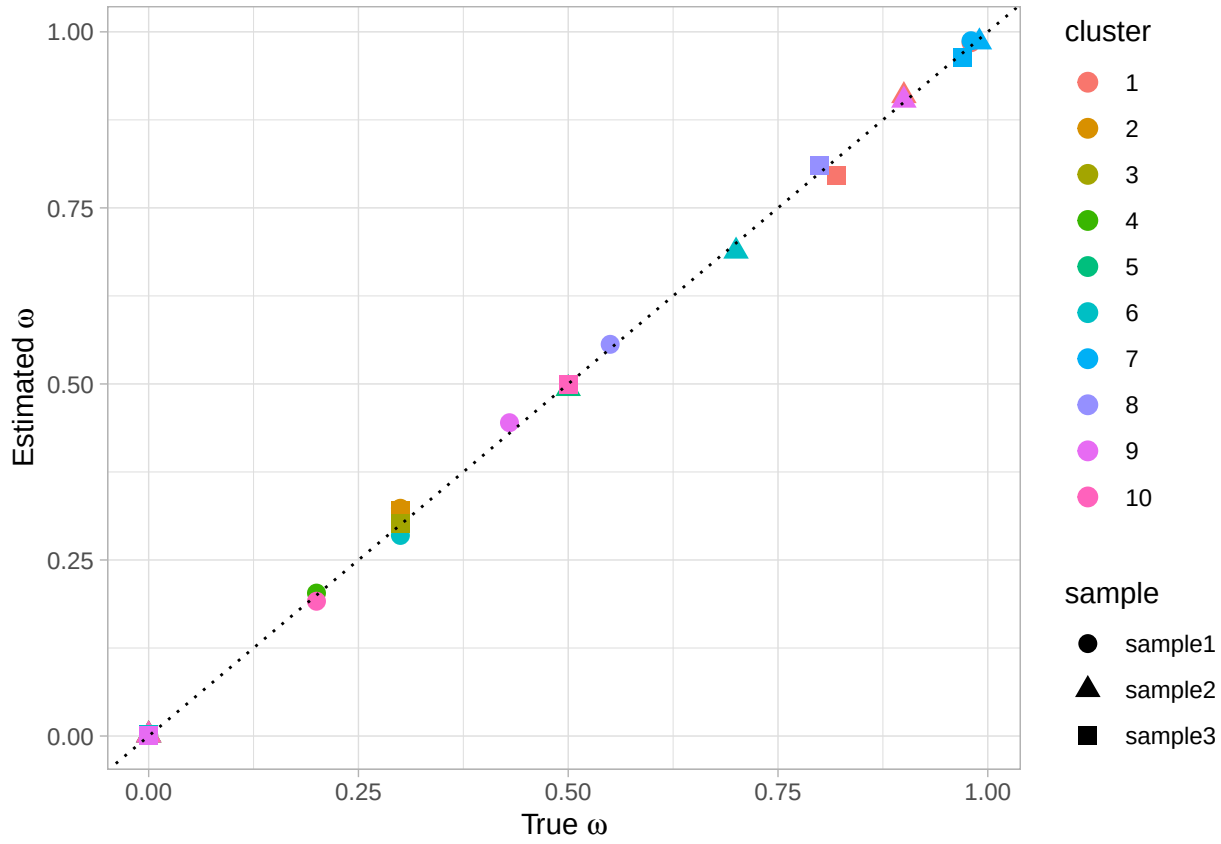
ω

```
mcmc_vals <- summary(samps)$statistics
mcmc_w <- mcmc_vals[substr(rownames(mcmc_vals), 1, 1) == "w", "Mean"]
mcmc_w <- matrix(mcmc_w, nrow=K)
colnames(mcmc_w) <- paste0("sample", 1:S)
round(mcmc_w, 2)
```

```
##       sample1 sample2 sample3
##  [1,]    0.98    0.91    0.80
##  [2,]    0.32    0.00    0.32
##  [3,]    0.00    0.00    0.30
##  [4,]    0.20    0.00    0.00
##  [5,]    0.00    0.49    0.00
##  [6,]    0.28    0.69    0.00
##  [7,]    0.99    0.99    0.96
##  [8,]    0.56    0.00    0.81
##  [9,]    0.44    0.90    0.00
## [10,]    0.19    0.00    0.50
```

```
mcmc_w_sd <- mcmc_vals[substr(rownames(mcmc_vals), 1, 1) == "w", "SD"]
mcmc_w_sd <- matrix(mcmc_w_sd, nrow=K)
colnames(mcmc_w_sd) <- paste0("sample", 1:S)

# order true w based on mcmc cluster numbering
mcmc_cluster_numbering  <- matrix(z.map, nrow = 10)
get_mode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
```

```
}
mcmc_cluster_numbering <- apply(mcmc_cluster_numbering, 2, get_mode)
true_to_mcmc_w_ordering <- match(1:K, mcmc_cluster_numbering)
w_ordered <- w[true_to_mcmc_w_ordering, ]

# scatter
mcmc_w_tb <- mcmc_w %>%
  as_tibble() %>%
  mutate(cluster=1:K) %>%
  gather("sample", "mcmc_w", -c(cluster))
w_master <- w_ordered %>%
  as_tibble() %>%
  mutate(cluster=1:K) %>%
  gather("sample", "true_w", -c(cluster)) %>%
  left_join(mcmc_w_tb, by=c("cluster", "sample")) %>%
  mutate(cluster=factor(cluster),
         sample=factor(sample))
ggplot(w_master, aes(true_w, mcmc_w)) +
  geom_point(size=3, aes(color = cluster, shape = sample)) +
  geom_abline(slope=1, intercept=0, linetype="dotted") +
  xlab(expression("True "*omega)) +
  ylab(expression("Estimated "*omega)) +
  theme_light()
```




```
#ggsave(file.path(figs.dir, "w_plot.pdf"), height = 7, width = 8)
```

Admat functions

```
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
  }

  while (sum(admat[1, ]) == 0) {
    admat <- mutate.admat(admat)
  }

  admat
}

base.admat <- function(w, zero.thresh=0.01) {
  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
}

init.admat <- function(w, zero.thresh) {
  base <- base.admat(w, zero.thresh)
