Only Functions

Qi

2022-08-30

```
rm(list = ls())
```

1, Creat Stump

```
create_stump <- function(x, y, num_trees = 1){

all_tree <- vector("list", length = num_trees)

# Make each tree information stored in all_tree[[i]]
# This is like creating a box, but it's empty for further storage

for (i in 1:num_trees) {

  # Make each element has two indexes
  all_tree[[i]] <- vector("list",2)
  names(all_tree[[i]]) <- c("info", "node")

  # Create tree information storage
  all_tree[[i]][[1]] <- matrix(NA, ncol = 8, nrow = 1)

  # Create node indices storage
  all_tree[[i]][[2]] <- rep(1, length(y))

  # Create column names for information matrix
  colnames(all_tree[[i]][[1]]) <- c("terminal","child_L","child_R","parent","split_variable","split_valu

  all_tree[[i]][[1]][1,] <- c(1, rep(NA, 5), mean(y), length(y))



}

return(all_tree)

}
```

2. Update Tree

```
update_tree <- function(x, y, type = c("grow", # For growing the existing tree
                                       "prune", # For merging one pair of the terminal node
                                       "change", # Change the splitting variable and splitting rule
                                       "swap", # Swap the splitting rules for two pairs of terminal nod
                                       ), curr_tree # The current sets of trees
                                        , node_min_size # The minimum size of a node to grow, avoid the

                        ){


  new_tree <- switch (type, grow = grow_tree(x, y, curr_tree, node_min_size),
                            prune = prune_tree(x, y, curr_tree),
                            change = change_tree(x, y, curr_tree, node_min_size),
                            swap = swap_tree(x, y, curr_tree, node_min_size))
  return(new_tree)
                                       }
```

3. Grow Tree

```
grow_tree <- function(x, y, curr_tree, node_min_size){

  # Give new tree information box
  new_tree <- curr_tree

  # Get the terminal nodes list
  terminal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 1)

  # Get the terminal node size
  terminal_node_size <- new_tree$info[terminal_node,"node_size"]

  # Initialize
  available_values <- NULL
  max_bad_trees <- 10
  count_bad_trees <- 0
  bad_trees = TRUE

  # If it's a bad_tree, for example, the size of one new terminal node is too small, then we need to re

  while (bad_trees) {

    # Information box for new tree
    new_tree <- curr_tree

    # Add two extra rows to the tree information, because we are now having more nodes of the tree.
    new_tree$info <- rbind(new_tree$info, c(1, rep(NA, 7)), c(1, rep(NA, 7)))

    # Choose a random terminal to split, if it has been smaller than the minimum size of our requiremen
    split_node <- sample(terminal_node, 1, prob = as.integer(as.numeric(terminal_node_size))> node_min_s
```

```
    # Choose a variable to split
    split_var <- sample(1:ncol(x),1)

    # The next step guaranteed that we are having a nice splitting value.
    # Available values are the range of the selected variables belonging to this node. Noting that node
    available_values <- sort(unique(x[new_tree$node == split_node, split_var]))

    # Select an available value to split

    if (length(available_values)==1){
      split_val <- available_values[1]
    }else if(length(available_values)==2){
        split_val <- available_values[2]
    }else{
      # We cannot select the biggest or smallest value as the splitting rule
        split_val <- gdata::resample(available_values[-c(1, length(available_values))],1)
    }
       # Make sure the current parent exist there. If it's the root node, then it's NA.
      curr_parent <- new_tree$info[split_node, "parent"]
      new_tree$info[split_node, 1:6] <- c(0, # Because now it's not terminal node
                                          nrow(new_tree$info)-1, # The second last row is the left chil
                                          nrow(new_tree$info), # The last row is the right child
                                          curr_parent, # Record the current parent
                                          split_var, split_val)


      # Fill the parent of these two child nodes
      new_tree$info[nrow(new_tree$info)-1, "parent"] <- split_node
      new_tree$info[nrow(new_tree$info), "parent"] <- split_node

      #Fill the details including updating the node indices
      new_tree <- fill_tree_details(new_tree, x)

      # Check for bad trees, if it's a bad tree, then we cannot use this.
      if(any(new_tree$info[,"node_size"]<= node_min_size)){

        # Count how many bad trees that we have generated
        count_bad_trees = count_bad_trees + 1
      }else{
        bad_trees = FALSE
      }

      # If too many bad trees are generated, we return the current tree, which means now the trees has
      if(count_bad_trees == max_bad_trees){return(curr_tree)}


}
        return(new_tree)
}
```

4. Prune Tree

```
prune_tree <- function(x, y, curr_tree){

  # To begin with, we create a holder for the new tree, where we can store the information about the ne
  new_tree <- curr_tree

  # If the tree has been the stump, we cannot prune it anymore.
  if(nrow(new_tree$info) == 1){ return(new_tree)}

  # Then, we will get the list of the terminal nodes.
  terminal_nodes <- which(as.numeric(new_tree$info[,"terminal"])==1)

  # Then we randomly pick up a terminal to prune, but it's important that both of the terminal points h
  bad_node <- TRUE
  while (bad_node) {

    # Randomly pick up a point to prune
    selected_node <- sample(terminal_nodes, 1)

    # Then, find the parent of this node
    selected_parent <- as.numeric(new_tree$info[selected_node, "parent"])

    # Then, get the children of this parent
    children_left <- as.numeric(new_tree$info[selected_parent,"child_L"])
    children_right <- as.numeric(new_tree$info[selected_parent, "child_R"])

    # Check whether they are both terminal nodes
    children_left_ter <- as.numeric(new_tree$info[children_left,"terminal"])
    children_right_ter <- as.numeric(new_tree$info[children_right, "terminal"])

    # If both of the nodes are terminal nodes, then it's okay
    if( children_right_ter + children_left_ter == 2 ){ bad_node <- FALSE }


  }

  # After selecting the two terminal nodes, we need to delete the two rows of these nodes.
  new_tree$info <- new_tree$info[-c(children_left, children_right),]

  # Update the information for the parent node since now it's a terminal node, so it does not have the
  new_tree$info[selected_parent,c("terminal", "child_L", "child_R", "split_variable", "split_value")] <-

  # If the tree comes back to a stump, there is no need to fill the tree details.
  if(nrow(new_tree$info)==1){new_tree$node <- rep(1, nrow(x))}else{

    # Since we have deleted several rows, there are some row indices changing, if we didn't delete the
    if(selected_node <= nrow(new_tree$info)){

      # Find those whose parents are affected by deleting rows
      bad_parents <- which(as.numeric(new_tree$info[,"parent"])>= selected_node)

      # Since the deleting rows must be continuous two rows, -2 is to make sure the parents are correct
```

```
      new_tree$info[bad_parents, "parent"] <- as.numeric(new_tree$info[bad_parents, "parent"]) - 2

      for (i in selected_node:nrow( new_tree$info )) {

        # Update the child index who has been affected.
        # First, find the parent of the ith row, because we have updated parents before, this row now h
        curr_parent <- as.numeric(new_tree$info[i, "parent"])

        # Then, find the correct children index of the parent row to update the wrong children row inde
        curr_children <-  which(as.numeric(new_tree$info[,"parent"])==curr_parent)

        # Then, update the row indexes.
        new_tree$info[curr_parent, c("child_L", "child_R")] <- sort(curr_children)

      } # End loop for updating children nodes.

    } # End loop for updating tree information

      # Fill tree details
      new_tree <- fill_tree_details(new_tree, x)
  }

      return(new_tree)

}
```

5. Change Tree

```
get_children <- function(tree_info, parent){

  all_chilren <- NULL
  # If the current node is the terminal node, then return the children list and the parent so far.
  if(as.numeric(tree_info[parent, "terminal"]) == 1){return(c(all_chilren, parent))}else{

    # If the current node is not the terminal node, then we have to recursively get the node list.
    curr_child_left <- as.numeric(tree_info[parent, "child_L"])
    curr_child_right <- as.numeric(tree_info[parent, "child_R"])

    # Return the children and the children of the children recursively
    return(c(all_chilren, get_children(tree_info,curr_child_left), get_children(tree_info, curr_child_r

  }

}
```

```
change_tree <- function(x, y, curr_tree, node_min_size){

  # In this function, since we are changing the tree structure, we have to make sure that the new tree
  # It has at least some observations in each terminal node, which is decided by the "node_min_size".
```

```r
  # If it's a stump, there is nothing to change.
  if(nrow(curr_tree$info) == 1){return(curr_tree)}

  # Then, create a information box for the new three
  new_tree <- curr_tree

  # Since changing means change out the split variable and split rule, we need to make sure that this i
  internal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 0)
  terminal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 1)

  # Then, we can use a while loop to get a good tree.
  max_bad <- 100
  count_bad <- 0
  bad_tree <- TRUE

  while (bad_tree) {
    # When it's a bad tree, our changing has to be reset to make a new tree.
    new_tree <- curr_tree

    # Then we select a internal node to change.
    selected_node <- sample(internal_node, 1)

    # Use the get_children function to get all the children nodes of this node
    all_children <- get_children(new_tree$info, selected_node)

    # First, we find the nodes that corresponding to these children, then we delete those who are not i
    # This step is used to getting the further available value because we have to first find which poin
    use_node <- !is.na(match(new_tree$node, all_children))

    # Then we create a new split variable and a new split value.
    available_values <- NULL
    new_split <- sample(1:ncol(x), 1)

    # By using the selected points, we can get the available values.
    available_values <- sort(unique(x[use_node, new_split]))

    if(length(available_values) == 1){split_val <- available_values[1]}else if(length(available_values)
      split_val <- available_values[2]
    }else{

      split_val <- gdata::resample(available_values[-c(1, length(available_values))], 1)
    }

    # Then, we can update the tree information
    new_tree$info[selected_node, c("split_variable", "split_value")] <- c(new_split, split_val)

    # Updating the tree details using the fill tree function
    new_tree <- fill_tree_details(new_tree, x)

    # Now, we finished changing the splitting variable and splitting rule, but we have to check whether
    if(any(new_tree$info[terminal_node,"node_size"] <= node_min_size)){count_bad <- count_bad + 1}else{
      bad_tree <- FALSE
    }
```

```
    if(count_bad == max_bad){return(curr_tree)}

  }


  return(new_tree)

}
```

6. Swap Tree

```
swap_tree <- function(x, y, curr_tree, node_min_size){

  # Swap means we have to find two neighboring internal nodes, and then swaps their splitting values an

  # If the tree is a stump, then we cannot swap anymore
  if(nrow(curr_tree$info) == 1){ return(curr_tree) }

  # Create an information box for new tree
  new_tree <- curr_tree

  # Same as "change", we need to find which nodes are internal and which are terminal
  internal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 0)
  terminal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 1)

  # If the tree is too small, like it only has one or two internal node, then swapping is useless since
  if(length(internal_node) < 3 ){ return(curr_tree)}

  # Then we need to find a pair of neighboring internal nodes
  parent_internal <- as.numeric(new_tree$info[internal_node, "parent"])

  # This step, we bind two internal nodes by column and create the pairs of internal nodes.
  # -1 because the root node doesn't have a parent, so it's useless as a child.
  # Be careful that this step creates a p*2 matrix, because internal_node is a vector, and parent is al
  # This matrix describes:
      # 1st column: Which points are internal nodes
      # 2nd column: Which nodes are the parent of the 1st column correspondingly.
  pairs_internal <- cbind(internal_node, parent_internal)[-1,]


  # Then create a loop to get good trees.
  # Set the maximum number of bad trees.

  max_bad <- 10
  count_bad <- 0
  bad_tree <- TRUE

  while (bad_tree) {
    new_tree <- curr_tree
```

```
    # Pick up a random pair
    selected_node <- sample(1:nrow(pairs_internal),1)

    # Get the split variable and split value for this pair
    # 1 for some internal node, 2 for the parent of this node
    swap_1_rule <- as.numeric(new_tree$info[pairs_internal[selected_node,1], c("split_variable", "split
    swap_2_rule <- as.numeric(new_tree$info[pairs_internal[selected_node,2], c("split_variable", "split

    # Change the tree information matrix, interchange the splitting rules
    new_tree$info[pairs_internal[selected_node,1], c("split_variable", "split_value")] <- swap_2_rule
    new_tree$info[pairs_internal[selected_node,2], c("split_variable", "split_value")] <- swap_1_rule


    # Then we should fill in the tree details
    new_tree <- fill_tree_details(new_tree, x)

    # Check whether it's a bad tree
    if(any(as.numeric(new_tree$info[,"node_size"]) <= node_min_size)){ count_bad <- count_bad + 1}else{
      bad_tree <- FALSE
    }

    if(max_bad == count_bad){ return(curr_tree)}

  }

  return(new_tree)

}
```

7. Fill Tree Details

```
fill_tree_details <- function(curr_tree, x){

  # Collect old information from the tree
  tree_info <- curr_tree$info

  # Create a new matrix to overwrite the information
  new_tree_info <- tree_info

  # Start with default node
  node <- rep(1, nrow(x))

  # Get the number of observations falling in each node.
  # But we don't need the first node, because it's the root so all the observations will fall into the

    for (i in 2:nrow(tree_info)) {

    #Get the parent
    curr_parent <- as.numeric(tree_info[i,"parent"])

    #Get the splitting variable and splitting value
```

```
    split_var <- as.numeric(tree_info[curr_parent, "split_variable"])
    split_val <- as.numeric(tree_info[curr_parent, "split_value"])
    direction <- ifelse(tree_info[curr_parent, "child_L"] == i, "L", "R")
    if(direction == "L"){
      # if the direction is the left, then it should use "smaller than" to confirm that this observatio
      # x[node == curr_parent,] selects the observations who belongs to the parent node.
      # Be careful that the "node" updated every time in the loop!
      # The node now haven't been updated, so it describes the category that this observation belongs t
      #So it's reasonable to say node == curr_parent because we need to filter among all the nodes belo
      new_tree_info[i, "node_size"] <- sum(x[node == curr_parent, split_var] < split_val)
      # This step we update the node indices for the i th row. So we now only update those who belongs
      node[node == curr_parent][x[node == curr_parent, split_var]<split_val] <- i
    }else{

      #Same as left, but change the inequality direction
      new_tree_info[i, "node_size"] <- sum(x[node == curr_parent, split_var] >= split_val)
      node[node == curr_parent][x[node == curr_parent, split_var] >= split_val] <- i

    }

  }

  return(list(info = new_tree_info, node = node))


}
```

