# baesRCM Simulation Build

Quinton Neville

9/15/2022

## Contents

## 1. Generate Some Simple Data

Here we generate 10 volumes of multivariate normal data for 10 subjects in a network of 4 rois, with 2 true connections or edges in the associated group graph.

```r
#Generate covariance structure for multivariate gaussian covariance matrices
volumes  <- 10
subjects <- 10
rois     <- 10 #keep it small to start testing
true_con <- 20 #true connections or no. of edges in G
#Group overall graph with true_con # of edges
set.seed(4)
G <-
  matrix(
    rbinom(volumes * subjects, 1, prob = true_con / (volumes * subjects)),
    nrow = volumes
  )
diag(G) <- 1
G <- Matrix::forceSymmetric(G, uplo = "U")
#G

#Overall Precision Matrix
set.seed(4)
Sigma_0 <-
  (G * matrix(rnorm((volumes * subjects), 0, 10), nrow = volumes)) |>
  Matrix::forceSymmetric(uplo = "U") |>
  (\(x) {as.matrix(Matrix::nearPD(x)$mat)})()
#Sigma_0

#Subject precision matrices based off of group
Sigma_k <- list()

#Fill matrices in list
for (n in 1:subjects) {
  set.seed(n)
  Sigma_k[[n]] <- (Sigma_0 + matrix(rnorm((volumes * subjects), 0.25/n, 1), nrow = volumes)) |>
  Matrix::forceSymmetric(uplo = "U") |>
  (\(x) {as.matrix(Matrix::nearPD(x)$mat)})()
}

#List of each subject's array in time/by volume
#randomly generated around mean 0 with subject specific precision
#i.e. stead state
data_list <- list()

#Loop through subjects and volumes to generate data
for (n in 1:subjects) { #assumes no temporal mean trend, centered at 0
    set.seed(n)
    data_list[[n]] <- mvtnorm::rmvnorm(volumes, rep(0, rois), Sigma_k[[n]])
}

#Save 'truth' to evaluate recovery
Omega_0 <- solve(Sigma_0)
Omega_k <- map(Sigma_k, solve)
```

## 2. Apply `bayesRCM` package functions to generate posterior samples of the model

In development, one can source the helper functions from the `./R/` directory and the `.cpp` file from `/src/`. However, at this moment the package is funcitonal and so one can download via github with `devtools::install_github("nevilleq/bayesRCM")` and then load the library like normal `library(bayesRCM)`.

### 2.1 Posterior Samples (Prelim Test)

Next, let's generate some preliminary posterior samples for 10 iterations and look at the results. There are still bugs we need to work out, especially in the graph update `log_H` and being able to do a cholesky decomp (i.e. encforcing symmetry and positive definite-ness).

```
result <- rcm(y = data_list, n_samples = 10)
#write_rds(result, "./results/prelim_test.RDS")
write_rds(result, "../results/prelim_test.RDS")
```

