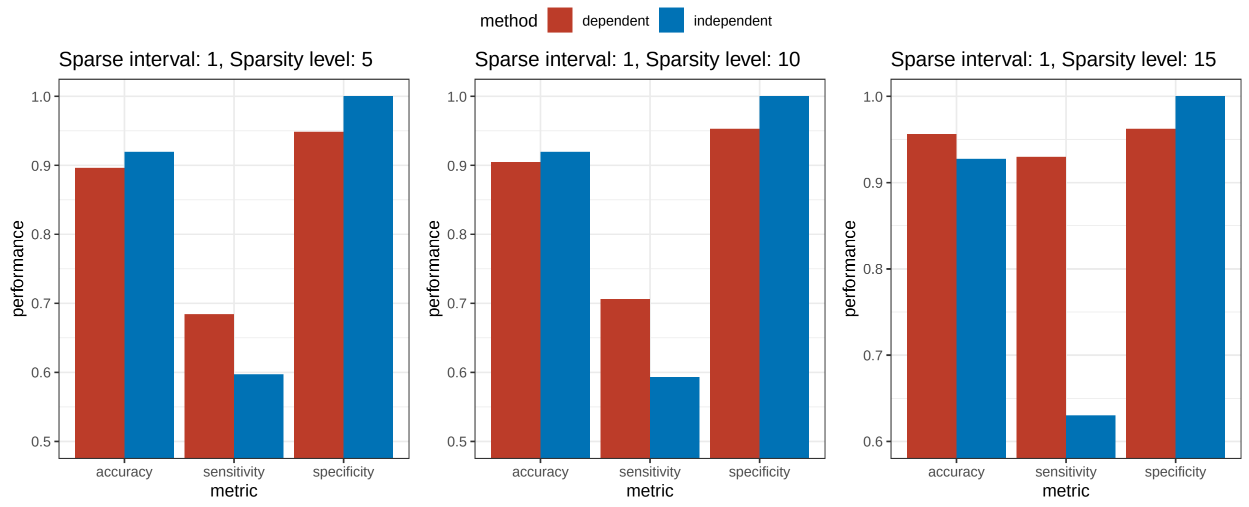
covdepGE versus competitor

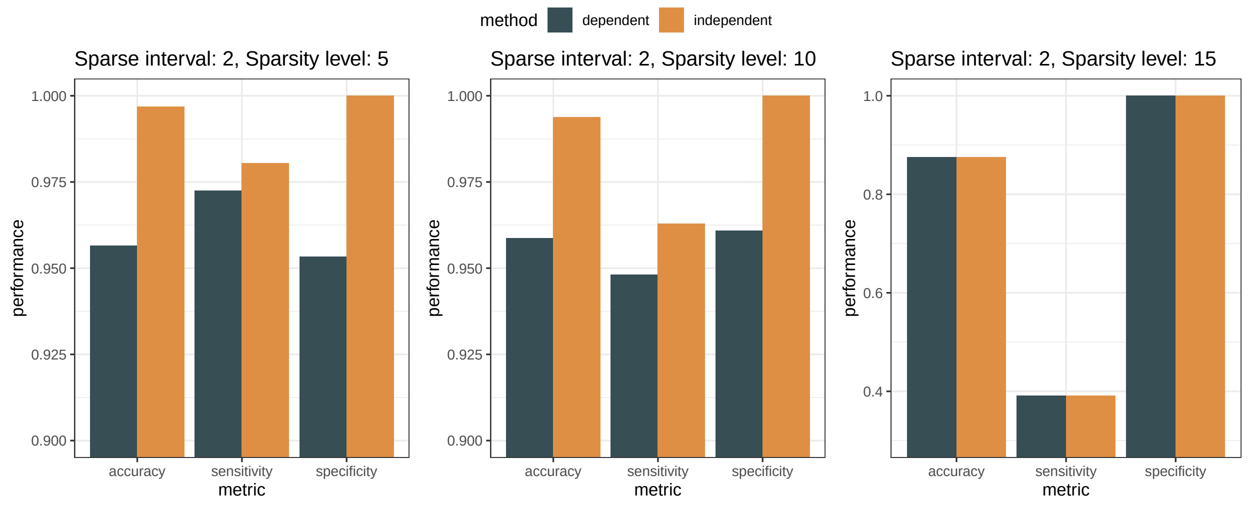
Summary

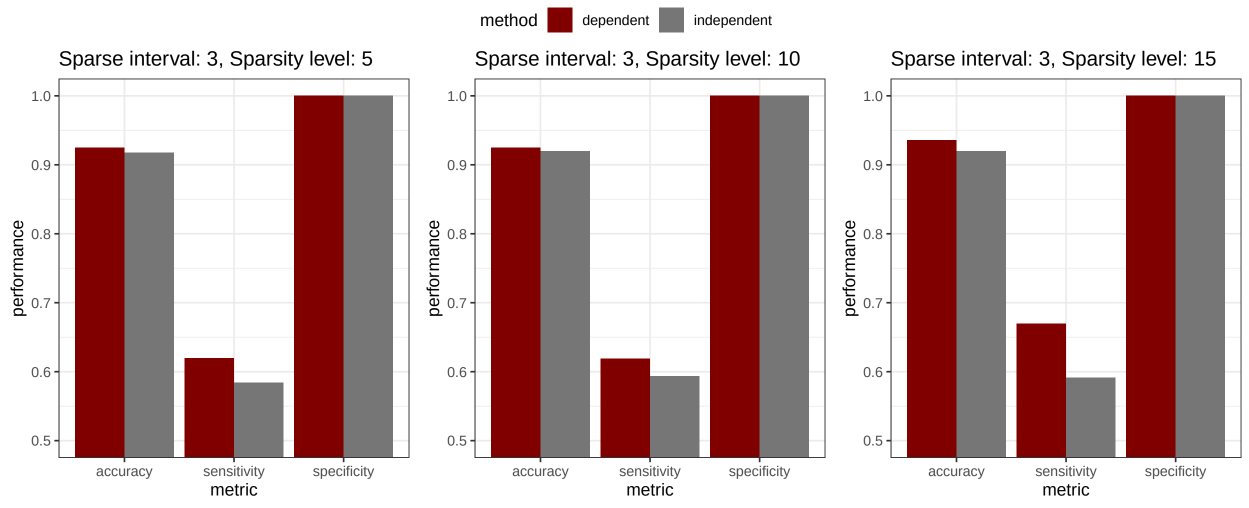
I repeated this experiment 9 times. In each experiment, I chose one of the three covariate intervals to be sparse, and the other two to be non-sparse. The non-sparse intervals had 60 individuals in them, while the sparsity level of the sparse interval varied from 5, 10, or 15 individuals. In each experiment, I recorded the following metrics for both the covariate independent and covariate dependent graphical estimation methods:

