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
```

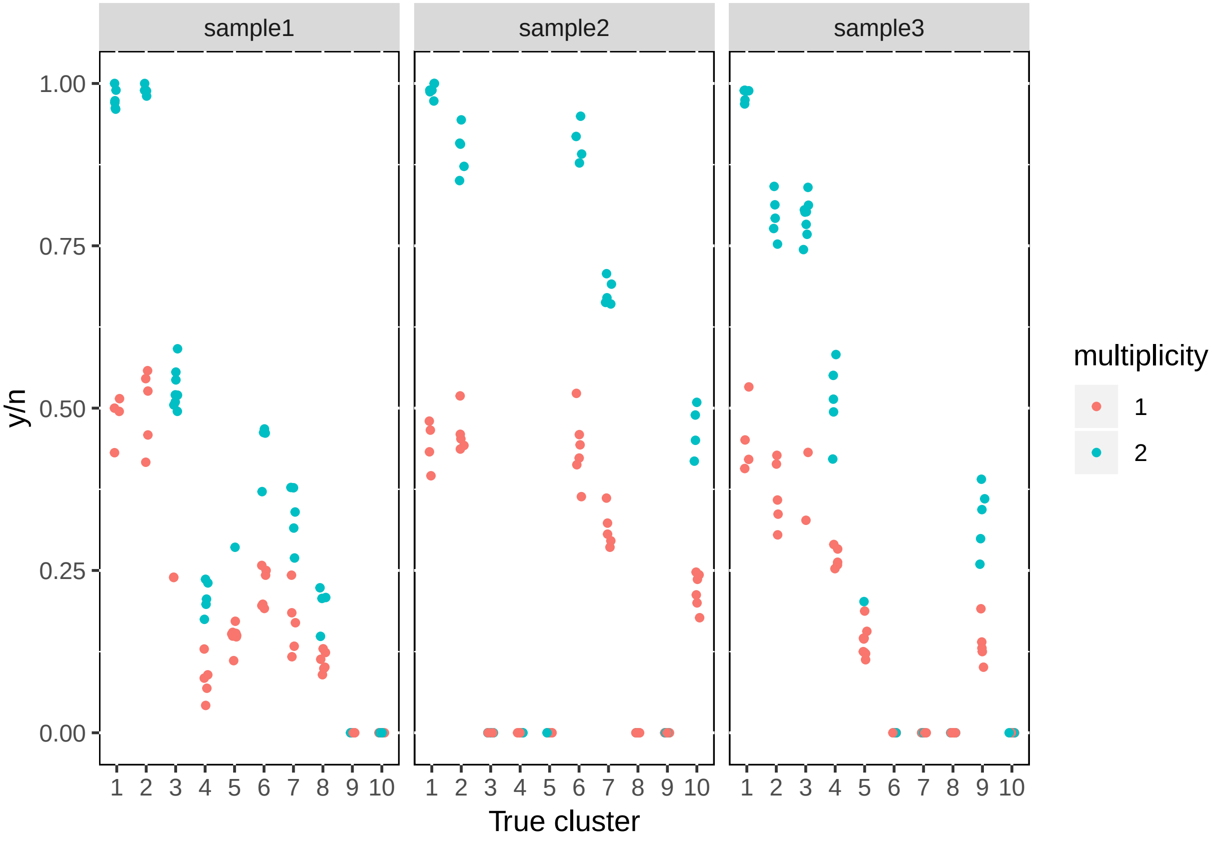
```
##       sample1 sample2 sample3
##  [1,]    0.98    0.99    0.97
##  [2,]    0.98    0.90    0.82
##  [3,]    0.55    0.00    0.80
##  [4,]    0.20    0.00    0.50
##  [5,]    0.30    0.00    0.30
##  [6,]    0.43    0.90    0.00
##  [7,]    0.30    0.70    0.00
##  [8,]    0.20    0.00    0.00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

