Homework 7
Labor Economics

Mark Agerton and Nick Frazier

Due Mon, Feb 23

1 Setup

Wages are

$$\begin{aligned} y_0 &= \delta_0 + \beta_0 x + \theta + \epsilon_0 \\ y_1 &= \delta_1 + \beta_1 x + \alpha_1 \theta + \epsilon_1 \end{aligned}$$

Also define the utility shifter function C and an index function I

$$\begin{aligned} C &= \gamma_0 + \gamma_2 z + \gamma_3 x + \alpha_C \theta \\ I &= E[y_1 - y_0 - C | \mathcal{F}] \\ &= \underbrace{(\delta_1 - \delta_0 - \gamma_0)}_{\widetilde{\delta}} + \underbrace{(\beta_1 - \beta_0 - \gamma_3)}_{\widetilde{\beta}} x_i - \gamma_2 z + \underbrace{(\alpha_1 - 1 - \alpha_C)}_{\widetilde{\alpha}} \theta - \epsilon_c \end{aligned}$$

The distribution of shocks is

$$\left. \begin{pmatrix} \epsilon_{i,0} \\ \epsilon_{i,1} \\ \epsilon_{i,C} \end{pmatrix} \right|_{x_i, z_i, \theta_i} \sim N \left[\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_0^2 & 0 & 0 \\ 0 & \sigma_1^2 & 0 \\ 0 & 0 & \sigma_C^2 \end{pmatrix} \right]$$

The information set for the agent is $\mathcal{F}$. The preference shock $\epsilon_C \in \mathcal{F}$, but $\{\epsilon_0, \epsilon_1\} \notin \mathcal{F}$. The decision rule is

$$s = 1 \iff E[I \geq 0 | \mathcal{F}]$$

2 Q1

There is no unobserved heterogeneity in this model since we know θ. Thus,

$$E\left[\begin{matrix} y_0 \\ y_1 \end{matrix} \middle| x_i, \theta_i, s = k \right] = \begin{bmatrix} \delta_0 + x\beta_0 + \theta \\ \delta_1 + x\beta_1 + \alpha_1 \theta \end{bmatrix} + \underbrace{E\left[\begin{matrix} \epsilon_0 \\ \epsilon_1 \end{matrix} \middle| x_i, \theta_i, \epsilon_c : big/small \right]}_{0}$$

This is straight-up OLS, which means we recover $\{\delta, \beta, \alpha_1, \sigma_0^2, \sigma_1^2\}$.

$$\begin{aligned} \Pr[S = 1 | \mathcal{F}] &= \Pr\left[\epsilon_c \leq \widetilde{\delta} + \widetilde{\beta} x - \gamma_2 z + \widetilde{\alpha}\theta \middle| \mathcal{F} \right] \\ &= \Phi \left[\left. \frac{\overbrace{[(\delta_1 - \delta_0) + (\beta_1 - \beta_0)x + (\alpha_1 - 1)\theta]}^{\text{known number}} - \gamma_0 - \gamma_2 z - \gamma_3 x - \alpha_c \theta}{\sigma_c} \right| \mathcal{F} \right] \end{aligned}$$

Now we can get $\left\{\gamma_0, \gamma_2, \gamma_3, \alpha_c, \sigma_c^2\right\}$

3 Q2

Now we don't know θ but agents do. However, we do have two measurement equations $m \in \{A, B\}$:

$$\begin{aligned} M_{iA} &= x_i^M \beta_A^M + \theta_i + \epsilon_{iA}^M \\ M_{iB} &= x_i^M \beta_B^M + \alpha_B \theta_i + \epsilon_{iB}^M \end{aligned}$$

where $\epsilon_m^M \sim N(0, \sigma_m^{M2})$ are i.i.d.

3.1 Heckman two-step

We can write

$$\begin{aligned} E[y_1|x,z,s=1] &= \delta_1 + \beta_1 x + E[\epsilon_1 + \alpha_1 \theta | x, z, I \geq 0] \\ &= \delta_1 + \beta_1 x + \alpha_1 E[\theta | x, z, I \geq 0] \\ &= \delta_1 + \beta_1 x + \alpha_1 \sigma^* E\left[\frac{\theta}{\sigma^*}\middle| 0 \leq \widetilde{\delta} + \widetilde{\beta} x - \gamma_2 z + \underbrace{(\alpha_1 - \alpha_0 - \alpha_c)\theta - \epsilon_c}_{\eta}\right] \\ &= \delta_1 + \beta_1 x + \alpha_1 \sigma^* E\left[\frac{\theta}{\sigma^*}\middle| \eta \geq -\left(\widetilde{\delta} + \widetilde{\beta} x - \gamma_2 z\right)\right] \end{aligned}$$

