pset 1 631

```r
#==============================================#
# ==== Inustrial Organization Problem Set 1 ====
#==============================================#


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


# Load packages
library(data.table)
library(stats4)
library(broom)
library(AER)
library(xtable)
library(Matrix)
library(BLPestimatoR)
library(SQUAREM)
library(BB)

# set save option

opt_save <- TRUE

# set path for output
f_out <- "C:/Users/Nmath_000/Documents/Code/courses/econ 631/ps1/tex/"

#=====================#
# ==== Question 1 ====
#=====================#

# laod data
q1dt <- fread("file:///C:/Users/Nmath_000/Documents/MI_school/Third Year/Econ 631/ps1/ps1.dat")



# create variable names
setnames(q1dt, colnames(q1dt), c("y", "x1", "x2", "z"))


#=================#
# ==== part 1 ====
#=================#

# write log likelihood function
Probit_llf <- function(th0, th1, th2){
```

```r
  mu <- q1dt[,  pnorm(th0 + th1*x1 + th2*x2)]

  -sum(q1dt[, y*log(mu) + (1-y)*log(1-mu)])
}

# create starting values
prob_start <- list(th0 = 0,
                   th1 = .01,
                   th2 = .01)

# run the mle function
probit_res <- mle(Probit_llf, start = prob_start)

# get the results I need
probit_res <- data.table(variable = rownames(summary(probit_res)@coef),
                         summary(probit_res)@coef)

# check my results
check <- glm( y ~ x1+ x2,
              family = binomial(link = "probit"),
              data = q1dt)

# looks good
summary(check)
probit_res

#================#
# ==== Part2 ====
#================#

probit_res$Estimate[3]*dnorm(probit_res$Estimate[1] + probit_res$Estimate[2]*mean(q1dt$x1) + probit_res$

#================#
# ==== Part3 ====
#================#

# define logit log likelihood
logit_llf <- function(th0, th1, th2){


  mu <- q1dt[,  plogis(th0 + th1*x1 + th2*x2)]

  -sum(q1dt[, y*log(mu) + (1-y)*log(1-mu)])
}

# create startign values
logit_start <- list(th0 = 0,
                    th1 = .01,
                    th2 = .01)

# run the mle function
logit_res <- mle(logit_llf, start = logit_start)
```

```r
# get the results I need
logit_res <- data.table(variable = rownames(summary(logit_res)@coef),
                        summary(logit_res)@coef)

# check my results
check <- glm( y ~ x1+ x2,
              family = binomial(link = "logit"),
              data = q1dt)
# looks good
summary(check)
logit_res

#==================#
# ==== Part 6 ====
#==================#


p6_llf <- function(th0, th1, th2, th3, th4, th5, p, sig){

  # define some intermediate terms
  q1dt[, psi := (p/sig^2)*(x2 - (th3 + th4*x1 + th5*z))]
  q1dt[, tau := 1-(p^2/sig^2)]
  q1dt[, m:= (-th0 - th1*x1 - th2*x2 - psi)/(tau^.5)]

  # get prob y equals one conditional on x1, x2, z, theta
  q1dt[, p_y_1 := 1- pnorm(m)]

  # get pro x2 = x2i given x1 z theta
  q1dt[, p_x := dnorm((x2 - th3 - th4*x1 - th5*z)/sig)]

  # get log likelihoods
  q1dt[, llf := y*log(p_y_1) + (1-y)*log(1-p_y_1) + log(p_x) - log(sig)]

  # sum them
  sum_llf <- q1dt[, sum(llf)]

  # remove extra vars
  q1dt[ ,`:=` (psi = NULL, tau = NULL, m = NULL, p_y_1 = NULL, p_x = NULL, llf = NULL)]

  #  return negative sum
  return(-sum_llf)
}

second_stage <- glm(y ~ x1+ x2,
                    family = binomial(link = "probit"),
                    data = q1dt)

first_stage <- lm(x2 ~ x1 + z, data = q1dt)

p6_start <- as.list(c(second_stage$coefficients, first_stage$coefficients, .1, 1))

names(p6_start) <- c( paste0("th", 0:5), "p", "sig")
```

```r
# run the mle function
p6_res <- mle(p6_llf, start = p6_start)

logLik(p6_res)