- Sensitivity: the percentage of edges recovered by the method
- Specificity: the percentage of non-edges recovered by the method
- Accuracy

```r
library(ggplot2)
library(ggpubr)
library(kableExtra)

# create a dataframe for aggregating results of experiments
sparsity_levels <- 5 * 1:3
sparse_intervals <- 1:3
experiments <- expand.grid(
  method = c("dependent", "independent"),
  sparsity_level = sparsity_levels,
  sparse_interval = sparse_intervals
)
metrics <- c("sensitivity", "specificity", "accuracy")
perf_mat <- matrix(NA, nrow(experiments), length(metrics), dimnames = list(NULL, metrics))
results <- cbind.data.frame(experiments, perf_mat)

# create a list for visualizations
viz <- vector("list")

# list of color schemes
color_schemes <- list(
  ggsci::scale_fill_nejm(), ggsci::scale_fill_jama(),
  ggsci::scale_fill_uchicago()
)

# iterate over the sparsity levels and intervals to add the corresponding data
# to results and create a visualization
for (sparse_interval in sparse_intervals) {
  for (sparsity_level in sparsity_levels) {

    # load the data corresponding to the interval and sparsity level
    load(paste0(
      "performance_data_int", sparse_interval, "_sparse",
      sparsity_level, ".Rda"
    ))

    # reshape the data and clean column names
    res_df_wide <- reshape(perf_res_df,
      idvar = "method", timevar = "metric",
      direction = "wide"
    )
    colnames(res_df_wide)[-1] <- sapply(sapply(
      colnames(res_df_wide)[-1],
      strsplit, "[.]"
    ), `[[`, 2)

    # find the rows of result corresponding to the interval and sparsity level
    res_rows <- sapply(1:nrow(results), function(row_ind) all(
      results[row_ind, c("sparsity_level", "sparse_interval")] ==
        t(c(sparsity_level, sparse_interval))))

    # add the data for the dependent and independent methods
    results[res_rows & results$method == "dependent", metrics] <- round(
      res_df_wide[res_df_wide$method == "dependent", metrics], 3
    )
    results[res_rows & results$method == "independent", metrics] <- round(
      res_df_wide[res_df_wide$method == "independent", metrics], 3
    )

    # create a visualization
    y_lower <- floor(min(perf_res_df$performance) * 10) / 10
    viz[[length(viz) + 1]] <- ggplot(perf_res_df, aes(metric, performance, fill = method)) +
      geom_bar(stat = "identity", position = position_dodge()) +
      coord_cartesian(ylim = c(y_lower, 1)) +
      theme_bw() +
      color_schemes[[sparse_interval]] +
      ggtitle(paste0("Sparse interval: ", sparse_interval, ", Sparsity level: ", sparsity_level))
  }
}

ggarrange(ggarrange(plotlist = viz[1:3], ncol = 3, nrow = 1, common.legend = T),
  ggarrange(plotlist = viz[4:6], ncol = 3, nrow = 1, common.legend = T),
  ggarrange(plotlist = viz[7:9], ncol = 3, nrow = 1, common.legend = T),
  ncol = 1, nrow = 3
)
```







```r
kbl(results, booktabs = T, linesep = "") %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

method	sparsity_level	sparse_interval	sensitivity	specificity	accuracy
dependent	5	1	0.684	0.949	0.896
independent	5	1	0.597	1.000	0.920
dependent	10	1	0.706	0.953	0.905
independent	10	1	0.594	1.000	0.920
dependent	15	1	0.930	0.962	0.956
independent	15	1	0.630	1.000	0.928
dependent	5	2	0.973	0.953	0.956
independent	5	2	0.980	1.000	0.997
dependent	10	2	0.948	0.961	0.959
independent	10	2	0.963	1.000	0.994
dependent	15	2	0.391	1.000	0.876
independent	15	2	0.391	1.000	0.876
dependent	5	3	0.619	1.000	0.924
independent	5	3	0.584	1.000	0.917
dependent	10	3	0.619	1.000	0.925
independent	10	3	0.594	1.000	0.920
dependent	15	3	0.670	1.000	0.935
independent	15	3	0.591	1.000	0.920

```r
# clear the environment
rm(list = ls())
```

Data generation

The extraneous covariate is created as the union of three disjoint intervals with nearly adjacent endpoints. Within each interval, the individuals' covariate values are equally spaced.

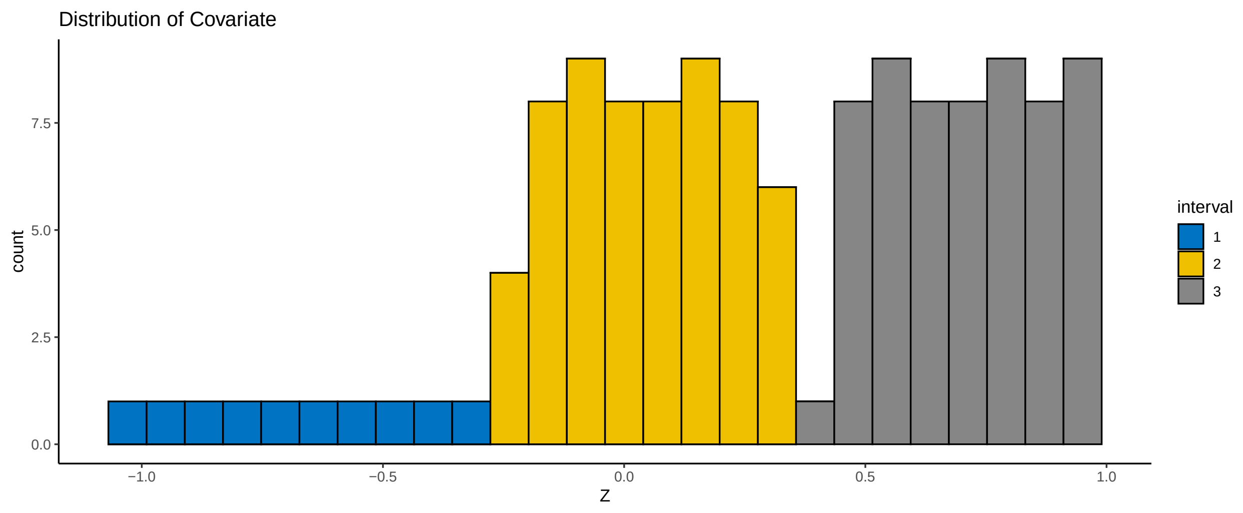
In this experiment, the second interval is the sparse part of the covariate space, with sparsity level 15.