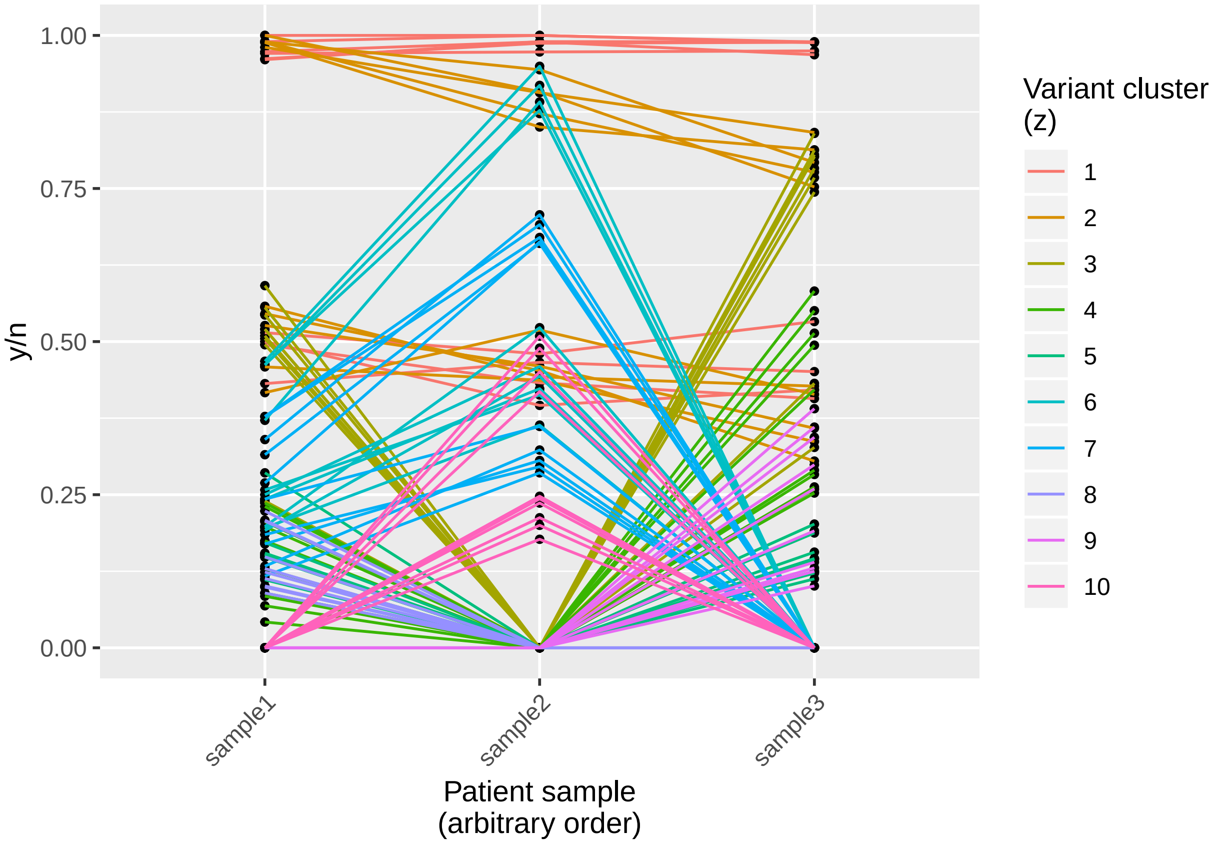
```
tcn <- matrix(2, nrow=I, ncol=S)
m <- matrix(rep(sample(1:2, size = I, replace = T), S),
            nrow=I, ncol=S)
W <- w[z, ]
calcTheta <- function(m, tcn, w) {
  (m * w) / (tcn * w + 2*(1-w))
}

theta <- calcTheta(m, tcn, W)

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

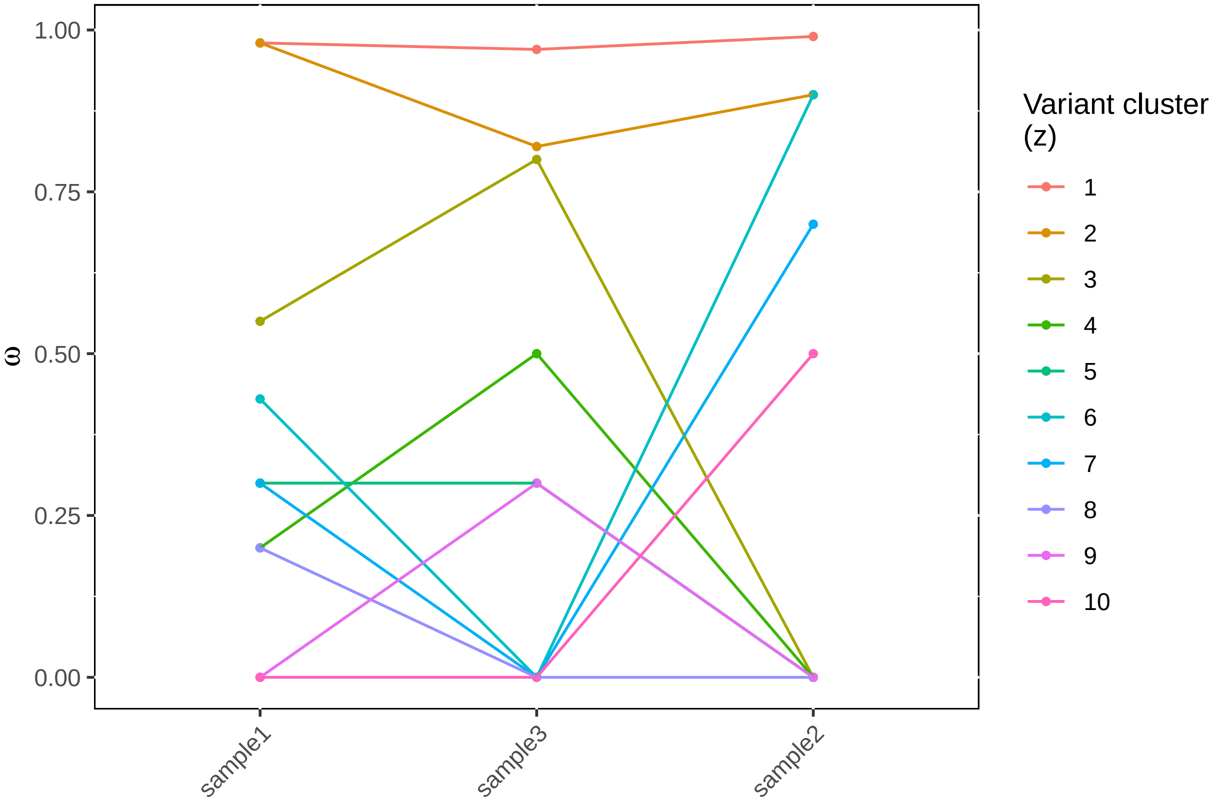
```r
for (i in 1:I) {
  for (s in 1:S) {
    y[i, s] <- rbinom(1, n[i, s], theta[i,s])
  }
}