  rand.admat(base)
}


mutate.admat <- function(admat, ncol.to.mutate) {

  # choose a column(s) to mutate
  K <- ncol(admat)
  rand.ks <- sample(1:K, size=ncol.to.mutate)

  # mutate columns
  new.admat <- admat
  for (rand.k in rand.ks) {
    ## possible positions (0's)
    possiblePos <- which(!is.na(admat[, rand.k]) & admat[, rand.k] != 1)
    ## current position with 1
    ind.1 <- which(admat[, rand.k] == 1)
    ## select new position
    if (length(possiblePos) == 1) {
      new.1 <- possiblePos
    } else {
      new.1 <- sample(possiblePos, size=1)
    }

    new.admat[ind.1, rand.k] <- 0
    new.admat[new.1, rand.k] <- 1
  }


  while (sum(new.admat[1, ]) == 0) {
    new.admat <- mutate.admat(admat)
  }
  new.admat
}
```

SCHISM tree scoring

```
decide.ht <- function(pval, alpha=0.05) {
  # 1 signals rejection event for null of i -> j
  if (pval <= alpha) return(1)
  else return(0)
```

```
}

create.cpov <- function(mcmc_w, mcmc_w_sd, alpha=0.05, zero.thresh=0.01) {
  cpov <- base.admat(mcmc_w, zero.thresh)
  S <- ncol(mcmc_w) # number of samples

  # root can go to anyone -- all 0's (default base admat value)

  for (r in 2:nrow(cpov)) {
    for (c in 1:ncol(cpov)) {

      if (is.na(cpov[r,c])) next # skip restricted position

      from <- r-1 # 'from' cluster node
      to <- c # 'to' cluster node

      statistic <- 0
      pval <- 0

      for(s in 1:S) {
        d <- mcmc_w[from,s] - mcmc_w[to,s]
        d_sd <- sqrt((mcmc_w_sd[from,s])^2 + (mcmc_w_sd[to,s])^2)
        I <- sum(d < 0)
        statistic <- statistic + (d / d_sd)^2 * I

        for (k in 0:S) {
        pval <- pval + ((1 - pchisq(statistic, k)) * choose(S, k) / (2^S))
        }
      }
      cpov[r,c] <- decide.ht(pval, alpha)
    }
  }
  cpov
}

calc.topology.cost <- function(admat, cpov) {

  TC <- 0
  edges <- which(admat == 1, arr.ind=T)
  for (i in 1:nrow(edges)) {
    cpov.temp <- ifelse(is.na(cpov[edges[i,1], edges[i,2]]), 1, cpov[edges[i,1], edges[i,2]])
    TC <- TC + cpov.temp
  }

  TC
}

calc.mass.cost <- function(admat, mcmc_w) {

  numChildren <- rowSums(admat, na.rm = T)
  nodes <- which(numChildren > 0, arr.ind = T) # not leaves