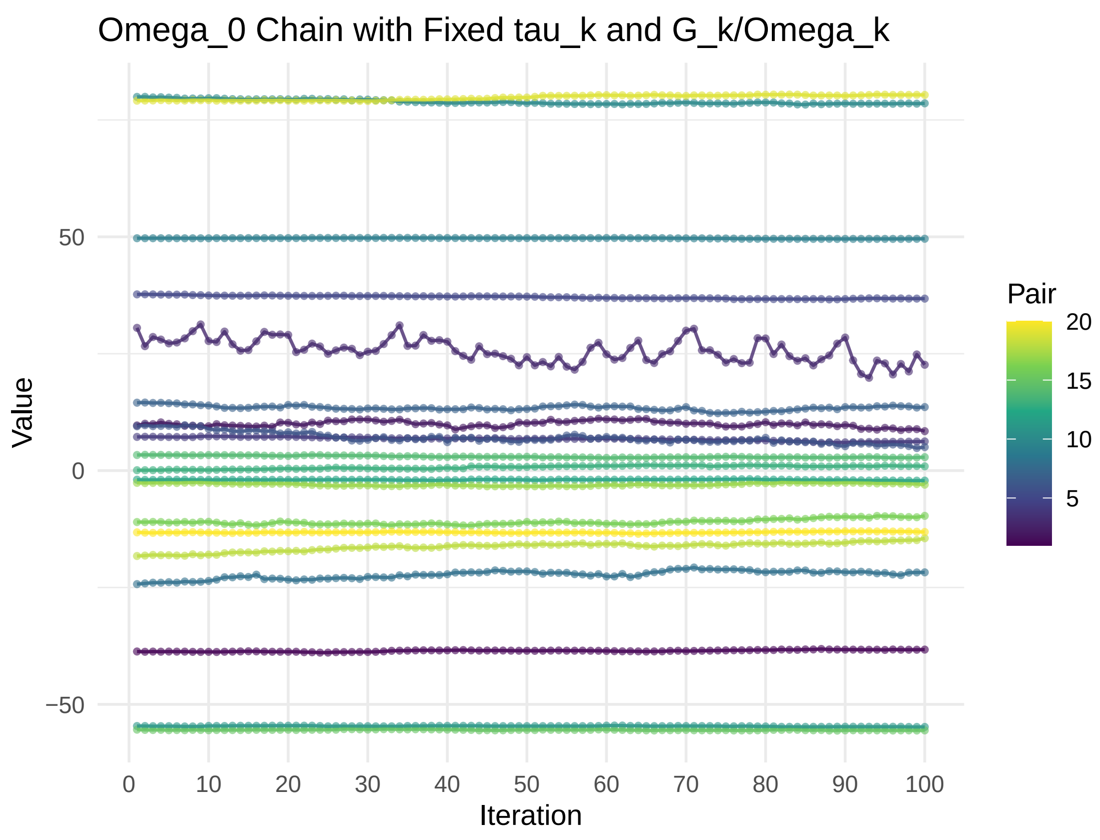
### 2.2 Testing Individual Updates by Parameter

Here, we fix all MCMC, graph updates, and/or non-direct sampling components of the algorithm, then observe the behaviour of the resulting Markov Chain(s). To do so, we are going to pull out the `rcm` source code, fix the desired elements at the "truth" (see above), and then sample the parameter(s) of interest. This should help troubleshoot and debug, especially for the $G_k/\Omega_k$ update.