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




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


ω
1.00
0.75
0.50
0.25
0.00
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

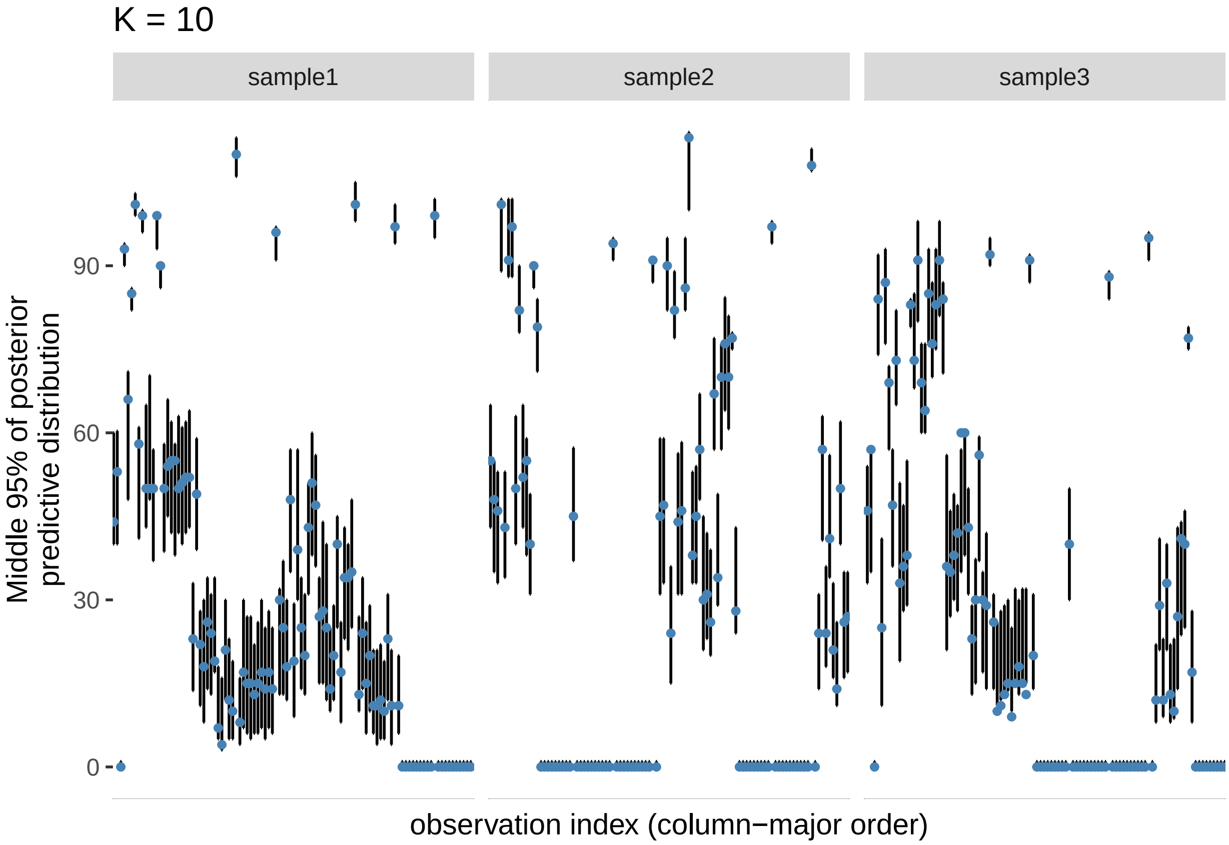
```
jags.file <- file.path(models.dir, "model.jags")
inits <- list(".RNG.name" = "base::Wichmann-Hill",
              ".RNG.seed" = 123)
test.data <- list("I" = I, "S" = S,
                  "y" = y, "n" = n,
                  "tcn" = tcn)
params <- c("z", "w", "ystar", "m")
```

```
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: 731
##    Total graph size: 4638
##
## Initializing model
```