Define $\eta \equiv (\alpha_1 - \alpha_0 - \alpha_c)\theta - \epsilon_c$. Then

$$\begin{pmatrix} \eta \\ \theta \end{pmatrix} \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma^{*2} & (\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2 \\ (\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2 & \sigma_\theta^2 \end{pmatrix}\right]$$

where $\sigma^{*2} = (\alpha_1 - \alpha_0 - \alpha_c)^2 \sigma_\theta^2 + \sigma_c^2$. We can project θ onto η, which means

$$\theta = \frac{\operatorname{Cov}(\eta, \theta)}{\operatorname{Var} \eta} \eta + \nu = \frac{(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2}{\sigma^{*2}} \eta + \nu$$

where

$$\nu \sim N\left(0, \sigma_\theta^2 \left(1 - \rho_{\eta\theta}^2\right)\right) \qquad \text{and} \qquad \rho_{\eta\theta} = \frac{(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2}{\sigma^* \sigma_\theta}$$

Letting $t \equiv -(\widetilde{\delta} + \widetilde{\beta} x - \gamma_2 z)/\sigma^*$, we can now write

$$\begin{aligned} E[y_0|x,z,s=1] &= \delta_0 + \beta_0 x + \alpha_0 \frac{(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2}{\sigma^*} \overbrace{\frac{-\phi(t)}{\Phi(t)}}^{\lambda_0} \\ &= \delta_0 + \beta_0 x + \alpha_0 \left(\rho_{\eta\theta} \sigma_\theta\right) \lambda_{0i} \\ E[y_1|x,z,s=1] &= \delta_1 + \beta_1 x + \alpha_1 \frac{(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2}{\sigma^*} \underbrace{\frac{\phi(t)}{1 - \Phi(t)}}_{\lambda_1} \\ &= \delta_1 + \beta_1 x + \alpha_1 \left(\rho_{\eta\theta} \sigma_\theta\right) \lambda_{1i} \end{aligned}$$

A probit first-step has given us $\{(\delta_1 - \delta_0 - \gamma_0)/\sigma_c, (\beta_1 - \beta_0 - \gamma_3)\sigma_c), \gamma_2/\sigma_c\}$. With the second step, we now get $\{\delta_1, \delta_0, \beta_1, \beta_0\}$ and the ratio α_1/α_0. ~~We can back out $\{\gamma_0/\sigma_c, \gamma_2/\sigma_c, \gamma_3/\sigma_c\}$ from the original probit equations. We also get the quantity $(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2$ since we have σ_c^2.~~ However, we have 3 αs and only 2 equations for them, so those aren't identified. We can now turn to variances and covariances. Recall

$$\rho_{\eta\theta}\sigma_\theta = \frac{(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2}{\sqrt{(\alpha_1 - \alpha_0 - \alpha_c)^2\sigma_\theta^2 + \sigma_c^2}}$$

It is clear to see that these are of little help since we have a bunch of parameters in the euqations for the variances:

$$\begin{aligned}
\mathrm{Var}(Y_0|\eta < -t) &= \alpha_0^2 \,\mathrm{Var}\,(\theta|\eta < t) + \sigma_0^2 \\
&= \alpha_0^2 \left(\rho_{\eta\theta}\sigma_\theta\right)^2 \left[1 - t\lambda_0 - \lambda_0^2\right] + \sigma_\theta^2 \left(1 - \rho_{\eta\theta}^2\right) + \sigma_0^2 \\
\mathrm{Var}(Y_1|\eta \geq -t) &= \alpha_1^2 \,\mathrm{Var}\,(\theta|\eta \geq t) + \sigma_1^2
\end{aligned}$$

Fortunately, with the measurement equations, we can say things. Recall $I = E[Y_1 - Y_0 - C|X, Z, \theta]$. If we had an estimate of I, we would be in business... and when we do EM/MLE, we do get an estimate of I (right). **Previously: "I have no idea what to do with the last 2 eqns b/c how do we compute I w/ out θ This is maybe why we need MLE and EM??"**

$$\mathrm{Cov}(Y_0 - \beta_0 X, M^A - X^M\beta_A) = \alpha_0\sigma_\theta^2 \tag{1}$$

$$\mathrm{Cov}(Y_0 - \beta_0 X, M^B - X^M\beta_B) = \alpha_0\alpha_B\sigma_\theta^2 \tag{2}$$

$$\mathrm{Cov}(Y_1 - \beta_1 X, M^A - X^M\beta_A) = \alpha_1\sigma_\theta^2 \tag{3}$$

$$\mathrm{Cov}(Y_1 - \beta_1 X, M^B - X^M\beta_B) = \alpha_1\alpha_B\sigma_\theta^2 \tag{4}$$

$$\mathrm{Cov}\left[I - \widetilde{\delta} - \widetilde{\beta}x - \gamma_2 z,\ \left(M^A - X^M\beta_A\right)\right] = (\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2 \tag{5}$$

$$\mathrm{Cov}\left[I - \widetilde{\delta} - \widetilde{\beta}x - \gamma_2 z,\ \left(M^B - X^M\beta_B\right)\right] = (\alpha_1 - \alpha_0 - \alpha_c)\alpha_B\sigma_\theta^2 \tag{6}$$

$$\mathrm{Cov}\left[I - \widetilde{\delta} - \widetilde{\beta}x - \gamma_2 z,\ (Y_0 - X\beta_0)\right] = (\alpha_1 - \alpha_0 - \alpha_c)\alpha_0\sigma_\theta^2 \tag{7}$$

$$\mathrm{Cov}\left[I - \widetilde{\delta} - \widetilde{\beta}x - \gamma_2 z,\ (Y_1 - X\beta_1)\right] = (\alpha_1 - \alpha_0 - \alpha_c)\alpha_1\sigma_\theta^2 \tag{8}$$

