

SUPPLEMENTARY MATERIAL: CODE

MCP_Rand.R

```

1 library(DoseFinding)
2 library(ggplot2)
3 library(logistf)
4 library(dplyr)
5 library(broom)
6
7 ### logit Functions #####
8 logit <- function(p) log(p / (1 - p))
9 inv_logit <- function(y) 1 / (1 + exp(-y))
10
11 #### Permuted Block Randomization #####
12 # This method is rerandomizing a sequence that was created PBR
13 ### x is the full randomization sequence that we which to randomize
14 ### B is the block size
15 PBR <- function(x, B=7){
16   num_blocks <- round(length(x)/B)
17   full_blocks <- num_blocks == round(length(x)/B)
18   for(i in 1:num_blocks){
19     x[((i-1)*B+1):(i*B)] <- sample(x[((i-1)*B+1):(i*B)])
20   }
21   if(!full_blocks){
22     x[num_blocks*B+1:length(x)] <- sample(x[num_blocks*B+1:length(x)])
23   }
24   return(x)
25 }
26
27 ##### Generating Data Example Function #####
28 generate_data <- function(seed=20, doses, mods, n, randMethod="PBR", B=7, x1,
29   timetrend=FALSE, p1=0.2, pK=0.2){
30   #### Data Generation #####
31   set.seed(seed, kind = "Mersenne-Twister", sample.kind = "Rejection",
32     normal.kind = "Inversion")
33   dose_block <- c(0,10,10,25,25,100,100)
34   if(randMethod=="CR"){
35     dose_vector <- sample(doses, size =sum(n), replace = T, prob=n/sum(n))
36   } else if (randMethod=="RA"){
37     dose_vector <- rep(dose_block, sum(n)/B)
38     dose_vector <- sample(dose_vector)
39   } else if (randMethod=="PBR") {
40     dose_vector <- rep(dose_block, sum(n)/B)
41     dose_vector <- PBR(dose_vector, B)
42   }
43   ## emax cal
44   emax_max=(logit(pK)-logit(p1))*110/100
45   ## Covariate
46   prob <- inv_logit(emax(dose_vector, logit(p1), emax_max, 10) + 0.6*x1)
47   #### Time Trend ####
48   if(timetrend){

```

```

49   prob <- prob + seq(-0.2,0.2,length.out=sum(n))
50 }
51 prob[prob > 1] <- 1; prob[prob < 0] <- 0  # Move time trend to logit-scale
52 dat <- data.frame(y = rbinom(sum(n), 1, prob), dose = dose_vector)
53 return(dat)
54 }
55
56
57 #reg_mod return mean and covariance matrix or residuals
58 #allows for glm and logistf
59 #pearson & working residuals are commented out but can be used for future research
60 reg_mod <- function(reg="glm", covariates = FALSE, test_type="population", dat, x1,
61                     doses){
62   if(test_type=="randomisation_residuals"){
63     if(reg=="glm" & !covariates){
64       fit <- glm(y~1 , data = dat, family = binomial)
65       ### Pearson Residuals
66       #residus <- residuals(fit, type="pearson")
67       ### Response Residuals
68       residus <- residuals(fit, type="response")
69       ### Working Residuals
70       #residus <- residuals(fit, type="working")
71     } else if(reg=="logistf" & !covariates){
72       fit <- logistf(y ~ 1, data = dat)
73       ### Pearson Residuals
74       #res <- dat$y - fit$predict
75       #residus <- res/(sqrt(fit$hat.diag))
76       ### Response Residuals
77       residus <- dat$y - fit$predict
78       ### Working Residuals
79       #res <- dat$y - fit$predict
80       #residus <- res/(fit$hat.diag)
81     } else if(reg=="glm" & covariates){
82       fit <- glm(y~1 + x1 , data = dat, family = binomial)
83       ### Pearson Residuals
84       #residus <- residuals(fit, type="pearson")
85       ### Response Residuals
86       residus <- residuals(fit, type="response")
87       ### Working Residuals
88       #residus <- residuals(fit, type="working")
89     } else if(reg=="logistf" & covariates){
90       fit <- logistf(y ~ 1+ x1, data = dat)
91       ### Pearson Residuals
92       #res <- dat$y - fit$predict
93       #residus <- res/(sqrt(fit$hat.diag))
94       ### Response Residuals
95       residus <- dat$y - fit$predict
96       ### Working Residuals
97       #res <- dat$y - fit$predict
98       #residus <- res/(fit$hat.diag)
99     }