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

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

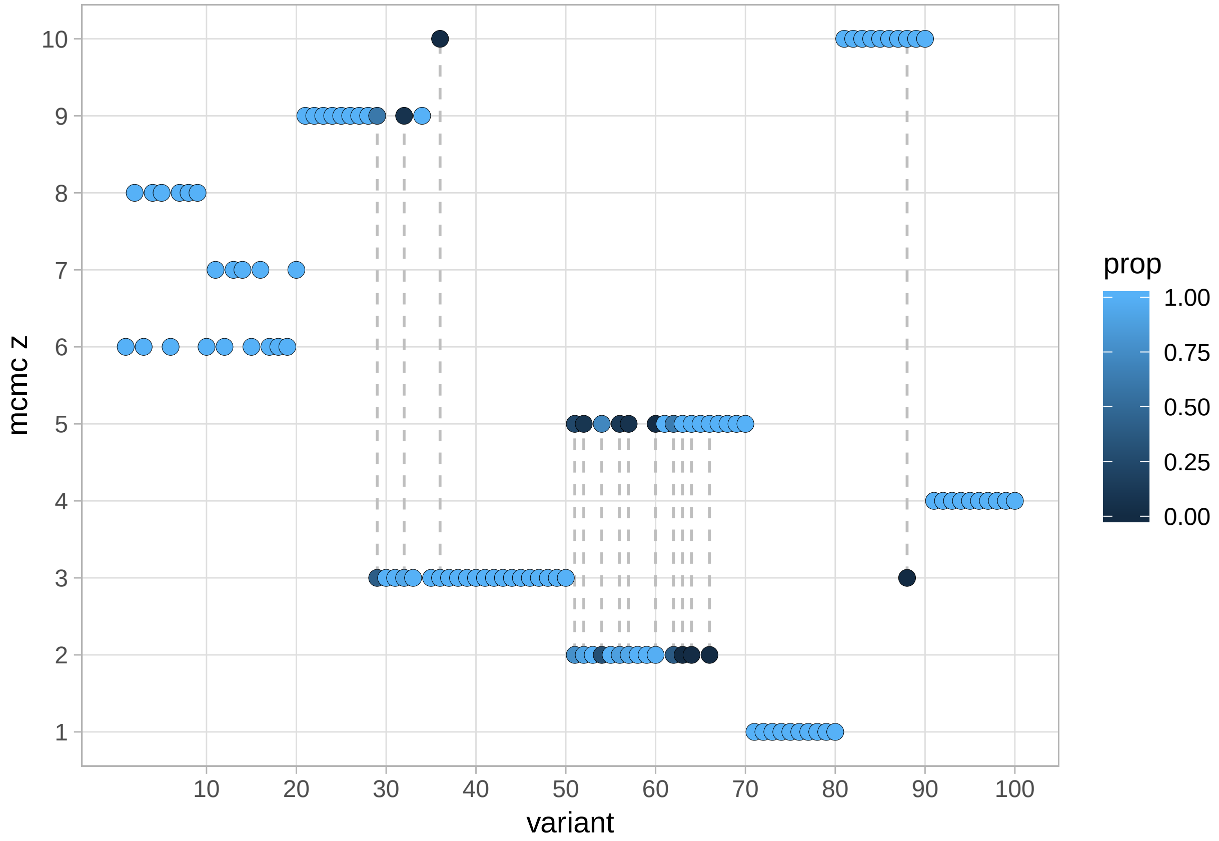


z

```
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 = round(count/sum(count), 2))

  z.chain.tb_simp
}

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

```
## # A tibble: 114 x 5
## # Groups:   variant [100]
##    variant mcmc_z true_z count  prop
##      <int>  <dbl>  <int> <int> <dbl>
##  1       1      6      1  1428     1
##  2       2      8      1  1428     1
##  3       3      6      1  1428     1
##  4       4      8      1  1428     1
##  5       5      8      1  1428     1
##  6       6      6      1  1428     1
##  7       7      8      1  1428     1
##  8       8      8      1  1428     1
##  9       9      8      1  1428     1
## 10      10      6      1  1428     1
## # ... with 104 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
```