```r
library(covdepGE)
library(mclust)
```

```
## Package 'mclust' version 5.4.9
## Type 'citation("mclust")' for citing this R package in publications.
```

```r
set.seed(1)

# create covariate for individuals in each of the three intervals

# define the dimensions of the data
sparse_interval <- 1
sparsity_level <- 10
interval_n <- list(n1 = 60, n2 = 60, n3 = 60)
interval_n[[sparse_interval]] <- sparsity_level
```

```r
attach(interval_n)
n <- sum(n1, n2, n3)
p <- 4

# define the limits of the intervals
limits1 <- c(-.990, -.331)
limits2 <- c(-.229, 0.329)
limits3 <- c(0.431, 0.990)

# define the covariate values within each interval
z1 <- seq(limits1[1], limits1[2], length = n1)
z2 <- seq(limits2[1], limits2[2], length = n2)
z3 <- seq(limits3[1], limits3[2], length = n3)
Z <- matrix(c(z1, z2, z3), n, 1)

# visualize the covariate
cov_df <- cbind.data.frame(rbind(cbind(1, z = z1), cbind(2, z = z2), cbind(3, z = z3)), 1:n)
names(cov_df) <- c("interval", "Z", "individual_index")
cov_df$interval <- factor(cov_df$interval)
ggplot(cov_df, aes(Z, fill = interval)) +
  geom_histogram(color = "black", bins = n %/% 5) +
  theme_classic() +
  ggsci::scale_fill_jco() +
  labs(title = "Distribution of Covariate")
```



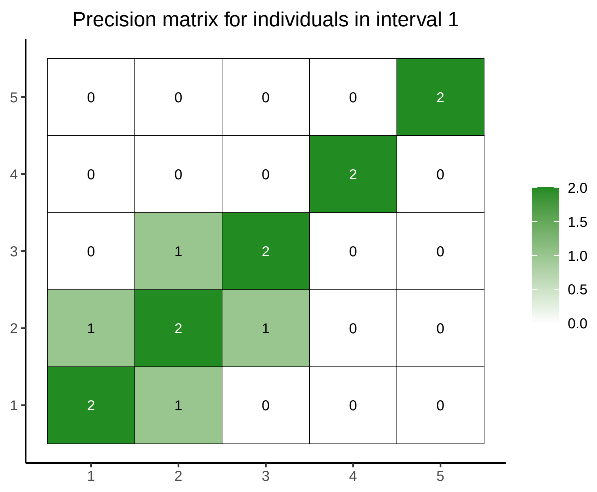
```r
# individual index by interval
ind_idx <- tapply(cov_df$individual_index, cov_df$interval,
                  function(indices) paste0(min(indices), ",...,", max(indices)))
data.frame(interval = names(ind_idx), individual_indices = ind_idx)
```

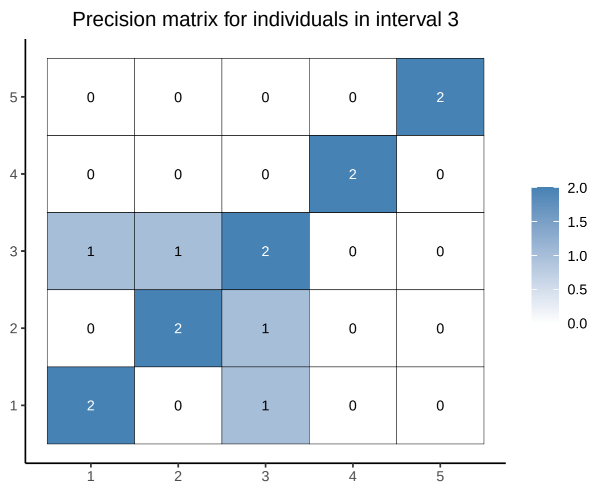
```
##   interval individual_indices
## 1        1           1,...,10
## 2        2          11,...,70
## 3        3         71,...,130
```

All of the individuals in interval 1 have the same precision matrix, as do all of the individuals in interval 3. The first individual in interval 2 has the same precision matrix as those in interval 1.

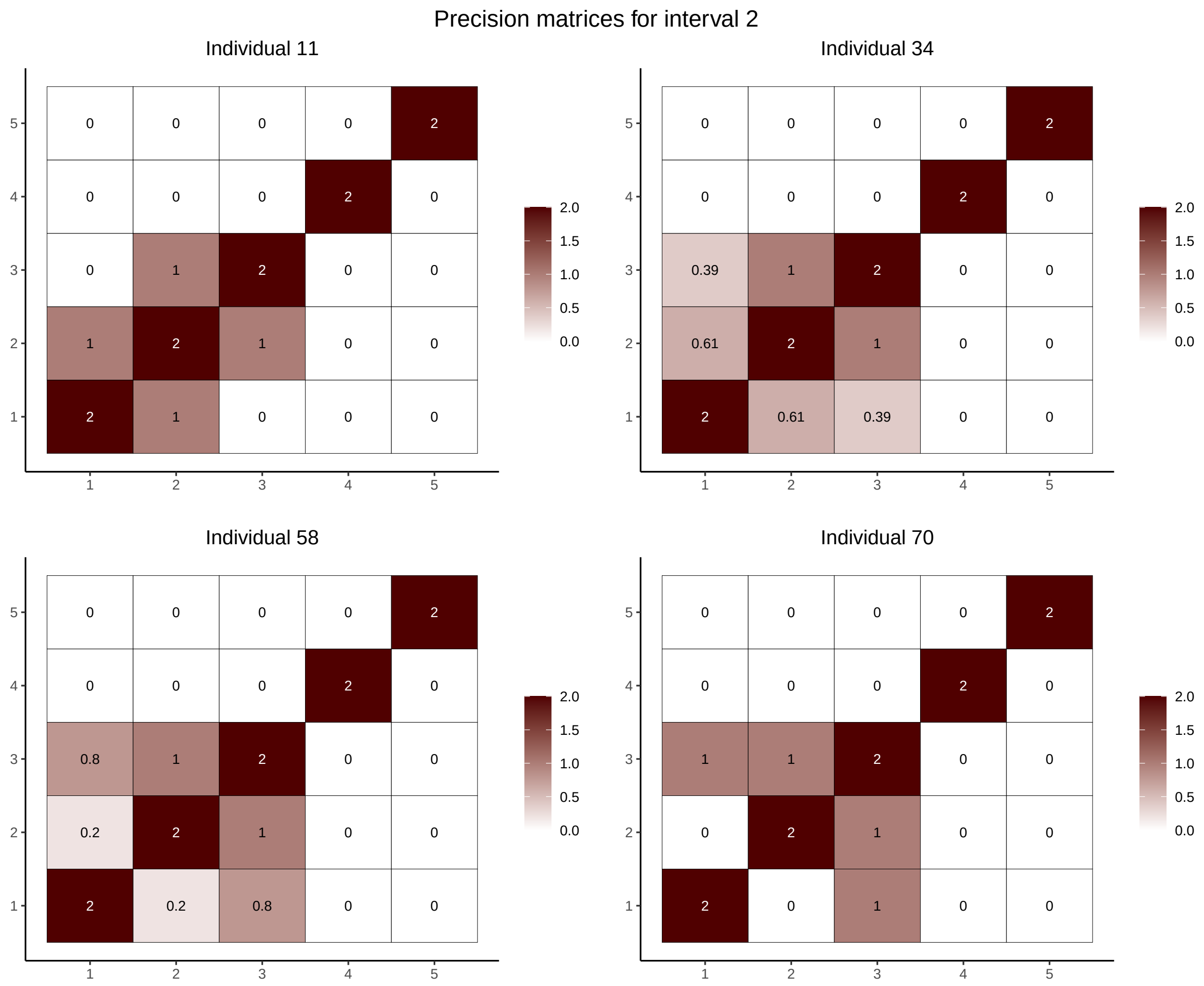
As the individual index in interval 2 increases, the precision matrix continuously shifts from the precision matrix in interval 1 to the precision matrix in interval 3 such that the last individual in interval 2 has the same precision matrix as the individuals in interval 3.