# get the results I need
p6_res_tab <- data.table(variable = rownames(summary(p6_res)@coef),
                         summary(p6_res)@coef)

p6_res_tab




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

  #=================#
  # ==== part 3 ====
  #=================#


    # load cereal data
    cereal <- data.table(readxl::read_excel("C:/Users/Nmath_000/Documents/MI_school/Third Year/Econ 631/


    # get total market share by city year
    cereal[, s0 := 1-sum(share), c("city", "year", "quarter" )]


    # create column for mean utility
    cereal[, m_u := log(share) - log(s0)]

    #============#
    # ==== a ====
    #============#

    # run ols, tidy it up, make it a data.table
    q2_p3_ols <- data.table(tidy(lm(m_u ~ mushy + sugar + price , data = cereal)))

    # round p value
    q2_p3_ols[, p.value := round(p.value, 6)]

    #============#
    # ==== b ====
    #============#


    # create  sugar instroment
    cereal[, (.N-1), c('firm_id', "city", "quarter")]
    cereal[, i1_sugar := (sum(sugar) - sugar)/ (.N-1), c('firm_id', "city", "quarter")]

    # create mush intrument
```

```r
    cereal[, i1_mushy := (sum(mushy) - mushy)/(.N-1), c('firm_id', "city", "quarter")]

    # create price instrument
    cereal[, i1_price := (sum(price) - price)/ (.N-1), c('firm_id', "city", "quarter")]

    # now do 2sls, tidy it up, make it a data.table
    q2_p3_iv1 <- data.table(tidy(ivreg(m_u ~ mushy + sugar + price | i1_sugar + i1_mushy +  mushy + suga

    q2_p3_iv1[, p.value := round(p.value,6)]


    #============#
    # ==== c ====
    #============#

    # now create second set of instroments
    # would be ideal to write functino for this, but who has the time
    # create  sugar instroment
    # start by getting sum of sugar for firm
    cereal[, f_sugar_sum := sum(sugar), c( "city", "quarter", "firm_id")]

    # get number of products by firm
    cereal[, f_nprod := .N, c( "city", "quarter", "firm_id")]

    # get sum of sugar in market, subtract off sum of sugar for the firm
    # divide by number of products minus the products for this firm that we subtracted off
    cereal[, i2_sugar := (sum(sugar) - f_sugar_sum)/ (.N-f_nprod), c("city", "quarter")]

    # now do the same for the other
    cereal[, f_mushy_sum := sum(mushy), c( "city", "quarter", "firm_id")]
    cereal[, i2_mushy := (sum(mushy) - f_mushy_sum)/ (.N-f_nprod), c("city", "quarter")]

    # #note dont actually need this
    # cereal[, f_price_sum := sum(price), c( "city", "quarter", "firm_id")]
    # cereal[, i2_price := (sum(price) - f_price_sum)/ (.N-f_nprod), c("city", "quarter")]

    # now do 2sls
    ivreg_out <- ivreg(m_u ~ mushy + sugar + price | i2_sugar + i2_mushy + mushy + sugar , data = cereal
    q2_p3_iv2 <- data.table(tidy(ivreg(m_u ~ mushy + sugar + price | i2_sugar + i2_mushy + mushy + sugar

    q2_p3_iv2[, p.value := round(p.value,6)]




#=====================#
# ==== Question 3 ====
#=====================#

  #=====================================#
  # ==== set up data for functions ====
  #=====================================#
```

```r
# load cereal data
cereal <- data.table(readxl::read_excel("C:/Users/Nmath_000/Documents/MI_school/Third Year/Econ 631/

#note these are data manipulations that can happen outside the outer loop. We
# just need to convert some things to matrices since we actually need to
# use matrix algebra in this question



# create a single market variale. Don't have to referece two variables then
cereal[, mkt := paste0(city, "_", quarter)]

# get total market share by city year
#note: should we do this before or after dropping obs with missing instruments?
