Appendix D R code

In this appendix we provide the essential R code to help the reader to reproduce our empirical results or adapt the code for their own applications. The latest updates to the code and the technical documentation are available at GitHub: https://github.com/Xun90/SCM-Debug.git. We assume the reader is familiar with the *Synth* R package (see Abadie et al., 2011 for an introduction), and suggest the use of the dataprep() function provided in *Synth* to pre-process the data.

Step 1: Load necessary R packages.

```
library("Synth")      #Synth package
#The two QP solvers used by "Synth" are employed here for a direct comparison with "Synth"
library(kernlab)      #QP solver 1: ipop
library(LowRankQP)    #QP solver 2: LowRankQP, whose results are reported in this study
library(lpSolve)      #LP solver
library(matrixcalc)   #for matrix calculations
```

Step 2: Re-examine the *Synth* results for the California tobacco control application with 1,000 random reorderings of predictors.

```
##loop on 1000 random orders
lossV <- matrix(0, 1000, 1)
lossW <- matrix(0, 1000, 1)
W <- matrix(0, 38, 1000)
V <- matrix(0, 7, 1000)
C <- matrix(0, 7, 1000)
set.seed(42)
for (i in 1:1000){
  row <- sample(nrow(X0))
  C[,i] <- row
  dataprep.out$X0 <- X0[row,]
  dataprep.out$X1 <- as.matrix(X1[row,])
  synth.out <- synth(data.prep.obj = dataprep.out, method = "BFGS")
  lossV[i,] <- synth.out$loss.v
  lossW[i,] <- synth.out$loss.w
  W[,i] <- synth.out$solution.w
  sorted <- cbind(row, t(synth.out$solution.v))
  sorted <- sorted[order(sorted[,"row"]),]
  V[,i] <- sorted[,2]}
```

Step 3: Re-examine the *Synth* results for the California tobacco control application with 1,000 random reorderings of donors.

```
##loop on 1000 random orders
lossV <- matrix(0, 1000, 1)
lossW <- matrix(0, 1000, 1)
W <- matrix(0, 38, 1000)
V <- matrix(0, 7, 1000)
C <- matrix(0, 38, 1000)
set.seed(42)
for (i in 1:1000){
  column <- sample(ncol(X0))
  C[,i] <- column
  dataprep.out$X0 <- X0[,column]
```

```
  dataprep.out$Z0 <- Y0pre[,column]
  synth.out <- synth(data.prep.obj = dataprep.out, method = "BFGS")
  lossV[i,] <- synth.out$loss.v
  lossW[i,] <- synth.out$loss.w
  V[,i] <- t(synth.out$solution.v)
  sorted <- cbind(column, synth.out$solution.w)
  sorted <- sorted[order(sorted[,"column"]),]
  W[,i] <- sorted[,2]}
```

Step 4: Implement the iterative algorithm proposed by Malo et al. (2020) to check for the feasibility of the unconstrained optimum and the possibility of corner solutions.

```
scm.corner <- function(Y1pre,Y0pre,X1,X0){
  ##step1
  Tpre <- dim(Y0pre)[1]
  nDonors <- dim(Y0pre)[2]
  #QP setup
  c1 <- -t(Y0pre) %*% Y1pre
  H1 <- t(Y0pre) %*% Y0pre
  A1 <- matrix(rep(1,nDonors), ncol = nDonors)
  b1 <- 1
  r1 <- 0
  l1 <- matrix(rep(0,nDonors), nrow = nDonors)
  u1 <- matrix(rep(1,nDonors), nrow = nDonors)
  #run QP
  step1_ipop <- ipop(c = c1, H = H1, A = A1, b = b1, l = l1, u = u1, r = r1,
                     margin = 0.0005, maxiter = 1000, sigf = 7, bound = 10) #QP_Solver1
  step1_lowr <- LowRankQP(Vmat = H1, dvec = c1, Amat = A1, bvec = b1, uvec = u1,
                          method = "LU") #QP_Solver2
  W_ipop <- matrix(step1_ipop@primal, nrow = nDonors)
  W_lowr <- step1_lowr$alpha
  L1_ipop <- (t(Y1pre) %*% Y1pre)/Tpre + 2/Tpre * (t(c1) %*% W_ipop
                                                   + 0.5 * t(W_ipop) %*% H1 %*% W_ipop)
  L1_lowr <- (t(Y1pre) %*% Y1pre)/Tpre + 2/Tpre * (t(c1) %*% W_lowr
                                                   + 0.5 * t(W_lowr) %*% H1 %*% W_lowr)
  ##step2
  #normalize X - Synth
  nvarsV <- dim(X0)[1]
  big.dataframe <- cbind(X0, X1)
  divisor <- sqrt(apply(big.dataframe, 1, var))
  scaled.matrix <- t(t(big.dataframe) %*% ( 1/(divisor)
                                            * diag(rep(dim(big.dataframe)[1], 1)) ))
  X0.scaled <- scaled.matrix[,c(1:(dim(X0)[2]))]
  if(is.vector(X0.scaled)==TRUE)
  {X0.scaled <- t(as.matrix(X0.scaled))}
  X1.scaled <- scaled.matrix[,dim(scaled.matrix)[2]]
  #LP setup
  f.obj_ipop <- (X1.scaled - X0.scaled %*% W_ipop)^2
  f.obj_lowr <- (X1.scaled - X0.scaled %*% W_lowr)^2
  f.con <- rbind(rep(1,nvarsV), diag(x = 1, nrow = nvarsV))
  f.dir <- c("=", rep(">=",nvarsV))
  f.rhs <- c(1, rep(0,nvarsV))
  #run LP
  step2_ipop <- lp ("min", f.obj_ipop, f.con, f.dir, f.rhs)
  step2_lowr <- lp ("min", f.obj_lowr, f.con, f.dir, f.rhs)
  V_ipop <- step2_ipop$solution
  V_lowr <- step2_lowr$solution
  L2_ipop <- step2_ipop$objval
  L2_lowr <- step2_lowr$objval

  scm.corner.out <- list(W = cbind(W_ipop,W_lowr), V = cbind(V_ipop,V_lowr),
                         Lv = c(L1_ipop,L1_lowr), Lw = c(L2_ipop,L2_lowr))
  return(scm.corner.out)}
```