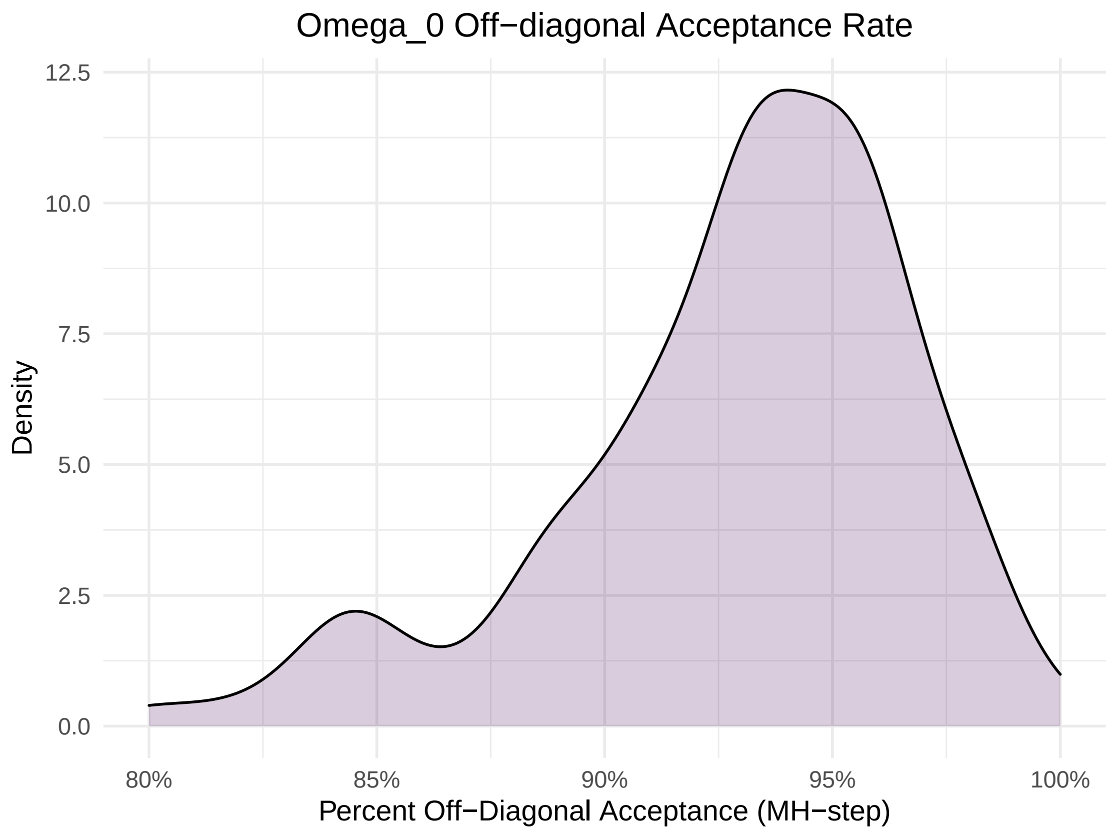
#### 2.2.1 $\Omega_0$ with fixed $\tau_k$

```
#Set params
y <- data_list
n_samples <- 100
n_burn <- 10
```

```r
#Grab no. of subjects, rois, volumes
K  <- length(y)
p  <- ncol(y[[1]])
vk <- map_dbl(y, nrow)
Sk <- map(.x = y, ~t(.x) %*% .x)

#Set lambda 1-3 penalty gamma a, b  hyperparams
alpha <- c(0.5, 1, 0.5) #1 - G_k, 2 - tau_k, 3 - Omega_0 (glasso)
beta  <- c(0.5, 1, 0.5)

#Set tau's MH stepsize
step_tau  <- rep(1, K)
n_updates <- 10

#Initialize estimates for Omega_k, Omega_0
#Omega_k - tune lambda by bic and then grab G and Omega_0
lambda_grid <- 10^seq(-3, 0, length.out = 10)
bic         <- vector(mode = "numeric", length = 0)

#Tune lambda for independent glasso
for (lam in lambda_grid) {
  omega_k <- ind_graphs(y, 0.1)
  bic   <- c(bic, bic_cal(y, omega_k))
}
#Compute initial est. via best bic
omega_k <- ind_graphs(y, lambda_grid[which.min(bic)])

#Loop through just to make sure PD
for (k in 1:K) {
  if (any(eigen(omega_k[[k]])$values < 0)) {
    #print(k)
    omega_k[[k]] <-
      omega_k[[k]] |>
      (\(x) {as.matrix(Matrix::nearPD(x)$mat)})()
  }
}

adj_k   <- map(.x = omega_k, ~abs(.x) > 0.001)
#Omega0  <- apply(abind::abind(omega_k, along=3),1:2,mean)
omega_0 <- Reduce("+", omega_k) / K

#Initialize estimates for tau_k
#Tau vector of subject specific regularization param on Omega_0
tau_vec <- vector(mode = "numeric", length = K)

#Iterate over each subject, find optimal tau_k based on posterior
for (k in 1:K) {
  #print(k)
  #Tau posterior for fixed omega_k, omega_0, and lambda_2 = 0
  f_opt <- function(tau) {
    -1 * log_tau_posterior(tau, omega_k[[k]], omega_0, lambda_2 = 0)
  }
  #Optimize in 1D
  tau_vec[k] <- c(optimize(f_opt, interval = c(0, 1000), tol = 10)$min)
}

#Set up storage for results
#Omegas
omegas_res <- array(NA, c(p * (p + 1) / 2, K, n_samples))
omega0_res <- array(NA, c(p * (p + 1) / 2, n_samples))
pct_omega_acc <- vector(mode = "integer", length = n_samples)
pct_k_acc  <- matrix(NA, nrow = n_samples, ncol = K)

#Taus
accept_mat    <- matrix(NA, nrow = 0, ncol = K)
step_tau_mat  <- step_tau #Adaptive window for MH tau
tau_res       <- array(NA, c(K, n_samples))

#Lambdas
lambda_res <- array(NA, c(3, n_samples), dimnames = list(str_c("lambda_", 1:3)))

#Set timer
timer   <- 0
t_start <- proc.time()
n_iter  <- (n_burn + n_samples)
#n_iter  <- 3

#Loop through sampling algorithm n_samples + n_burn # times
for (t in 1:n_iter) {
  #Print iteration for early testing
  print(paste0("Iteration: ", t))

  #Update Lambdas via direct sampling
  #Lambda 1 sparsity-inducing penalty on G_k