The top four equations give us two measurements for α_B. The bottom four plus knowledge of $(\alpha_1 - \alpha_0 - \alpha_c)\sigma_\theta^2$ from the two-step gives us α_0 and α_1 (just divide them). With α_ks in hand plus α_B give us multiple measurements for σ_θ^2. We plug these in to the variances for $\mathrm{Var}(Y_k|X, Z, s = k)$ and get σ_k^2. Done.

3.2 MLE approach

The contribution to the likelihood of any given individual i is now the product of the likelihood of the wage and choice times the product of the likelihoods of the test equations.

$$\begin{aligned}
L_i = &\left[f(y_{1i}|X, \theta, s_i = 1)\Pr(s_i = 1|X, Z, \theta)\right]^{s_i} \\
&\times \left[f(y_{0i}|X, \theta, s_i = 0)\Pr(s_i = 0|X, Z, \theta)\right]^{1-s_i} \\
&\times f(m_i^A|X_i^M, \theta) \\
&\times f(m_i^B|X_i^M, \theta) \\
&\times f(\theta)
\end{aligned}$$

Define $q_i \equiv 2s_i - 1$. Since we only observe y_{1i} or y_{i0}, we simply use y_i in the likelihood equation. We can log everything and integrate w/ respect to θ.

$$\begin{aligned}
\mathcal{L}_i = \int_\theta & \log\left[1 - \Phi\left(q_i \times \frac{(\delta_1 - \delta_0 - \gamma_0) + (\beta_1 - \beta_0 - \gamma_3)X_i - \gamma_2 Z_i + (\alpha_1 - \alpha_0 - \alpha_c)\theta}{\sigma_c}\right)\right] \\
& + s_i \log\left[\phi\left(\frac{y_i - \delta_1 - \beta_1 x_i - \alpha_1\theta}{\sigma_1}\right)\right] \\
& + (1 - s_i)\log\left[\phi\left(\frac{y_i - \delta_0 - \beta_0 x_i - \alpha_0\theta}{\sigma_0}\right)\right] \\
& + \log\left[\phi\left(\frac{M_i^A - X_i^M \beta_A - \theta}{\sigma_A}\right)\right] \\
& + \log\left[\phi\left(\frac{M_i^B - X_i^M \beta_B - \alpha_B\theta}{\sigma_B}\right)\right] \\
& + \log\left[\phi\left(\frac{\theta}{\sigma_\theta}\right)\right] d\theta
\end{aligned}$$

