Homework 7
Labor Economics

Mark Agerton and Nick Frazier

Due Mon, Feb 23

1 Setup

Wages are

$$y_0 = \delta_0 + \beta_0 x + \theta + \epsilon_0$$
$$y_1 = \delta_1 + \beta_1 x + \alpha_1 \theta + \epsilon_1$$

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

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

3.1.1 Step 1

We know that $S = 1 \Leftrightarrow \eta \geq -(\widetilde{\delta} + \widetilde{\beta} X - \widetilde{\gamma} Z) + \eta \geq 0$. So, in the first step of the procedure, we estimate

$$\begin{aligned} \Pr[S = 1 | \mathcal{F}] &= 1 - \Phi\left[\overbrace{\frac{\delta_1 - \delta_0 - \gamma_0}{\sigma^*}}^{\widetilde{\delta}} + \overbrace{\frac{\beta_1 - \beta_0 - \gamma_3}{\sigma^*}}^{\widetilde{\beta}} X - \overbrace{\frac{\gamma_2}{\sigma^*}}^{\widetilde{\gamma}} Z\right] \\ &= 1 - \Phi\left[\widetilde{\delta} + \widetilde{\beta} X - \widetilde{\gamma}_2 Z\right] \end{aligned}$$

This gives us $\left\{\widetilde{\delta}, \widetilde{\beta}, \widetilde{\gamma}\right\}$.

3.2 Step 2

Letting $t \equiv -(\widetilde{\delta} + \widetilde{\beta}X - \widetilde{\gamma}Z)$, we can now write

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

This gives us estimates for $\{\delta_0, \delta_1, \beta_0, \beta_1, (\alpha_0 \rho_{\eta\zeta}\sigma_\theta), (\alpha_1 \rho_{\eta\zeta}\sigma_\theta)\}$

3.3 Step 3

Now we can use the covariances. We have

$$\begin{aligned}
\mathrm{Cov}(Y_0 - \delta_0 - \beta_0 X,\ M^A - \beta_A X^M) &= \mathrm{Cov}(Y_0, M^A|X, X^M, Z) &&= \alpha_0 \sigma_\theta^2 \\
\mathrm{Cov}(Y_0 - \delta_0 - \beta_0 X,\ M^B - \beta_B X^M) &= \mathrm{Cov}(Y_0, M^B|X, X^M, Z) &&= \alpha_0 \alpha_B \sigma_\theta^2 \\
\mathrm{Cov}(Y_1 - \delta_1 - \beta_1 X,\ M^A - \beta_A X^M) &= \mathrm{Cov}(Y_1, M^A|X, X^M, Z) &&= \alpha_1 \sigma_\theta^2 \\
\mathrm{Cov}(Y_1 - \delta_1 - \beta_1 X,\ M^B - \beta_B X^M) &= \mathrm{Cov}(Y_1, M^B|X, X^M, Z) &&= \alpha_1 \alpha_B \sigma_\theta^2 \\
\mathrm{Cov}(M^A - \beta_A X^M,\ M^B - \beta_B X^M) &= \mathrm{Cov}(M^A, M^B|X^M) &&= \alpha_B \sigma_\theta^2
\end{aligned}$$

We use these covariances to identify the rest of the model.