cereal[, s0 := 1-sum(share), mkt]

# create column for mean utility
cereal[, m_u := log(share) - log(s0)]

# create  sugar instrument
cereal[, (.N-1), c('firm_id', "city", "quarter")]
cereal[, i1_sugar := (sum(sugar) - sugar)/ (.N-1), c('firm_id', "city", "quarter")]

# create mush instrument
cereal[, i1_mushy := (sum(mushy) - mushy)/(.N-1), c('firm_id', "city", "quarter")]

# drop observations with missing instruments. Missing becasue firm only has one product
cereal <- cereal[!is.na(i1_sugar)]

# defint some variables
# I'm not crazy about doing this. I think this either needs to be written as a function
# or else we should just hard code the column names. This is like a weird middle ground
# of generalizability that isn't that helpful
share.fld =     "share"
prod.id.fld =   "product_id"
mkt.id.fld =    "mkt"
prc.fld =       "price"
x.var.flds =    c("sugar",
                  "mushy")

# order data and get nrows
cereal <- setorder(cereal, "city", "product_id")
JM <- nrow(cereal)

# make matrix of controls X
X <- as.matrix(cereal[, c(x.var.flds), with = FALSE])
K <- ncol(X)

# make a matrix that includes price
cereal[, n_price := -price]
XP <- as.matrix(cereal[, c("sugar", "mushy", "n_price"), with = FALSE])

# make a vector for price
```

```r
  P <- as.matrix(cereal[, price])

  # put market into a vector
  mkt.id <- cereal[, get(mkt.id.fld)]

  # put shares into a vector
  s.jm <- as.vector(cereal[, get(share.fld)]);

  # put shates into a vector
  s.j0 <- cereal[, s0 ]

  # set number of simulated consumers
  n.sim = 20

  # Standard normal distribution draws, one for each characteristic in X
  # columns are simulated consumers, rows are variables in X (including constant and price)
  # as Ying pointed out we want to do this once outside the outer loop to get faster convergence
  v = matrix(rnorm(K * n.sim), nrow = K, ncol = n.sim)

  # Z matrix needed for gmm function
  Z <- as.matrix(cereal[!is.na(i1_sugar),c("sugar", "mushy", "i1_sugar", "i1_mushy"), with = FALSE])

  # PZ matrix needed for GMM function
  PZ <- Z %*% solve(t(Z) %*% Z) %*% t(Z)

  # make weighting matrix. Also used in GMM function
  W.inv <- diag(1, 4, 4 )

#================================#
# ==== functions for search ====
#================================#

  # these are all the functions we will need for a given sd_mush sd_sug guess

  # hard code estimates for variance paremter to test funcitons
  # this is what the outside loop will be optimizing over
  sd_mush_test <- .1
  sd_sug_test <- .2

  # get inital delta guess
  #note doesn't need to come from cereal data.table, this is just an initial guess
  delta.initial <- cereal[, m_u]

  # this function takes the data.table, V sims,  and the sd parameter estimates and returns
  # the mu (individual utility) estimates
  #note I am hard coding this for mushy and sugar as the relavent variables.
  # could write this in a more general way if we wanted and have it take a list of theta parms
  # and a list of corresponding variabe names
  find_mu <- function(X, sd_sug.in,  sd_mush.in, v.in){

     X %*% diag(x = c(sd_sug.in, sd_mush.in ), 2, 2) %*%v.in
  }
```

```r
# test it out
mu_mat <- find_mu(X, sd_sug_test, sd_mush_test, v )



# This computes a matrix of share estiamtes. rows are estimates for every product
# columns are estimates for every simulated V
ind_sh <- function(delta.in, mu.in, mkt.id.in){
  # This function computes the "individual" probabilities of choosing each brand
  numer <- exp(mu.in) * matrix(rep(exp(delta.in), n.sim), ncol = n.sim)

  denom <- as.matrix(do.call("rbind", lapply(mkt.id.in, function(tt){
    1 + colSums(numer[mkt.id.in %in% tt, ])

  })))
  return(numer / denom);
}

# test it out
sj_mat <- ind_sh(delta.initial, mu_mat, mkt.id)