8. Log Likelihood Function

```
get_like <- function(curr_tree, x, y, sigma){

  # Find which node is terminal node and find its mean

  terminal_node <- which(as.numeric(curr_tree$info[,"terminal"])==1)
  terminal_mean <- curr_tree$info[terminal_node, "mu"]

  res <- 0

  for (i in 1:length(terminal_node)) {

    group_index <- which(curr_tree$node == terminal_node[i])
    res <- res + sum(dnorm(y[group_index], mean = terminal_mean[i], sd = sqrt(sigma), log = TRUE))

  }

  return(res)


}
```

9. Tree Marginal Function

```
tree_marginal <- function(tree, x, y, c, alpha_sig, beta_sig){
  # Calculate the marginal distribution for the tree to get a sample from.
  I <- length(table(tree$node))
  ni <- table(tree$node)
  terminal_node <- which(as.numeric(tree$info[,"terminal"]) == 1 )
  part1 <-  -1/2 * sum(log(ni+c))


  M <- 0

  for (i in 1:I) {
    y_i <- y[tree$node==terminal_node[i]]
    M <- M + (sum(y_i^2) - sum(y_i)^2/(ni[i] + c))/2

  }

  part2 <- -(length(y)/2 + alpha_sig)*log(beta_sig+M)

  # Get the log marginal likelihood
  Log_mar <- part1 + part2
  return(Log_mar)
}
```

10. Sample Mu and Sigma

```
# Since the distribution of mu and sigma are conjugate, we can directly sample from the posterior distr
# This is a function that sample the mean of each terminal node as a vector together.



fill_update_mu <- function(y, sigma, c, curr_tree){
  ni <- table(curr_tree$node)
  mean_vec <- NULL
  var_vec <- NULL
  # Update each mean and variance, since they are iid distributed, I can generate them together by usin
  mean_vec <- aggregate(x = y, by = list(curr_tree$node), FUN = sum)$V1 / (ni+c)
  var_vec <- sigma / (ni+c)

#  for (i in 1:length(ni)) {
#    mean_vec[i] <- (sum(y[curr_tree$node==names(ni)[i]])) / (ni[i] + c )
#    var_vec[i] <- sigma / (ni[i] + c )
#  }

  mu_update <- mvtnorm::rmvnorm(1, mean = mean_vec,  sigma = diag(var_vec, ncol = length(ni)))
  new_tree <- curr_tree

  new_tree$info[which(as.numeric(new_tree$info[,"terminal"])==1), "mu"] <- mu_update
  return(new_tree)
```

```
}

sample_sigma <- function(y, mu, alpha_sig, beta_sig, tree, c){

  # Get the posterior parameters for the inverse gamma distribution
  ni <- table(tree$node)
  param1 <- length(y)/2 + length(ni)/2 + alpha_sig
  SS <- NULL
  for (i in 1:length(ni)) {
    SS[i] <- sum((y[tree$node==ni[i]]-mu[i])^2)
  }
  param2 <- sum(SS)/2 + c*sum(mu^2)/2 + beta_sig
  sigma_update <- 1/rgamma(1, shape = param1, rate = param2)
  return(sigma_update)
}
```

11. Tree Prior Function

```
get_tree_prior <- function(tree, alpha, beta){
  # First, we need to work the depth of the tree
  # So we need to find the level of each node, then the depth is the maximum of the level

  level <- rep(NA, nrow(tree$info))
  # The first node is the 0 depth of the tree
  level[1] <- 0

  if(nrow(tree$info)==1){return(log(1-alpha))} # Because the tree depth is 0.

  for (i in 2:nrow(tree$info)) {
    # We need to first find the current parent
    curr_parent <- as.numeric(tree$info[i,"parent"])

    # Then this child must have one more level than its parent
    level[i] <- level[curr_parent] + 1
  }

  # If we only compute the internal nodes
  internal_node <- which(as.numeric(tree$info[,"terminal"])==0)
  log_prior <- 0
  for (i in 1:length(internal_node)) {
    log_prior <- log_prior + log(alpha) - beta*log(1 + level[internal_node[i]])
  }

  # Also, in each terminal node, it does not split
  terminal_node <- which(as.numeric(tree$info[,"terminal"])==1)
  for (i in 1:length(terminal_node)) {
    log_prior <- log_prior + log(1-alpha*(1+level[terminal_node[i]])^(-beta))

  }
```

```
  return(log_prior)
}
```

12. Main Function

```
train_bart <- function(x, y, num_trees = 1, # number of trees
                       control = list(node_min_size = 5), # control the minimum terminal node size
                       hyper = list(alpha = 0.95, beta = 2, c = 10, alpha_sig=1, beta_sig=1), # Give va
                       init = list(sigma = 0.1), # sigma2 initial value
                       mcmc_par = list(iter = 1000, # number of total iterations
                                       burn = 100, # number of burn-in
                                       thin = 1) # how to thin the chain

){

  # Scale the covariates
  # We need to record how we scaled them so that we can predict new observations.

  # Only calculate those columns who are numeric


  # Get the columns that are numeric
  # Create a holder

  if(is.null(ncol(x))){
    center_x_num = mean(x)
    scale_x_num = sd(x)
    }else{
      center_x <- apply(x, 2, mean)
      scale_x <- apply(x, 2, sd)}

    x <- scale(x)

      center_y <- mean(y)
      scale_y <- sd(y)
      y <- scale(y)




  # Extract control parameters
  node_min_size <- control$node_min_size

  # Extract initial values
  sigma <- init$sigma
  log_lik <- 0

  # Extract hyperparameters
  alpha <- hyper$alpha
  beta <- hyper$beta
```

```r
c <- hyper$c
alpha_sig <- hyper$alpha_sig
beta_sig <- hyper$beta_sig


# Extract MCMC details
iter <- mcmc_par$iter
burn <- mcmc_par$burn
thin <- mcmc_par$thin
totIter <- burn + iter*thin

# Create containers
store_size <- iter
tree_store <- vector("list", store_size)
sigma2_store <- rep(NA, store_size)
log_like_store <- rep(NA, store_size)
num_node <- rep(NA, store_size)
pred <- matrix(NA, nrow = store_size, ncol = length(y))
log_mar_tree <- rep(NA, store_size)

if(num_trees == 1 ){full_condition_store <- rep(NA, store_size)}else{
  full_condition_store <- matrix(NA, ncol = num_trees, nrow = store_size)
}


# Create a tree stump
curr_tree <- create_stump(num_trees = num_trees, y = y, x = x)[[1]]

# Initialize
new_tree <- curr_tree


pb = utils::txtProgressBar(min = 1, max = totIter,
                           style = 3, width = 60,
                           title = 'Running Models...')


# MCMC Iteration
for(i in 1:totIter){
  utils::setTxtProgressBar(pb, i)
  if((i > burn ) & ((i-burn)%%thin == 0 )){

    curr <- (i-burn)/thin
    tree_store[[curr]] <- curr_tree
    sigma2_store[curr] <- sigma
    log_like_store[curr] <- log_lik
    pred[curr,] <- prediction
    num_node[curr] <- num.node
    log_mar_tree[curr] <- log_mar_tree_curr
  }

  # Propose a new tree.
  # To prevent the tree from being too small, we need to grow at the beginning
```

```
type <- sample(c("grow","prune","change","swap"), 1)

#if(i < max( floor(0.1*burn), 10) ){type = "grow"}
if(nrow(curr_tree$info) <= 3){type = "grow"}
# This is a proposed tree, but we need to figure out whether we should use this tree.
new_tree <- update_tree(x,y, type = type, curr_tree = curr_tree, node_min_size = node_min_size)


# New tree, compute the log of marginalized likelihood plus the log of the tree prior
# First, subtract the mean of the tree



# Use Metropolis-Hasting to sample a new tree here, by using the marginal distribution of tree

# Get the new log probability
l_new <- tree_marginal(new_tree, x, y, c, alpha_sig, beta_sig) + get_tree_prior(new_tree, alpha = a

# Get the old log probability
l_old <- tree_marginal(curr_tree, x, y, c, alpha_sig, beta_sig) + get_tree_prior(curr_tree, alpha =


# Since the proposal is symmetric, we can record the transition probability as follows:
a <- exp(l_new - l_old)
l <- runif(1)




# Need to save the full conditional to check the convergence
if(  (i>burn) & ( ((i-burn)%%thin) ==0) ) {full_condition_store[curr] <- l_old}

if(a > l){curr_tree <- new_tree }

# Then, we should update the mu for each terminal node and update the sigma

curr_tree <- fill_update_mu(y, sigma = sigma, c = c, curr_tree = curr_tree)

mu <- curr_tree$info[which(as.numeric(curr_tree$info[,"terminal"])==1),"mu"]

sigma <- sample_sigma(y, mu = mu, alpha_sig, beta_sig, tree = curr_tree, c)

# Get the log likelihood for the model

log_lik <- get_like(curr_tree = curr_tree, x, y, sigma = sigma)


#For now, I will also get the prediction of this iteration
# We have subtracted mu in the previous step

all_mean <- curr_tree$info[,"mu"]
pred_mean <- all_mean[curr_tree$node]
prediction_scaled <- rnorm(length(y), mean = pred_mean, sd = sqrt(sigma))
```

```
    prediction <- prediction_scaled*scale_y + center_y


    num.node <- sum(as.numeric(curr_tree$info[,"terminal"]==1))

    log_mar_tree_curr <- tree_marginal(curr_tree, x, y, c, alpha_sig, beta_sig) + get_tree_prior(curr_t

  }
  # End the iteration loop
  cat('\n')

  return(list(
    tree = tree_store,
    sigma2_scaled = sigma2_store,
    log_like = log_like_store,
    center_x = center_x,
    scale_x = scale_x,
    center_y = center_y,
    scale_y = scale_y,
    iter = iter,
    burn = burn,
    thin = thin,
    store_size = store_size,
    prediction = pred,
    num_node = num_node,
    log_mar_tree = log_mar_tree

  ))


}
```

13. Simulated Dataset

```
set.seed(0)
x_grid <- seq(from = 0.5, to = 9.5, by = 1)
x1 <- sample(x_grid, 1000, replace = TRUE)
x2 <- sample(1:4, replace = TRUE, size = 1000)
Y <- rep(NA, 1000)
res <- 5*rnorm(1000, mean = 0, sd = 1)

Y[ x1<=5 & (x2==1 | x2==2)] <- 8
Y[ x1>5 & (x2==1 | x2==2)] <- 2
Y[ x1<=3 & (x2==3 | x2==4)] <- 1
Y[ (x1>3 & x1<=7) & (x2==3 | x2==4)] <- 5
Y[ x1>7 & (x2==3 | x2==4)] <- 8

true_mean <- Y

cat <- c("A","B","C","D")
x2 <- cat[x2]
```

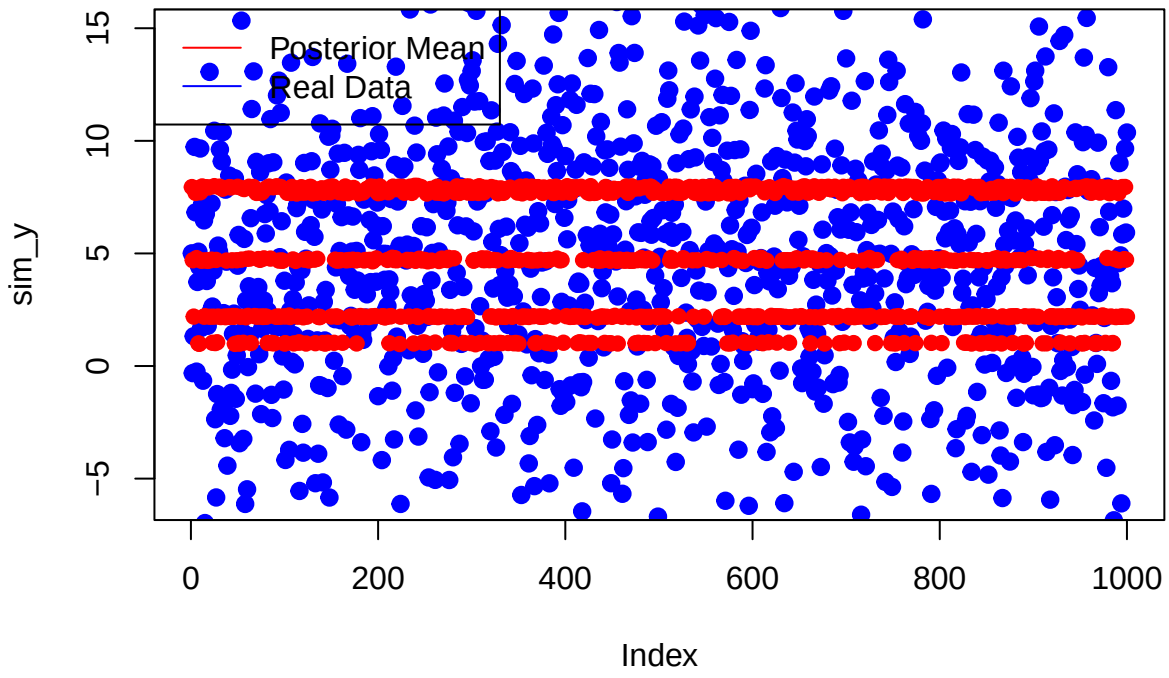
```
sim_dat <- data.frame(y = as.numeric(Y+res), x1= as.numeric(x1), x2=x2)
sim_x <- model.matrix(~ -1 + sim_dat[,2]+sim_dat[,3])
sim_y <- sim_dat[,1]
```