$$\begin{aligned}
\widehat{\alpha_0/\alpha_1} &= (\alpha_0\rho\sigma_\theta^2)/(\alpha_0\rho\sigma_\theta^2) \\
\widehat{\alpha}_B &= \operatorname{Cov}(Y_0, M^B)/\operatorname{Cov}(Y_0, M^A) \\
\widehat{\sigma}_\theta^2 &= \operatorname{Cov}(M^A, M^B)/\widehat{\alpha}_B \\
\widehat{\alpha}_0 &= \operatorname{Cov}(Y_0, M^A) \\
\widehat{\alpha}_1 &= \operatorname{Cov}(Y_1, M^B) \\
\widehat{\sigma}_A^2 &= \operatorname{Var}(M^A) - \sigma_\theta^2 \\
\widehat{\sigma}_B^2 &= \operatorname{Var}(M^B) - \widehat{\alpha}_B^2\widehat{\sigma}_\theta^2 \\
\widehat{\rho} &= \widehat{\alpha_0(\rho\sigma_\theta)}/(\widehat{\alpha}_0\sqrt{\widehat{\sigma}_\theta^2}) \\
&= \widehat{\alpha_1(\rho\sigma_\theta)}/(\widehat{\alpha}_1\sqrt{\widehat{\sigma}_\theta^2}) \\
\widehat{\sigma}_0^2 &= \operatorname{Var}(Y_0) - \widehat{\alpha_0(\rho\sigma_\theta)}^2\left[1 - t\widehat{\lambda}_0 + \widehat{\lambda}_0^2\right] - \widehat{\sigma}_\theta^2(1-\widehat{\rho}^2) \\
\widehat{\sigma}_1^2 &= \operatorname{Var}(Y_1) - \widehat{\alpha_1(\rho\sigma_\theta)}^2\left[1 - t\widehat{\lambda}_1 + \widehat{\lambda}_1^2\right] - \widehat{\sigma}_\theta^2(1-\widehat{\rho}^2) \\
\widehat{\alpha}_c &= \left\{\alpha_c \in R \,\middle|\, (\widehat{\rho}\widehat{\sigma}_\theta) - (\widehat{\alpha}_1 - \widehat{\alpha}_0 - \alpha_c)\widehat{\sigma}_\theta^2\left[(\widehat{\alpha}_1 - \widehat{\alpha}_0 - \alpha_c)^2\widehat{\sigma}_\theta^2 + 1\right]^{-1/2} = 0\right\} \\
\widehat{\gamma}_0 &= \left(\widehat{\delta}_1 - \widehat{\delta}_0\right) - \widehat{\widehat{\delta}} \times \widehat{\sigma^*} \\
\widehat{\gamma}_3 &= \left(\widehat{\beta}_1 - \widehat{\beta}_0\right) - \widehat{\widehat{\beta}} \times \widehat{\sigma^*} \\
\widehat{\gamma}_2 &= \widehat{\widehat{\gamma}} \times \widehat{\sigma^*}
\end{aligned}$$

3.4 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.

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

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.

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

We maximize the $\sum_{i=1}^N \sum_{j=1}^J w_j \mathcal{L}_i(\theta_j)\phi\left(\frac{\theta_j}{\sigma_\theta}\right)$. To update σ_θ, we do

$$
(\sigma_\theta^2)^{(t)} = \mathrm{Var}\left(\frac{(\widehat{Y}_i - Y_i) + (\widehat{M}_i^A - M_i^A) + (\widehat{M}_i^B - M_i^B)}{3}\right)
$$

4 Q4

We lose equations 1, 2 and 7. Additionally, we are more likely to observe one side of the distribution of θ now since agents select on θ. Thus, we need to include a control function in our measurement equations and the analogous component of the likelihood equation. If we do this, we still have identification because we can use

$$
\frac{\mathrm{Cov}\left[I - \widetilde{\delta} - \widetilde{\beta}x - \gamma_2 z,\ (Y_1 - X\beta_1)\right]}{\mathrm{Cov}\left[I - \widetilde{\delta} - \widetilde{\beta}x - \gamma_2 z,\ (M^A - X^M\beta_A)\right]} = \alpha_1
$$

Ta-da!