Step 5: Implement the two-step procedure described in Section 4.1. This implementation is currently a hybrid of Section 4.1 and Malo et al. (2020). Since there are currently no reliable solvers in R for the second-stage convex programming problem, we solve the non-Archimedean problem (8) iteratively, decreasing ε towards zero until the objective function reaches the optimal solution of the first-stage QP problem.

```
two.step.iterative <- function(Y1pre,Y0pre,X1.scaled,X0.scaled,SV){
  #SV - predictor weights defined by the user
  ##Solve non-Archimedean problem (8)
  Tpre <- dim(Y0pre)[1]
  nDonors <- dim(Y0pre)[2]
  #QP setup
  A <- matrix(rep(1,nDonors), ncol = nDonors)
  b <- 1
  r <- 0
  l <- matrix(rep(0,nDonors), nrow = nDonors)
  u <- matrix(rep(1,nDonors), nrow = nDonors)
  #Loop on 10 epsilon values (0.1^1 ... 0.1^10) to find the best performer
  L_upper = matrix(0, 10, 2)
  L_lower = matrix(0, 10, 2)
  W_ipop = matrix(0, nDonors, 10)
  W_lowr = matrix(0, nDonors, 10)
  for (i in 1:10){
    eps <- 0.1^(i) #epsilon - penalty term
    c <- (-t(X0.scaled) %*% diag(SV1) %*% X1.scaled) - eps * t(Y0pre) %*% Y1pre
    H <- t(X0.scaled) %*% diag(SV1) %*% X0.scaled + eps * t(Y0pre) %*% Y0pre
    #run QP
    QP_ipop <- ipop(c = c, H = H, A = A, b = b, l = l, u = u, r = r,
                    margin = 0.0005, maxiter = 1000, sigf = 7, bound = 10) #QP_Solver1
    QP_lowr <- LowRankQP(Vmat = H, dvec = c, Amat = A, bvec = b, uvec = u,
                         method = "LU") #QP_Solver2
    W_ipop[,i] <- matrix(QP_ipop@primal, nrow = nDonors)
    W_lowr[,i] <- QP_lowr$alpha
    L_upper[i,1] <- 1/Tpre * t(Y1pre - Y0pre %*% W_ipop[,i]) %*%
      (Y1pre - Y0pre %*% W_ipop[,i])
    L_upper[i,2] <- 1/Tpre * t(Y1pre - Y0pre %*% W_lowr[,i]) %*%
      (Y1pre - Y0pre %*% W_lowr[,i])
    L_lower[i,1] <- t(X1.scaled - X0.scaled %*% W_ipop[,i]) %*% diag(SV1) %*%
      (X1.scaled - X0.scaled %*% W_ipop[,i])
    L_lower[i,2] <- t(X1.scaled - X0.scaled %*% W_lowr[,i]) %*% diag(SV1) %*%
      (X1.scaled - X0.scaled %*% W_lowr[,i])}
  ##Use the first step of the two-step procedure in Section 4.1 to determine epsilon
  c1 <- (-t(X0.scaled) %*% diag(SV) %*% X1.scaled)
  H1 <- t(X0.scaled) %*% diag(SV) %*% X0.scaled
  #run QP
  QP_ipop1 <- ipop(c = c1, H = H1, A = A, b = b, l = l, u = u, r = r,
                   margin = 0.0005, maxiter = 1000, sigf = 7, bound = 10) #QP_Solver1
  QP_lowr1 <- LowRankQP(Vmat = H1, dvec = c1, Amat = A, bvec = b, uvec = u,
                        method = "LU") #QP_Solver2
  W_ipop1 <- matrix(QP_ipop1@primal, nrow = nDonors)
  W_lowr1 <- QP_lowr1$alpha
  W1 <- cbind(W_ipop1, W_lowr1)
  obj_left <- t(X1.scaled) %*% diag(SV) %*% X1.scaled
  Lw_ipop1 <- obj_left + 2 * (t(c1) %*% W_ipop1 + 1/2 * t(W_ipop1) %*% H1 %*% W_ipop1)
  Lw_lowr1 <- obj_left + 2 * (t(c1) %*% W_lowr1 + 1/2 * t(W_lowr1) %*% H1 %*% W_lowr1)
  Lw1 <- c(Lw_ipop1, Lw_lowr1)

  two.step.iterative.out <- list(W = cbind(W_ipop,W_lowr), W1 = W1, V = SV,
                           L_upper = L_upper, L_lower = L_lower, Lw1 = Lw1)
  return(two.step.iterative.out)}
```