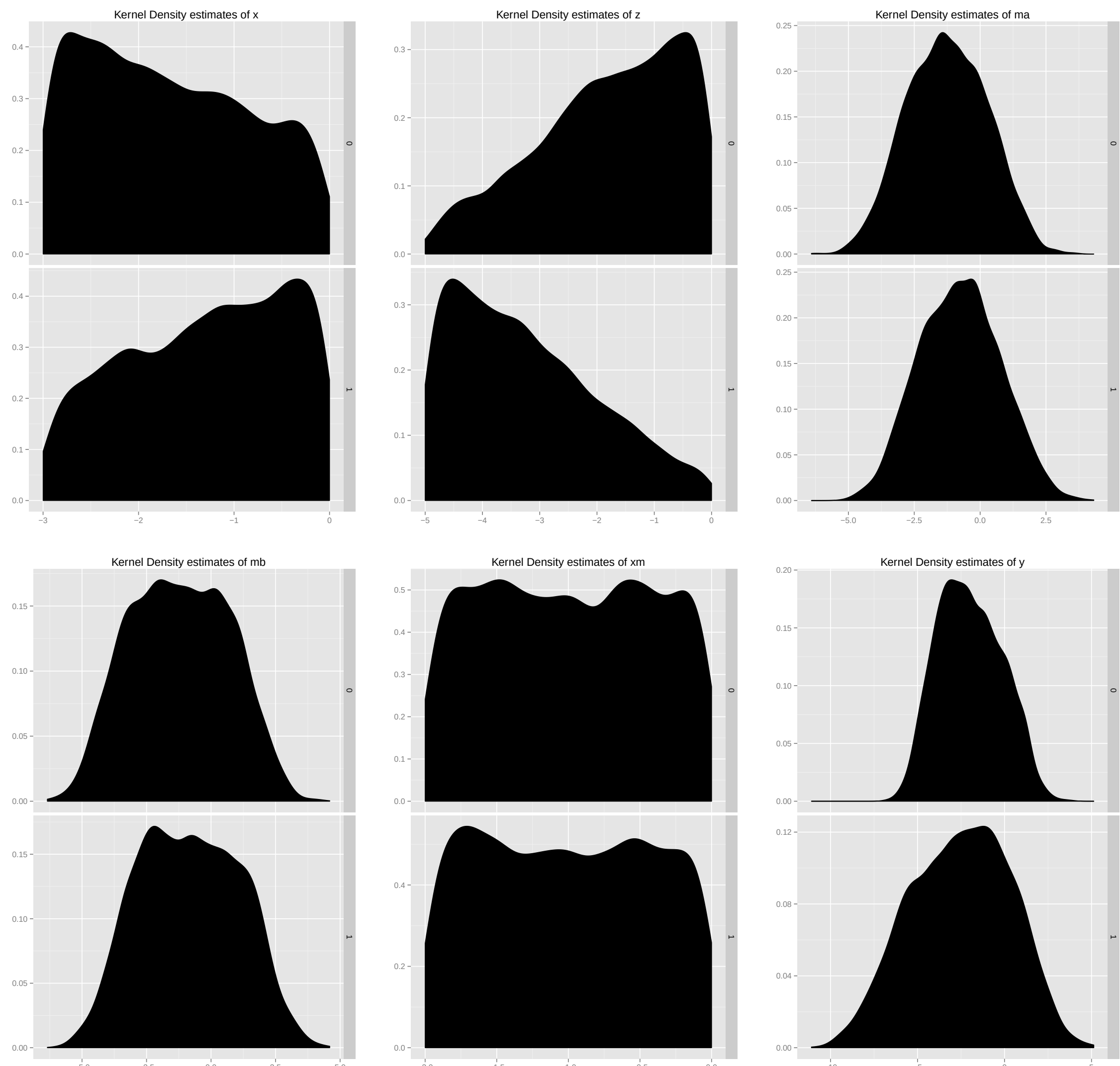

Figure 1: Histograms

4 Results

5 Main call

Listing 1: Main call

```
using DataFrames
using Distributions
using Optim

####################################################
######### Structure of Code
####################################################
# Process data
# OLS
# Run probit
  # Probit value function
  # Probit gradient
  # Probit hessian
# Process Probit results
# recover structural parameters
# estimate variance of structural parameters
# EM algorithm

####################################################
######### Basic Parameters
####################################################
data_dir = "C:/Users/Nick/SkyDrive/One_data/LaborEcon/PS7/"
cd(data_dir)
fs = "data_ps7_spring2015.raw"
namevec = [symbol("id"),symbol("S"),symbol("Y"),symbol("M_a"),symbol("M_b"),symbol("X"),symbol("Z
    "),symbol("X_m")]

data = readtable(fs, separator = ' ', header = true, names = namevec)



code_dir = "C:/Users/Nick/SkyDrive/One_data/LaborEcon/PS7/code"
cd(code_dir)
include("functions.jl")

####################################################
######### Read in data. Use DataFrames
####################################################


####################################################
######### Process Data
####################################################

# TODO flip data to make it (obs x var)

N = int(size(data,1))
N_1 = sum(data[:S])
N_0 = N - N_1

# create constant
data[:C] = vec(ones(N,1))

data[:Y_0] = NaN
data[:Y_1] = NaN
data[data[:S] .== 1,:Y_0]  = data[data[:S] .== 1,:Y]
data[data[:S] .== 0,:Y_1]  = data[data[:S] .== 0,:Y]

sel0 = data[:S] .== 0
sel1 = data[:S] .== 1
```

```
K_A       = 2
K_B       = 2
numparams = 10 # just a guess

###################################################
######### Step 1
###################################################

# notice that they all have the same mean
# mean(data[:M_a])
# mean(data[:M_b])
# mean(data[:X_m])


# OLS on measurement equations
(β_A,σ_a,VCV_a) = least_sq(data[:X_m],data[:M_a])
se_β_A = sqrt(VCV_a)

(β_B,σ_b,VCV_b) = least_sq(data[:X_m],data[:M_b])
se_β_B = sqrt(VCV_b)

# TODO publish params

###################################################
######### Heckman 2-step
###################################################

# Step 1: Probit
# probit data
X = [vec(data[:C])  vec(data[:X]) vec(data[:Z]) ]
d = convert(Array,data[:S])

# optimization
iters = 1
f = probit_LL
g! = probit_gradient!
h! = probit_hessian!

initials = squeeze( (X'X)\X'd, 2).*2.*rand(size(X,2))

probit_opt = []
for kk = 1:iters
  probit_opt = Optim.optimize(f,g!,h!,vec(initials),
    xtol = 1e-32,
    ftol = 1e-32,
    grtol = 1e-14,
    iterations = 3000)
  initials = probit_opt.minimum
end
probit_opt

probit_res         = probit_results(probit_opt.minimum,g!,h!)
param_probit       = probit_res["θ"]
param_probit_se    = probit_res["std_hess"]
VCV_probit         = probit_res["vcv_hessian"]
param_probit_z     = probit_res["z_stat"]
param_probit_pval = probit_res["pvals"]
# probit_res["ME1"]
# probit_res["ME2"]

# Step 2: OLS

t = -(X*param_probit)
λ_0 = -normpdf(t)./normcdf(t)


# Y_0
```

```
Y0 = convert(Array,data[sel0,:Y])
X0 = [vec(data[sel0,:C]) vec(data[sel0,:X]) λ_0[sel0] ]

(ρ_0,~,VCV_ρ_0) = least_sq(X0,Y0)
se_ρ_0 = sqrt(diag(VCV_ρ_0))

# Y_1
Y1 = convert(Array,data[sel1,:Y])
X1 = [vec(data[sel1,:C]) vec(data[sel1,:X]) λ_0[sel1] ]

(ρ_1,~,VCV_ρ_1) = least_sq(X1,Y1)
se_ρ_1 = sqrt(diag(VCV_ρ_1))

# publish params

δ_0    = ρ_0[1]
β_0    = ρ_0[2]
π_0    = ρ_0[3]
δ_0_se = se_ρ_0[1]