```
ggsave(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      6      1  1428     1
##  2       2      8      1  1428     1
##  3       3      6      1  1428     1
##  4       4      8      1  1428     1
##  5       5      8      1  1428     1
##  6       6      6      1  1428     1
##  7       7      8      1  1428     1
##  8       8      8      1  1428     1
##  9       9      8      1  1428     1
## 10      10      6      1  1428     1
## # ... with 90 more rows
```

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

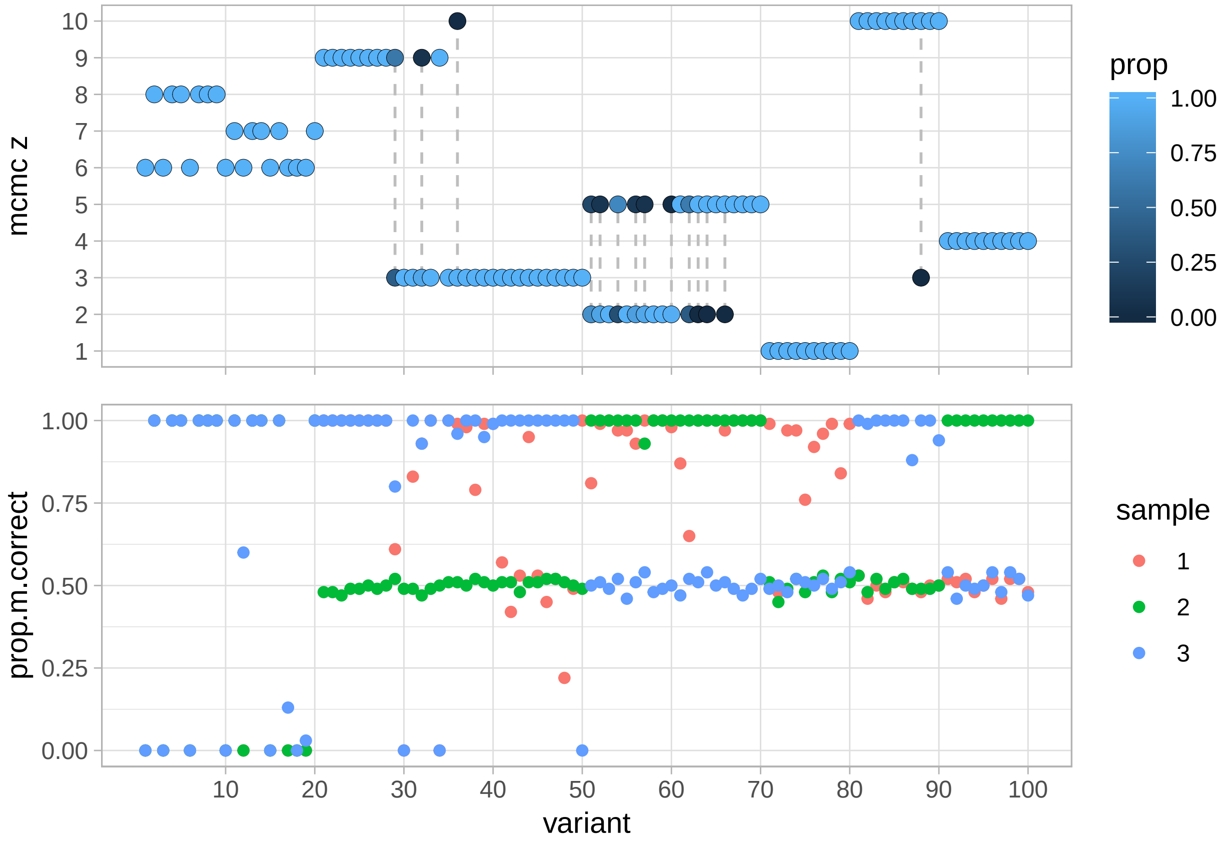
```
##   [1]  6  8  6  8  8  6  8  8  8  6  7  6  7  7  6  7  6  6  6  7  9  9  9
##  [24]  9  9  9  9  9  9  3  3  3  3  9  3  3  3  3  3  3  3  3  3  3  3  3
##  [47]  3  3  3  3  2  2  2  5  2  2  2  2  2  2  5  5  5  5  5  5  5  5  5
##  [70]  5  1  1  1  1  1  1  1  1  1  1 10 10 10 10 10 10 10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

