pset 2 Labor

```
#===========================#
# ==== Metrics 675 ps 2 ====
#===========================#




#=======================================#
# ==== load packages and clear data ====
#=======================================#

library(data.table)
library(doParallel)
library(foreach)
library(ggplot2)
library(Matrix)

# clear data and consol
rm(list = ls(pos = ".GlobalEnv"), pos = ".GlobalEnv")
options(scipen = 999)
cat("\f")

# set options
opt_test_run <- TRUE

# set attributes for plot to default ea theme
plot_attributes <- theme( plot.background = element_rect(fill = "lightgrey"),
                          panel.grid.major.x = element_line(color = "gray90"),
                          panel.grid.minor  = element_blank(),
                          panel.background = element_rect(fill = "white", colour = "black") ,
                          panel.grid.major.y = element_line(color = "gray90"),
                          text = element_text(size= 20),
                          plot.title = element_text(vjust=0, colour = "#0B6357",face = "bold", size = 4

#=================================================#
# ==== Question 1: Kernel Density Estimation ====
#=================================================#

#=================#
# ==== Part a ====
#=================#


# now to find the theoretically optimal H I need to calculate integral of second derivative.
# second dericative of normal function is
phi_2 <- function(x, mean, v){

  dnorm(x=x,mean=mean,sd=sqrt(v))*(((x - mean)/v)^2-(1/v))

}
```

```r
# now create the function to integrate
f_int <- function(x){

  f_out <- (.5*phi_2(x=x, -1.5,1.5) + .5*phi_2(x=x, 1,1))^2
  return(f_out)
}

# and the integral is
v2k <- integrate(f_int, lower = -Inf, upper = Inf)$val

# so optimal bandwith is
h_opt <- (15/(v2k*1000))^(1/5)

#=================#
# ==== part b/d ====
#=================#

# set parms
n     <- 1000
M <- ifelse(opt_test_run, 10, 1000)


# kernal function
K0 <- function(u){

  out <- .75 * (1-u^2) * (abs(u) <= 1)
  return(out)
}

# define the true f(x) function
f_x <- function(x){

  .5*dnorm(x, -1.5,sqrt(1.5))+.5*dnorm(x,1,1)
}


#==============================#
# ==== Make imse function ====
#==============================#

# define variables for debug
# in_data <- r_dt
# x_v <- "rdraw"

# generate data for debugging functions
# start data.table for random data, take a random draw for weighted normals
# r_dt <- data.table( r1 = sample(1:2,prob=c(.5,.5),size=n,replace=T) )
#
# # draw a random number from appropriate normal dist according to r1
# r_dt[r1 == 1, rdraw := rnorm(.N,-1.5,1.5)]
# r_dt[r1 == 2, rdraw := rnorm(.N,1,1)]
# r_dt[, r1 := NULL]
# in_data <- r_dt
# h_v <- c(.5,.6)
# x_v <- "rdraw"
# i <- 1

imse_f <- function(in_data, x_v, h_v = NULL, f_x = f_x){

  # copy the data to aviod editing it in global enviorment
  data <- copy(in_data)

  # add a constant for the merge
  in_data[, const := 1]

  # cartesian merge to get all pairs
  paired_dt <- merge(in_data, in_data, by = "const", allow.cartesian = TRUE)

  # get new variable names after the merge. This kind of annoyingly general for a HW assingment. I regr
  x_vx <- paste0(x_v, ".x")
  x_vxi <- paste0(x_v, ".y")

  # initialize a list for output from each h
  ouput_list <- vector("list", length= length(h_v))

  # now do the imse calculations for each h in h_v
  for(i in 1:length(h_v)){

    h <- h_v[[i]]

    # get the kernal thing for each pair
    paired_dt[, k_x := K0((get(x_vxi) - get(x_vx))/h)]

    # now mean the kernal by rdraw.x and devide by h
    f_hats <- paired_dt[, list(f_hat_x = mean(k_x)/h), by = x_vx]

    # now get the f_hats for the leave one out by deleating the observation where x= xi. This will be r
    # 1, M+2, 2M+3, 3M+4 ... so eq(1, M*M, M+1) should take care of those
    paired_dt_lo <- paired_dt[-c(seq(1, n*n, n+1)), ]

    # now get the mean of the f_hats leacing out the x
    f_hats_lo <- paired_dt_lo[, list(f_hat_x = mean(k_x)/h), by = x_vx]

    # now add in f_x for each
    f_hats[, f_x := f_x(get(x_vx))]
    f_hats_lo[, f_x := f_x(get(x_vx))]

    # now do squared error
    f_hats[, sq_er := (f_hat_x - f_x)^2]
    f_hats_lo[, sq_er := (f_hat_x - f_x)^2]

    # now get imse
    imse_li <- f_hats[, mean(sq_er)]
    imse_lo <- f_hats_lo[, mean(sq_er)]

    # now put into a data.table and put in list
    ouput_list[[i]] <- data.table(imse_li = imse_li, imse_lo= imse_lo, h = h)
  }

  output <- rbindlist(ouput_list)

  return(output[])

}


#==========================#
# ==== run simulations ====
#==========================#