```r
# create precision matrices

# the shared part of the structure for all three intervals is a 2 on the diagonal
# and a 1 in the (2, 3) position
common_str <- diag(p + 1)
common_str[2, 3] <- 1

# define constants for the structure of interval 2
const1 <- 0.23
const2 <- 0.56

# interval 2 has two different linear functions of Z in the (1, 2) position and
# (1, 3) positions; define structures for each of these components
int2_str12 <- int2_str13 <- matrix(0, p + 1, p + 1)
int2_str12[1, 2] <- int2_str13[1, 3] <- 1

# define the precision matrices for each of the individuals in interval 2
int2_prec <- lapply(z2, function(z) common_str +
                      ((1 - (z + const1) / const2) * int2_str12) +
                      ((z + const1) / const2 * int2_str13))

# interval 1 has a 1 in the (1, 2) and interval 3 has a 1 in the (1, 3) position;
# define structures for each of these components
int1_str12 <- int3_str13 <- matrix(0, p + 1, p + 1)
int1_str12[1, 2] <- int3_str13[1, 3] <- 1

# define the precision matrices for each of the individuals in interval 1 and interval 3
int1_prec <- rep(list(common_str + int1_str12), n1)
int3_prec <- rep(list(common_str + int3_str13), n3)

# put all of the precision matrices into one list
prec_mats <- c(int1_prec, int2_prec, int3_prec)

# symmetrize the precision matrices
prec_mats <- lapply(prec_mats, function(mat) t(mat) + mat)

# visualize the precision matrices for each interval

# all of the individuals in interval 1 and 3 have the same precision matrix
int1_g <- gg_adjMat(prec_mats[[1]], color1 = "forestgreen") +
  ggtitle("Precision matrix for individuals in interval 1")
int3_g <- gg_adjMat(prec_mats[[n1 + n2 + 1]], color1 = "steelblue") +
  ggtitle("Precision matrix for individuals in interval 3")

ggarrange(int1_g, int3_g)
```





```r
# the first individual in interval 2 has the same precision matrix as the
# individuals in interval 1; as the subject index grows, the (1, 3)/ (3, 1)
# entry changes from 0 to 1 and the (1, 2)/ (2, 1) entry changes from 1 to 0.
# The last individual has the same precision matrix as the individuals in interval 3
int2_g_inds <- n1 + c(1, 2 * n2 %/% 5, 4 * n2 %/% 5, n2)
int2_graphs <- lapply(int2_g_inds, function(indv_idx) gg_adjMat(prec_mats[[indv_idx]]) +
                        labs(title = paste0("Individual ", indv_idx)))
annotate_figure(ggarrange(plotlist = int2_graphs),
                top = text_grob("Precision matrices for interval 2", size = 15))
```



```r
# invert the precision matrices to get the covariance matrices
cov_mats <- lapply(prec_mats, solve)

# generate the data using the covariance matrices
data_mat <- t(sapply(cov_mats, MASS::mvrnorm, n = 1, mu = rep(0, p + 1)))
```

Covariate dependent graph estimation