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

m

```
m.tb <- m %>%
  as_tibble() %>%
  mutate(variant = 1:nrow(m)) %>%
  gather(sample, m, -c(variant)) %>%
  mutate(sample = as.integer(gsub("sample", "", .$sample))) %>%
  group_by(variant, sample)

m.chain.tb <- m.chain %>%
  as_tibble() %>%
  mutate(iter = 1:nrow(m.chain)) %>%
  gather(label, mcmc_m, -c(iter))
m.chain.tb <- m.chain.tb %>%
  mutate(variant = as.integer(gsub(".*\\[(.*),.*", "\\1", m.chain.tb$label))) %>%
  mutate(sample = as.integer(gsub(".*,(.*)\\].*", "\\1", m.chain.tb$label))) %>%
  mutate(true_m = rep(m.tb$m, each = nrow(m.chain)))
m.chain.tb.simp <- m.chain.tb %>%
  group_by(variant, sample) %>%
  mutate(prop.m.correct = round(sum(mcmc_m == true_m) / nrow(m.chain), 2)) %>%
  ungroup() %>%
  mutate(iter = NULL) %>%
  mutate(mcmc_m = NULL)
m.chain.tb.simp <- distinct(m.chain.tb.simp) %>%
  mutate(z.map = rep(z.map, times = S))
m.chain.tb.simp <- m.chain.tb.simp %>% mutate(sample = as.character(m.chain.tb.simp$sample))
```

```
m.plot <- ggplot(m.chain.tb.simp, aes(x=variant, prop.m.correct, colour = sample)) +
  geom_point() +
  scale_x_continuous(breaks = seq(10,100,10), minor_breaks=NULL) +
  theme_light()
z.plot.2 <- z.plot + theme(axis.title.x = element_blank(), axis.text.x = element_blank())
grid.newpage()
grid.draw(rbind(ggplotGrob(z.plot.2), ggplotGrob(m.plot), size = "last"))
```