# note: pretty sure it would be faster yet to just include the simulations in the by group of the data
# operations in the IMSE function. Probably marginally faster but kind of hard to wrap my head around.
# update: I tried this an it exceeded R's vector length limit. Might be a workaround, unsure.

# define squared phi_2 function for part d
phi_2_sq <- function(x , mean, v){

  phi_2(x =x , mean = mean, v = v)^2
}


# now set up function to run simulations, make sure to pass in user defined functons/vars or foreach ca
sim_function <- function(i, n, f_x, phi_2, h_v){

  # generate data
  # start data.table for random data, take a random draw for weighted normals
  r_dt <- data.table( r1 = sample(1:2,prob=c(.5,.5),size=n,replace=T) )

  # draw a random number from appropriate normal dist according to r1
  r_dt[r1 == 1, rdraw := rnorm(.N,-1.5,1.5)]
  r_dt[r1 == 2, rdraw := rnorm(.N,1,1)]
  r_dt[, r1 := NULL]


  # get IMSE
  results_i <- imse_f(in_data = r_dt, x_v = "rdraw" ,f_x = f_x ,h_v =h_v)
  results_i[, sim := i]

  # now get mean and SE or part d
  mean_i <- r_dt[, mean(rdraw)]
  var_i <- r_dt[, var(rdraw)]

  # calculate "optimal bandwidth" under the procdure from part D
  vok <- 3/5
  u2k2 <- (1/5)^2

  # and the integral is
  v2phi <- integrate(phi_2_sq, mean = mean_i, v = var_i, lower = -Inf, upper = Inf)$val
  # now calculate h optimal
  h_opt <- (vok/ (u2k2 *v2phi*n))^(1/5)

  # put that bad boy in the table
  results_i[, d_h_hat := h_opt]

  # return the rsults for all of q2
  return(results_i[])

}


# make a vector of h's
h_v <- seq(.5, 1.5,.1)

# lets time this sucker
start_t <- Sys.time()


# parallel setup
cl <- makeCluster(4, type = "PSOCK")
registerDoParallel(cl)

# run simulations in parallel
output_list <- foreach(sim = 1 : M,
                       .inorder = FALSE,
                       .packages = "data.table",
                       .options.multicore = list(preschedule = FALSE, cleanup = 9)) %dopar% sim_functio

# stop clusters
stopCluster(cl)


# AND TIME
run_time1 <- Sys.time() - start_t

  # bind list
  output_dt <- rbindlist(output_list)

  # now take the mean of imse
  part_b_res <- output_dt[, list(imse_li = mean(imse_li), imse_lo = mean(imse_lo), d_h_hat = mean(d_h_ha

  # make them pretty
  part_b_res_pretty <- signif(part_b_res, 3)
  part_b_res_pretty[, colnames(part_b_res_pretty)] <- lapply(part_b_res_pretty[,colnames(part_b_res_pre

  # make the graph
  # melt the data to work better with ggplot
  part_b_res[, d_h_hat := NULL ]
  plot_data <-  melt.data.table(part_b_res, id.vars = "h", variable.name = "Error Type")
  plot_1_3_b <- ggplot(data = plot_data, aes(x = h, y = value, color = `Error Type`, shape = `Error Typ

  plot_1_3_b <- plot_1_3_b + geom_point() + geom_line() + plot_attributes
  plot_1_3_b




#====================#
# ==== save data ====
#====================#

  # only save data if this isn't a test run
  if(!opt_test_run){
    # save IMSE by h results
    print(xtable(part_b_res_pretty, type = "latex",
                 digits = 3),
          file = "C:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/Q1_p3_b.tex",
          include.rownames = FALSE,
          floating = FALSE)

    # save the plot
    png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_1_3_b.png", height = 800, wid
    print(plot_1_3_b)
    dev.off()

  }



  #====================#
  # ==== Question 2 ====
  #====================#