```r
# use varbvs to get the hyperparameter sigma
sigmasq <- sapply(1:(p + 1), function(col_ind) mean(varbvs::varbvs(
  data_mat[, -col_ind], Z, data_mat[, col_ind], verbose = F)$sigma))
sigmasq
```

```
## [1] 0.5806405 0.5500376 0.5278778 0.3899514 0.4302449
```

```r
mean(sigmasq)
```

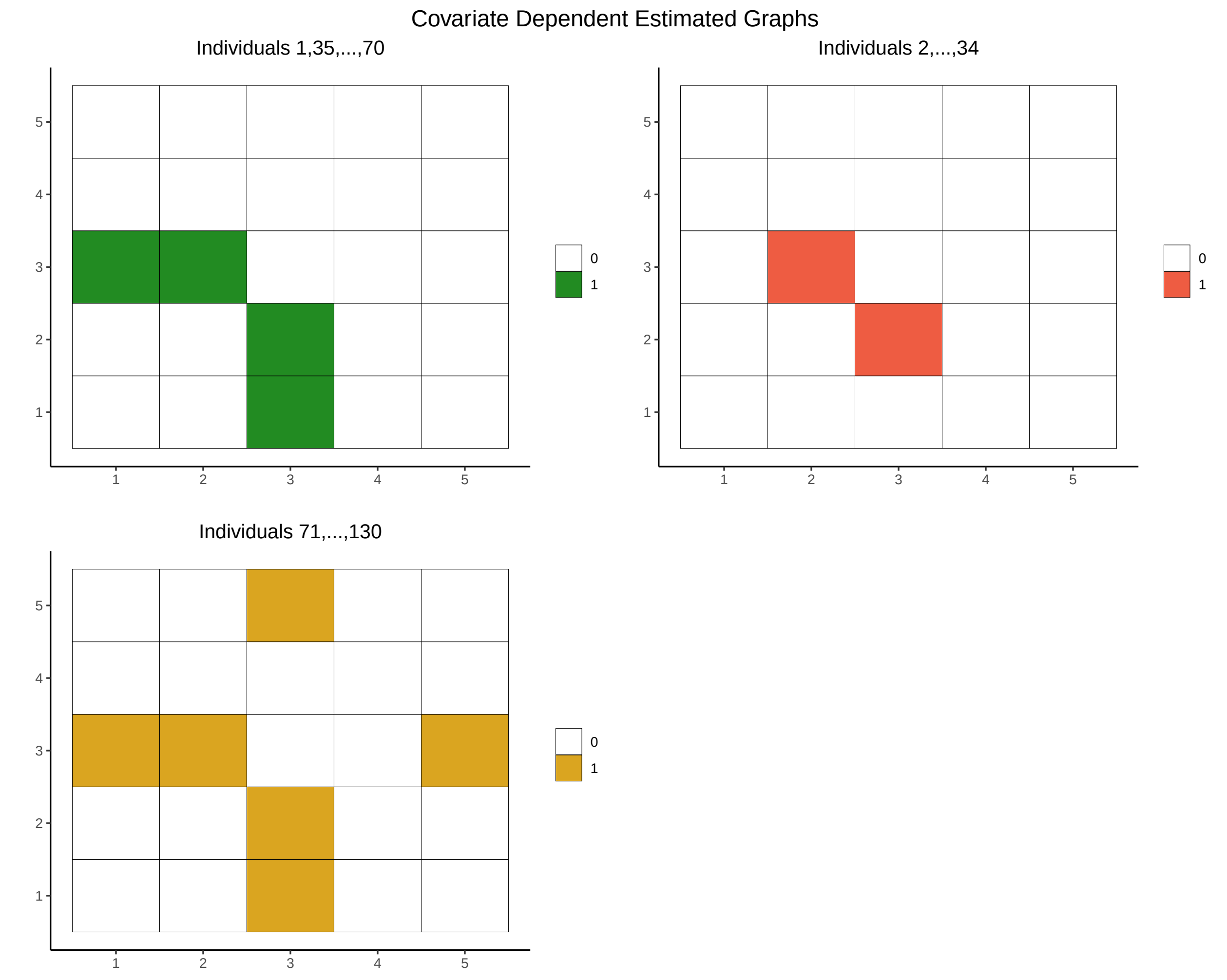
```
## [1] 0.4957504
```

```r
# estimate the covariance structure dependent of the covariate
out_dep <- covdepGE(data_mat,
  Z, # extraneous covariates
  sigmasq = mean(sigmasq), # hyperparameter residual variance
  var_min = 1e-3, # smallest sigmabeta_sq grid value
  var_max = 5, # largest sigmabeta_sq grid value
  n_sigma = 50, # length of the sigmabeta_sq grid
  pi_vec = 0.1, # prior inclusion probability
  tolerance = 1e-10, # variational parameter exit condition 1
  max_iter = 1e3, # variational parameter exit condition 2
  print_time = T
)
```

```
## Warning in covdepGE(data_mat, Z, sigmasq = mean(sigmasq), var_min = 0.001, :
## Response 3: 1/50 candidate models did not converge in 1000 iterations
```

```
## Time difference of 4.416915 secs
```

```r
# visualize each of the unique graphs
colors <- c("forestgreen", "tomato2", "goldenrod", "steelblue")
annotate_figure(ggarrange(plotlist = gg_adjMats(out_dep, colors)),
                top = text_grob("Covariate Dependent Estimated Graphs", size = 15))
```



Covariate independent graph estimation

Gaussian Mixture Model clustering will first be applied to the extraneous covariate. The number of clusters is selected by optimizing BIC.

For all of the individuals within each of the clusters identified by GMM, the shared graph will be estimated by applying `covdepGE` using a constant value for the extraneous covariate. Using a constant value for the extraneous covariate will result in the same estimate for all individuals within each cluster.