blp_inner <- function(delta.in, mu.in, s.jm.in, mkt.id.in) {
  # Computes a single update of the BLP (1995) contraction mapping.
  # of market level predicted shares.
  # This single-update function is required by SQUAREM, see Varadhan and
  # Roland (SJS, 2008), and Roland and Varadhan (ANM, 2005)
  # INPUT
  #   delta.in : current value of delta vector
  #   mu.in: current mu matrix
  # OUTPUT
  #   delta.out : delta vector that equates observed with predicted market shares
  pred.s <- rowMeans(ind_sh(delta.in, mu.in, mkt.id.in));
  delta.out <- delta.in + log(s.jm.in) - log(pred.s)
  return(delta.out)
}

# test it out
delta_new <- blp_inner(delta.initial, mu_mat, s.jm, mkt.id)


# The function to iterate the contraction mapping is genereic. WE don't need to write it
# here is an example #note it is dependent on output of above examples
#note I have to pass additonal variables so that we are not pulling objects from global enviorment
  squarem.output <- squarem(par       = delta.initial,
                            fixptfn   = blp_inner,
                            mu.in     = mu_mat,
                            s.jm.in   = s.jm,
                            mkt.id.in = mkt.id,
                            control   = list(trace = TRUE))
  delta <- squarem.output$par
  summary(delta.initial - delta)
```

```r
# function to runn GMM
# a lot of inputs here but this is how you get around using global objects
# THis is supposed to be better practice but it doe sget a bit wild with all these
gmm_obj_f <- function(parm_vector.in, delta.in, X.in,  XP.in,  Z.in, PZ.in, W.inv.in, s.jm.in, mkt.i

  sd_sug.in <- exp(parm_vector.in[[1]])
  sd_mush.in <- exp(parm_vector.in[[2]])

   # get new MU matrix
  mu <- find_mu(X.in, sd_sug.in = sd_sug.in, sd_mush.in = sd_mush.in, v.in)

  # update delta
  squarem.output <- squarem(par       = delta.in,
                            fixptfn   = blp_inner,
                            mu.in     = mu,
                            s.jm.in   = s.jm.in,
                            mkt.id.in = mkt.id.in,
                            control   = list(trace = TRUE))
  delta_new <- squarem.output$par

  # first step
  PX.inv <- solve(t(XP.in) %*% PZ.in %*% XP.in)

  # finsih getting theta
  theta1 <- PX.inv %*% t(XP.in) %*% PZ.in %*% delta_new

  # get xi hat
  xi.hat <- delta_new - XP.in %*% theta1

  result <- t(xi.hat) %*% Z.in %*% W.inv.in %*% t(Z.in) %*% xi.hat
  # get function value
  return(result)

}

# make sd_vector
parm_vector <- c(3, 3)

# test it out
f <- gmm_obj_f(parm_vector.in = parm_vector,
               delta.in    = delta.initial,
               X.in        = X,
               XP.in       = XP,
               Z.in        = Z,
               PZ.in       = PZ,
               W.inv.in    = W.inv,
               s.jm.in     = s.jm,
               mkt.id.in   = mkt.id,
               v.in        = v)


Results <-  optim(par         = c(3,3),
```

```r
                      fn          =  gmm_obj_f,
                    # # uncomment if you want fast convergence
                      # control       = list(reltol = 5),
                      delta.in    = delta.initial,
                      X.in        = X,
                      XP.in       = XP,
                      Z.in        = Z,
                      PZ.in       = PZ,
                      W.inv.in    = W.inv,
                      s.jm.in     = s.jm,
                      mkt.id.in   = mkt.id,
                      v.in        = v)

# save these real quick in case something happens
save(Results, file = paste0(f_out, "results_save.R"))

# grab out values
sd_final <- exp(Results$par)
sd_final



# get betas
# get new MU matrix
mu <- find_mu(X, sd_sug.in = sd_final[[1]], sd_mush.in = sd_final[[2]], v)

# update delta
squarem.output <- squarem(par       = delta.initial,
                          fixptfn   = blp_inner,
                          mu.in     = mu,
                          s.jm.in   = s.jm,
                          mkt.id.in = mkt.id,
                          control   = list(trace = TRUE))
delta_final <- squarem.output$par

# first step
PX.inv <- solve(t(XP) %*% PZ %*% XP)

# finsih getting theta
theta_final <- PX.inv %*% t(XP) %*% PZ %*% delta_new


# get m(parms) moment condition
temp <- delta_final - XP%*%theta_final
temp <- cbind(temp, temp, temp, temp)
m_final <- Z*(temp)


S_final <- (1/nrow(m_final))*t(m_final)%*%m_final

# get derivative of moment condition for Beta
temp <- cbind(X[,1],X[,1],X[,1],X[,1])
dmdb1 <- -Z*temp
```

```r
temp <- cbind(X[,2],X[,2],X[,2],X[,2])
dmdb2 <- -Z*temp

# now for p
temp <- cbind(P,P,P,P)
dmda <- Z*temp


#========================#
# ==== now get dmds ====
#========================#

  #========================#
  # ==== Get first one ====
  #========================#

    # set phi