```

```

100   return(list(residus=residus))
101 } else {
102   if(reg=="glm" & !covariates){
103     fit <- glm(y ~ factor(dose) + 0, data = dat, family = binomial)
104     mu_hat <- coef(fit); S_hat <- vcov(fit)
105   } else if(reg=="logistf" & !covariates){
106     fit <- logistf(y ~ factor(dose) + 0, data = dat)
107     mu_hat <- fit$coefficients; S_hat <- fit$var
108   } else if(reg=="glm" & covariates){
109     fit <- glm(y ~ factor(dose) + x1 -1, data = dat, family = binomial)
110     mu_hat <- coef(fit)[1:length(doses)]
111     S_hat <- vcov(fit)[1:length(doses), 1:length(doses)]
112   } else if(reg=="logistf" & covariates){
113     fit <- logistf(y ~ factor(dose) + x1 - 1, data = dat)
114     mu_hat <- fit$coefficients[1:length(doses)]
115     S_hat <- fit$var[1:length(doses), 1:length(doses)]
116   }
117 }
118
119 return(list(mu_hat=mu_hat, S_hat=S_hat))
120 }
121
122 ##### MCP Test based on Response Residuals #####
123 MCP_RD <- function(dat, doses, models, contMat, residus, n) {
124   r_mean <- tapply(residus, dat$dose, mean)
125   sZ_mean <- tapply(residus, dat$dose, var)
126
127   SZ <- apply(contMat, 2, function(col) {
128     sum(col * r_mean) / sqrt(sum(col^2 * sZ_mean / n))
129   })
130
131   return(SZ)
132 }
133
134
135 ### rerandomizing randomisation sequence
136 get_randomised_data <- function(dat, randMethod, doses, B, N, n) {
137   dose_vector <- switch(randMethod,
138     "CR" = sample(doses, size = N, replace = TRUE, prob = n/N),
139     "RA" = sample(dat$dose),
140     "PBR" = PBR(dat$dose, B))
141   dat$dose <- dose_vector
142   return(dat)
143 }
144
145 MCP_Rand <- function(dat=dat, doses=doses, n=n, test_type="population",
146   reg="glm", randMethod="PBR", B=7, x1, covariate=FALSE,
147   nrand=1000){
148   doses <- c(0, 10, 25, 100)
149   mods <- DoseFinding::Mods(emax = c(10, 50),
150     sigEmax = rbind(c(5, 3), c(25, 3)),

```

```

151         #betaMod = c(0.2475, 2.025),      # original betaMod
152         betaMod = c(0.1529, 0.5809),      # new betaMod
153         doses = doses)
154 reg_val <- reg_mod(dat=dat, reg=reg, x1=x1, test_type=test_type,
155                   covariates=covariate, doses)
156 if(test_type=="population"){
157   pvals <- min(attr(MCTtest(dose=doses, resp=reg_val$mu_hat, S=reg_val$S_hat,
158                           models = mods, alternative = "one.sided", alpha = 0.1,
159                           type = "general")$tStat, "pVal"))
160 } else if(test_type=="randomisation_mean"){
161   results <- rep(0, nrand + 1)
162   results[1] <- max(MCTtest(doses, reg_val$mu_hat, S = reg_val$S_hat,
163                           models = mods, type = "general", pVal = FALSE, critV = 0)$tStat)
164   for(j in 1:nrand){
165     datR <- get_randomised_data(dat[, 1:2], randMethod, doses, B, sum(n), n)
166     rand_reg_val <- reg_mod(dat=datR, reg=reg, x1=x1, test_type=test_type,
167                           covariates=covariate, doses)
168     results[j+1] <- max(MCTtest(doses, rand_reg_val$mu_hat,
169                               S = rand_reg_val$S_hat, models = mods, type = "general",
170                               pVal = FALSE, critV = 0)$tStat)
171   }
172   pvals <- sum(results[-1] - results[1] >= 0) / nrand
173 } else if(test_type=="randomisation_residuals") {
174   reg_val <- reg_mod(dat=dat, reg=reg, x1=x1, covariates=covariate,
175                     doses, test_type="population")
176   contMat <- optContr(models = mods, w=n)$contMat
177   reg_val <- reg_mod(dat=dat, reg=reg, x1=x1, covariates=covariate,
178                     test_type=test_type)
179   residus <- reg_val$residus
180   results <- rep(0, nrand + 1)
181   results[1] <- max(MCP_RD(dat, doses, models, contMat, residus, n))
182   for(j in 1:nrand){
183     datR <- get_randomised_data(dat[, 1:2], randMethod, doses, B, sum(n), n)
184     results[j+1] <- max(MCP_RD(datR, doses, models, contMat, residus, n))
185   }
186   pvals <- sum(results[-1] - results[1] >= 0) / nrand
187 }
188 return(pvals)
189 }

```

runsimMP.R

```

1   rm(list = ls())
2   usethis::edit_r_profile()
3   library(detectseparation)
4   library(clustermq)
5   library(DoseFinding)
6   library(ggplot2)
7   library(logistf)
8   library(dplyr)

```

```

9 library(broom)
10 library(tidyr)
11
12 ## Set LSF as scheduler and path to template
13 options(clustermq.scheduler = "lsf")
14
15 source("MCP_Rand.R") ## load auxiliary code
16 export <- list(logit=logit, inv_logit=inv_logit, PBR = PBR,
17               generate_data=generate_data, reg_mod=reg_mod, MCP_RD = MCP_RD,
18               get_randomised_data=get_randomised_data, MCP_Rand=MCP_Rand,
19               MCTtest=MCTtest, detect_separation=detect_separation)
20
21 ## setup scenarios (toy example)
22 doses <- c(0, 10, 25, 100)
23 covariate <- c(TRUE)
24 reg <- c("glm", "logistf")
25 test_type <- c("population", "randomisation_mean", "randomisation_residuals")
26
27 # Create the full factorial grid first
28 params_changed <- expand.grid(test_type=test_type, reg=reg, covariate=covariate)
29
30 # Filter for MCP benchmark configurations only
31 filtered_params <- params_changed[!(params_changed$test_type == "population" &
32   params_changed$reg != "glm"),]
33 filtered_params <- filtered_params[!(filtered_params$test_type == "population" &
34   filtered_params$covariate == FALSE),]
35
36 # Convert remaining combinations to a data frame
37 filtered_params <- as.data.frame(filtered_params)
38
39 # Define the custom function
40 proportion_less_than_005 <- function(x) {
41   # Remove NAs from the input vector
42   x <- na.omit(x)
43   if (length(x) == 0) return(NA) # Handle case where no data points left after
44   # omitting NAs
45   sum(x < 0.05) / length(x)
46 }
47
48 # Define the custom function
49 proportion_less_than_01 <- function(x) {
50   # Remove NAs from the input vector
51   x <- na.omit(x)
52   if (length(x) == 0) return(NA) # Handle case where no data points left
53   # after omitting NAs
54   sum(x < 0.1) / length(x)
55 }
56
57 MCR_Rand_dat <- function(seed=1, randMethod= "PBR", doses=doses, n=n, B=7, p1=0.2,
58   pK=0.2, nrand=nrand, timetrend = FALSE,
59   filtered_params=filtered_params){