```
# estimate the dependence structure independent of the covariate

# apply Gaussian Mixture model clustering; selects number of clusters based on
# the model that results in the best BIC
gmm <- Mclust(Z)

# find accuracy of the clustering
fossil::rand.index(gmm$classification, as.numeric(cov_df$interval))
```

```
## [1] 0.9837806
```

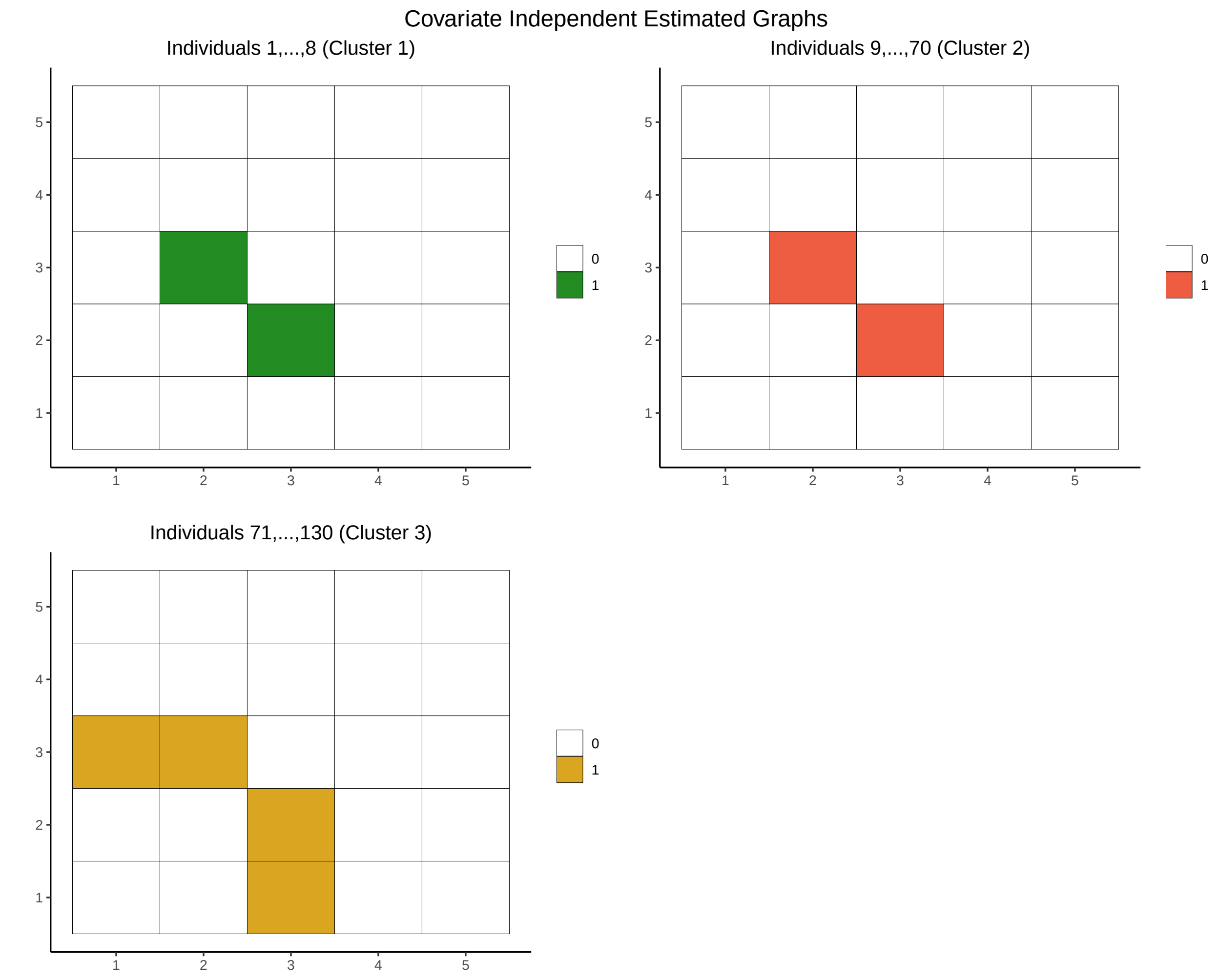
```r
# find number of clusters in final clustering
(num_clusters <- length(unique(gmm$classification)))
```

```
## [1] 3
```

```r
out_indep <- vector("list", num_clusters)

# iterate over each of the clusters identified by GMM
for (k in 1:num_clusters) {

  # fix the datapoints in the k-th cluster
  data_mat_k <- data_mat[gmm$classification == k, ]

  # use varbvs to get the hyperparameter sigma
  sigmasq_k <- sapply(1:(p + 1), function(col_ind) mean(varbvs::varbvs(
    data_mat_k[, -col_ind], NULL, data_mat_k[, col_ind], verbose = F)$sigma))

  # apply the GGM using covdepGE with constant Z, save the resulting graph
  out_indep[[k]] <- covdepGE(data_mat_k,
    rep(0, nrow(data_mat_k)), # extraneous covariates
    sigmasq = mean(sigmasq_k), # hyperparameter residual variance
    var_min = 1e-3, # smallest sigmabeta_sq grid value
    var_max = 5, # largest sigmabeta_sq grid value
    n_sigma = 50, # length of the sigmabeta_sq grid
    pi_vec = 0.1, # prior inclusion probability
    tolerance = 1e-10, # variational parameter exit condition 1
    max_iter = 1e3, # variational parameter exit condition 2
    print_time = T,
    kde = F, # whether to use kde to calculate bandwidths
    scale = F # whether to scale the extraneous covariates
  )
}
```

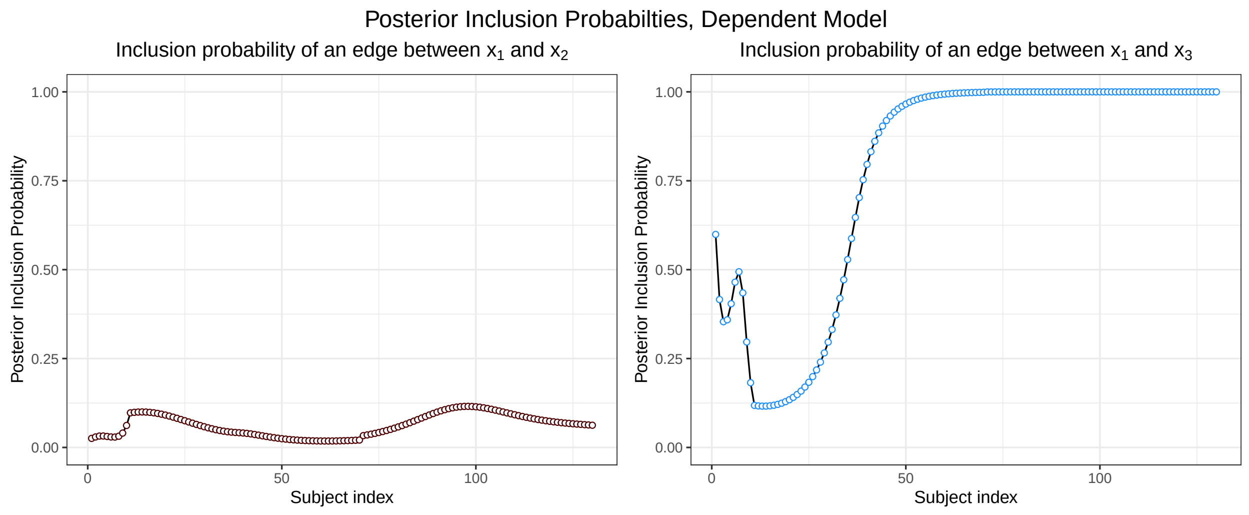
```
## Warning in covdepGE(data_mat_k, rep(0, nrow(data_mat_k)), sigmasq =
## mean(sigmasq_k), : For 3/5 responses, the selected value of sigmabeta_sq was on
## the grid boundary. See return value ELBO for details