β_0_se = se_ρ_0[2]
π_0_se = se_ρ_0[3]

δ_1    = ρ_1[1]
β_1    = ρ_1[2]
π_1    = ρ_1[3]
δ_1_se = se_ρ_1[1]
β_1_se = se_ρ_1[2]
π_1_se = se_ρ_1[3]

###################################################
######### Recover some more parameters
###################################################

# cannot get gammas? Need them for next step. Estimate of Ī



###################################################
######### Use covariances
###################################################

Y_0_Xβ = convert(Array,data[sel0,:Y])
    - [vec(data[sel0,:C]) vec(data[sel0,:X])]*[δ_0; β_0]
Y_1_Xβ = convert(Array,data[sel1,:Y])
    - [vec(data[sel1,:C]) vec(data[sel1,:X])]*[δ_1; β_1]
M_A0_Xβ = convert(Array,data[sel0,:M_a])
    - vec(data[sel0,:X_m]).*β_A
M_B0_Xβ = convert(Array,data[sel0,:M_b])
    - vec(data[sel0,:X_m]).*β_B
M_A1_Xβ = convert(Array,data[sel1,:M_a])
    - vec(data[sel1,:X_m]).*β_A
M_B1_Xβ = convert(Array,data[sel1,:M_b])
    - vec(data[sel1,:X_m]).*β_B

cov_0_A = (1/N_0)*sum(Y_0_Xβ'*M_A0_Xβ)
cov_0_B = (1/N_0)*sum(Y_0_Xβ'*M_B0_Xβ)
cov_1_A = (1/N_1)*sum(Y_1_Xβ'*M_A1_Xβ)
cov_1_B = (1/N_1)*sum(Y_1_Xβ'*M_A1_Xβ)

###################################################
######### EM algorithm
###################################################


include("HG_wts.jl")

σ_θ = 1
initials = ones(18)
```

```
initials[1:4] = [ρ_0[1] ρ_1[1] ρ_0[2] ρ_1[2]]
opt_out = []

# Is the idea we try for 100 iterations b/w inner MLE and outer σ_θ optimization?
# what about a loop w/ "while (abs( opt_out.f_minimum - opt_out_old.f_minimum  ) > ftol) || (
    count < maxit) " ?
for i = 1:100
  count = 0

  opt_out = Optim.optimize(wtd_LL,vec(initials),
      xtol = 1e-32,
      ftol = 1e-32,
      grtol = 1e-14,
      iterations = 2000,
      autodiff=true)
  initials = opt_out.minimum

  update = unpackparams(opt_out.minimum)
  δ_0 = update["δ_0"]
  δ_1 = update["δ_1"]
  β_0 = update["β_0"]
  β_1 = update["β_1"]
  α_0 = update["α_0"]
  α_1 = update["α_1"]
  α_C = update["α_C"]
  β_A = update["β_A"]
  α_B = update["α_B"]
  β_B = update["β_B"]

  Y0         = convert(Array,data[sel0,:Y])
  X0         = [vec(data[sel0,:C]) vec(data[sel0,:X])]
  Y1         = convert(Array,data[sel1,:Y])
  X1         = [vec(data[sel1,:C]) vec(data[sel1,:X])]

  # Why are we getting a θ_hat? Is this to get an estimate for σ_θ?
  θ_hat = zeros(N)
  θ_A = data[:M_a]  - data[:X_m] .* β_A
  θ_B = (data[:M_b] - data[:X_m] .* β_B)./α_B
  θ_hat[sel1] = (1/3).* ( θ_A[sel1] + θ_B[sel1] +
          ( (Y1 - X1*[δ_1; β_1])  )./α_1 )
  θ_hat[sel0] = (1/3).* ( θ_A[sel0] + θ_B[sel0] +
          ( (Y0 - X0*[δ_0; β_0])  )./α_0 )
  σ_θ = var(θ_hat)
end


opt_out.minimum

ρ_0
ρ


str = ["δ_0", "δ_1", "β_0","β_1",
    " _0"," _2"," _3","α_0","α_1",
    "α_C","σ_C","σ_1","σ_2","β_A",
    "α_B","σ_A","β_B","σ_B"]

numparams = length(opt_out.minimum)
for i = 1:numparams

  @sprintf("%s  :  %5.3f ", [str[i] opt_out.minimum[i]])

end

# println("Coefficients from model 1: ")
# println("           [β_0,β_1,β_2] = $(round(beta_MLE,3))")
# println("  [SE(β0),SE(β1),SE(β1)] = $(round(beta_SE,3))")
# println("Coefficients from model 2:")
```

```
# println("               [ 0 , 1 , 2 ] = $(round(gamma,3))")
# println("  [SE( 0 ),SE( 1 ),SE( 1 )] = $(round(gamma_SE,3))")
# println(" ")
# println(" ")
# println("                      ρ is: $(round(rho_mle,3))")
# println("                  SE(ρ) is: $(round(rho_SE,3))")
# println("                Sigma_v is: $(round(sigma_v,3))")
# println("")
# println("            MAX Log(L) is: $(round(probit_opt.f_minimum,3))")
```