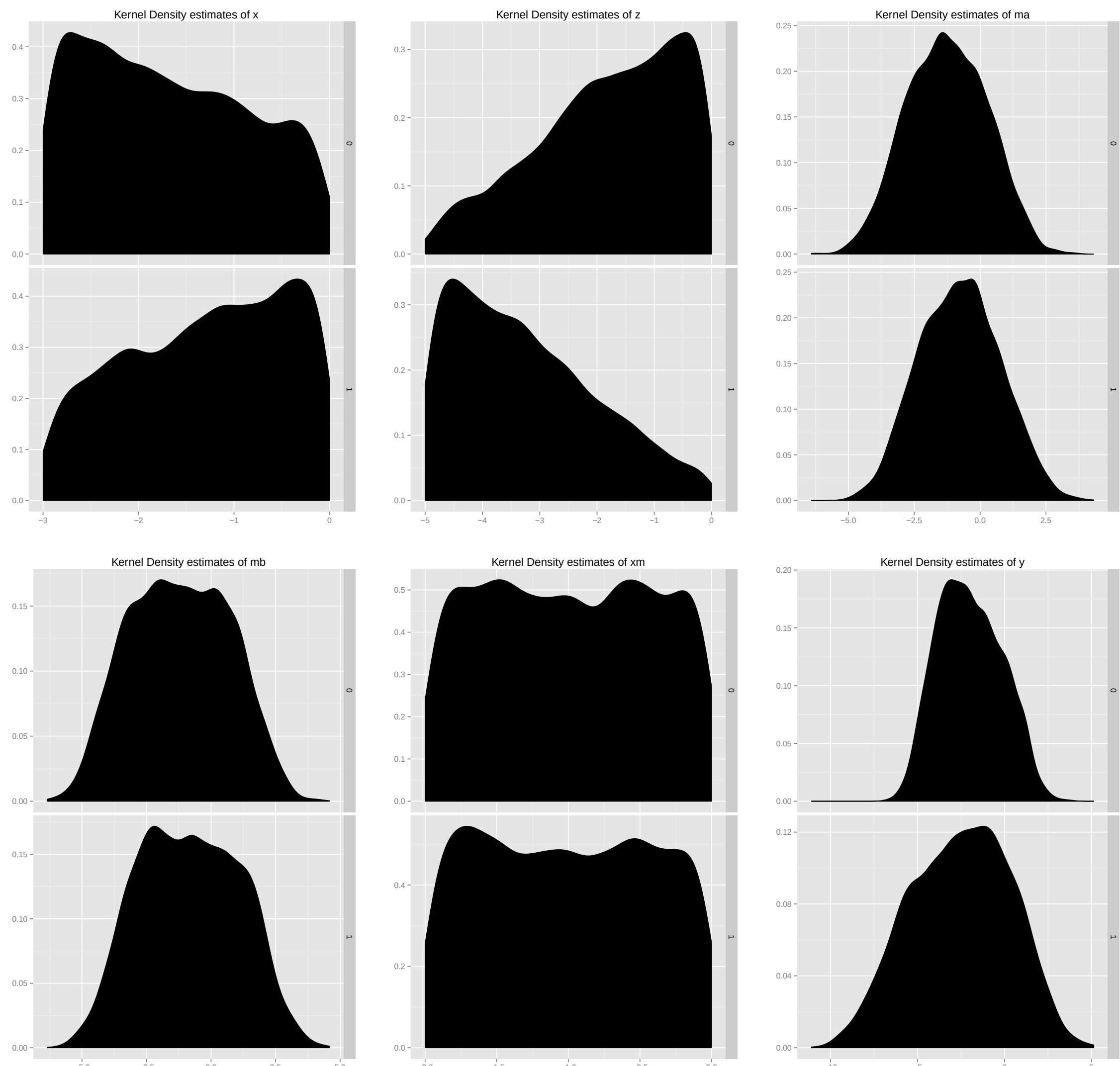

Figure 1: Histograms

5 Results

Listing 1: Results

```
------------------------------------------------------------------
Results for Homework 5
Started run at 2015-02-24T10:24:13
------------------------------------------------------------------
No outer product for probit
Two-step parameters
     δ_0    = 1.48
     β_0    = 1.97
     π_0    = 0.26

     δ_0_se = 0.04
     β_0_se = 0.01
     π_0_se = 0.03

     δ_1    = 0.98
     β_1    = 2.99
     π_1    = 0.59

     δ_1_se = 0.05
     β_1_se = 0.03
     π_1_se = 0.06

Cov of outcome and measurements from two-step
    cov_0_A = 2.71
    cov_0_B = 2.58
    cov_1_A = 2.89
    cov_1_B = 2.89

Final Parameters
  α_B    = 0.952

  σ_θ    = 1.975

  α_1    = 0.742

  α_0    = 0.695

  σ_A_sq = -1.497

  σ_B    = 0.491

  γ_0    = 0.257

  γ_1    = 0.592

  γ_3    = -0.538


  ρ_ηθ   = 0.191 # using ρ_ηθ_0

  σ_0    = [13.199]

  σ_1    = [14.127]

  α_c    = -0.051

  σ_c    = 1.0 # normalize σ_c =1


  Incidental parameters
  σ_star = 1.019

  ρ_ηθ_0 = 0.191
```

```
  ρ_ηθ_1 = 0.4
Two-step parameters
     δ_0    = 1.48
     β_0    = 1.97
     π_0    = 0.26

     δ_0_se = 0.04
     β_0_se = 0.01
     π_0_se = 0.03

     δ_1    = 0.98
     β_1    = 2.99
     π_1    = 0.59

     δ_1_se = 0.05
     β_1_se = 0.03
     π_1_se = 0.06

------------------------------------------------------------------
Finished run at 2015-02-24T10:24:14
------------------------------------------------------------------
```