## Time difference of 0.01695585 secs

## Warning in covdepGE(data_mat_k, rep(0, nrow(data_mat_k)), sigmasq =
## mean(sigmasq_k), : For 2/5 responses, the selected value of sigmabeta_sq was on
## the grid boundary. See return value ELBO for details

## Time difference of 0.381978 secs

## Warning in covdepGE(data_mat_k, rep(0, nrow(data_mat_k)), sigmasq =
## mean(sigmasq_k), : Response 1: 5/50 candidate models did not converge in 1000
## iterations

## Warning in covdepGE(data_mat_k, rep(0, nrow(data_mat_k)), sigmasq =
## mean(sigmasq_k), : For 2/5 responses, the selected value of sigmabeta_sq was on
## the grid boundary. See return value ELBO for details
## Time difference of 1.149925 secs
```

```r
# get the graphs and pip matrices for each cluster
pip_mats <- sapply(out_indep, function(out) unique(out$inclusion_probs))
adj_mats <- sapply(out_indep, function(out) unique(out$graphs))

# find the individuals in each of the clusters
clust_inds <- lapply(1:num_clusters, function(cl_ind) which(gmm$classification == cl_ind))
clust_inds_sum <- sapply(clust_inds, function(clust) paste0(lapply(split(
  clust, cumsum(c(1, diff(clust) != 1))), function(idx_seq) paste0(
    min(idx_seq), ",...,", max(idx_seq))), collapse = ","))

# create a list of n graphs according to the independent estimate for each
# cluster and the gmm cluster assignment for each individual
indep_graphs <- sapply(1:n, function(ind_idx) adj_mats[gmm$classification[ind_idx]])

# create  a list of n posterior inclusion probabilities
indep_pip <- sapply(1:n, function(ind_idx) pip_mats[gmm$classification[ind_idx]])

# visualize the resulting graphs
cl_gr_viz <- lapply(1:num_clusters, function(cl_ind) gg_adjMat(
  adj_mats[[cl_ind]], color1 = colors[[cl_ind]]) + labs(
    title = paste0("Individuals ", clust_inds_sum[cl_ind], " (Cluster ", cl_ind, ")")))
cl_gr_vis <- rep(list(ggplot() + theme_void()), 4)
cl_gr_vis[1:length(cl_gr_viz)] <- cl_gr_viz
annotate_figure(ggarrange(plotlist = cl_gr_vis),
                top = text_grob("Covariate Independent Estimated Graphs", size = 15))
```

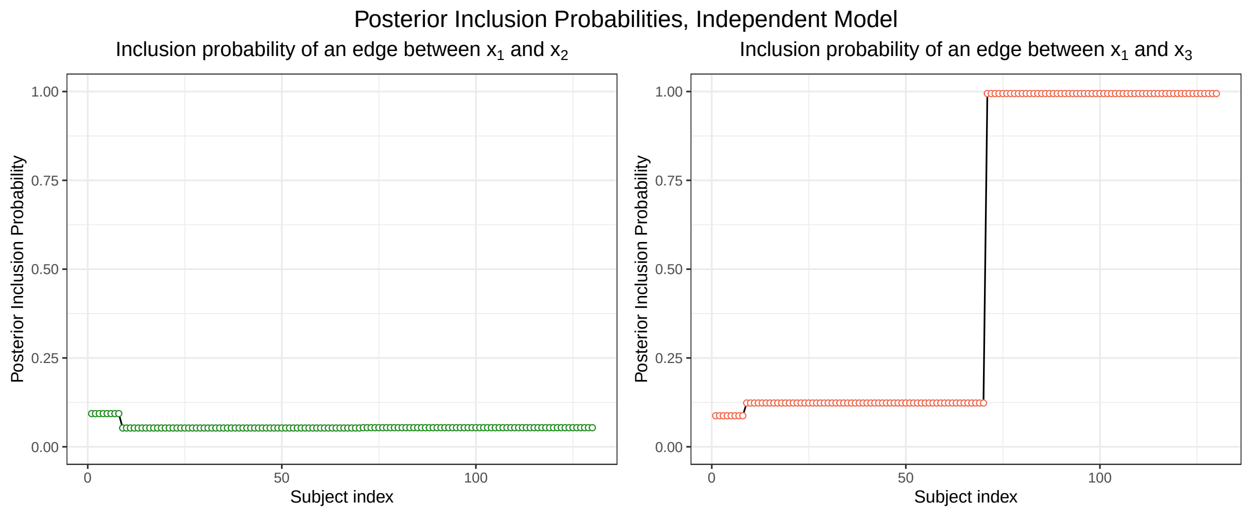


Performance Analysis