  mc.node <- rep(0, length(nodes))
```

```
  for (i in 1:length(nodes)) {
    node <- nodes[i]

    # root node: MCF = 1
    parent.w <- rep(1, ncol(mcmc_w))
    # not root node: look up MCF in mcmc_w
    if (node != 1) {
      parent.w <- mcmc_w[node-1,]
    }

    kids <- which(admat[node,] == 1, arr.ind = T)
    if (numChildren[node] > 1) {
      children.w <- colSums(mcmc_w[kids,])
    } else {
      children.w <- mcmc_w[kids,]
    }

    mc.s <- ifelse(parent.w >= children.w, 0, children.w - parent.w)
    mc.node[i] <- sqrt(sum(mc.s^2))
  }

  sum(mc.node)

}

calc.tree.fitness <- function(admat, cpov, mcmc_w, scaling.coeff=5) {
  TC <- calc.topology.cost(admat, cpov)
  MC <- calc.mass.cost(admat, mcmc_w)
  Z <- TC + MC
  fitness <- exp(-scaling.coeff * Z)
  fitness
}
```

```
sample.w <- function(w.chain, K) {
  nIter <- nrow(w.chain)
  randIter <- sample(nIter, size = 1)
  chosen <- w.chain[randIter, ]
  matrix(chosen, nrow = K)
}
```

Tree MCMC

```
set.seed(1234)

# true_edges <- data.frame(from = c(1, 2, 3, 3, 4, 4, 5, 6, 7, 8),
#                          to = c(1, 2, 3, 6, 4, 5, 8, 9, 7, 10))
# true_edges_reorder <- true_edges
# true_edges_reorder[2:(K), 1] <- match(true_edges[2:(K), 1] - 1,
#                                       true_to_mcmc_w_ordering) + 1
# true_edges_reorder[, 2] <- match(true_edges[, 2],
#                                  true_to_mcmc_w_ordering)

answer <- base.admat(mcmc_w)
```

```
# for (i in 1:K) {
#   from <- true_edges_reorder[i, 1]
#   to <- true_edges_reorder[i, 2]
#   answer[from, to] <- 1
# }
answer[1,1] <- answer[2,2] <- answer[5,3] <- answer[9,4] <- answer[10,5] <- answer[11,6] <-answer[3,7] <
answer
```

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

```
cpov <- create.cpov(mcmc_w, mcmc_w_sd)

calc.tree.fitness(answer, cpov, mcmc_w)
```

```
## [1] 3.239112e-16
```

```
calc.tree.fitness(answer, cpov, w_ordered)
```

```
## [1] 2.77147e-16
```

```
# Initialize chain ------------------------------------------------------------

#admat.chain <- list(init.admat(mcmc_w, zero.thresh=0.01))

# start at best tree
best.admat.mcmc <- base.admat(mcmc_w)