```

```

60  n_methods <- dim(filtered_params)[1]
61  p_vals <- rep(0, n_methods)
62  # Initialize run_time to store the runtime of each method
63  run_time <- rep(0, n_methods)
64  no_respon = FALSE; fail_MCP = FALSE; completesep = FALSE
65  mods <- DoseFinding::Mods(emax = c(10, 50),
66                            sigEmax = rbind(c(5, 3), c(25, 3)),
67                            #betaMod = c(0.2475, 2.025),
68                            betaMod = c(0.1529, 0.5809),
69                            doses = doses)
70  plotMods(mods, trafo = inv_logit)
71  x1 <- rnorm(sum(n), 0, 1)
72  dat <- generate_data(seed=seed, doses=doses, mods=mods, randMethod=randMethod,
73                      B=B, n=n, x1=x1, p1=p1, pK=pK, timetrend = timetrend )
74  result <- glm(y ~ 1, data = dat, family = binomial, method = "detect_separation")
75  completesep <- result$outcome
76
77  if(sum(dat$y)==0){
78    p_vals = rep(NA, n_methods)
79    run_time = p_vals
80  } else {
81    for(i in 1:n_methods){
82      if(sum(dat[dat$dose == 0,]$y)==0){completesep=TRUE}
83      start_time <- Sys.time() # Capture start time
84
85      p_vals[i] <- MCP_Rand(dat=dat, doses=doses, n=n,
86                          test_type=filtered_params$test_type[i],
87                          reg=filtered_params$reg[i],
88                          randMethod="PBR", B=B, x1=x1,
89                          covariate=filtered_params$covariate[i], nrand=nrand)
90
91      end_time <- Sys.time() # Capture end time
92
93      run_time[i] <- end_time - start_time # Calculate runtime
94      if(filtered_params$test_type[i]=="population" & is.na(p_vals[i])){
95        fail_MCP = TRUE
96      }
97    }
98  }
99
100  list(p_vals = p_vals, run_time = run_time, no_respon = no_respon,
101       fail_MCP = fail_MCP, completesep = completesep)
102  # Return both p-values and runtimes
103 }
104
105 simPlots <- function(results, paramscom, sim_matrix_p_vals=0){
106   # Convert covariate to factor for better visualization
107   results$reg <- factor(results$reg, labels = c("glm", "logistf"))
108
109   # Add an additional column to combine "test_type" and "reg"
110   # for purely visual purposes

```

```

111 results$combined <- interaction(results$test_type, results$reg)
112 # Defining the colors
113 colors <- c(rgb(215,25,28, maxColorValue = 255),
114             rgb(253, 174, 97, maxColorValue = 255),
115             rgb(255, 255, 51, maxColorValue = 255),
116             rgb(171, 221, 164, maxColorValue = 255),
117             rgb(43, 131, 186, maxColorValue = 255))
118 # Define the legend labels
119 methods <- c(
120   "Population-based Test",
121   "Randomisation Test (MLE)",
122   "Randomisation Test (MLE; residual-based)",
123   "Randomisation Test (penalized MLE)",
124   "Randomisation Test (pen. MLE; residual-based)"
125 )
126 legend_labels <- methods
127
128 # Create the bar plot using ggplot
129 t1 <- ggplot(results, aes(x = combined, y = TypeError_Power*100,
130                          fill = combined)) +
131   geom_bar(stat = "identity", position = "dodge", color = "black", size = 0.5) +
132   scale_fill_manual(values = colors, labels = legend_labels) +
133   labs(title = "Type I Error Rate for Different Tests",
134        x = "", y = "Type I Error Rate in %",
135        fill = "Legend") +
136   ylim(0,12) +
137   # Applying a minimal theme with larger base font size
138   theme_minimal(base_size = 15) +
139   theme(
140     axis.text.x = element_blank(),           # Remove the text under each bar
141     axis.ticks.x = element_blank(),          # Remove the ticks under each bar
142     legend.title = element_blank(),          # Remove legend title
143     legend.text.align = 0,                   # Align legend text
144     # Center the title, increase size, and make it bold
145     plot.title = element_text(hjust = 0.5, size = 18, face = "bold"),
146     legend.position = "bottom",              # Position the legend to the right
147     # Increase space between legend elements
148     legend.spacing.y = unit(0.5, 'cm'),
149     legend.text = element_text(size = 12) # Increase the legend text size
150   ) +
151   geom_text(aes(label = round(TypeError_Power*100, 3)),
152            position = position_dodge(width = 0.9), vjust = -0.5, size = 4)
153 # Save the plot to a PDF file
154 ggsave(paste0("t1_", paramscom, ".pdf"), plot = t1, width = 14, height = 7)
155
156
157 # Create the bar plot using ggplot
158 ptt <- ggplot(results, aes(x = combined, y = RunTime, fill = combined)) +
159   geom_bar(stat = "identity", position = "dodge") +
160   scale_fill_manual(values = colors, labels = legend_labels) +