  #===========================#
  # ==== A: generate data ====
  #===========================#

    gen_data_2.5.a <- function(){

      # start data.table with random x's. get a chi squared too cause i need that for the epsilon
      r_dt <- data.table(x = runif(n,-1,1), chi_sq = rchisq(n,5))

      # create a noise clumn epsilon,
      r_dt[, eps := x^2*(chi_sq-5)]

      # now calcualte y
      r_dt[, y := exp(-0.1*(4*x-1)^2)*sin(5*x) + eps]

      # drop the chi_sq column
      r_dt[, chi_sq := NULL]

      # return the random data
      return(r_dt[])
    }


  #===========================#
  # ==== B do experiment ====
  #===========================#

  # generate some random data
  r_dt <- gen_data_2.5.a()


  # write a function to apply accross simulations
  power_s_fun <- function(sim = NULL){

    r_dt <- gen_data_2.5.a()
    r_dt[, const :=1]

    # store results in a list
    results <- vector("list", length = 20)
    # make the 20 squared variables
    #note: im makeing an extra column. Ill fix this if I have time but this is easy for now
    for(i in 1:20){

    r_dt[, temp := x^i]
    setnames(r_dt, "temp", paste0("x_exp_", i))


    # conver things to matrices to get the y hats
    x_mat <- as.matrix(r_dt[, c(grep("x_exp", colnames(r_dt), value = TRUE), "const"), with = FALSE])
    y_mat <- as.matrix(r_dt[, y])

    # get the projection matrix
    X.Q <- qr.Q(qr(x_mat))
    XX <- tcrossprod(X.Q)
    Y.hat <- XX %*% y_mat

    # now put this crap in a data.table to calculate cv
    res <- data.table(y_hat = Y.hat, w = diag(XX), y = r_dt[, y])

    # now calculate cv
    res[, cv_n := ((y - y_hat.V1)/(1-w))^2]

    # now get the mean of cv_i to get cv
    res <-  data.table(cv = res[, mean(cv_n)], k = i)
    setnames(res, "cv", paste0("cv_", sim))
    results[[i]] <- res

    }

    print(sim)
    # bind results
  return(rbindlist(results))
```

```
}

start_t <- Sys.time()

# parallel setup
cl <- makeCluster(4, type = "PSOCK")
registerDoParallel(cl)

# run simulations in parallel
all_out <- foreach(sim_i = 1 : M,
                       .inorder = FALSE,
                       .packages = "data.table",
                       .options.multicore = list(preschedule = FALSE, cleanup = 9)) %dopar% power_s_fu


# now merge all results
all_out_dt <-Reduce(function(x, y) merge(x, y, by = "k"), all_out)

# stop clusters
stopCluster(cl)

# check time
run_time2 <- Sys.time() - start_t

# row sum my data to get the averaage cv for each k
all_out_dt[, k := NULL]
mean_cv <- data.table( cv_means = rowMeans(all_out_dt), k = 1:20)

# now plot that bad boy

# initialize base data mapping for plot
plot_2_5_b <- ggplot(data = mean_cv, aes(x = k, y = cv_means))

plot_2_5_b <- plot_2_5_b + geom_point(size = 1) + geom_line() + plot_attributes
plot_2_5_b

#=================#
# ==== part c ====
#=================#


    # write a function to apply accross simulations
    B_fun <- function(sim = NULL){

      r_dt <- gen_data_2.5.a()
      r_dt[, const :=1]


      # make the y vars
      for(i in 1:7){

        r_dt[, temp := x^i]
        setnames(r_dt, "temp", paste0("x_exp_", i))
      }

        # conver things to matrices to get the y hats
        x_mat <- as.matrix(r_dt[, c(grep("x_exp", colnames(r_dt), value = TRUE), "const"), with = FALS
        y_mat <- as.matrix(r_dt[, y])

        # get betas
        B <- Matrix::solve(Matrix::crossprod(x_mat, x_mat))%*%(Matrix::crossprod(x_mat, y_mat))