```r
# analyze results

# visualize the posterior inclusion probabilties for both models
annotate_figure(ggarrange(gg_inclusionCurve(out_dep, 1, 2), gg_inclusionCurve(
  out_dep, 1, 3, point_color = "dodgerblue")), top = text_grob(
    "Posterior Inclusion Probabilties, Dependent Model", size = 15))
```



```
out_dep_copy <- out_dep
out_dep_copy$inclusion_probs <- indep_pip
annotate_figure(ggarrange(gg_inclusionCurve(
  out_dep_copy, 1, 2, point_color = "forestgreen"), gg_inclusionCurve(
    out_dep_copy, 1, 3, point_color = "coral2")), top = text_grob(
      "Posterior Inclusion Probabilities, Independent Model", size = 15))
```

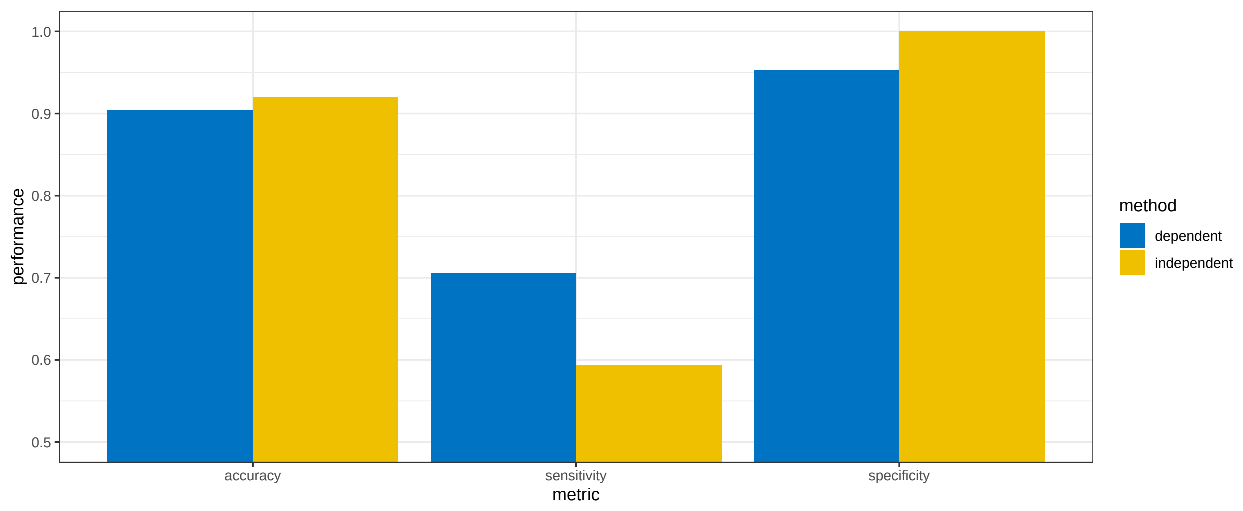


```
# find sensitivity, specificity, and accuracy for each method

# find the true graph for each individual
true_graphs <- lapply(prec_mats, function(mat) ((mat - diag(diag(mat))) != 0) * 1)

# find the total number of entries in all of the graphs
true_gr_vector <- unlist(true_graphs)
num_entries <- length(true_gr_vector)
# num_entries == n * (p + 1)^2

# find the total number of 1's and 0's in all the graphs
tot1 <- sum(true_gr_vector)
```

```r
tot0 <- sum(-true_gr_vector + 1)
# tot1 + tot0 == num_entries

# create a list of lists; the j-th value in the outer list is a list with three
# values: the true, estimated (covariate dependent), and estimated (covariate
# independent) graphs for the j-th individual
true_est_graphs <- lapply(1:n, function(ind_idx) list(
  true = true_graphs[[ind_idx]], est_dep = out_dep$graphs[[ind_idx]],
  est_indep = indep_graphs[[ind_idx]]))

# find the number of true positives for each method
TP_dep <- sum(sapply(1:n, function(ind_idx) sum(
  (true_est_graphs[[ind_idx]]$true == 1) & (true_est_graphs[[ind_idx]]$est_dep == 1))))
TP_indep <- sum(sapply(1:n, function(ind_idx) sum(
  (true_est_graphs[[ind_idx]]$true == 1) & (true_est_graphs[[ind_idx]]$est_indep == 1))))

# find the number of true negatives for each method
TN_dep <- sum(sapply(1:n, function(ind_idx) sum(
  (true_est_graphs[[ind_idx]]$true == 0) & (true_est_graphs[[ind_idx]]$est_dep == 0))))
TN_indep <- sum(sapply(1:n, function(ind_idx) sum(
  (true_est_graphs[[ind_idx]]$true == 0) & (true_est_graphs[[ind_idx]]$est_indep == 0))))

# find sensitivity, specificity, and accuracy for each method
sensitivity_dep <- TP_dep / tot1
sensitivity_indep <- TP_indep / tot1

specificity_dep <- TN_dep / tot0
specificity_indep <- TN_indep / tot0

accuracy_dep <- (TN_dep + TP_dep) / num_entries
accuracy_indep <- (TN_indep + TP_indep) / num_entries

# visualize performance
(perf_res_df <- data.frame(performance = c(
  sensitivity_dep, sensitivity_indep, specificity_dep, specificity_indep,
  accuracy_dep, accuracy_indep), method = rep(c("dependent", "independent"), 3),
  metric = rep(c("sensitivity", "specificity", "accuracy"), each = 2)))
```

```
##   performance      method      metric
## 1   0.7062500   dependent sensitivity
## 2   0.5937500 independent sensitivity
## 3   0.9532567   dependent specificity
## 4   1.0000000 independent specificity
## 5   0.9046154   dependent    accuracy
## 6   0.9200000 independent    accuracy
```

```r
y_lower <- floor(min(perf_res_df$performance) * 10) / 10
ggplot(perf_res_df, aes(metric, performance, fill = method)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  coord_cartesian(ylim = c(y_lower, 1)) +
  theme_bw() +
  ggsci::scale_fill_jco()
```



```
# save the performance data
setwd("~/Jacob/covdepGE/dev/analyses_and_demos")
save(perf_res_df, file = paste0("performance_data_int", sparse_interval,
                                 "_sparse", sparsity_level, ".Rda"))
```