```

```

162 labs(title = "RunTime for Different Tests",
163       x = "",
164       y = "RunTime (Seconds)",
165       fill = "Test Type") +
166 # Applying a minimal theme with larger base font size
167 theme_minimal(base_size = 15) +
168 theme(
169   axis.text.x = element_blank(),           # Remove the text under each bar
170   axis.ticks.x = element_blank(),           # Remove the ticks under each bar
171   legend.title = element_blank(),           # Remove legend title
172   legend.text.align = 0,                    # Align legend text
173   # Center the title, increase size, and make it bold
174   plot.title = element_text(hjust = 0.5, size = 18, face = "bold"),
175   legend.position = "bottom",               # Position the legend to the right
176   # Increase space between legend elements
177   legend.spacing.y = unit(0.5, 'cm'),
178   legend.text = element_text(size = 12)     # Increase the legend text size
179 ) +
180 geom_text(aes(label = round(RunTime, 3)),
181           position = position_dodge(width = 0.9), vjust = -0.5, size = 4)
182
183 # Save the plot to a PDF file
184 ggsave(paste0("ptt_", paramscom, ".pdf"), plot = ptt, width = 11, height = 5)
185
186 # Convert matrix to long format
187 # df <- sim_matrix_p_vals[-1,]
188 df <- sim_matrix_p_vals
189 df <- df %>%
190   # mutate(method = methods[-1]) %>%
191   mutate(method = methods) %>%
192   pivot_longer(cols = -method, names_to = "variable", values_to = "p_value")
193 # Factorize the method variable with specified levels' order
194 df <- df %>%
195   mutate(method = factor(method, levels = methods))
196 # mutate(method = factor(method, levels = methods[-1]))
197
198 # Exclude NAs and create the plot
199 plot_pval <- ggplot(df %>% filter(!is.na(p_value)),
200                   aes(x = p_value, fill = method)) +
201   geom_histogram(bins = 50, alpha = 0.5, position = "identity", boundary = 0,
202                 breaks = seq(0, 1, length.out = 51)) +
203   scale_fill_manual(values = colors) +
204   labs(title = "Histogram p-values by test", x = "P-value", y = "Frequency",
205        fill = "Test") +
206   theme_minimal() +
207   theme(legend.position = "bottom",
208         plot.title = element_text(hjust = 0.5, size = 18, face = "bold")) +
209   facet_wrap(~ method, scales = "free_y", ncol = 2) + guides(fill="none")
210
211 ggsave(paste0("plot_pval_", paramscom, ".pdf"), plot = plot_pval,
212       width = 11, height = 8)

```

```

213 }
214
215 # Define the vectors
216 n <- c(7, 14, 14, 14)
217 time_trend <- c(TRUE, FALSE)
218 rand_method <- c("CR", "RA", "PBR")
219 sample_size <- list(n, 2*n, 10*n)
220 pK <- seq(0.2, 0.9, 0.1)
221
222 # Generate the combination grid
223 grid <- rbind(
224   expand.grid(
225     pK = pK,
226     rand_method = rand_method[2:3],
227     sample_size = I(sample_size[1]),
228     time_trend = time_trend,
229     KEEP.OUT.ATTRS = FALSE,
230     stringsAsFactors = FALSE
231   ),
232   expand.grid(
233     pK = c(0.61, seq(0.2, 0.7, 0.1)),
234     rand_method = rand_method[2:3],
235     sample_size = I(sample_size[2]),
236     time_trend = time_trend,
237     KEEP.OUT.ATTRS = FALSE,
238     stringsAsFactors = FALSE
239   ),
240   expand.grid(
241     pK = c(0.364, seq(0.2, 0.45, 0.05)),
242     rand_method = rand_method,
243     sample_size = I(sample_size[3]),
244     time_trend = time_trend,
245     KEEP.OUT.ATTRS = FALSE,
246     stringsAsFactors = FALSE
247   )
248 )
249
250 # Loop over all combinations
251 for (i in 1:nrow(grid)) {
252   current_combination <- grid[i,]
253   # Extracting current values (you can use these values in your tasks)
254   current_time_trend <- current_combination$time_trend
255   current_rand_method <- current_combination$rand_method
256   current_sample_size <- current_combination$sample_size[[1]]
257   current_pK <- current_combination$pK
258
259   print(current_combination)
260
261   # Create the full factorial grid first
262   param_constatnt <- list(nrand=1000, doses=doses,
263     n=current_combination$sample_size[[1]], B=7,