        # get weights
        X.Q <- qr.Q(qr(x_mat))
        XX <- tcrossprod(X.Q)
        weights <- diag(XX)
        Y.hat <- XX %*% y_mat

        # now square the weights
        weights_sq <- weights^2

        # now get se
        se <- sqrt(sum(weights_sq) * var(y_mat - Y.hat))



        # put the stuff in a list
        output <- list()
        output[["B"]] <- B
        output[["se"]] <- se

      # return the betas
      return(output)

    }

    start_t <- Sys.time()

    # okay now run this shit 1000 times
   bw_stuff <- lapply(c(1:M), B_fun)

   run_time3 <- Sys.time() - start_t

   # now do some dumb stuff because its late
   b_list <- list()
   se_list <- list()
   for(i in 1:M){

     b_list[[i]] <- bw_stuff[[i]][["B"]]
     se_list[[i]] <- bw_stuff[[i]][["se"]]
   }

   b_mat <- do.call(cbind, b_list)
   se_mat <- do.call(cbind, se_list)

   # sum the rows
```

```
  betas <- rowMeans(b_mat)
  se <-   rowMeans(se_mat)

  # now write a function to plot the u hat funciton
  u_hat_fun <- function(x){

    betas[[8]] + betas[[1]]*x + betas[[2]]*x^2 + betas[[3]]*x^3 + betas[[4]]*x^4 + betas[[5]]*x^5 +
  }


  # write out true function
  true_fun <- function(x){

    exp(-0.1*(4*x-1)^2)*sin(5*x)

  }


 #======================#
 # ==== part c plot ====
 #======================#


  # plot the true functin
  plot_2_5_c <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
  plot_2_5_c <- plot_2_5_c + stat_function(fun = true_fun,
                                           color = "blue")
  plot_2_5_c <- plot_2_5_c + plot_attributes + xlim(-1,1)


  # now add u hat function
  plot_2_5_c <- plot_2_5_c + stat_function(fun = u_hat_fun,
                                             color = "red", linetype = 2)

  plot_2_5_c <- plot_2_5_c + scale_colour_identity("Function", guide="legend",
                                                    labels = c("U hat", "True U"),
                                                    breaks = c("red", "blue")) + theme(axis.title.y=element_
  # create some data to plot with the standard errors
  plot_data <- data.table(x = seq(-1,1,.2))
  plot_data[, y_hat := u_hat_fun(x)]
  plot_data[, se := se]

  plot_2_5_c <- plot_2_5_c + geom_point(data = plot_data, mapping = aes(x = x, y = y_hat),
                                           color = "red")

  plot_2_5_c <- plot_2_5_c + geom_errorbar(data = plot_data, aes(ymin=y_hat-se, ymax=y_hat+se), wid

  # print it out to see if it looks alright
  plot_2_5_c

#======================#
# ==== part d pot ====
#======================#
```

```
    # create derivative funciton
    # write out true function
    true_fun_d <- function(x){

      exp(-0.1*(4*x-1)^2)*(5*cos(5*x) - 0.8*(4*x-1)*sin(5*x))

    }

    # write out estimated polynomial
    est_fun_d <- function(x){
     betas[[1]] + 2*betas[[2]]*x + 3*betas[[3]]*x^2 + 4*betas[[4]]*x^3 + 5*betas[[5]]*x^4 + 6*betas[[(

    }

    # plot the true functin
    plot_2_5_d <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
    plot_2_5_d <- plot_2_5_d + stat_function(fun = true_fun_d,
                                              color = "blue")
    plot_2_5_d <- plot_2_5_d + plot_attributes + xlim(-1,1)


    # now add u hat function
    plot_2_5_d <- plot_2_5_d + stat_function(fun = est_fun_d,
                                              color = "red", linetype = 2)