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)
```

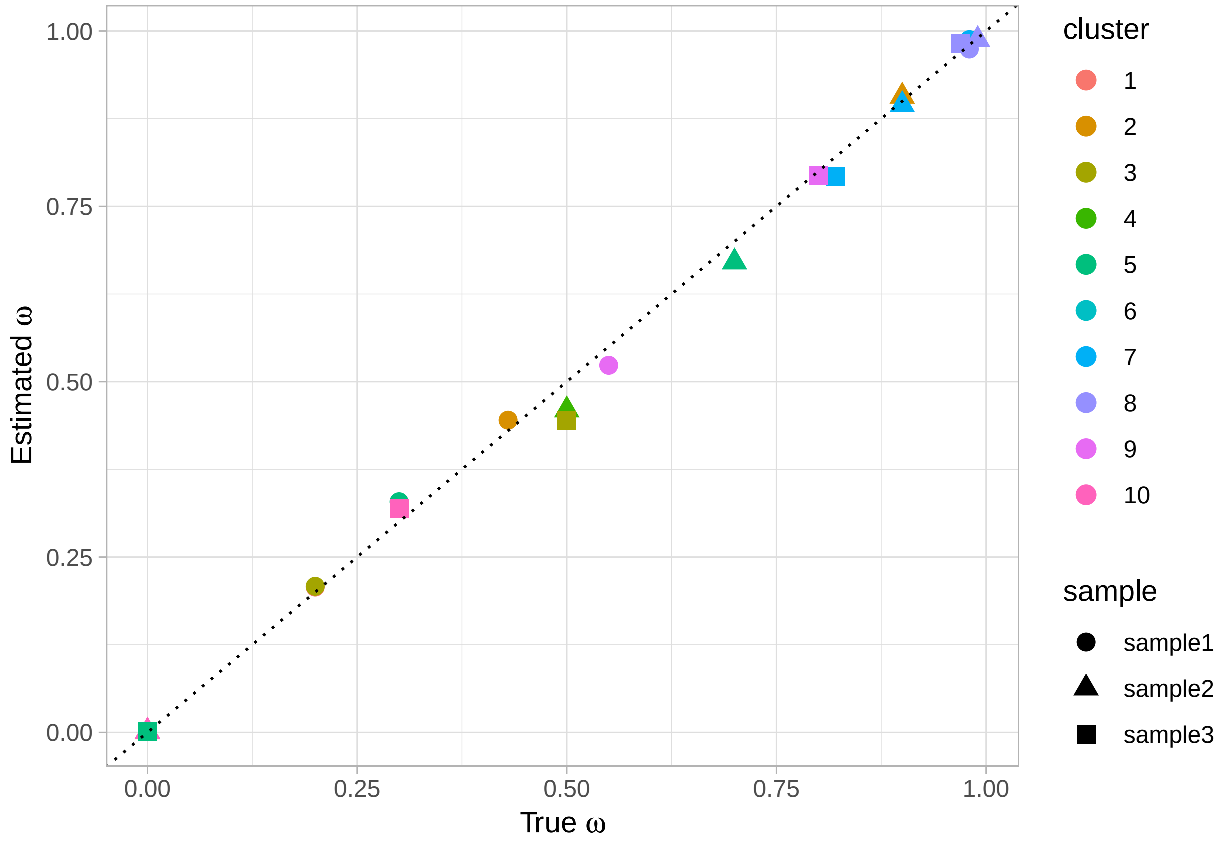
```
##       sample1 sample2 sample3
##  [1,]    0.21    0.00    0.00
##  [2,]    0.45    0.91    0.00
##  [3,]    0.21    0.00    0.45
##  [4,]    0.00    0.46    0.00
##  [5,]    0.33    0.67    0.00
##  [6,]    0.49    0.45    0.42
##  [7,]    0.99    0.89    0.79
##  [8,]    0.97    0.99    0.98
##  [9,]    0.52    0.00    0.79
## [10,]    0.00    0.00    0.32
```

```
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)
mcmc_cluster_numbering
```

```
##  [1]  8  7  9  3  3  2  5  1 10  4
```

```
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()
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



```
#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) {
  cpov <- base.admat(mcmc_w, zero.thresh = 0.01)
  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)) {
    TC <- TC + cpov[edges[i,1], edges[i,2]]
  }

  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)

}
```

```r
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
}
```

Tree MCMC

Mutate 2 columns

Mutate 3 columns

Mutate 4 columns