```

```

264         timetrend = current_combination$time_trend,
265         pK= current_combination$pK,
266         randMethod= current_combination$rand_method,
267         filtered_params=filtered_params)
268
269     # Number of Simulations
270     nsim = 10000
271     seeds <- data.frame(seed=seq(1,nsim))
272
273     sim_out <- Q_rows(fun = MCR_Rand_dat,
274                     df = seeds,
275                     const = param_constatnt,
276                     seed = 23456,
277                     n_jobs = nrow(seeds),
278                     template = list(
279                         walltime = 300,
280                         job_name = "MCP_Rand",
281                         log_file = "logs.txt"),
282                     pkgs = c("data.table", "DoseFinding", "logistf"),
283                     export = export)
284
285     paramscom <- paste("nsim", nsim, "nrand", param_constatnt$nrand, "pK",
286                      round(current_combination$pK*100), "rand_method",
287                      current_combination$rand_method, "N",
288                      sum(current_combination$sample_size[[1]]),
289                      'TimeTrend', current_time_trend, sep="_")
290     filename <- paste("simulation", paramscom, "FULL", sep="_")
291     saveRDS(sim_out, file= filename)
292
293     percent_completesep = sum(do.call(rbind, lapply(sim_out,
294                                                    function(x) x$completesep)))/nsim
295     percent_fail_MCP = sum(do.call(rbind, lapply(sim_out,
296                                                  function(x) x$fail_MCP)))/nsim
297     percent_no_respon = sum(do.call(rbind, lapply(sim_out,
298                                                  function(x) x$no_respon)))/nsim
299     print(c(percent_completesep, percent_fail_MCP, percent_no_respon))
300
301     p_vals_list <- lapply(sim_out, function(x) x$p_vals)
302     run_time_list <- lapply(sim_out, function(x) x$run_time)
303
304     # Convert the list into a matrix
305     sim_matrix_p_vals <- as.data.frame(t(do.call(rbind, p_vals_list)))
306     run_times <- as.data.frame(t(do.call(rbind, run_time_list)))
307
308     row.names(filtered_params) <- NULL
309     # Convert the matrix to a data frame
310     results <- cbind(filtered_params, RunTime = rowMeans(run_times),
311                     TypeError_Power=apply(sim_matrix_p_vals, 1, proportion_less_than_01))
312     print(results)
313
314     filename <- paste("simulation", paramscom, "RESULTS", sep="_")

```

```

315     saveRDS(results, file= filename)
316
317     #plot
318     simPlots(results, paramscom, sim_matrix_p_vals=sim_matrix_p_vals)
319 }
320
321 ##### Power Plot Figure 2 in Paper #####
322 ## Other Plots can be created by changing imported file names
323
324 pK_values <- as.integer(seq(0.2, 0.9, 0.1)*100)
325 #pK_values <- sort(as.integer(c(0.61, seq(0.2, 0.7, 0.1))*100))
326 #pK_values <- sort(as.integer(c(0.364, seq(0.2, 0.45, 0.05))*100))
327
328 # Initialize a list to store the last columns of the files
329 last_columns <- list()
330
331 # Loop through each pK value and read the corresponding RDS file
332 for (pK in pK_values) {
333     # Construct the filename
334     filename <- sprintf("/home/pinlul/Documents/simulation_nsim_10000_nrand_1000_pK_
335                        %d_rand_method_PBR_N_49_TimeTrend_FALSE_RESULTS", pK)
336
337     # Print the filename (optional, for debugging)
338     cat("Reading file:", filename, "\n")
339
340     # Read the RDS file
341     data <- readRDS(filename)
342
343     # Extract the last column and save it in the list
344     last_columns[[as.character(pK)]] <- data[[ncol(data)]]
345 }
346
347 # Optionally, combine all the last columns into a single data frame
348 # You can name the columns using the corresponding pK values
349 combined_last_columns <- do.call(cbind, last_columns)
350 colnames(combined_last_columns) <- names(last_columns)
351
352 # Specify the output file and open a PNG graphics device
353 pdf(file = "power_nsim_10000_nrand_1000_rand_method_PBR_N_49_TimeTrend_FALSE
354        _RESULTS.pdf", width = 10, height = 7)
355
356 # Plot the data
357 ##colors <- c("#d7191c", "#fdae61", "#ffffbf", "#abdda4", "#2b83ba")
358 colors <- c(rgb(215,25,28, maxColorValue = 255),
359            rgb(253, 174, 97, maxColorValue = 255),
360            rgb(255, 255, 51, maxColorValue = 255),
361            rgb(171, 221, 164, maxColorValue = 255),
362            rgb(43, 131, 186, maxColorValue = 255))
363 methods <- c(
364     "Population-based Test",
365     "Randomisation Test (MLE)",