    plot_2_5_d <- plot_2_5_d + scale_colour_identity("Function", guide="legend",
                                                      labels = c("U hat", "True U"),
                                                      breaks = c("red", "blue")) + theme(axis.title.y=e
    # create some data to plot with the standard errors
    plot_data <- data.table(x = seq(-1,1,.2))
    plot_data[, y_hat := est_fun_d(x)]
    plot_data[, se := se]

    plot_2_5_d <- plot_2_5_d + geom_point(data = plot_data, mapping = aes(x = x, y = y_hat),
                                           color = "red")

    plot_2_5_d <- plot_2_5_d + geom_errorbar(data = plot_data, aes(ymin=y_hat-se, ymax=y_hat+se), widt

    # print it out to see if it looks alright
    plot_2_5_d

#======================#
# ==== save plots ====
#======================#

    # only save data if this isn't a test run
    if(!opt_test_run){

      # save the plot
      png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_2_5_b.png", height = 800,
      print(plot_2_5_b)
      dev.off()
```

```
        # save the plot
        png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_2_5_c.png", height = 800,
        print(plot_2_5_c)
        dev.off()

        # save the plot
        png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_2_5_d.png", height = 800,
        print(plot_2_5_d)
        dev.off()



      }



#=====================#
# ==== question 3 ====
#=====================#

  #=================#
  # ==== Part a ====
  #=================#

    d   = 5
    theta_n = 1


    data_gen <- function(n) {
      X <- matrix(runif(n*d,-1,1), n, d)
      V <- rnorm(n)
      x.norm = sapply(1:n,function(i) t(X[i,])%*%X[i,])
      E      = 0.3637899*(1+x.norm)*V
      g0.x   =exp(x.norm)

      U <- rnorm(n)
      tt <- as.numeric((sqrt(x.norm)+U)>1)
      Y <- tt + g0.x + E
      return(list(Y=Y, X=X, tt=tt))
    }


    # generate the polynomial basis
    gen.P = function(Z,K) {
      if (K==0)   out = NULL;
      if (K==1)   out = poly(Z,degree=1,raw=TRUE);
      if (K==2)  {out = poly(Z,degree=1,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^2);}
      if (K==2.5) out = poly(Z,degree=2,raw=TRUE);
      if (K==3)  {out = poly(Z,degree=2,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^3);}
      if (K==3.5) out = poly(Z,degree=3,raw=TRUE);
      if (K==4)  {out = poly(Z,degree=3,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^4);}
      if (K==4.5) out = poly(Z,degree=4,raw=TRUE);
      if (K==5)  {out = poly(Z,degree=4,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^5);}
      if (K==5.5) out = poly(Z,degree=5,raw=TRUE);
      if (K>=6)  {out = poly(Z,degree=5,raw=TRUE); for (k in 6:K) for (j in 1:ncol(Z)) out = cbind(out,
      ## RETURN POLYNOMIAL BASIS
      return(out)
    }

  #==================#
  # ==== part b ====
  #==================#

    n   <- 500
    K   <- c(1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 7, 8, 9, 10)
    K.r <- c(6, 11, 21, 26, 56, 61, 126, 131, 252, 257, 262, 267, 272, 277)
    nK  <- length(K)
    M <- ifelse(opt_test_run, 10, 1000)
    theta.hat <- matrix(NaN, ncol=nK, nrow=M)
    se.hat    <- theta.hat

    set.seed(123)
    ptm <- proc.time()
    for (m in 1:M) {
      data <- data_gen(n)
      X <- data$X
      Y <- data$Y
      tt <- data$tt
      for (k in 1:nK) {
        X.pol <- cbind(1, gen.P(X, K[k]))
        X.Q   <- qr.Q(qr(X.pol))