14. In Sample Prediction Result

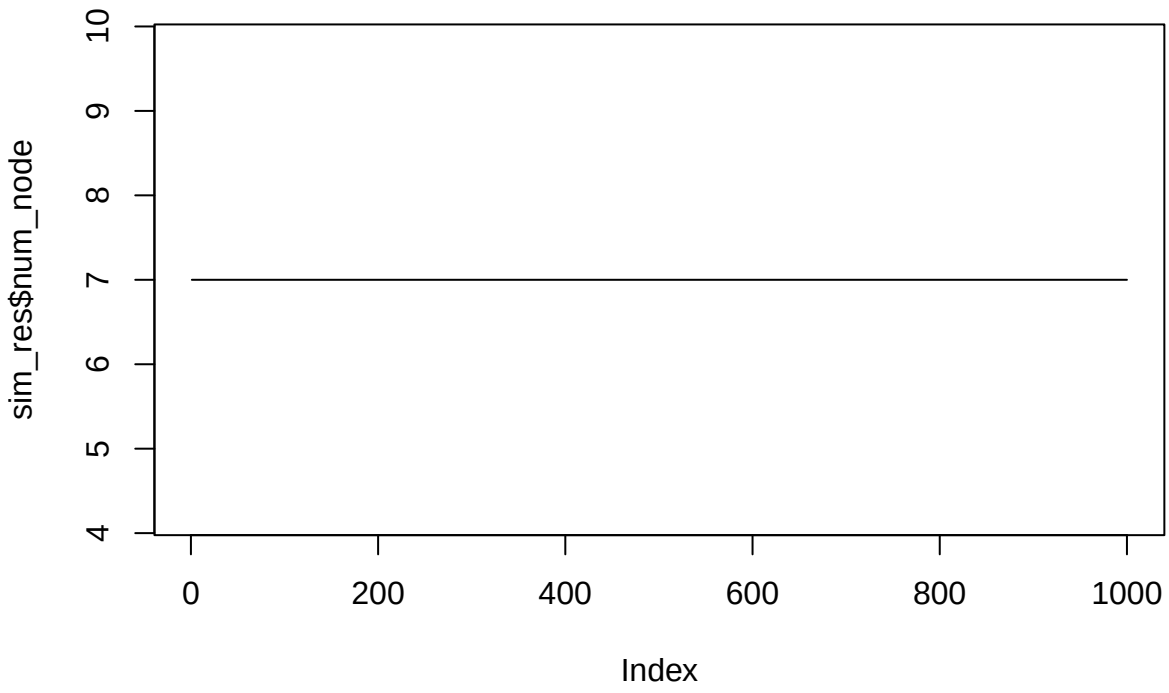
```
sim_res <- train_bart(x = sim_x, y = sim_y, mcmc_par = list(iter = 1000, burn = 200, thin = 2),hyper = l
```

```
##   |                                                                      |
```

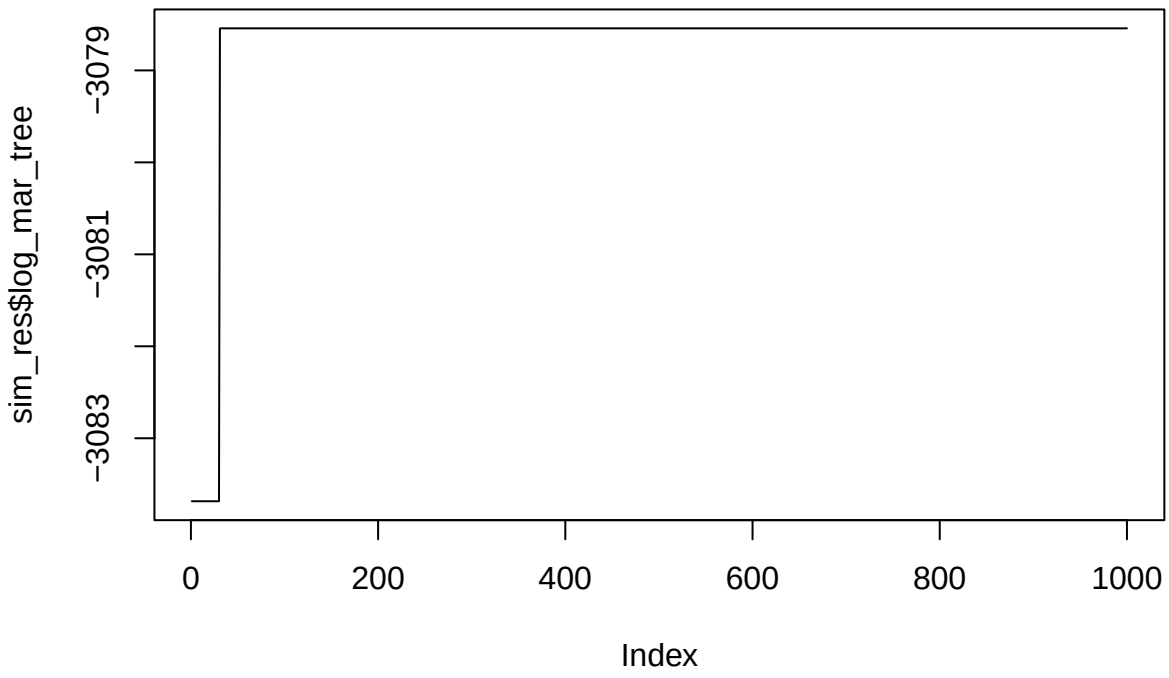
```
pred_y_mean_all <- apply(sim_res$prediction, 2, mean)
plot(sim_y, type = "p", col = "blue", lwd = 2, ylim = c(-6, 15), pch = 19)
lines(pred_y_mean_all, type = "p", col = "red", pch = 19)
legend("topleft",c("Posterior Mean","Real Data"), col = c("red","blue"), lty = c(1,1))
```



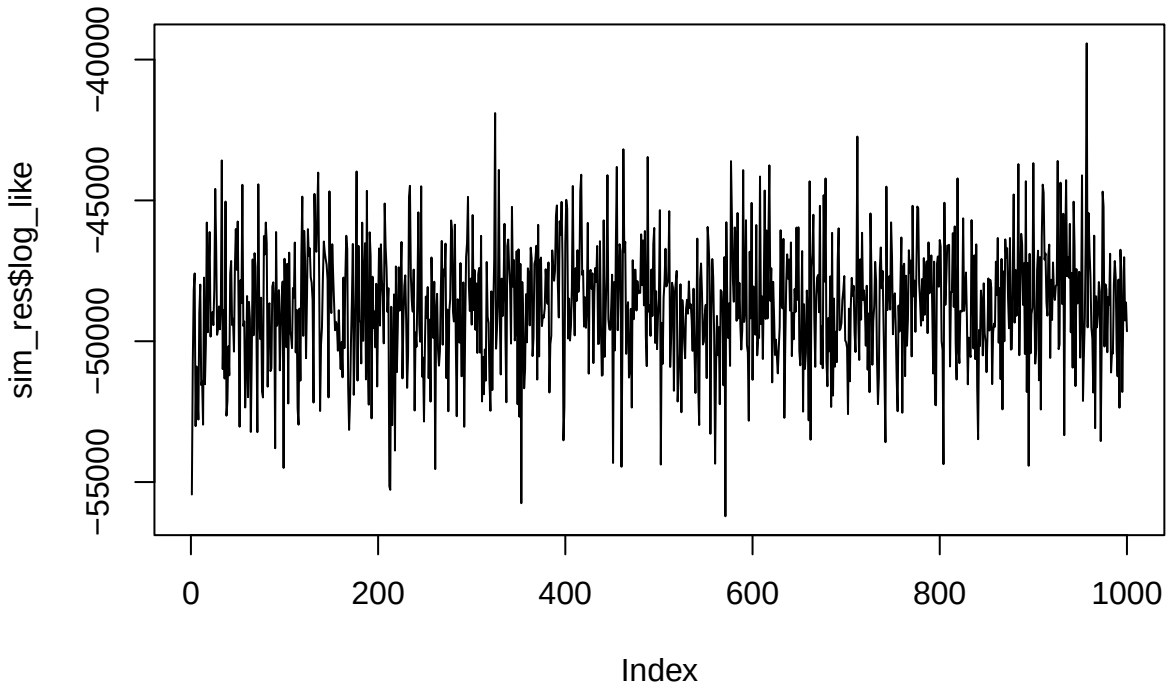

```
plot(sim_res$num_node, type = 'l')
```




```
plot(sim_res$log_mar_tree, type = 'l')
```



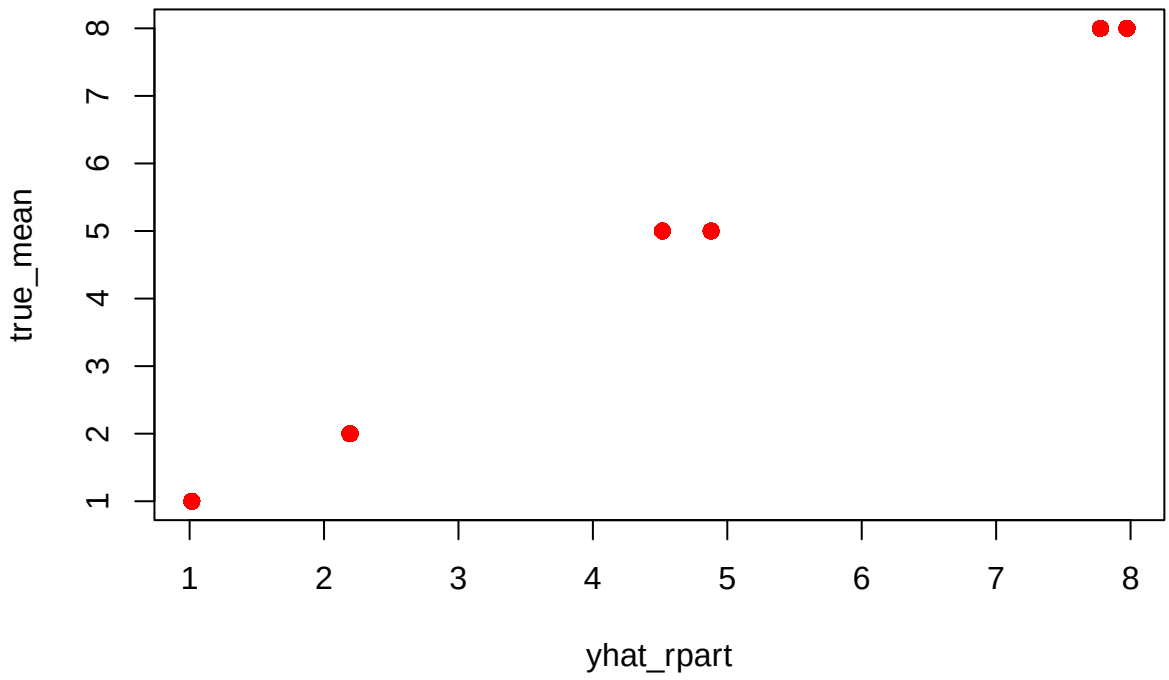

```
plot(sim_res$log_like, type = 'l')
```



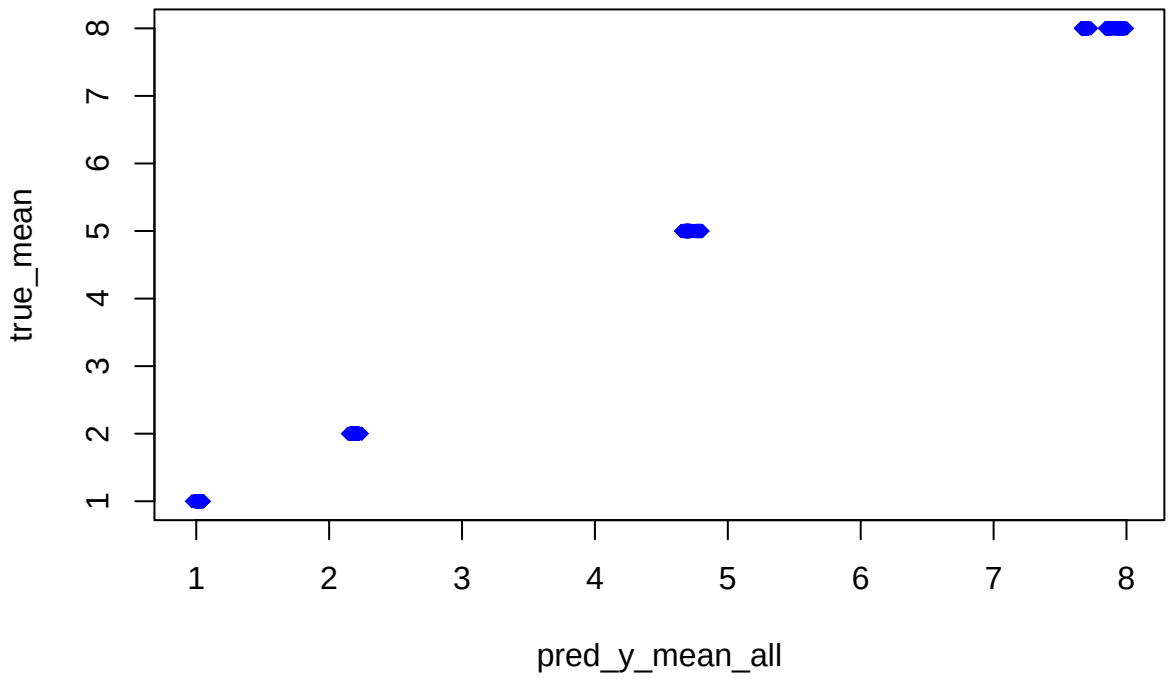
15. Comparing with Rpart

```
library(rpart)
rmod <- rpart(y~., data = sim_dat)
yhat_rpart <- predict(rmod, sim_dat[,2:3])
#plot(yhat_rpart, type = 'p', pch = 19, col = "red")
#lines(pred_y_mean_all, type = 'p', pch = 18, col = "blue")
#legend("topright", c("rpart","cart"), pch = c(19,18), col = c("red","blue"))


plot(x = yhat_rpart, y = true_mean, type = 'p', pch = 19, col = "red")
```