6 Main call

Listing 2: Main call

```
using DataFrames
using Distributions
using Optim
using Dates
using Roots

doLog = false
originalSTDOUT = STDOUT

if doLog == true
  (outRead, outWrite) = redirect_stdout()
  println("-------------------------------------------------------------------")
  println("Results for Homework 5\nStarted run at " * string(now()))
  println("-------------------------------------------------------------------")
end

###################################################
######### 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
###################################################
dir = "C:/Users/Nick/SkyDrive/One_data/LaborEcon/PS7/"
# dir = "C:/mja3/SkyDrive/Rice/Class/econ515-Labor/PS7/"
fs  = "data_ps7_spring2015.raw"

cd(dir)
```

```
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)

cd("./code/")
include("./functions.jl")

###################################################
######### 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)
λ_1 = normpdf(t)./(1-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_1[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]

println("Two-step parameters
     δ_0    = $(round( ρ_0[1]   , 2 ))
     β_0    = $(round( ρ_0[2]   , 2 ))
     π_0    = $(round( ρ_0[3]   , 2 ))

     δ_0_se = $(round( se_ρ_0[1], 2 ))
     β_0_se = $(round( se_ρ_0[2], 2 ))
     π_0_se = $(round( se_ρ_0[3], 2 ))

     δ_1    = $(round( ρ_1[1]   , 2 ))
     β_1    = $(round( ρ_1[2]   , 2 ))
     π_1    = $(round( ρ_1[3]   , 2 ))
```

```
     δ_1_se = $(round( se_ρ_1[1], 2 ))
     β_1_se = $(round( se_ρ_1[2], 2 ))
     π_1_se = $(round( se_ρ_1[3], 2 )) \n " )

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

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

###################################################
######### 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
M_A_Xβ = convert(Array,data[:M_a])
    - vec(data[:X_m]).*β_A
M_B_Xβ = convert(Array,data[:M_b])
    - vec(data[: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)
cov_A_B = (1/N)*sum(M_A_Xβ'*M_B_Xβ)

α_B = cov_0_B/cov_0_A
σ_θ = sqrt(cov_A_B/α_B)
α_1 = cov_1_A/(σ_θ^2)
α_0 = cov_0_A/(σ_θ^2)

# σ_A = sqrt( var(convert(Array,data[:M_a])) - σ_θ^2  )
#########################################################
σ_A_sq =  var(convert(Array,data[:M_a])) - σ_θ^2  ### < NEGATIVE VARIANCE
σ_B = sqrt(  var(convert(Array,data[:M_b])) -α_B^2*σ_θ^2  )



ρ_ηθ_0 = π_0 / (α_0*σ_θ)
ρ_ηθ_1 = π_1 / (α_1*σ_θ)
########################################################
ρ_ηθ   = ρ_ηθ_0 # ρ_ηθ_1  #### < NOT THE SAME NUMEBR

δ_t_0 = λ_0[sel0]'*(λ_0[sel0] - t[sel0])
σ_0 = sqrt(
    var(convert(Array,data[sel0,:Y])) - α_0^2.*(ρ_ηθ*σ_θ)^2*
    (1 -δ_t_0) - σ_θ^2*(1-ρ_ηθ^2)
  ) # variance on income. Large

δ_t_1 = λ_1[sel1]'*(λ_1[sel1] - t[sel1])
σ_1 = sqrt(
    var(convert(Array,data[sel1,:Y])) - α_1^2.*(ρ_ηθ*σ_θ)^2*
    (1 -δ_t_1) - σ_θ^2*(1-ρ_ηθ^2)
  ) # variance on income. Large

f_c(α_c) = ρ_ηθ*σ_θ - (α_1 - α_0 - α_c)*σ_θ^2*(
  (α_1 - α_0 - α_c)^2*σ_θ^2 + 1)^(-.5)
```

```
α_c = fzero(f_c, -1000, 1000)

# normalize σ_c =1
σ_c = 1

σ_star = sqrt( (α_1 - α_0 - α_c)^2*σ_θ^2 + σ_c^2)


γ_0 = δ_1 - δ_0 - param_probit[1]*σ_star
γ_1 = β_1 - β_0 - param_probit[2]*σ_star
γ_3 = param_probit[3]*σ_star

# these are wrong b/c ignore variance of σ_star
# (b/c it looks hard)
V(kk) =[VCV_ρ_1[kk]     0           0;
    0         VCV_ρ_0[kk]       0;
    0              0     VCV_probit[kk]]
γ_0_se = [1 -1 -1]*V(1)*[1 -1 -1]'
γ_1_se = [1 -1 -1]*V(2)*[1 -1 -1]'
γ_3_se = [0  0  1]*V(3)*[0  0  1]'



println("Cov of outcome and measurements from two-step
    cov_0_A = $(round(cov_0_A, 2))
    cov_0_B = $(round(cov_0_B, 2))
    cov_1_A = $(round(cov_1_A, 2))
    cov_1_B = $(round(cov_1_B, 2)) \n")


println("Final Parameters
  α_B     = $(round(α_B    ,3))
  σ_θ     = $(round(σ_θ    ,3))
  α_1     = $(round(α_1    ,3))
  α_0     = $(round(α_0    ,3))
  σ_A_sq  = $(round(σ_A_sq,3)) squared: $(round(NaN,3))
  σ_B     = $(round(σ_B    ,3)) squared: $(round(σ_B.^2,3))