best.admat.mcmc[1,1] <- best.admat.mcmc[2,2] <- best.admat.mcmc[3,7] <-
  best.admat.mcmc[3,8] <- best.admat.mcmc[5,6] <- best.admat.mcmc[8,9] <-
  best.admat.mcmc[9,4] <- best.admat.mcmc[9,10] <- best.admat.mcmc[10,5] <-
  best.admat.mcmc[11,3] <- 1

# sample mcf from posterior
admat.chain <- list(best.admat.mcmc)
mcf.chain <- list(sample.w(w.chain, K))
cpov.chain <- list(create.cpov(mcf.chain[[1]], mcmc_w_sd))
score.chain <- calc.tree.fitness(admat.chain[[1]], cpov.chain[[1]], mcf.chain[[1]])

# MCMC ------------------------------------------------------------------------

numAccept = 0
ncol.to.mutate <- 1
numIter <- 1000
for (i in 1:numIter) {
  fit.prev <- score.chain[i]
```

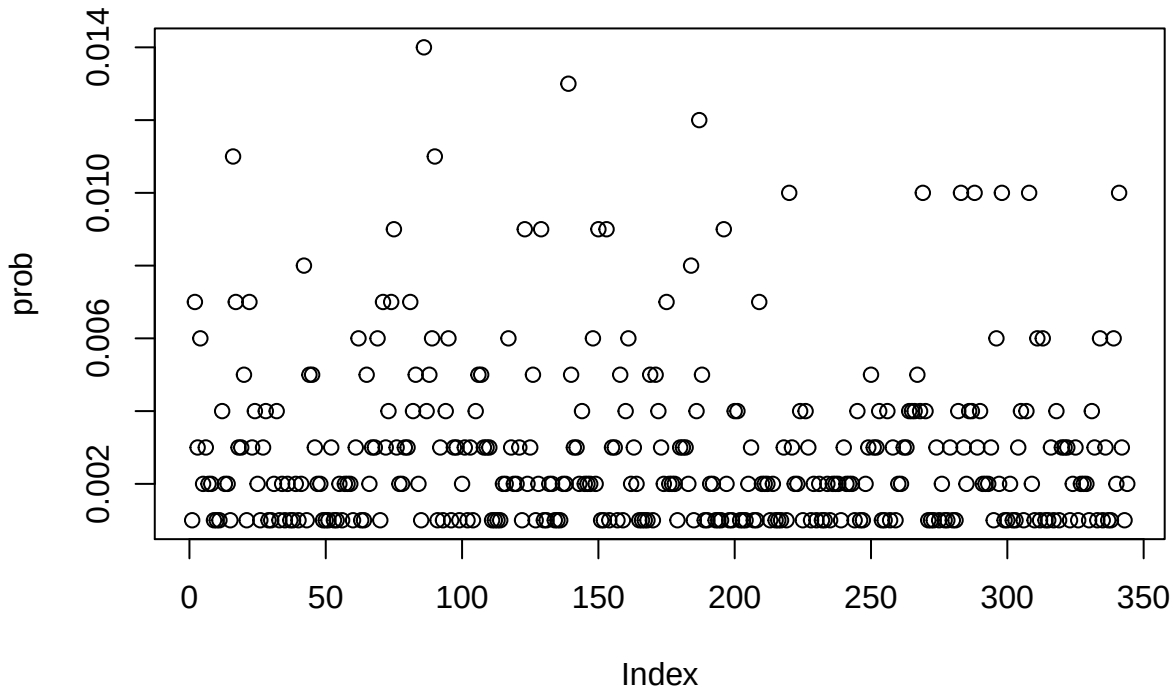
```
  # new admat
  admat.star <- mutate.admat(admat.chain[[i]], ncol.to.mutate)

  # sample w
  w.star <- sample.w(w.chain, K)

  # create cpov and calc fitness
  cpov.star <- create.cpov(w.star, mcmc_w_sd)
  fit.star <- calc.tree.fitness(admat.star, cpov.star, w.star)

  r <- fit.star / fit.prev
  u <- runif(1,0,1)
  if(u <= r) {
    admat.chain[[i+1]] <- admat.star
    score.chain[i+1] <- fit.star
    cpov.chain[[i+1]] <- cpov.star
    mcf.chain[[i+1]] <- w.star
    numAccept <- numAccept + 1
  } else {
    admat.chain[[i+1]] <- admat.chain[[i]]
    score.chain[i+1] <- score.chain[i]
    cpov.chain[[i+1]] <- cpov.chain[[i]]
    mcf.chain[[i+1]] <- mcf.chain[[i]]
  }
}

acceptRate <- numAccept/(numIter)
acceptRate
```

```
## [1] 0.361
```

```
# max(score.chain)
# max.score.ind <- which(score.chain == max(score.chain))
# if(length(max.score.ind) > 1) max.score.ind <- max.score.ind[1]
# max.admat <- admat.chain[[max.score.ind]]
# max.admat


##
## posterior distribution of trees
##
numericRepresentation <- function(x){
    x[is.na(x)] <- 0
    x <- as.numeric(x)
    paste(x, collapse="")
}
trees <- sapply(admat.chain, numericRepresentation)
tab <- table(trees)
length(tab)
```

```
## [1] 344
```

```
freq <- as.numeric(tab)
prob <- freq/1000
plot(prob)
```




```
tab2 <- tab[prob > 0.01]
## is the true tree among those with highest probability
tr <- numericRepresentation(answer)
tr %in% names(tab2)
```

```
## [1] FALSE
```