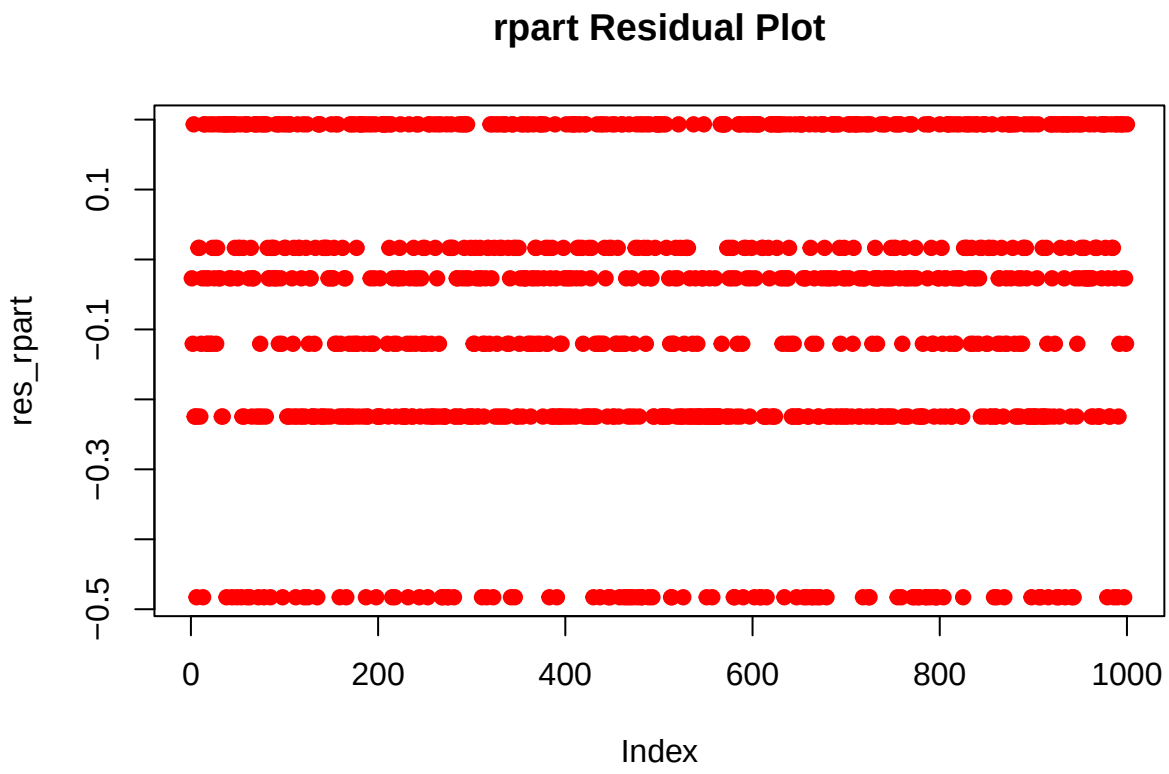



```
plot(x = pred_y_mean_all, y = true_mean, type = 'p', pch = 18, col = "blue")
```



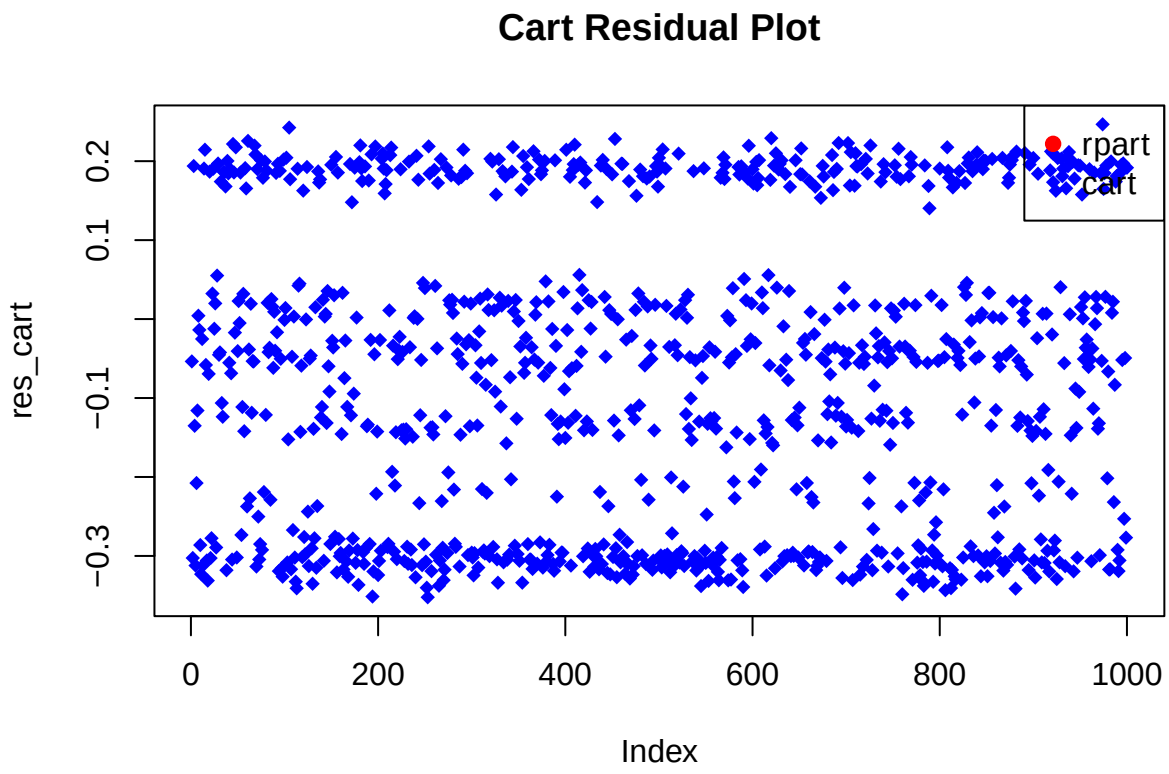

```
res_rpart <- yhat_rpart - true_mean
res_cart <- pred_y_mean_all - true_mean

plot(res_rpart, type = 'p', pch = 19, col = "red", main = "rpart Residual Plot")
```




```
plot(res_cart, type = 'p', pch = 18, col = "blue", main = "Cart Residual Plot")

legend("topright", c("rpart","cart"), pch = c(19,18), col = c("red","blue"))
```




```
sd(res_rpart)
```

```
## [1] 0.2069735
```

```
sd(res_cart)
```

```
## [1] 0.1914061
```

```
sum(res_rpart^2)
```

```
## [1] 47.87081
```

```
sum(res_cart^2)
```

```
## [1] 41.78285
```