  γ_0     = $(round(γ_0    ,3))
  γ_1     = $(round(γ_1    ,3))
  γ_3     = $(round(γ_3    ,3))

  ρ_ηθ    = $(round(ρ_ηθ   ,3)) # using ρ_ηθ_0

  σ_0     = $(round(σ_0    ,3)) squared: $(round(σ_0.^2,3))
  σ_1     = $(round(σ_1    ,3)) squared: $(round(σ_1.^2,3))
  α_c     = $(round(α_c    ,3))
  σ_c     = $(round(σ_c    ,3)) # normalize σ_c =1

  Incidental parameters
  σ_star = $(round(σ_star,3))
  ρ_ηθ_0 = $(round(ρ_ηθ_0,3))
  ρ_ηθ_1 = $(round(ρ_ηθ_1,3))")




# ####################################################
# ######### 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 = []
```

```
# println("Doing EM!\n")

# # Loop w/ "while (abs( opt_out.f_minimum - opt_out_old.f_minimum  ) > ftol) || (count < maxit)
    " ?
# for i = 1:5

#   global count = 0
#   println("\n ----------Update $i: σ_θ = $(σ_θ) --------------\n\n")

#   opt_out = Optim.optimize(wtd_LL,vec(initials),
#       xtol = 1e-32,
#       ftol = 1e-32,
#       grtol = 1e-14,
#       iterations = 500,
#       autodiff=true)

#   initials = opt_out.minimum

#   println("\nResults: \t $opt_out \n\n")

#   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])]

#   # Form an updated estimate for θ_hat
#   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 = sqrt(var(θ_hat))

# println("\t
#   δ_0 = $(round(opt_out.minimum[1]  , 2))
#   δ_1 = $(round(opt_out.minimum[2]  , 2))
#   β_0 = $(round(opt_out.minimum[3]  , 2))
#   β_1 = $(round(opt_out.minimum[4]  , 2))

#   γ_0 = $(round(opt_out.minimum[5]  , 2))
#   γ_2 = $(round(opt_out.minimum[6]  , 2))
#   γ_3 = $(round(opt_out.minimum[7]  , 2))

#   α_0 = $(round(opt_out.minimum[8]  , 2))
#   α_1 = $(round(opt_out.minimum[9]  , 2))
#   α_C = $(round(opt_out.minimum[10] , 2))

#   σ_C = $(round(opt_out.minimum[11] , 2))
#   σ_1 = $(round(opt_out.minimum[12] , 2))
#   σ_2 = $(round(opt_out.minimum[13] , 2))

#   α_A = 1 (normalized)
#   β_A = $(round(opt_out.minimum[14] , 2))
#   σ_A = $(round(opt_out.minimum[15] , 2))
```

```
#   α_B = $(round(opt_out.minimum[16] , 2))
#   β_B = $(round(opt_out.minimum[17] , 2))
#   σ_B = $(round(opt_out.minimum[18] , 2))")

# end


if doLog == true

  println("--------------------------------------------------------------------")
  println("Finished run at " * string(now()))
  println("--------------------------------------------------------------------")

  close(outWrite)
  stringOut = readavailable(outRead)
  close(outRead)
  redirect_stdout(originalSTDOUT)

  f = open("../Hwk7-Results.txt", "w")
  write(f, stringOut )
  close(f)

  println(stringOut)
end
```

7 Functions and weights

Listing 3: 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
###################################################

## 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)
    out + (out .== 0.0)*eps(1.0) - (out .== 1.0)*eps(1.0)
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

###################################################
######### 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

  out = - sum(ll)
  countPlus!( out )
  return(out)
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"]
  γ_0 = out["γ_0"]
  γ_2 = out["γ_2"]
  γ_3 = out["γ_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_1)
  φ_0[sel0] = normpdf( (Y0 - X0 * [δ_0; β_0; α_0]) ./ σ_2)

  Φ_s       = normcdf(
                q.* (
                        (δ_1 - δ_0 - γ_0).*data[:C] +
                        (β_1 - β_0 - γ_3).*data[:X] +
                        (-γ_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

function countPlus!(out::Float64)
  global count += 1
  if printCounter(count)
    println("Eval $(count): value = $(round(out,5))")
  end
    return count
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]
  γ_0 = θ[5]
  γ_2 = θ[6]
  γ_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,
  "γ_0" => γ_0,
  "γ_2" => γ_2,
  "γ_3" => γ_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 4: 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]
```