        MP     <- diag(rep(1,n)) - X.Q %*% t(X.Q)
        Y.M <- MP %*% Y
        tt.M <- MP %*% tt
        theta.hat[m, k] <- (t(tt.M) %*% Y.M) / (t(tt.M) %*% tt.M)
        Sigma <- diag((as.numeric((Y.M - tt.M*theta.hat[m, k])))^2)
        se.hat[m, k] <- sqrt(t(tt.M) %*% Sigma %*% tt.M) / (t(tt.M) %*% tt.M)
      }
    }
    proc.time() - ptm

    table <- matrix(NaN, ncol=6, nrow=nK)
    for (k in 1:nK) {
      table[k, 1] <- K.r[k]
      table[k, 2] <- mean(theta.hat[, k]) - 1                          # bias
      table[k, 3] <- sd(theta.hat[, k])                                # standard deviation
      table[k, 4] <- table[k, 2]^2 + table[k, 3]^2                     # mse
      table[k, 5] <- mean(se.hat[, k])                                 # mean standard error
      table[k, 6] <- mean((theta.hat[, k] - 1.96 * se.hat[, k] > 1) |
                            (theta.hat[, k] + 1.96 * se.hat[, k] < 1))   # rejection rate
    }

    table <- data.table(table)
    setnames(table, colnames(table), c("K", "Theta", "Bias", "S.D", "V_HCO", "Rejection rate"))

#======================#
```

```
# ==== save table ====
#=====================#

    # save IMSE by h results
    print(xtable(table, type = "latex",
                 digits = 3),
          file = "C:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/Q3_4_b.tex",
          include.rownames = FALSE,
          floating = FALSE)


#=====================#
# ==== Q3. 4. (c) ====
#=====================#




    # cross validation function
    K.CV <- function(tt, X, Y) {
      temp <- rep(NaN, nK)
      for (k in 1:nK) {
        X.pol <- cbind(1, tt, gen.P(X, K[k]))
        X.Q   <- qr.Q(qr(X.pol))
        XX <- X.Q %*% t(X.Q)
        Y.hat <- XX %*% Y
        W <- diag(XX)
        temp[k] <- mean(((Y-Y.hat) / (1-W))^2)
      }
      return(which.min(temp))
    }

    theta.hat2 <- rep(NaN, M)
    se.hat2    <- theta.hat2
    K.hat2     <- theta.hat2

    set.seed(123)
    ptm <- proc.time()
    for (m in 1:M) {
      data <- data_gen(n)
      X <- data$X; Y <- data$Y; tt <- data$tt
      k.opt <- K.CV(tt, X, Y)
      X.pol <- cbind(1, gen.P(X, K[k.opt]))
      X.Q   <- qr.Q(qr(X.pol))
      MP    <- diag(rep(1,n)) - X.Q %*% t(X.Q)
      Y.M   <- MP %*% Y
      tt.M  <- MP %*% tt
      theta.hat2[m] <- (t(tt.M) %*% Y.M) / (t(tt.M) %*% tt.M)
      Sigma         <- diag((as.numeric((Y.M - tt.M*theta.hat[m, k])))^2)
      se.hat2[m]    <- sqrt(t(tt.M) %*% Sigma %*% tt.M) / (t(tt.M) %*% tt.M)
      K.hat2[m]     <- K.r[k.opt]
    }
    time4 <- proc.time() - ptm
```

```r
# summary of the cross validation
table(K.hat2)

# estimator
summary(theta.hat2)
sd(theta.hat2)
summary(se.hat2)
sd(se.hat2)

par(mfrow=c(1,2))
hist(theta.hat2, freq=FALSE, xlab="theta-hat", ylab="", main="")
lines(c(mean(theta.hat2), mean(theta.hat2)), c(-1, 20), col="red", lwd=3)
hist(se.hat2, freq=FALSE, xlab="s.e.", ylab="", main="")
lines(c(mean(se.hat2), mean(se.hat2)), c(-1, 80), col="red", lwd=3)


par(mfrow=c(1,2))
CI.l <- theta.hat2 - 1.96 * se.hat2
CI.r <- theta.hat2 + 1.96 * se.hat2

# rejection rate
mean(1 < CI.l | 1 > CI.r)
plot(1:M, CI.l, type="l", ylim=c(0,2), xlab="simulations", ylab="CI")
lines(1:M, CI.r)
abline(1, 0, col="red", lwd=2)

temp <- sort(CI.l, index.return=TRUE)
CI.l <- temp$x
CI.r <- CI.r[temp$ix]
plot(1:M, CI.l, type="l", ylim=c(0,2), xlab="simulations", ylab="CI")
lines(1:M, CI.r)
abline(1, 0, col="red", lwd=2)
```