6 Functions and weights

Listing 2: Functions used

```
# functions

# least_sq
# pdf wrappers
# Probit


####################################################
######### Process Data
####################################################

function least_sq(X::Array,Y::Array;N=int(size(X,1)), W=1)
  l = minimum(size(X))
  A = X'*W*X
  if sum(size(A))== 1
    inv_term = 1./A
  else
    inv_term = A\eye(int(size(X,2)))
  end
  β = inv_term * X'*W*Y
  if l == 1
    sigma_hat = sqrt(sum((1/N).* (Y - (β*X')')'*(Y - (β*X')'  ) ) ) #sum converts to Float64
  else
    sigma_hat = sqrt(sum((1/N).* (Y - (X*β))'*(Y - (X*β)  ) ) ) #sum converts to Float64
  end
  VCV = (sigma_hat).^2 * inv_term * eye(l)
  return β, sigma_hat, VCV
end

function least_sq(X::DataArray,Y::DataArray;N=int(size(X,1)), W=1)
  l = minimum( [size(X,2),size(X,1)]) # b/c array has size 0
  X = convert(Array{Float64,1},X)
  Y = convert(Array{Float64,1},Y)
  A = X'*W*X
  if sum(size(A))== 1
    inv_term = 1./A
  else
    inv_term = A\eye(int(size(X,2)))
  end
  β = inv_term * X'*W*Y
  if l == 1
    sigma_hat = sqrt(sum((1/N).* (Y - (β*X')')'*(Y - (β*X')'  ) ) ) #sum converts to Float64
  else
    sigma_hat = sqrt(sum((1/N).* (Y - (X*β))'*(Y - (X*β)  ) ) ) #sum converts to Float64
  end
  VCV = (sigma_hat).^2 * inv_term * eye(l)
  return β, sigma_hat, VCV
end

####################################################
######### pdf wrappers
```

```julia
#####################################################

## Normal PDF
function normpdf(x::Union(Vector{Float64}, Float64, DataArray) ;mean=0,var=1) # a type-union
    should work here and keep code cleaner
    out = Distributions.pdf(Distributions.Normal(mean,var), x)
end

# function normpdf(x::Float64;mean=0,var=1)
#   out = Distributions.pdf(Distributions.Normal(mean,var), x)
# end

# function normpdf(x::DataArray;mean=0,var=1)
#   out = Distributions.pdf(Distributions.Normal(mean,var), x)
# end

## Normal CDF
function normcdf(x::Union(Vector{Float64}, Float64, DataArray);mean=0,var=1)
    out = Distributions.cdf(Distributions.Normal(mean,var), x)
    out + (out .== 0.0)*eps(1.0) - (out .== 1.0)*eps(1.0)
end

# function normcdf(x::Vector{Float64};mean=0,var=1)
#   out = Distributions.cdf(Distributions.Normal(mean,var), x)
#   out + (out .== 0.0)*eps(1.0) - (out .== 1.0)*eps(1.0)
# end

# function normcdf(x::DataArray;mean=0,var=1)
#   out = Distributions.cdf(Distributions.Normal(mean,var), x)
#   out + (out .== 0.0)*eps(1.0) - (out .== 1.0)*eps(1.0)
# end

#####################################################
######### Probit
#####################################################

function λ(θ::Vector{Float64})
    q = 2d-1
    q .* normpdf(q .* X*θ) ./ normcdf(q.*X*θ)
end


function probit_LL(θ::Vector{Float64})
    out = - sum( log( normcdf( (2d-1) .* X*θ) ) )
end

function probit_LL_g(θ::Vector{Float64}, grad::Vector{Float64})
    out = - sum( log( normcdf( (2d-1) .* X*θ) ) )
    if length(grad) > 0
        grad[:] = - sum( λ(θ) .* X,  1 )
    end
    out
end

function probit_gradient!(θ::Vector{Float64}, grad::Vector{Float64})
  grad[:] = - sum( λ(θ) .* X,  1 )
end

function probit_hessian!(θ::Vector{Float64}, hessian::Matrix{Float64})
  hh = zeros(size(hessian))
  A = λ(θ) .* ( λ(θ) + X*θ )
  for i in 1:size(X)[1]
    hh += A[i] * X[i,:]'*X[i,:]
  end
  hessian[:] = hh
end

function probit_vcov_score(θ::Vector{Float64}, g!)
```

```
    K = length(θ)
    N = maximum(size(X))
    score - zeros(K,1)
    g!(θ,score)
    vcv_hessian = N*(score*score') \ eye(K)
end

function probit_vcov_hessian(θ::Vector{Float64}, h!)
    K       = length(θ)
    hessian = zeros((K,K))
    h!(θ, hessian)
    vcv_hessian = N*(hessian\eye(K))
end

function probit_results(θ::Vector,g!,h!)

    K = length(θ)