16. Comparing with rbart

```
library(rbart)
bart_res <- rbart(sim_x, sim_y, ntree = 1, nskip = 200, ndpost = 2000, k = 2, power = 2, base = 0.95)
```

```
## *****Into main of cpsambrt
## row, cols chvm: 5, 5
## **********************
## n: 1000
## p: 5
## first row: 8.500000, 0.000000
## second row: 3.500000, 0.000000
## last row: 9.500000, 0.000000
## first and last y: 0.140864, 5.506611
## no test observations
## number of trees mean: 1
## number of trees stan dev: 40
## tau: 8.393757
## overalllambda: 34.056930
## overallnu: 10.000000
## burn (nskip): 200
## nd (ndpost): 2000
## nadapt: 1000
## adaptevery: 100
## mean tree prior base: 0.950000
## mean tree prior power: 2.000000
## variance tree prior base: 0.950000
## variance tree prior power: 2.000000
## thread count: 1
## first and last sigmav: 1.000000, 1.000000
## chgv first row: 1.000000, 0.068920
## chgv last row: 0.068920, 1.000000
## mean trees prob birth/death: 0.700000
## mean trees prob birth: 0.500000
## variance trees prob birth/death: 0.700000
## variance trees prob birth: 0.500000
## mean trees initial step width pert move: 0.100000
## variance trees initial step width pert move: 0.100000
## mean trees prob of a change var move : 0.100000
## variance trees prob of a change var move : 0.100000
## mean trees min num obs in bottom node: 5
## variance trees min num obs in bottom node: 5
## *****printevery: 100
## &&& made xinfo
## Variable 0 has numcuts=100 : 0.589109 ... 9.41089
## Variable 1 has numcuts=100 : 0.00990099 ... 0.990099
## Variable 2 has numcuts=100 : 0.00990099 ... 0.990099
## Variable 3 has numcuts=100 : 0.00990099 ... 0.990099
## Variable 4 has numcuts=100 : 0.00990099 ... 0.990099
## Starting MCMC...
##
## Adapt ambrt[0]:pert_rate=0.938931 pertalpha=0.213393 chgv_rate=0.517241   m_rate=0.11
## Adapt sbrt[0]:pert_rate=0.980952 pertalpha=0.222944 chgv_rate=0.833333   m_rate=0.29
## Adapt sbrt[1]:pert_rate=0.987342 pertalpha=0.224396 chgv_rate=0.769231   m_rate=0.12
## Adapt sbrt[2]:pert_rate=0.974684 pertalpha=0.221519 chgv_rate=0.777778   m_rate=0.2
## Adapt sbrt[3]:pert_rate=0.884615 pertalpha=0.201049 chgv_rate=0.666667   m_rate=0.22
## Adapt sbrt[4]:pert_rate=0.976471 pertalpha=0.221925 chgv_rate=0.705882   m_rate=0.31
## Adapt sbrt[5]:pert_rate=0.947826 pertalpha=0.215415 chgv_rate=0.733333   m_rate=0.43
## Adapt sbrt[6]:pert_rate=0.96 pertalpha=0.218182 chgv_rate=0.6   m_rate=0.28
```

```
## Adapt sbrt[7]:pert_rate=0.988372 pertalpha=0.22463 chgv_rate=0.5   m_rate=0.26
## Adapt sbrt[8]:pert_rate=0.987013 pertalpha=0.224321 chgv_rate=0.818182   m_rate=0.33
## Adapt sbrt[9]:pert_rate=0.952381 pertalpha=0.21645 chgv_rate=0.692308   m_rate=0.24
## Adapt sbrt[10]:pert_rate=0.988506 pertalpha=0.22466 chgv_rate=0.777778   m_rate=0.14
## Adapt sbrt[11]:pert_rate=0.974359 pertalpha=0.221445 chgv_rate=0.727273   m_rate=0.25
## Adapt sbrt[12]:pert_rate=0.988764 pertalpha=0.224719 chgv_rate=0.666667   m_rate=0.17
## Adapt sbrt[13]:pert_rate=0.989011 pertalpha=0.224775 chgv_rate=0.6875   m_rate=0.28
## Adapt sbrt[14]:pert_rate=0.97 pertalpha=0.220455 chgv_rate=0.769231   m_rate=0.24
## Adapt sbrt[15]:pert_rate=0.977778 pertalpha=0.222222 chgv_rate=0.75   m_rate=0.25
## Adapt sbrt[16]:pert_rate=0.962963 pertalpha=0.218855 chgv_rate=0.857143   m_rate=0.26
## Adapt sbrt[17]:pert_rate=0.948052 pertalpha=0.215466 chgv_rate=0.666667   m_rate=0.28
## Adapt sbrt[18]:pert_rate=0.988506 pertalpha=0.22466 chgv_rate=0.8   m_rate=0.19
## Adapt sbrt[19]:pert_rate=0.962963 pertalpha=0.218855 chgv_rate=0.777778   m_rate=0.26
## Adapt sbrt[20]:pert_rate=0.974026 pertalpha=0.22137 chgv_rate=0.75   m_rate=0.16
## Adapt sbrt[21]:pert_rate=0.93617 pertalpha=0.212766 chgv_rate=0.75   m_rate=0.24
## Adapt sbrt[22]:pert_rate=0.980198 pertalpha=0.222772 chgv_rate=0.727273   m_rate=0.22
## Adapt sbrt[23]:pert_rate=0.989796 pertalpha=0.224954 chgv_rate=0.75   m_rate=0.25
## Adapt sbrt[24]:pert_rate=0.988636 pertalpha=0.22469 chgv_rate=0.833333   m_rate=0.2
## Adapt sbrt[25]:pert_rate=0.975904 pertalpha=0.221796 chgv_rate=0.769231   m_rate=0.29
## Adapt sbrt[26]:pert_rate=0.939759 pertalpha=0.213582 chgv_rate=0.833333   m_rate=0.3
## Adapt sbrt[27]:pert_rate=0.990741 pertalpha=0.225168 chgv_rate=0.75   m_rate=0.24
## Adapt sbrt[28]:pert_rate=0.979592 pertalpha=0.222635 chgv_rate=0.833333   m_rate=0.35
## Adapt sbrt[29]:pert_rate=0.988889 pertalpha=0.224747 chgv_rate=0.846154   m_rate=0.22
## Adapt sbrt[30]:pert_rate=0.975806 pertalpha=0.221774 chgv_rate=0.588235   m_rate=0.29
## Adapt sbrt[31]:pert_rate=0.959016 pertalpha=0.217958 chgv_rate=0.583333   m_rate=0.38
## Adapt sbrt[32]:pert_rate=0.950617 pertalpha=0.216049 chgv_rate=0.75   m_rate=0.28
## Adapt sbrt[33]:pert_rate=0.95283 pertalpha=0.216552 chgv_rate=0.714286   m_rate=0.25
## Adapt sbrt[34]:pert_rate=0.977273 pertalpha=0.222107 chgv_rate=0.833333   m_rate=0.23
## Adapt sbrt[35]:pert_rate=0.988372 pertalpha=0.22463 chgv_rate=0.777778   m_rate=0.17
## Adapt sbrt[36]:pert_rate=0.979167 pertalpha=0.222538 chgv_rate=0.75   m_rate=0.27
## Adapt sbrt[37]:pert_rate=0.924051 pertalpha=0.210012 chgv_rate=0.8   m_rate=0.21
## Adapt sbrt[38]:pert_rate=0.990385 pertalpha=0.225087 chgv_rate=0.5   m_rate=0.27
## Adapt sbrt[39]:pert_rate=0.920455 pertalpha=0.209194 chgv_rate=0.833333   m_rate=0.26
## Adapt ambrt[0]:pert_rate=0.898182 pertalpha=0.435605 chgv_rate=0.471264   m_rate=0.126214
## Adapt sbrt[0]:pert_rate=0.955556 pertalpha=0.484171 chgv_rate=0.861111   m_rate=0.165049
## Adapt sbrt[1]:pert_rate=0.971963 pertalpha=0.495692 chgv_rate=0.772727   m_rate=0.213592
## Adapt sbrt[2]:pert_rate=0.960784 pertalpha=0.483709 chgv_rate=0.842105   m_rate=0.174757
## Adapt sbrt[3]:pert_rate=0.967033 pertalpha=0.441866 chgv_rate=0.818182   m_rate=0.23301
## Adapt sbrt[4]:pert_rate=0.966667 pertalpha=0.487563 chgv_rate=0.72   m_rate=0.194175
## Adapt sbrt[5]:pert_rate=0.990099 pertalpha=0.484732 chgv_rate=0.862069   m_rate=0.271845
## Adapt sbrt[6]:pert_rate=0.980392 pertalpha=0.486145 chgv_rate=0.666667   m_rate=0.165049
## Adapt sbrt[7]:pert_rate=0.894737 pertalpha=0.456784 chgv_rate=0.666667   m_rate=0.184466
## Adapt sbrt[8]:pert_rate=0.932039 pertalpha=0.475173 chgv_rate=0.857143   m_rate=0.213592
## Adapt sbrt[9]:pert_rate=0.944444 pertalpha=0.464603 chgv_rate=0.724138   m_rate=0.165049
## Adapt sbrt[10]:pert_rate=0.944444 pertalpha=0.482226 chgv_rate=0.8   m_rate=0.165049
## Adapt sbrt[11]:pert_rate=0.929204 pertalpha=0.467654 chgv_rate=0.75   m_rate=0.300971
## Adapt sbrt[12]:pert_rate=0.945652 pertalpha=0.482968 chgv_rate=0.730769   m_rate=0.145631
## Adapt sbrt[13]:pert_rate=0.978723 pertalpha=0.499984 chgv_rate=0.72   m_rate=0.203883
## Adapt sbrt[14]:pert_rate=0.952941 pertalpha=0.477455 chgv_rate=0.851852   m_rate=0.145631
## Adapt sbrt[15]:pert_rate=0.991228 pertalpha=0.50062 chgv_rate=0.875   m_rate=0.165049
## Adapt sbrt[16]:pert_rate=0.988764 pertalpha=0.491809 chgv_rate=0.846154   m_rate=0.203883
## Adapt sbrt[17]:pert_rate=0.950617 pertalpha=0.465514 chgv_rate=0.75   m_rate=0.15534
## Adapt sbrt[18]:pert_rate=0.928571 pertalpha=0.474121 chgv_rate=0.833333   m_rate=0.223301
## Adapt sbrt[19]:pert_rate=0.977011 pertalpha=0.485964 chgv_rate=0.823529   m_rate=0.15534
```

```
## Adapt sbrt[20]:pert_rate=0.959596 pertalpha=0.482785 chgv_rate=0.8   m_rate=0.15534
## Adapt sbrt[21]:pert_rate=0.98 pertalpha=0.473888 chgv_rate=0.823529   m_rate=0.203883
## Adapt sbrt[22]:pert_rate=0.964602 pertalpha=0.488378 chgv_rate=0.791667   m_rate=0.203883
## Adapt sbrt[23]:pert_rate=0.941748 pertalpha=0.481476 chgv_rate=0.8   m_rate=0.223301
## Adapt sbrt[24]:pert_rate=0.978022 pertalpha=0.499436 chgv_rate=0.923077   m_rate=0.291262
## Adapt sbrt[25]:pert_rate=0.95098 pertalpha=0.479373 chgv_rate=0.75   m_rate=0.145631
## Adapt sbrt[26]:pert_rate=0.922222 pertalpha=0.447658 chgv_rate=0.833333   m_rate=0.213592
## Adapt sbrt[27]:pert_rate=0.938776 pertalpha=0.480415 chgv_rate=0.772727   m_rate=0.165049
## Adapt sbrt[28]:pert_rate=0.944444 pertalpha=0.477877 chgv_rate=0.875   m_rate=0.126214
## Adapt sbrt[29]:pert_rate=0.942529 pertalpha=0.481434 chgv_rate=0.851852   m_rate=0.135922
## Adapt sbrt[30]:pert_rate=0.973684 pertalpha=0.490768 chgv_rate=0.642857   m_rate=0.23301
## Adapt sbrt[31]:pert_rate=0.940594 pertalpha=0.465932 chgv_rate=0.65   m_rate=0.223301
## Adapt sbrt[32]:pert_rate=0.94382 pertalpha=0.463436 chgv_rate=0.857143   m_rate=0.135922
## Adapt sbrt[33]:pert_rate=0.957895 pertalpha=0.471442 chgv_rate=0.814815   m_rate=0.213592
## Adapt sbrt[34]:pert_rate=0.873684 pertalpha=0.441027 chgv_rate=0.894737   m_rate=0.194175
## Adapt sbrt[35]:pert_rate=0.926606 pertalpha=0.473053 chgv_rate=0.851852   m_rate=0.23301
## Adapt sbrt[36]:pert_rate=0.92233 pertalpha=0.466485 chgv_rate=0.8   m_rate=0.262136
## Adapt sbrt[37]:pert_rate=0.987952 pertalpha=0.471548 chgv_rate=0.851852   m_rate=0.194175
## Adapt sbrt[38]:pert_rate=0.962963 pertalpha=0.492616 chgv_rate=0.740741   m_rate=0.174757
## Adapt sbrt[39]:pert_rate=0.969072 pertalpha=0.460737 chgv_rate=0.818182   m_rate=0.271845
## Adapt ambrt[0]:pert_rate=0.826725 pertalpha=0.818467 chgv_rate=0.464516   m_rate=0.0776699
## Adapt sbrt[0]:pert_rate=0.920455 pertalpha=1.01286 chgv_rate=0.888889   m_rate=0.184466
## Adapt sbrt[1]:pert_rate=0.964706 pertalpha=1.08681 chgv_rate=0.75   m_rate=0.184466
## Adapt sbrt[2]:pert_rate=0.939394 pertalpha=1.03271 chgv_rate=0.833333   m_rate=0.213592
## Adapt sbrt[3]:pert_rate=0.901639 pertalpha=0.905463 chgv_rate=0.9   m_rate=0.252427
## Adapt sbrt[4]:pert_rate=0.902174 pertalpha=0.999696 chgv_rate=0.775   m_rate=0.203883
## Adapt sbrt[5]:pert_rate=0.917526 pertalpha=1.01081 chgv_rate=0.871795   m_rate=0.23301
## Adapt sbrt[6]:pert_rate=0.843537 pertalpha=0.932003 chgv_rate=0.772727   m_rate=0.242718
## Adapt sbrt[7]:pert_rate=0.965909 pertalpha=1.00275 chgv_rate=0.689655   m_rate=0.174757
## Adapt sbrt[8]:pert_rate=0.90566 pertalpha=0.978057 chgv_rate=0.914286   m_rate=0.194175
## Adapt sbrt[9]:pert_rate=0.885057 pertalpha=0.934546 chgv_rate=0.72093   m_rate=0.223301
## Adapt sbrt[10]:pert_rate=0.876106 pertalpha=0.960184 chgv_rate=0.805556   m_rate=0.320388
## Adapt sbrt[11]:pert_rate=0.893617 pertalpha=0.94978 chgv_rate=0.8   m_rate=0.271845
## Adapt sbrt[12]:pert_rate=0.917647 pertalpha=1.00726 chgv_rate=0.75   m_rate=0.271845
## Adapt sbrt[13]:pert_rate=0.873786 pertalpha=0.992906 chgv_rate=0.72973   m_rate=0.320388
## Adapt sbrt[14]:pert_rate=0.939394 pertalpha=1.01936 chgv_rate=0.878049   m_rate=0.165049
## Adapt sbrt[15]:pert_rate=0.941748 pertalpha=1.0715 chgv_rate=0.875   m_rate=0.31068
## Adapt sbrt[16]:pert_rate=0.831461 pertalpha=0.929364 chgv_rate=0.75   m_rate=0.203883
## Adapt sbrt[17]:pert_rate=0.941748 pertalpha=0.996355 chgv_rate=0.769231   m_rate=0.203883
## Adapt sbrt[18]:pert_rate=0.95 pertalpha=1.02367 chgv_rate=0.709677   m_rate=0.184466
## Adapt sbrt[19]:pert_rate=0.894737 pertalpha=0.988204 chgv_rate=0.846154   m_rate=0.262136
## Adapt sbrt[20]:pert_rate=0.858491 pertalpha=0.941969 chgv_rate=0.806452   m_rate=0.252427
## Adapt sbrt[21]:pert_rate=0.890909 pertalpha=0.959525 chgv_rate=0.821429   m_rate=0.184466
## Adapt sbrt[22]:pert_rate=0.924731 pertalpha=1.02641 chgv_rate=0.8   m_rate=0.271845
## Adapt sbrt[23]:pert_rate=0.913462 pertalpha=0.999568 chgv_rate=0.848485   m_rate=0.262136
## Adapt sbrt[24]:pert_rate=0.928571 pertalpha=1.054 chgv_rate=0.891892   m_rate=0.223301
## Adapt sbrt[25]:pert_rate=0.929293 pertalpha=1.01245 chgv_rate=0.777778   m_rate=0.300971
## Adapt sbrt[26]:pert_rate=0.818182 pertalpha=0.832423 chgv_rate=0.892857   m_rate=0.242718
## Adapt sbrt[27]:pert_rate=0.954023 pertalpha=1.04165 chgv_rate=0.857143   m_rate=0.174757
## Adapt sbrt[28]:pert_rate=0.943396 pertalpha=1.02461 chgv_rate=0.846154   m_rate=0.330097
## Adapt sbrt[29]:pert_rate=0.977011 pertalpha=1.06901 chgv_rate=0.894737   m_rate=0.203883
## Adapt sbrt[30]:pert_rate=0.898876 pertalpha=1.00259 chgv_rate=0.666667   m_rate=0.271845
## Adapt sbrt[31]:pert_rate=0.928571 pertalpha=0.983299 chgv_rate=0.766667   m_rate=0.165049
## Adapt sbrt[32]:pert_rate=0.92381 pertalpha=0.973015 chgv_rate=0.882353   m_rate=0.213592
```