```

```

366 "Randomisation Test (MLE; residual-based)",
367 "Randomisation Test (penalized MLE)",
368 "Randomisation Test (pen. MLE; residual-based)"
369 )
370 lty <- c(3,1,1,2,2)
371 matplot(pK_values, t(combined_last_columns), type = 'l', lty = lty, col = colors,
372         lwd = 3, xlab = "Success Rate on Arm K-1 (%)", ylab = "Power",
373         main = "Power Comparison of Different Tests",
374         cex.main = 1.5, cex.axis = 1.2, cex.lab = 1.4)
375 abline(h=seq(0,1,by=0.05), lty=3, col="lightgrey")
376 abline(v=seq(20,90,by=10), lty=3, col="lightgrey")
377 matlines(pK_values, t(combined_last_columns), type = 'l', lty = lty,
378          col = colors, lwd = 2,)
379
380 # Add dots to indicate actual data points
381 for (i in 1:nrow(combined_last_columns)) {
382   points(pK_values, combined_last_columns[i, ], col = colors[i], pch = 16, cex = 1)
383 }
384
385 abline(h = 0.8, col = "black", lty = 2)
386
387 # Add legend in the upper-left corner with more space between elements
388 legend("bottomright", legend = methods, col = colors, lty = lty, cex = 0.8,
389        bty = "n", inset = 0.04, y.intersp = 1.5, lwd=2)
390
391 dev.off()
392
393 print(round(combined_last_columns*100,2))
394
395
396 #### Print percentage of complete separation and failed MCP-Mod
397 # for original data scenario but other can analysed by changing file name
398 filename <- sprintf("/home/pinlul/Documents/simulation_nsim_10000_nrand_1000_pK_
399                    %d_rand_method_PBR_N_49_TimeTrend_FALSE_FULL", 20)
400
401 # Print the filename (optional, for debugging)
402 cat("Reading file:", filename, "\n")
403
404 # Read the RDS file
405 data <- readRDS(filename)
406
407 percent_completesep = sum(do.call(rbind, lapply(data,
408         function(x) x$completesep)))/10000
409 percent_fail_MCP = sum(do.call(rbind, lapply(data, function(x) x$fail_MCP)))/10000
410 c(percent_completesep, percent_fail_MCP)

```

runsimPMX_example.R

```

1 ## Generate potential outcomes based on https://doi.org/10.1002/sim.8913
2 ## Use scenario of strong drug effect,

```

```

3 # 50 patients in total (Scenario III) and Emax model
4 ## Code by Lukas Pin and Bjoern Bornkamp
5
6 # Load necessary libraries
7 library(DoseFinding)
8
9 source("MCP_Rand.R") ## load auxiliary code
10
11 export <- list(logit=logit, inv_logit=inv_logit, PBR = PBR,
12               generate_data=generate_data, reg_mod=reg_mod, MCP_RD = MCP_RD,
13               get_randomised_data=get_randomised_data,
14               proportion_less_than_005=proportion_less_than_005,
15               MCP_Rand=MCP_Rand, MCTtest=MCTtest,
16               detect_separation=detect_separation, richtmyer=richtmyer,
17               gen_pot_outcomes=gen_pot_outcomes)
18
19 gen_pot_outcomes <- function(u){
20   N <- nrow(u)
21   ## generate potential outcomes
22   doses <- c(0, 100, 200, 400, 1000)
23   ## time-point of interest
24   t_end <- 12*30.4375
25   ## generate individual parameters (based on quantile function)
26   VA0i <- 55*exp(qnorm(u[,1], 0, sqrt(0.07)))
27   VAssi <- 30*exp(qnorm(u[,2], 0, sqrt(0.2)))
28   k <- 0.005*exp(qnorm(u[,3], 0, sqrt(0.2)))
29   Emaxi <- 30 + qnorm(u[,4], 0, sqrt(100))
30   ED50 <- 150
31   eps <- qnorm(u[,5], 0, 5.3)
32   Y <- matrix(nrow = N, ncol = 5)
33   for(i in 1:N){
34     dr <- VA0i[i] - VAssi[i]*(1 - exp(-k[i]*t_end)) + Emaxi[i]*doses/(ED50 + doses)
35     Y[i,] <- dr + eps[i]
36   }
37   out <- data.frame(Y, base = VA0i)
38   names(out)[1:5] <- paste0("Y", doses)
39   out
40 }
41
42 ##### Permuted Block Randomization #####
43 # This method is rerandomizing a sequence that was created PBR
44 ### x is the full randomization sequence that we which to randomize
45 ### B is the block size
46 PBR <- function(x, B=7){
47   num_blocks <- round(length(x)/B)
48   full_blocks <- num_blocks == round(length(x)/B)
49   for(i in 1:num_blocks){
50     x[((i-1)*B+1):(i*B)] <- sample(x[((i-1)*B+1):(i*B)])
51   }
52   if(!full_blocks){
53     x[num_blocks*B+1:length(x)] <- sample(x[num_blocks*B+1:length(x)])

```

```

54   }
55   return(x)
56 }
57
58 # Define the custom function
59 proportion_less_than_005 <- function(x) {
60   # Remove NAs from the input vector
61   x <- na.omit(x)
62   # Handle case where no data points left after omitting NAs
63   if (length(x) == 0) return(NA)
64   sum(x < 0.05) / length(x)
65 }
66
67
68 ## generate quasi-random sequence using the Richtmyer sequence
69 richtmyer <- function(N, s){
70   primes <- c(2,3,5,7,11,13,17,19)
71   if(s > length(primes))
72     stop("s too large")
73   gen_vec <- sqrt(primes[1:s])
74   out_mat <- matrix(nrow = N, ncol = s)
75   for(i in 1:N){
76     out_mat[i,] <- (i*gen_vec)%%1
77   }
78   out_mat
79 }
80
81 # Generate quasi-random sequence
82 n_group <- 10
83 u <- richtmyer(5*n_group, 5)
84 plot(u[,1],u[,2])
85 dat <- gen_pot_outcomes(u)
86
87 # Dose levels
88 doses <- c(0, 100, 200, 400, 1000)
89
90 ## plot individual dose-response curves
91 matplot(doses, t(dat[,1:5]), type="l", col = 1, lty=1, ylab="VA change")
92 ## add true population dose-response
93 lines(doses, colMeans(dat[,1:5]), lwd = 3, col=2)
94
95 ## generate a simulated data-set, by randomly selecting from the potential outcomes
96 Z <- sample(rep(1:5, each=n_group)) ## random allocation rule
97 resp <- dose <- numeric(5*n_group)
98 for(i in 1:(5*n_group)){
99   dose[i] <- doses[Z[i]]
100   resp[i] <- dat[i,Z[i]]
101 }
102 obs_dat <- data.frame(dose = dose, resp = resp, base = dat$base)
103 tapply(obs_dat$resp, obs_dat$dose, mean)
104 summary(lm(resp ~base, obs_dat))