  card_k   <- (sapply(adj_k, sum) - p)/2 #Cardinality of G_k / # edges
  lambda_1 <- rgamma(1, alpha[1] + K, rate = beta[1] + sum(card_k))

  #Lambda 2 Exponential rate parameter for df/shrinkage tau_k prior
  lambda_2 <- rgamma(1, alpha[2] + K + 1, beta[2] + sum(tau_vec))

  #Lambda 3 Sparse L-1 penalty on group precision omega_0 prior
  card_0 <- (sum(abs(omega_0) > 0.001) + p) / 2 #Cardinality omega_0 / # Edges
  lambda_3 <- rgamma(1, alpha[3] + card_0, beta[3] + norm(omega_0, type = "1"))

  #Invert for Covariance & randomly select row_col pair
  sigma_0 <- matinv(omega_0)
  row_col <- sample(1:p, 1)

  #Set up foreach:: combine into 2 list, multicombine = TRUE
  # my_combine <- function(x, ...) {
  #   lapply(seq_along(x),
  #          function(i) c(x[[i]], lapply(list(...), function(y) y[[i]])))
  # }

  #Update G_k, Omega_k via modified BIPS proposal & update scheme - Wang and Li (2012)
  omega_k <- omega_k
  #Tau_k update
  tau_vec <- tau_vec

  #Update Omega_0 via Wang and Li (2012) + step-proposal distribution
  D       <- apply(mapply('/', omega_k, tau_vec, SIMPLIFY = 'array'), 1:2, sum)
  omega_0 <- omega0_update(omega_0, D, sum(tau_vec), lambda_3)
  pct_accept <- omega_0$pct_accept #Off-diagonal acceptance%
  omega_0 <- omega_0$omega #Precision matrix itself

  #Save those results after burn-in
  if(t > n_burn) {
    t_burn <- t - n_burn
    #omegas_res[, , t_burn] <- sapply(omega_k, function(x) x[upper.tri(x, diag = TRUE)])
    omega0_res[, t_burn]   <- omega_0[upper.tri(omega_0, diag = TRUE)]
    #tau_res[, t_burn]      <- tau_k
    #lambda_res[, t_burn]   <- c(lambda_1, lambda_2, lambda_3)
    pct_omega_acc[t_burn]  <- pct_accept
    #pct_k_acc[t_burn, ]    <- accept_k
  }

  #Track temporal progress (every 20% progress update)
  if (t %% floor(0.2 * (n_samples + n_burn))) {
    t_now   <- proc.time()
    timer   <- c(timer, (t_now - t_start)[3])
    t_start <- t_now
  }
}

#Result list of results
omega0_result <-
  list(
    omega_0   = omega0_res,
    omega_acc = pct_omega_acc,
    timer     = timer
  )

#Write out for safekeeping
write_rds(omega0_result, "../results/prelim_omega0.RDS")
```



Omega_0 Chain with Fixed tau_k and G_k/Omega_k
Value
50
0
−50
0
10
20
30
40
50
60
70
80
90
100
Iteration
Pair
20
15
10
5



Omega_0 Off−diagonal Acceptance Rate
Density
12.5
10.0
7.5
5.0
2.5
0.0
80%
85%
90%
95%
100%
Percent Off−Diagonal Acceptance (MH−step)

#### 2.2.2 $\Omega_k$ & $G_k$

#### 2.2.3 $\tau_k$

#### 2.2.4 Sensitivity to Regularization Hyperparameters $(\lambda_j \,|\, a_j, b_j \sim \Gamma(a_j, b_j))$

#### 2.2.1 Posterior Distribution

#### 2.2.2. Accept/Reject Rates for Proposals