```
## Adapt sbrt[33]:pert_rate=0.953271 pertalpha=1.02139 chgv_rate=0.815789   m_rate=0.281553
## Adapt sbrt[34]:pert_rate=0.925926 pertalpha=0.928087 chgv_rate=0.793103   m_rate=0.291262
## Adapt sbrt[35]:pert_rate=0.866667 pertalpha=0.931771 chgv_rate=0.837838   m_rate=0.145631
## Adapt sbrt[36]:pert_rate=0.904255 pertalpha=0.958685 chgv_rate=0.789474   m_rate=0.184466
## Adapt sbrt[37]:pert_rate=0.921348 pertalpha=0.98741 chgv_rate=0.857143   m_rate=0.242718
## Adapt sbrt[38]:pert_rate=0.932039 pertalpha=1.04349 chgv_rate=0.775   m_rate=0.31068
## Adapt sbrt[39]:pert_rate=0.946237 pertalpha=0.990832 chgv_rate=0.866667   m_rate=0.223301
## Adapt ambrt[0]:pert_rate=0.642633 pertalpha=1.1954 chgv_rate=0.449541   m_rate=0.0582524
## Adapt sbrt[0]:pert_rate=0.921348 pertalpha=2 chgv_rate=0.867925   m_rate=0.126214
## Adapt sbrt[1]:pert_rate=0.821782 pertalpha=2 chgv_rate=0.73913   m_rate=0.194175
## Adapt sbrt[2]:pert_rate=0.863636 pertalpha=2 chgv_rate=0.833333   m_rate=0.15534
## Adapt sbrt[3]:pert_rate=0.848837 pertalpha=1.7468 chgv_rate=0.878788   m_rate=0.184466
## Adapt sbrt[4]:pert_rate=0.891566 pertalpha=2 chgv_rate=0.795918   m_rate=0.194175
## Adapt sbrt[5]:pert_rate=0.797619 pertalpha=1.83236 chgv_rate=0.888889   m_rate=0.165049
## Adapt sbrt[6]:pert_rate=0.852632 pertalpha=1.80603 chgv_rate=0.806452   m_rate=0.223301
## Adapt sbrt[7]:pert_rate=0.873563 pertalpha=1.99084 chgv_rate=0.816327   m_rate=0.184466
## Adapt sbrt[8]:pert_rate=0.755814 pertalpha=1.68007 chgv_rate=0.9   m_rate=0.15534
## Adapt sbrt[9]:pert_rate=0.846939 pertalpha=1.79887 chgv_rate=0.736842   m_rate=0.213592
## Adapt sbrt[10]:pert_rate=0.870968 pertalpha=1.90066 chgv_rate=0.8125   m_rate=0.242718
## Adapt sbrt[11]:pert_rate=0.892473 pertalpha=1.92649 chgv_rate=0.833333   m_rate=0.135922
## Adapt sbrt[12]:pert_rate=0.835052 pertalpha=1.91162 chgv_rate=0.755556   m_rate=0.320388
## Adapt sbrt[13]:pert_rate=0.872093 pertalpha=1.96797 chgv_rate=0.791667   m_rate=0.194175
## Adapt sbrt[14]:pert_rate=0.813725 pertalpha=1.88518 chgv_rate=0.872727   m_rate=0.213592
## Adapt sbrt[15]:pert_rate=0.892473 pertalpha=2 chgv_rate=0.90625   m_rate=0.194175
## Adapt sbrt[16]:pert_rate=0.854545 pertalpha=1.80496 chgv_rate=0.771429   m_rate=0.271845
## Adapt sbrt[17]:pert_rate=0.830189 pertalpha=1.87992 chgv_rate=0.833333   m_rate=0.271845
## Adapt sbrt[18]:pert_rate=0.876289 pertalpha=2 chgv_rate=0.727273   m_rate=0.184466
## Adapt sbrt[19]:pert_rate=0.933962 pertalpha=2 chgv_rate=0.860465   m_rate=0.320388
## Adapt sbrt[20]:pert_rate=0.913043 pertalpha=1.95468 chgv_rate=0.790698   m_rate=0.165049
## Adapt sbrt[21]:pert_rate=0.841463 pertalpha=1.83501 chgv_rate=0.787879   m_rate=0.116505
## Adapt sbrt[22]:pert_rate=0.838384 pertalpha=1.95573 chgv_rate=0.851064   m_rate=0.194175
## Adapt sbrt[23]:pert_rate=0.906977 pertalpha=2 chgv_rate=0.833333   m_rate=0.349515
## Adapt sbrt[24]:pert_rate=0.915094 pertalpha=2 chgv_rate=0.866667   m_rate=0.271845
## Adapt sbrt[25]:pert_rate=0.884298 pertalpha=2 chgv_rate=0.790698   m_rate=0.174757
## Adapt sbrt[26]:pert_rate=0.840909 pertalpha=1.59089 chgv_rate=0.916667   m_rate=0.174757
## Adapt sbrt[27]:pert_rate=0.870968 pertalpha=2 chgv_rate=0.893617   m_rate=0.174757
## Adapt sbrt[28]:pert_rate=0.865169 pertalpha=2 chgv_rate=0.833333   m_rate=0.194175
## Adapt sbrt[29]:pert_rate=0.885417 pertalpha=2 chgv_rate=0.886792   m_rate=0.184466
## Adapt sbrt[30]:pert_rate=0.918605 pertalpha=2 chgv_rate=0.722222   m_rate=0.184466
## Adapt sbrt[31]:pert_rate=0.888889 pertalpha=1.98646 chgv_rate=0.820513   m_rate=0.223301
## Adapt sbrt[32]:pert_rate=0.896 pertalpha=1.98141 chgv_rate=0.84375   m_rate=0.184466
## Adapt sbrt[33]:pert_rate=0.804878 pertalpha=1.8684 chgv_rate=0.829787   m_rate=0.213592
## Adapt sbrt[34]:pert_rate=0.929412 pertalpha=1.9604 chgv_rate=0.820513   m_rate=0.223301
## Adapt sbrt[35]:pert_rate=0.906542 pertalpha=1.91975 chgv_rate=0.822222   m_rate=0.0776699
## Adapt sbrt[36]:pert_rate=0.92381 pertalpha=2 chgv_rate=0.807692   m_rate=0.271845
## Adapt sbrt[37]:pert_rate=0.895349 pertalpha=2 chgv_rate=0.810345   m_rate=0.271845
## Adapt sbrt[38]:pert_rate=0.860215 pertalpha=2 chgv_rate=0.816327   m_rate=0.174757
## Adapt sbrt[39]:pert_rate=0.920792 pertalpha=2 chgv_rate=0.829268   m_rate=0.252427
## Adapt ambrt[0]:pert_rate=0.63622 pertalpha=1.72849 chgv_rate=0.461268   m_rate=0.0582524
## Adapt sbrt[0]:pert_rate=0.952381 pertalpha=2 chgv_rate=0.887097   m_rate=0.271845
## Adapt sbrt[1]:pert_rate=0.970588 pertalpha=2 chgv_rate=0.754717   m_rate=0.174757
## Adapt sbrt[2]:pert_rate=0.963636 pertalpha=2 chgv_rate=0.847826   m_rate=0.213592
## Adapt sbrt[3]:pert_rate=0.988889 pertalpha=2 chgv_rate=0.883721   m_rate=0.184466
## Adapt sbrt[4]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.753846   m_rate=0.281553
```

```
## Adapt sbrt[5]:pert_rate=0.978723 pertalpha=2 chgv_rate=0.913793   m_rate=0.23301
## Adapt sbrt[6]:pert_rate=0.975904 pertalpha=2 chgv_rate=0.780488   m_rate=0.135922
## Adapt sbrt[7]:pert_rate=0.967391 pertalpha=2 chgv_rate=0.824561   m_rate=0.252427
## Adapt sbrt[8]:pert_rate=0.899083 pertalpha=2 chgv_rate=0.90566   m_rate=0.242718
## Adapt sbrt[9]:pert_rate=0.98 pertalpha=2 chgv_rate=0.763889   m_rate=0.242718
## Adapt sbrt[10]:pert_rate=0.903614 pertalpha=2 chgv_rate=0.848485   m_rate=0.223301
## Adapt sbrt[11]:pert_rate=0.965909 pertalpha=2 chgv_rate=0.830508   m_rate=0.194175
## Adapt sbrt[12]:pert_rate=0.876289 pertalpha=2 chgv_rate=0.77193   m_rate=0.203883
## Adapt sbrt[13]:pert_rate=0.989362 pertalpha=2 chgv_rate=0.814815   m_rate=0.135922
## Adapt sbrt[14]:pert_rate=0.983193 pertalpha=2 chgv_rate=0.882353   m_rate=0.116505
## Adapt sbrt[15]:pert_rate=0.983333 pertalpha=2 chgv_rate=0.863636   m_rate=0.262136
## Adapt sbrt[16]:pert_rate=0.891892 pertalpha=2 chgv_rate=0.780488   m_rate=0.223301
## Adapt sbrt[17]:pert_rate=0.94 pertalpha=2 chgv_rate=0.854167   m_rate=0.23301
## Adapt sbrt[18]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.74   m_rate=0.291262
## Adapt sbrt[19]:pert_rate=0.94382 pertalpha=2 chgv_rate=0.807018   m_rate=0.281553
## Adapt sbrt[20]:pert_rate=0.990291 pertalpha=2 chgv_rate=0.773585   m_rate=0.145631
## Adapt sbrt[21]:pert_rate=0.972477 pertalpha=2 chgv_rate=0.744186   m_rate=0.271845
## Adapt sbrt[22]:pert_rate=0.957895 pertalpha=2 chgv_rate=0.854545   m_rate=0.174757
## Adapt sbrt[23]:pert_rate=0.966667 pertalpha=2 chgv_rate=0.836735   m_rate=0.281553
## Adapt sbrt[24]:pert_rate=0.95098 pertalpha=2 chgv_rate=0.883333   m_rate=0.194175
## Adapt sbrt[25]:pert_rate=0.914286 pertalpha=2 chgv_rate=0.821429   m_rate=0.23301
## Adapt sbrt[26]:pert_rate=0.909091 pertalpha=2 chgv_rate=0.890909   m_rate=0.184466
## Adapt sbrt[27]:pert_rate=0.9 pertalpha=2 chgv_rate=0.894737   m_rate=0.31068
## Adapt sbrt[28]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.869565   m_rate=0.184466
## Adapt sbrt[29]:pert_rate=0.857143 pertalpha=2 chgv_rate=0.876923   m_rate=0.184466
## Adapt sbrt[30]:pert_rate=0.923077 pertalpha=2 chgv_rate=0.741935   m_rate=0.213592
## Adapt sbrt[31]:pert_rate=0.972222 pertalpha=2 chgv_rate=0.836735   m_rate=0.281553
## Adapt sbrt[32]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.847826   m_rate=0.252427
## Adapt sbrt[33]:pert_rate=0.970588 pertalpha=2 chgv_rate=0.825397   m_rate=0.271845
## Adapt sbrt[34]:pert_rate=0.989362 pertalpha=2 chgv_rate=0.840909   m_rate=0.174757
## Adapt sbrt[35]:pert_rate=0.988506 pertalpha=2 chgv_rate=0.811321   m_rate=0.116505
## Adapt sbrt[36]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.816667   m_rate=0.0970874
## Adapt sbrt[37]:pert_rate=0.966667 pertalpha=2 chgv_rate=0.819444   m_rate=0.281553
## Adapt sbrt[38]:pert_rate=0.980198 pertalpha=2 chgv_rate=0.827586   m_rate=0.145631
## Adapt sbrt[39]:pert_rate=0.988764 pertalpha=2 chgv_rate=0.854167   m_rate=0.174757
## Adapt ambrt[0]:pert_rate=0.670947 pertalpha=2 chgv_rate=0.430518   m_rate=0.0291262
## Adapt sbrt[0]:pert_rate=0.977778 pertalpha=2 chgv_rate=0.901408   m_rate=0.135922
## Adapt sbrt[1]:pert_rate=0.946429 pertalpha=2 chgv_rate=0.742424   m_rate=0.252427
## Adapt sbrt[2]:pert_rate=0.968085 pertalpha=2 chgv_rate=0.85   m_rate=0.213592
## Adapt sbrt[3]:pert_rate=0.978947 pertalpha=2 chgv_rate=0.913793   m_rate=0.194175
## Adapt sbrt[4]:pert_rate=0.96 pertalpha=2 chgv_rate=0.746835   m_rate=0.223301
## Adapt sbrt[5]:pert_rate=0.989899 pertalpha=2 chgv_rate=0.893939   m_rate=0.223301
## Adapt sbrt[6]:pert_rate=0.981818 pertalpha=2 chgv_rate=0.759259   m_rate=0.291262
## Adapt sbrt[7]:pert_rate=0.964539 pertalpha=2 chgv_rate=0.810811   m_rate=0.262136
## Adapt sbrt[8]:pert_rate=0.888889 pertalpha=2 chgv_rate=0.876923   m_rate=0.15534
## Adapt sbrt[9]:pert_rate=0.932584 pertalpha=2 chgv_rate=0.783133   m_rate=0.23301
## Adapt sbrt[10]:pert_rate=0.95 pertalpha=2 chgv_rate=0.855263   m_rate=0.213592
## Adapt sbrt[11]:pert_rate=0.963855 pertalpha=2 chgv_rate=0.826667   m_rate=0.23301
## Adapt sbrt[12]:pert_rate=0.958333 pertalpha=2 chgv_rate=0.783784   m_rate=0.330097
## Adapt sbrt[13]:pert_rate=0.974359 pertalpha=2 chgv_rate=0.825397   m_rate=0.291262
## Adapt sbrt[14]:pert_rate=0.941176 pertalpha=2 chgv_rate=0.868421   m_rate=0.252427
## Adapt sbrt[15]:pert_rate=0.907216 pertalpha=2 chgv_rate=0.846154   m_rate=0.116505
## Adapt sbrt[16]:pert_rate=0.910714 pertalpha=2 chgv_rate=0.765957   m_rate=0.213592
## Adapt sbrt[17]:pert_rate=0.980583 pertalpha=2 chgv_rate=0.836066   m_rate=0.262136
```