```

```

105
106 # Define models for MCTtest
107 mods <- Mods(linear = NULL, emax = 150, sigEmax = c(175, 2),
108             linlog = NULL,
109             doses = doses, addArgs = list(off = 1))
110
111 # MCT Test without covariate
112 mm <- MCTtest(dose, resp, data=obs_dat, models = mods)
113 min(attr(mm$tStat, "pVal"))
114 # MCT Test with adjustment for baseline
115 mm2 <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)
116 min(attr(mm2$tStat, "pVal"))
117
118
119 ## Scenarios
120 ## (i) Original Scenario
121 ## (ii) Order the potential outcome data by "base" (i.e. time trend in recruitment)
122 ## (iii) Null hypothesis: Assume all potential outcomes are equal to Y(0)
123
124 ## Methods to compare
125 ## - Population MCPMod as above (with and without covariates)
126 ## - Randomization MCP-Mod with and without covariates
127 # (both can be based on residual based approach)
128
129 # Main function to perform pharmacological simulations
130 pharmaco <- function(seed=1, rand_prod = "RA", nrand = 1000){
131   #Parameters
132   set.seed(seed)
133   n_group <- 10
134   u <- richtmyer(5*n_group, 5)
135   dat <- gen_pot_outcomes(u)
136   doses <- c(0, 100, 200, 400, 1000)
137   mods <- Mods(linear = NULL, emax = 150, sigEmax = c(175, 2),
138               linlog = NULL,
139               doses = doses, addArgs = list(off = 1))
140
141   # Select randomization procedure
142   if(rand_prod == "RA"){
143     Z <- sample(rep(1:5, each=n_group)) ## random allocation rule
144   } else if (rand_prod == "PBR") {
145     Z <- PBR(x=rep(c(1,1,2,2,3,3,4,4,5,5),5), B=10) # Permuted Block Design
146   }
147
148
149   pvalues_all <- rep(NA, 12)
150
151   # Scenario (i) Original Scenario
152   resp <- dose <- numeric(5*n_group)
153   for(i in 1:(5*n_group)){
154     dose[i] <- doses[Z[i]]
155     resp[i] <- dat[i, Z[i]]

```

```

156 }
157
158 obs_dat <- data.frame(dose = dose, resp = resp, base = dat$base)
159
160 #Population MCPMod without covariates
161 mm <- MCTtest(dose, resp, data=obs_dat, models = mods)
162 pvalues_all[1] = min(attr(mm$tStat, "pVal"))
163
164 #Population MCPMod with covariates
165 mm2 <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)
166 pvalues_all[2] = min(attr(mm2$tStat, "pVal"))
167
168 #Randomization MCP-Mod without covariates
169 results <- rep(0, nrand)
170 obvs <- MCTtest(dose, resp, data=obs_dat, models = mods)$tStat
171 for(j in 1:nrand){
172   datR <- get_randomised_data(obs_dat[, 1:2], randMethod= rand_prod, doses, B=10,
173                               N=50, n=rep(10,5))
174   results[j] <- max(MCTtest(dose, resp, data=datR, models = mods)$tStat)
175 }
176 pvals <- rep(0,4)
177 for(i in 1:4){
178   pvals[i]<- sum(results- obvs[i]>= 0 )/nrand
179 }
180 pvalues_all[3] = min(pvals)
181
182 #Randomization MCP-Mod with covariates
183 results <- rep(0, nrand)
184 obvs <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)$tStat
185 for(j in 1:nrand){
186   datR <- get_randomised_data(obs_dat[, 1:3], randMethod=rand_prod, doses, B=10,
187                               N=50, n=rep(10,5))
188   results[j] <- max(MCTtest(dose, resp, data=datR, models = mods,
189                             addCovars = ~base)$tStat)
190 }
191 pvals <- rep(0,4)
192 for(i in 1:4){
193   pvals[i]<- sum(results- obvs[i]>= 0 )/nrand
194 }
195 pvalues_all[4] = min(pvals)
196
197 # Scenario (ii) Order the potential outcome data by "base"
198 dat_T <- dat[order(dat$base), ]
199 resp <- dose <- numeric(5*n_group)
200 for(i in 1:(5*n_group)){
201   dose[i] <- doses[Z[i]]
202   resp[i] <- dat_T[i,Z[i]]
203 }
204 obs_dat <- data.frame(dose = dose, resp = resp, base = dat_T$base)
205
206 #Population MCPMod without covariates