    phi <- .001
    # first get delta with + phi
    # get new MU matrix
    mu <- find_mu(X, sd_sug.in = sd_final[[1]] + phi, sd_mush.in = sd_final[[2]], v)

    # update delta
    squarem.output <- squarem(par       = delta.initial,
                              fixptfn   = blp_inner,
                              mu.in     = mu,
                              s.jm.in   = s.jm,
                              mkt.id.in = mkt.id,
                              control   = list(trace = TRUE))
    delta_phi_p0 <- squarem.output$par
    temp <- delta_phi_p0 - XP%*%theta_final
    temp <- cbind(temp, temp, temp, temp)
    m_p0 <- Z*(temp)


    # do it again
    mu <- find_mu(X, sd_sug.in = sd_final[[1]] - phi, sd_mush.in = sd_final[[2]], v)

    # update delta
    squarem.output <- squarem(par       = delta.initial,
                              fixptfn   = blp_inner,
                              mu.in     = mu,
                              s.jm.in   = s.jm,
                              mkt.id.in = mkt.id,
                              control   = list(trace = TRUE))
    delta_phi_m0 <- squarem.output$par
    temp <- delta_phi_m0 - XP%*%theta_final
    temp <- cbind(temp, temp, temp, temp)
    m_m0 <- Z*(temp)

    # now get first dmds
    dmds1 <- (m_p0 - m_m0)/(2*phi)
```

```r
 #=========================#
 # ==== get second one ====
 #=========================#

  mu <- find_mu(X, sd_sug.in = sd_final[[1]], sd_mush.in = sd_final[[2]] + phi, v)

  # update delta
  squarem.output <- squarem(par       = delta.initial,
                            fixptfn   = blp_inner,
                            mu.in     = mu,
                            s.jm.in   = s.jm,
                            mkt.id.in = mkt.id,
                            control   = list(trace = TRUE))
  delta_phi_0p <- squarem.output$par

  temp <- delta_phi_0p - XP%*%theta_final
  temp <- cbind(temp, temp, temp, temp)
  m_0p <- Z*(temp)


  mu <- find_mu(X, sd_sug.in = sd_final[[1]], sd_mush.in = sd_final[[2]] - phi, v)

  # update delta
  squarem.output <- squarem(par       = delta.initial,
                            fixptfn   = blp_inner,
                            mu.in     = mu,
                            s.jm.in   = s.jm,
                            mkt.id.in = mkt.id,
                            control   = list(trace = TRUE))
  delta_phi_0m <- squarem.output$par


  temp <- delta_phi_0m - XP%*%theta_final
  temp <- cbind(temp, temp, temp, temp)
  m_0m <- Z*(temp)

  # Now finish getting dmds

  dmds2 <- (m_0p - m_0m)/(2*phi)

#==============================#
# ==== get standard errors ====
#==============================#

  isSymmetric(S_final)

  S_final
  m_final

  dmdb1
  dmdb2
  dmda
  dmds1
```

```
      dmds2

      Rho <- t(rbind(colMeans(dmdb1),
                     colMeans(dmdb2),
                     colMeans(dmda),
                     colMeans(dmds1),
                     colMeans(dmds2)))

    SE_mat_final <- (1/JM)*(solve(t(Rho) %*% Rho, tol = 10^(-25)) %*% (t(Rho) %*% S_final %*% Rho) %*% s

    SE_final <- sqrt(diag(SE_mat_final))

    # put estiamates and SE into a table
    q3_p5 <- data.table(Variable = c("Sigma Sugar", "Sigma Mushy", "Beta Sugar", "Deta Mushy", "Alpha")



#================================#
# ==== save output to latex ====
#================================#

  if(opt_save){


    print(xtable(probit_res, type = "latex"),
          file = paste0(f_out, "q1_p1.tex"),
          include.rownames = FALSE,
          floating = FALSE)


    print(xtable(logit_res, type = "latex"),
          file = paste0(f_out, "q1_p3.tex"),
          include.rownames = FALSE,
          floating = FALSE)


    print(xtable(p6_res_tab, type = "latex"),
          file = paste0(f_out, "q1_p6.tex"),
          include.rownames = FALSE,
          floating = FALSE)

    print(xtable(q2_p3_ols, type = "latex"),
          file = paste0(f_out, "q2_p3a.tex"),
          include.rownames = FALSE,
          floating = FALSE)

    print(xtable(q2_p3_iv1, type = "latex"),
          file = paste0(f_out, "q2_p3b.tex"),
          include.rownames = FALSE,
          floating = FALSE)

    print(xtable(q2_p3_iv2, type = "latex"),
          file = paste0(f_out, "q2_p3c.tex"),
```

```
        include.rownames = FALSE,
        floating = FALSE)

  print(xtable(q3_p5, type = "latex"),
        file = paste0(f_out, "q3_p5.tex"),
        include.rownames = FALSE,
        floating = FALSE)


}

  #=====================================#
  # ==== run markdown to print code ====
  #=====================================#

  rmarkdown::render(input =  "C:/Users/Nmath_000/Documents/Code/courses/econ 631/ps1/ps1_r_markdown.Rm
                    output_format = "pdf_document",
                    output_file = paste0(f_out, "assignment_1_r_code_pdf.pdf"))
```