    vcv_hessian = repmat([NaN],K,K)
    try
        vcv_hessian = probit_vcov_hessian(θ, h!)
    catch
        println("No hessian for probit")
    end

    vcv_score = repmat([NaN],K,K)
    try
        vcv_score   = probit_vcov_score(θ, g!)
    catch
        println("No outer product for probit")
    end

    std_h = sqrt(diag(vcv_hessian))
    std_s = sqrt(diag(vcv_score))

    z_stat = θ./std_h
    pvals = Distributions.cdf(Distributions.Normal(), -abs(z_stat))

    X_bar = mean(X,1)
    # # Partial Effect at the Average
    ME1 = normpdf(vec(X_bar'.*θ)) .* θ
    # # Average Partial Effect (pg. 5)
    ME2 = mean(  normpdf( vec(X*θ) )  ) * θ

    return [
     "θ"=>θ ,
     "std_hess" => std_h,"std_score" => std_s,
     "vcv_hessian" => vcv_hessian, "vcv_score" => vcv_score,
     "z_stat"=> z_stat, "pvals"=> pvals,
     "ME1"=>ME1,"ME2"=>ME2]
end

###################################################
######### EM algorithm
###################################################

function wtd_LL(ρ::Vector{Float64})
  NN = length(X)
  ll = zeros(NN,N)
  for (j,x_j) in  enumerate(X)
    # Should weights be additive?
    ll[j,:] = W[j] .* ( LL_term(ρ, σ_θ .* x_j) + normpdf(x_j) )'
  end
  countPlus!()
  - sum(ll)
end

function LL_term(ρ::Vector{Float64}, x::Float64)
```

```
  θ   = x.*ones(N)
  out = unpackparams(ρ)
  δ_0 = out["δ_0"]
  δ_1 = out["δ_1"]
  β_0 = out["β_0"]
  β_1 = out["β_1"]
   _0 = out[" _0"]
   _2 = out[" _2"]
   _3 = out[" _3"]
  α_0 = out["α_0"]
  α_1 = out["α_1"]
  α_C = out["α_C"]
  σ_C = out["σ_C"]
  σ_1 = out["σ_1"]
  σ_2 = out["σ_2"]
  β_A = out["β_A"]
  α_B = out["α_B"]
  σ_A = out["σ_A"]
  β_B = out["β_B"]
  σ_B = out["σ_B"]

  Y0        = convert(Array,data[sel0,:Y])
  X0        = [vec(data[sel0,:C]) vec(data[sel0,:X]) θ[sel0]]
  Y1        = convert(Array,data[sel1,:Y])
  X1        = [vec(data[sel1,:C]) vec(data[sel1,:X]) θ[sel1]]
  q         = 2.*convert(Array,data[:S]) -1

  φ_M_A     = normpdf( (data[:M_a] - data[:X_m].*β_A - θ)    ./σ_A)
  φ_M_B     = normpdf( (data[:M_b] - data[:X_m].*β_B - θ*α_B)./σ_B)

  φ_1 = φ_0 = zeros(N)
  φ_1[sel1] = normpdf( (Y1 - X1 * [δ_1; β_1; α_1]) ./ σ_A)
  φ_0[sel0] = normpdf( (Y0 - X0 * [δ_0; β_0; α_0]) ./ σ_B)

  Φ_s       = normcdf(
                q.* (
                        (δ_1 - δ_0 -  _0).*data[:C] +
                        (β_1 - β_0 -  _3).*data[:X] +
                        (- _2) .* data[:Z] +
                        (α_1 - α_0 - α_C) .* θ
                    ) .*σ_C  )

  log(1 - Φ_s) + log(φ_0) + log(φ_1) + log(φ_M_A) + log(φ_M_B)
end

function printCounter(count)
    if count <= 5
        denom = 1
    elseif count <= 50
        denom = 10
    elseif count <= 200
        denom = 25
    elseif count <= 500
        denom = 50
    elseif count <= 2000
        denom = 100
    else
        denom = 500
    end
    mod(count, denom) == 0
end


function countPlus!()
  global count += 1
  if printCounter(count)
    println("Eval $(count)")
```

```
  end
end

####################################################
######### PS_7 functions
####################################################

function unpackparams(θ::Vector{Float64})
  d = minimum(size(θ))
  θ = squeeze(θ,d)
  δ_0 = θ[1]
  δ_1 = θ[2]
  β_0 = θ[3]
  β_1 = θ[4]
   _0 = θ[5]
   _2 = θ[6]
   _3 = θ[7]
  α_0 = θ[8]
  α_1 = θ[9]
  α_C = θ[10]
  σ_C = θ[11]
  σ_1 = θ[12]
  σ_2 = θ[13]
  β_A = θ[14]
  σ_A = θ[15]
  α_B = θ[16]
  β_B = θ[17]
  σ_B = θ[18]

  return [  "δ_0" => δ_0,
  "δ_1" => δ_1,
  "β_0" => β_0,
  "β_1" => β_1,
  " _0" =>  _0,
  " _2" =>  _2,
  " _3" =>  _3,
  "α_0" => α_0,
  "α_1" => α_1,
  "α_C" => α_C,
  "σ_C" => σ_C,
  "σ_1" => σ_1,
  "σ_2" => σ_2,
  "β_A" => β_A,
  "σ_A" => σ_A,
  "α_B" => α_B,
  "β_B" => β_B,
  "σ_B" => σ_B]
end
```