```

```

207 mm <- MCTtest(dose, resp, data=obs_dat, models = mods)
208 pvalues_all[5] = min(attr(mm$tStat, "pVal"))
209
210 #Population MCPMod with covariates
211 mm2 <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)
212 pvalues_all[6] = min(attr(mm2$tStat, "pVal"))
213
214 #Randomization MCP-Mod without covariates
215 results <- rep(0, nrand)
216 obvs <- MCTtest(dose, resp, data=obs_dat, models = mods)$tStat
217 for(j in 1:nrand){
218   datR <- get_randomised_data(obs_dat[, 1:2], randMethod=rand_prod, doses, B=10,
219                               N=50, n=rep(10,5))
220   results[j] <- max(MCTtest(dose, resp, data=datR, models = mods)$tStat)
221 }
222 pvals <- rep(0,4)
223 for(i in 1:4){
224   pvals[i]<- sum(results- obvs[i]>= 0 )/nrand
225 }
226 pvalues_all[7] = min(pvals)
227
228 #Randomization MCP-Mod with covariates
229 results <- rep(0, nrand)
230 obvs <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)$tStat
231 for(j in 1:nrand){
232   datR <- get_randomised_data(obs_dat[, 1:3], randMethod=rand_prod, doses, B=10,
233                               N=50, n=rep(10,5))
234   results[j] <- max(MCTtest(dose, resp, data=datR, models = mods,
235                             addCovars = ~base)$tStat)
236 }
237 pvals <- rep(0,4)
238 for(i in 1:4){
239   pvals[i]<- sum(results- obvs[i]>= 0 )/nrand
240 }
241 pvalues_all[8] = min(pvals)
242
243 ## (iii) Null hypothesis: Assume all potential outcomes are equal to Y(0)
244 dat_N <- dat
245 dat_N[,2:5] <- dat_N[,1]
246 resp <- dose <- numeric(5*n_group)
247 for(i in 1:(5*n_group)){
248   dose[i] <- doses[Z[i]]
249   resp[i] <- dat_N[i,Z[i]]
250 }
251 obs_dat <- data.frame(dose = dose, resp = resp, base = dat_N$base)
252
253 #Population MCPMod without covariates
254 mm <- MCTtest(dose, resp, data=obs_dat, models = mods)
255 pvalues_all[9] = min(attr(mm$tStat, "pVal"))
256
257 #Population MCPMod with covariates

```

```

258 mm2 <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)
259 pvalues_all[10] = min(attr(mm2$tStat, "pVal"))
260 results <- rep(0, nrand)
261
262 #Randomization MCP-Mod without covariates
263 obvs <- MCTtest(dose, resp, data=obs_dat, models = mods)$tStat
264 for(j in 1:nrand){
265   datR <- get_randomised_data(obs_dat[, 1:2], randMethod=rand_prod, doses, B=10,
266                               N=50, n=rep(10,5))
267   results[j] <- max(MCTtest(dose, resp, data=datR, models = mods)$tStat)
268 }
269 pvals <- rep(0,4)
270 for(i in 1:4){
271   pvals[i]<- sum(results- obvs[i]>= 0 )/nrand
272 }
273 pvalues_all[11] = min(pvals)
274
275 #Randomization MCP-Mod with covariates
276 results <- rep(0, nrand)
277 obvs <- MCTtest(dose, resp, data=obs_dat, models = mods, addCovars = ~base)$tStat
278 for(j in 1:nrand){
279   datR <- get_randomised_data(obs_dat[, 1:3], randMethod=rand_prod, doses, B=10,
280                               N=50, n=rep(10,5))
281   results[j] <- max(MCTtest(dose, resp, data=datR, models = mods,
282                             addCovars = ~base)$tStat)
283 }
284 pvals <- rep(0,4)
285 for(i in 1:4){
286   pvals[i]<- sum(results- obvs[i]>= 0 )/nrand
287 }
288 pvalues_all[12] = min(pvals)
289 return(pvalues_all)
290 }
291
292 grid <- expand.grid(
293   rand_prod = c("RA", "PBR") ,
294   KEEP.OUT.ATTRS = FALSE,
295   stringsAsFactors = FALSE)
296
297
298 for (i in 1:nrow(grid)) {
299   param_constatnt <- list(nrand=1000, rand_prod= grid[i,])
300
301   nsim = 10000
302   seeds <- data.frame(seed=seq(1,nsim))
303
304   sim_out <- Q_rows(fun = pharmaco,
305                     df = seeds,
306                     const = param_constatnt,
307                     seed = 23456,
308                     n_jobs = nrow(seeds),

```

```
309         template = list(  
310             walltime = 300,  
311             job_name = "Pharmaco",  
312             log_file = "logs.txt"),  
313         pkgs = c("data.table", "DoseFinding", "logistf"),  
314         export = export)  
315  
316     filename <- paste("simulation_pharmaco", grid[i,], param_constatnt$nrnd, "FULL",  
317                     sep="_")  
318     saveRDS(sim_out, file= filename)  
319  
320     sim_matrix_p_vals <- as.data.frame(t(do.call(rbind, sim_out)))  
321  
322     results <- cbind(TypeError_Power=apply(sim_matrix_p_vals, 1,  
323                                           proportion_less_than_005))  
324     print(results)  
325  
326     filename <- paste("simulation_pharmaco", grid[i,], param_constatnt$nrnd,  
327                     "RESULTS", sep="_")  
328     saveRDS(results, file= filename)  
329 }
```