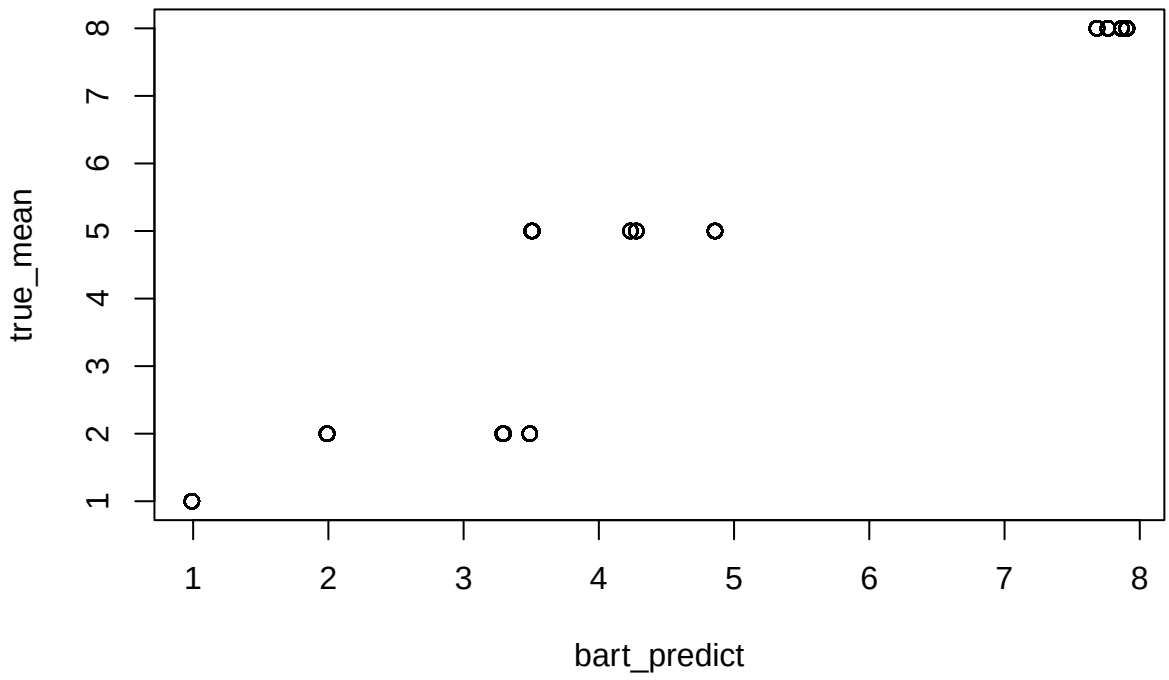
```
## Adapt sbrt[18]:pert_rate=0.977528 pertalpha=2 chgv_rate=0.78125   m_rate=0.184466
## Adapt sbrt[19]:pert_rate=0.978261 pertalpha=2 chgv_rate=0.818182   m_rate=0.262136
## Adapt sbrt[20]:pert_rate=0.987952 pertalpha=2 chgv_rate=0.793651   m_rate=0.23301
## Adapt sbrt[21]:pert_rate=0.93 pertalpha=2 chgv_rate=0.769231   m_rate=0.135922
## Adapt sbrt[22]:pert_rate=0.947368 pertalpha=2 chgv_rate=0.867647   m_rate=0.135922
## Adapt sbrt[23]:pert_rate=0.969697 pertalpha=2 chgv_rate=0.844828   m_rate=0.291262
## Adapt sbrt[24]:pert_rate=0.905263 pertalpha=2 chgv_rate=0.885714   m_rate=0.23301
## Adapt sbrt[25]:pert_rate=0.987654 pertalpha=2 chgv_rate=0.791667   m_rate=0.213592
## Adapt sbrt[26]:pert_rate=0.962963 pertalpha=2 chgv_rate=0.852941   m_rate=0.300971
## Adapt sbrt[27]:pert_rate=0.969072 pertalpha=2 chgv_rate=0.870968   m_rate=0.213592
## Adapt sbrt[28]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.892857   m_rate=0.135922
## Adapt sbrt[29]:pert_rate=0.928058 pertalpha=2 chgv_rate=0.869048   m_rate=0.417476
## Adapt sbrt[30]:pert_rate=0.92 pertalpha=2 chgv_rate=0.742424   m_rate=0.271845
## Adapt sbrt[31]:pert_rate=0.90678 pertalpha=2 chgv_rate=0.827586   m_rate=0.23301
## Adapt sbrt[32]:pert_rate=0.981818 pertalpha=2 chgv_rate=0.8   m_rate=0.242718
## Adapt sbrt[33]:pert_rate=0.990099 pertalpha=2 chgv_rate=0.828571   m_rate=0.194175
## Adapt sbrt[34]:pert_rate=0.878788 pertalpha=2 chgv_rate=0.872727   m_rate=0.184466
## Adapt sbrt[35]:pert_rate=0.978495 pertalpha=2 chgv_rate=0.833333   m_rate=0.0970874
## Adapt sbrt[36]:pert_rate=0.970297 pertalpha=2 chgv_rate=0.828571   m_rate=0.242718
## Adapt sbrt[37]:pert_rate=0.965812 pertalpha=2 chgv_rate=0.807229   m_rate=0.262136
## Adapt sbrt[38]:pert_rate=0.990291 pertalpha=2 chgv_rate=0.814286   m_rate=0.184466
## Adapt sbrt[39]:pert_rate=0.989691 pertalpha=2 chgv_rate=0.864407   m_rate=0.203883
## Adapt ambrt[0]:pert_rate=0.698529 pertalpha=2 chgv_rate=0.418764   m_rate=0.126214
## Adapt sbrt[0]:pert_rate=0.989362 pertalpha=2 chgv_rate=0.869565   m_rate=0.252427
## Adapt sbrt[1]:pert_rate=0.975207 pertalpha=2 chgv_rate=0.728395   m_rate=0.194175
## Adapt sbrt[2]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.871429   m_rate=0.116505
## Adapt sbrt[3]:pert_rate=0.971698 pertalpha=2 chgv_rate=0.90411   m_rate=0.23301
## Adapt sbrt[4]:pert_rate=0.928571 pertalpha=2 chgv_rate=0.772727   m_rate=0.23301
## Adapt sbrt[5]:pert_rate=0.927835 pertalpha=2 chgv_rate=0.886076   m_rate=0.213592
## Adapt sbrt[6]:pert_rate=0.919643 pertalpha=2 chgv_rate=0.769231   m_rate=0.291262
## Adapt sbrt[7]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.8   m_rate=0.174757
## Adapt sbrt[8]:pert_rate=0.988506 pertalpha=2 chgv_rate=0.878378   m_rate=0.165049
## Adapt sbrt[9]:pert_rate=0.964286 pertalpha=2 chgv_rate=0.783505   m_rate=0.145631
## Adapt sbrt[10]:pert_rate=0.99115 pertalpha=2 chgv_rate=0.811765   m_rate=0.213592
## Adapt sbrt[11]:pert_rate=0.979592 pertalpha=2 chgv_rate=0.833333   m_rate=0.223301
## Adapt sbrt[12]:pert_rate=0.931034 pertalpha=2 chgv_rate=0.788235   m_rate=0.15534
## Adapt sbrt[13]:pert_rate=0.988506 pertalpha=2 chgv_rate=0.855263   m_rate=0.15534
## Adapt sbrt[14]:pert_rate=0.978723 pertalpha=2 chgv_rate=0.89011   m_rate=0.203883
## Adapt sbrt[15]:pert_rate=0.913043 pertalpha=2 chgv_rate=0.868852   m_rate=0.213592
## Adapt sbrt[16]:pert_rate=0.980769 pertalpha=2 chgv_rate=0.761905   m_rate=0.262136
## Adapt sbrt[17]:pert_rate=0.929412 pertalpha=2 chgv_rate=0.857143   m_rate=0.174757
## Adapt sbrt[18]:pert_rate=0.979167 pertalpha=2 chgv_rate=0.805195   m_rate=0.203883
## Adapt sbrt[19]:pert_rate=0.954545 pertalpha=2 chgv_rate=0.842105   m_rate=0.184466
## Adapt sbrt[20]:pert_rate=0.913978 pertalpha=2 chgv_rate=0.808219   m_rate=0.165049
## Adapt sbrt[21]:pert_rate=0.931624 pertalpha=2 chgv_rate=0.8   m_rate=0.223301
## Adapt sbrt[22]:pert_rate=0.93617 pertalpha=2 chgv_rate=0.871795   m_rate=0.174757
## Adapt sbrt[23]:pert_rate=0.944 pertalpha=2 chgv_rate=0.859155   m_rate=0.31068
## Adapt sbrt[24]:pert_rate=0.909091 pertalpha=2 chgv_rate=0.884615   m_rate=0.349515
## Adapt sbrt[25]:pert_rate=0.987805 pertalpha=2 chgv_rate=0.790123   m_rate=0.174757
## Adapt sbrt[26]:pert_rate=0.967742 pertalpha=2 chgv_rate=0.85   m_rate=0.165049
## Adapt sbrt[27]:pert_rate=0.990099 pertalpha=2 chgv_rate=0.867647   m_rate=0.135922
## Adapt sbrt[28]:pert_rate=0.959184 pertalpha=2 chgv_rate=0.878788   m_rate=0.242718
## Adapt sbrt[29]:pert_rate=0.990741 pertalpha=2 chgv_rate=0.880435   m_rate=0.194175
## Adapt sbrt[30]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.779221   m_rate=0.174757
```

```
## Adapt sbrt[31]:pert_rate=0.947368 pertalpha=2 chgv_rate=0.838235   m_rate=0.194175
## Adapt sbrt[32]:pert_rate=0.990385 pertalpha=2 chgv_rate=0.805195   m_rate=0.165049
## Adapt sbrt[33]:pert_rate=0.89916 pertalpha=2 chgv_rate=0.831169   m_rate=0.23301
## Adapt sbrt[34]:pert_rate=0.955056 pertalpha=2 chgv_rate=0.880597   m_rate=0.223301
## Adapt sbrt[35]:pert_rate=0.96875 pertalpha=2 chgv_rate=0.831325   m_rate=0.15534
## Adapt sbrt[36]:pert_rate=0.968421 pertalpha=2 chgv_rate=0.839506   m_rate=0.116505
## Adapt sbrt[37]:pert_rate=0.976744 pertalpha=2 chgv_rate=0.82   m_rate=0.165049
## Adapt sbrt[38]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.82716   m_rate=0.135922
## Adapt sbrt[39]:pert_rate=0.989899 pertalpha=2 chgv_rate=0.866667   m_rate=0.15534
## Adapt ambrt[0]:pert_rate=0.662957 pertalpha=2 chgv_rate=0.412916   m_rate=0.0679612
## Adapt sbrt[0]:pert_rate=0.962963 pertalpha=2 chgv_rate=0.87   m_rate=0.15534
## Adapt sbrt[1]:pert_rate=0.960396 pertalpha=2 chgv_rate=0.738636   m_rate=0.165049
## Adapt sbrt[2]:pert_rate=0.98913 pertalpha=2 chgv_rate=0.871795   m_rate=0.0970874
## Adapt sbrt[3]:pert_rate=0.981481 pertalpha=2 chgv_rate=0.901235   m_rate=0.213592
## Adapt sbrt[4]:pert_rate=0.987805 pertalpha=2 chgv_rate=0.769231   m_rate=0.23301
## Adapt sbrt[5]:pert_rate=0.959184 pertalpha=2 chgv_rate=0.878049   m_rate=0.223301
## Adapt sbrt[6]:pert_rate=0.989011 pertalpha=2 chgv_rate=0.783784   m_rate=0.223301
## Adapt sbrt[7]:pert_rate=0.990196 pertalpha=2 chgv_rate=0.805825   m_rate=0.165049
## Adapt sbrt[8]:pert_rate=0.977273 pertalpha=2 chgv_rate=0.877778   m_rate=0.23301
## Adapt sbrt[9]:pert_rate=0.90099 pertalpha=2 chgv_rate=0.805556   m_rate=0.252427
## Adapt sbrt[10]:pert_rate=0.982143 pertalpha=2 chgv_rate=0.806452   m_rate=0.194175
## Adapt sbrt[11]:pert_rate=0.946809 pertalpha=2 chgv_rate=0.835165   m_rate=0.213592
## Adapt sbrt[12]:pert_rate=0.98913 pertalpha=2 chgv_rate=0.797872   m_rate=0.184466
## Adapt sbrt[13]:pert_rate=0.886076 pertalpha=2 chgv_rate=0.842697   m_rate=0.213592
## Adapt sbrt[14]:pert_rate=0.944444 pertalpha=2 chgv_rate=0.894231   m_rate=0.281553
## Adapt sbrt[15]:pert_rate=0.988889 pertalpha=2 chgv_rate=0.867647   m_rate=0.223301
## Adapt sbrt[16]:pert_rate=0.931818 pertalpha=2 chgv_rate=0.760563   m_rate=0.300971
## Adapt sbrt[17]:pert_rate=0.979592 pertalpha=2 chgv_rate=0.833333   m_rate=0.223301
## Adapt sbrt[18]:pert_rate=0.969388 pertalpha=2 chgv_rate=0.804598   m_rate=0.223301
## Adapt sbrt[19]:pert_rate=0.964286 pertalpha=2 chgv_rate=0.853659   m_rate=0.194175
## Adapt sbrt[20]:pert_rate=0.92233 pertalpha=2 chgv_rate=0.817073   m_rate=0.194175
## Adapt sbrt[21]:pert_rate=0.953488 pertalpha=2 chgv_rate=0.767123   m_rate=0.223301
## Adapt sbrt[22]:pert_rate=0.968421 pertalpha=2 chgv_rate=0.849462   m_rate=0.145631
## Adapt sbrt[23]:pert_rate=0.954545 pertalpha=2 chgv_rate=0.837209   m_rate=0.184466
## Adapt sbrt[24]:pert_rate=0.977528 pertalpha=2 chgv_rate=0.886364   m_rate=0.184466
## Adapt sbrt[25]:pert_rate=0.94 pertalpha=2 chgv_rate=0.769231   m_rate=0.23301
## Adapt sbrt[26]:pert_rate=0.978261 pertalpha=2 chgv_rate=0.853933   m_rate=0.291262
## Adapt sbrt[27]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.858974   m_rate=0.174757
## Adapt sbrt[28]:pert_rate=0.955056 pertalpha=2 chgv_rate=0.868421   m_rate=0.135922
## Adapt sbrt[29]:pert_rate=0.941748 pertalpha=2 chgv_rate=0.888889   m_rate=0.184466
## Adapt sbrt[30]:pert_rate=0.927083 pertalpha=2 chgv_rate=0.790698   m_rate=0.194175
## Adapt sbrt[31]:pert_rate=0.931034 pertalpha=2 chgv_rate=0.857143   m_rate=0.194175
## Adapt sbrt[32]:pert_rate=0.977528 pertalpha=2 chgv_rate=0.813953   m_rate=0.116505
## Adapt sbrt[33]:pert_rate=0.97 pertalpha=2 chgv_rate=0.8   m_rate=0.339806
## Adapt sbrt[34]:pert_rate=0.891304 pertalpha=2 chgv_rate=0.8875   m_rate=0.194175
## Adapt sbrt[35]:pert_rate=0.946429 pertalpha=2 chgv_rate=0.854167   m_rate=0.223301
## Adapt sbrt[36]:pert_rate=0.968085 pertalpha=2 chgv_rate=0.820225   m_rate=0.0776699
## Adapt sbrt[37]:pert_rate=0.880435 pertalpha=2 chgv_rate=0.834862   m_rate=0.281553
## Adapt sbrt[38]:pert_rate=0.99 pertalpha=2 chgv_rate=0.83908   m_rate=0.23301
## Adapt sbrt[39]:pert_rate=0.976923 pertalpha=2 chgv_rate=0.833333   m_rate=0.194175
## Adapt ambrt[0]:pert_rate=0.688958 pertalpha=2 chgv_rate=0.406305   m_rate=0.106796
## Adapt sbrt[0]:pert_rate=0.915254 pertalpha=2 chgv_rate=0.877358   m_rate=0.23301
## Adapt sbrt[1]:pert_rate=0.97561 pertalpha=2 chgv_rate=0.752577   m_rate=0.15534
## Adapt sbrt[2]:pert_rate=0.979167 pertalpha=2 chgv_rate=0.863636   m_rate=0.0776699
```