Listing 3: Quadrature weights (from MATLAB)

```
# X W
#  N = 10

A = [
3.4361591188377 0.0000076404329
2.5327316742328 0.0013436457468
1.7566836492999 0.0338743944555
1.0366108297895 0.2401386110823
0.3429013272237 0.6108626337353
-0.3429013272237 0.6108626337353
-1.0366108297895 0.2401386110823
-1.7566836492999 0.0338743944555
-2.5327316742328 0.0013436457468
-3.4361591188377 0.0000076404329 ]

# X W
```

```
#  N = 10

# A = [
# 10.1591092461801 0.0000000000000
# 9.5209036770133 0.0000000000000
# 8.9923980014049 0.0000000000000
# 8.5205692841176 0.0000000000000
# 8.0851886542490 0.0000000000000
# 7.6758399375049 0.0000000000000
# 7.2862765943956 0.0000000000000
# 6.9123815321893 0.0000000000000
# 6.5512591670629 0.0000000000000
# 6.2007735579934 0.0000000000000
# 5.8592901963942 0.0000000000000
# 5.5255210861387 0.0000000000000
# 5.1984265345763 0.0000000000006
# 4.8771500774732 0.0000000000149
# 4.5609737579358 0.0000000002899
# 4.2492864359560 0.0000000044568
# 3.9415607339262 0.0000000547555
# 3.6373358761707 0.0000005433516
# 3.3362046535476 0.0000043942869
# 3.0378033382307 0.0000291874190
# 2.7418037480697 0.0001602773347
# 2.4479069023077 0.0007317735570
# 2.1558378712292 0.0027913248290
# 1.8653415312330 0.0089321783603
# 1.5761790119750 0.0240612727661
# 1.2881246748689 0.0547189709322
# 1.0009634995607 0.1052987636978
# 0.7144887816726 0.1717761569189
# 0.4285000642206 0.2378689049587
# 0.1428012387034 0.2798531175228
# -0.1428012387034 0.2798531175228
# -0.4285000642206 0.2378689049587
# -0.7144887816726 0.1717761569189
# -1.0009634995607 0.1052987636978
# -1.2881246748689 0.0547189709322
# -1.5761790119750 0.0240612727661
# -1.8653415312330 0.0089321783603
# -2.1558378712292 0.0027913248290
# -2.4479069023077 0.0007317735570
# -2.7418037480697 0.0001602773347
# -3.0378033382307 0.0000291874190
# -3.3362046535476 0.0000043942869
# -3.6373358761707 0.0000005433516
# -3.9415607339262 0.0000000547555
# -4.2492864359560 0.0000000044568
# -4.5609737579358 0.0000000002899
# -4.8771500774732 0.0000000000149
# -5.1984265345763 0.0000000000006
# -5.5255210861387 0.0000000000000
# -5.8592901963942 0.0000000000000
# -6.2007735579934 0.0000000000000
# -6.5512591670629 0.0000000000000
# -6.9123815321893 0.0000000000000
# -7.2862765943956 0.0000000000000
# -7.6758399375049 0.0000000000000
# -8.0851886542490 0.0000000000000
# -8.5205692841176 0.0000000000000
# -8.9923980014049 0.0000000000000
# -9.5209036770133 0.0000000000000
# -10.1591092461801 0.0000000000000]

X = A[:,1]
W = A[:,2]
```