```
## Adapt sbrt[3]:pert_rate=0.909091 pertalpha=2 chgv_rate=0.9   m_rate=0.194175
## Adapt sbrt[4]:pert_rate=0.957627 pertalpha=2 chgv_rate=0.773438   m_rate=0.203883
## Adapt sbrt[5]:pert_rate=0.99 pertalpha=2 chgv_rate=0.879121   m_rate=0.194175
## Adapt sbrt[6]:pert_rate=0.97 pertalpha=2 chgv_rate=0.781609   m_rate=0.281553
## Adapt sbrt[7]:pert_rate=0.944882 pertalpha=2 chgv_rate=0.794872   m_rate=0.23301
## Adapt sbrt[8]:pert_rate=0.987342 pertalpha=2 chgv_rate=0.883495   m_rate=0.213592
## Adapt sbrt[9]:pert_rate=0.930435 pertalpha=2 chgv_rate=0.803279   m_rate=0.194175
## Adapt sbrt[10]:pert_rate=0.981308 pertalpha=2 chgv_rate=0.805825   m_rate=0.320388
## Adapt sbrt[11]:pert_rate=0.967391 pertalpha=2 chgv_rate=0.826923   m_rate=0.116505
## Adapt sbrt[12]:pert_rate=0.990099 pertalpha=2 chgv_rate=0.796117   m_rate=0.15534
## Adapt sbrt[13]:pert_rate=0.98913 pertalpha=2 chgv_rate=0.84375   m_rate=0.165049
## Adapt sbrt[14]:pert_rate=0.882812 pertalpha=2 chgv_rate=0.897436   m_rate=0.291262
## Adapt sbrt[15]:pert_rate=0.978022 pertalpha=2 chgv_rate=0.860759   m_rate=0.223301
## Adapt sbrt[16]:pert_rate=0.882979 pertalpha=2 chgv_rate=0.759036   m_rate=0.223301
## Adapt sbrt[17]:pert_rate=0.988764 pertalpha=2 chgv_rate=0.842697   m_rate=0.0873786
## Adapt sbrt[18]:pert_rate=0.989899 pertalpha=2 chgv_rate=0.804348   m_rate=0.174757
## Adapt sbrt[19]:pert_rate=0.965517 pertalpha=2 chgv_rate=0.87234   m_rate=0.145631
## Adapt sbrt[20]:pert_rate=0.941748 pertalpha=2 chgv_rate=0.815217   m_rate=0.223301
## Adapt sbrt[21]:pert_rate=0.936842 pertalpha=2 chgv_rate=0.782051   m_rate=0.252427
## Adapt sbrt[22]:pert_rate=0.948454 pertalpha=2 chgv_rate=0.865385   m_rate=0.242718
## Adapt sbrt[23]:pert_rate=0.896907 pertalpha=2 chgv_rate=0.842105   m_rate=0.165049
## Adapt sbrt[24]:pert_rate=0.972222 pertalpha=2 chgv_rate=0.881188   m_rate=0.271845
## Adapt sbrt[25]:pert_rate=0.98913 pertalpha=2 chgv_rate=0.792079   m_rate=0.145631
## Adapt sbrt[26]:pert_rate=0.975904 pertalpha=2 chgv_rate=0.85   m_rate=0.15534
## Adapt sbrt[27]:pert_rate=0.934066 pertalpha=2 chgv_rate=0.853933   m_rate=0.184466
## Adapt sbrt[28]:pert_rate=0.968421 pertalpha=2 chgv_rate=0.875   m_rate=0.126214
## Adapt sbrt[29]:pert_rate=0.968085 pertalpha=2 chgv_rate=0.890909   m_rate=0.23301
## Adapt sbrt[30]:pert_rate=0.934066 pertalpha=2 chgv_rate=0.8   m_rate=0.194175
## Adapt sbrt[31]:pert_rate=0.967391 pertalpha=2 chgv_rate=0.837209   m_rate=0.135922
## Adapt sbrt[32]:pert_rate=0.955056 pertalpha=2 chgv_rate=0.836735   m_rate=0.23301
## Adapt sbrt[33]:pert_rate=0.932692 pertalpha=2 chgv_rate=0.794118   m_rate=0.271845
## Adapt sbrt[34]:pert_rate=0.959596 pertalpha=2 chgv_rate=0.877778   m_rate=0.262136
## Adapt sbrt[35]:pert_rate=0.968421 pertalpha=2 chgv_rate=0.853211   m_rate=0.194175
## Adapt sbrt[36]:pert_rate=0.990909 pertalpha=2 chgv_rate=0.821782   m_rate=0.203883
## Adapt sbrt[37]:pert_rate=0.987805 pertalpha=2 chgv_rate=0.830508   m_rate=0.203883
## Adapt sbrt[38]:pert_rate=0.978723 pertalpha=2 chgv_rate=0.85567   m_rate=0.15534
## Adapt sbrt[39]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.843137   m_rate=0.135922
## Adapt ambrt[0]:pert_rate=0.702362 pertalpha=2 chgv_rate=0.400621   m_rate=0.0970874
## Adapt sbrt[0]:pert_rate=0.981651 pertalpha=2 chgv_rate=0.873874   m_rate=0.23301
## Adapt sbrt[1]:pert_rate=0.946809 pertalpha=2 chgv_rate=0.759259   m_rate=0.23301
## Adapt sbrt[2]:pert_rate=0.953271 pertalpha=2 chgv_rate=0.88   m_rate=0.23301
## Adapt sbrt[3]:pert_rate=0.972973 pertalpha=2 chgv_rate=0.883495   m_rate=0.23301
## Adapt sbrt[4]:pert_rate=0.954198 pertalpha=2 chgv_rate=0.758865   m_rate=0.203883
## Adapt sbrt[5]:pert_rate=0.939394 pertalpha=2 chgv_rate=0.861386   m_rate=0.184466
## Adapt sbrt[6]:pert_rate=0.973684 pertalpha=2 chgv_rate=0.809524   m_rate=0.145631
## Adapt sbrt[7]:pert_rate=0.971429 pertalpha=2 chgv_rate=0.784   m_rate=0.203883
## Adapt sbrt[8]:pert_rate=0.935484 pertalpha=2 chgv_rate=0.893805   m_rate=0.223301
## Adapt sbrt[9]:pert_rate=0.989362 pertalpha=2 chgv_rate=0.810606   m_rate=0.126214
## Adapt sbrt[10]:pert_rate=0.939759 pertalpha=2 chgv_rate=0.82906   m_rate=0.242718
## Adapt sbrt[11]:pert_rate=0.990909 pertalpha=2 chgv_rate=0.824561   m_rate=0.15534
## Adapt sbrt[12]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.818966   m_rate=0.0970874
## Adapt sbrt[13]:pert_rate=0.965909 pertalpha=2 chgv_rate=0.828829   m_rate=0.15534
## Adapt sbrt[14]:pert_rate=0.915663 pertalpha=2 chgv_rate=0.897638   m_rate=0.116505
## Adapt sbrt[15]:pert_rate=0.912088 pertalpha=2 chgv_rate=0.869048   m_rate=0.23301
```

```
## Adapt sbrt[16]:pert_rate=0.966667 pertalpha=2 chgv_rate=0.769231   m_rate=0.213592
## Adapt sbrt[17]:pert_rate=0.965517 pertalpha=2 chgv_rate=0.84375   m_rate=0.184466
## Adapt sbrt[18]:pert_rate=0.925234 pertalpha=2 chgv_rate=0.808081   m_rate=0.126214
## Adapt sbrt[19]:pert_rate=0.947917 pertalpha=2 chgv_rate=0.886792   m_rate=0.194175
## Adapt sbrt[20]:pert_rate=0.941748 pertalpha=2 chgv_rate=0.810811   m_rate=0.194175
## Adapt sbrt[21]:pert_rate=0.981132 pertalpha=2 chgv_rate=0.767442   m_rate=0.135922
## Adapt sbrt[22]:pert_rate=0.876289 pertalpha=2 chgv_rate=0.867257   m_rate=0.291262
## Adapt sbrt[23]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.843137   m_rate=0.23301
## Adapt sbrt[24]:pert_rate=0.962406 pertalpha=2 chgv_rate=0.866071   m_rate=0.213592
## Adapt sbrt[25]:pert_rate=0.988506 pertalpha=2 chgv_rate=0.8125   m_rate=0.213592
## Adapt sbrt[26]:pert_rate=0.886792 pertalpha=2 chgv_rate=0.857143   m_rate=0.262136
## Adapt sbrt[27]:pert_rate=0.929412 pertalpha=2 chgv_rate=0.835052   m_rate=0.262136
## Adapt sbrt[28]:pert_rate=0.988889 pertalpha=2 chgv_rate=0.880435   m_rate=0.145631
## Adapt sbrt[29]:pert_rate=0.988235 pertalpha=2 chgv_rate=0.882353   m_rate=0.174757
## Adapt sbrt[30]:pert_rate=0.948276 pertalpha=2 chgv_rate=0.81982   m_rate=0.213592
## Adapt sbrt[31]:pert_rate=0.96875 pertalpha=2 chgv_rate=0.845361   m_rate=0.203883
## Adapt sbrt[32]:pert_rate=0.939394 pertalpha=2 chgv_rate=0.831776   m_rate=0.213592
## Adapt sbrt[33]:pert_rate=0.961538 pertalpha=2 chgv_rate=0.8   m_rate=0.174757
## Adapt sbrt[34]:pert_rate=0.979798 pertalpha=2 chgv_rate=0.886598   m_rate=0.223301
## Adapt sbrt[35]:pert_rate=0.939759 pertalpha=2 chgv_rate=0.85124   m_rate=0.262136
## Adapt sbrt[36]:pert_rate=0.960396 pertalpha=2 chgv_rate=0.816514   m_rate=0.15534
## Adapt sbrt[37]:pert_rate=0.989691 pertalpha=2 chgv_rate=0.840909   m_rate=0.165049
## Adapt sbrt[38]:pert_rate=0.990566 pertalpha=2 chgv_rate=0.839286   m_rate=0.213592
## Adapt sbrt[39]:pert_rate=0.943089 pertalpha=2 chgv_rate=0.866142   m_rate=0.31068
## draw  burn 0
## draw  burn 100
## draw  keep 0
## draw  keep 100
## draw  keep 200
## draw  keep 300
## draw  keep 400
## draw  keep 500
## draw  keep 600
## draw  keep 700
## draw  keep 800
## draw  keep 900
## draw  keep 1000
## draw  keep 1100
## draw  keep 1200
## draw  keep 1300
## draw  keep 1400
## draw  keep 1500
## draw  keep 1600
## draw  keep 1700
## draw  keep 1800
## draw  keep 1900
```

```
bart_predict <- predict(bart_res, sim_x)$mmean
plot(x = bart_predict, y = true_mean)
```




```
res_bart <- (bart_predict-sim_y)^2
```

```
res_all <- matrix(c(sum(res_bart), sum(res_cart^2), sum(res_rpart^2)), ncol = 3)
colnames(res_all) <- c("bart","cart","rpart")
res_all
```

```
##          bart     cart    rpart
## [1,] 26749.51 41.78285 47.87081
```