

Homework 8

Labor Economics

Mark Agerton and Nick Frazier

Due Mon, April 13

1 Setup

Individual knows $\Omega_a = (x_a, y, \epsilon_a)$. His wage is

$$\log w_a^* = \alpha_1 + \alpha_2 x_a + \epsilon_a$$

We observe

$$\log w_a = \log w_a^* + \nu_a$$

Individual gets per-period payoff

$$u(\Omega_a) = \begin{cases} y + \gamma_1 + \gamma_2 y & \text{if } p_a = 0 \text{ (eg, no work)} \\ y + \exp\{f(x_a) + \epsilon_a\} & \text{if } p_a = 1 \text{ (eg, work)} \end{cases}$$

Assume iid shocks:

$$\begin{pmatrix} \epsilon_a \\ \nu_a \end{pmatrix} \sim N\left(0, \begin{bmatrix} \sigma_\epsilon^2 & 0 \\ 0 & \sigma_\nu^2 \end{bmatrix}\right)$$

Objective is

$$\max_{\{p_a\}_{a=1}^A} \beta^{a-1} E[u(\Omega)|\Omega_a]$$

2 Recursive formulation

Write problem recursively for lazy:

$$V_a^0(x, y, \epsilon_a) = \gamma_1 + (1 + \gamma_2)y + \beta E[V_{a+1}(\text{cyan}x + 1, y, \epsilon_{a+1})]$$

and working:

$$V_a^1(x, y, \epsilon_a) = \exp\{f(x_a) + \epsilon_a\} + y + \beta E[V_{a+1}(x + 1, y, \epsilon_{a+1})]$$

Value is

$$V_a(x, y, \epsilon_a) = \max\{V_a^0(x, y, \epsilon_a), V_a^1(x, y, \epsilon_a)\}$$

We normalize the value of afterlife to 0 after assuming earthly actions can't affect it

$$V_{A+1}(x, y, \epsilon) = 0$$

Let \mathcal{W}_a be the event that we work, which is

$$p_a = 1 \quad \Leftrightarrow \quad \epsilon_a \geq \underbrace{\log \left(\gamma_1 + \gamma_2 y + E[V_{a+1}^0(x, y)] - E[V_{a+1}^1(x, y)] \right) - \alpha_1 - \alpha_2 x_a}_{g(x, y, a)}$$

Then

$$\Pr(\mathcal{W}_a) = 1 - \Phi(g(x, y, a)/\sigma_\epsilon)$$

3 Backward induction

3.1 Last period

Last period's value is

$$V_A(x, y, \epsilon_A) = \max\{\gamma_1 + \gamma_2 y, \exp(\alpha_1 + \alpha_2 x + \epsilon_A)\} + y$$

Now

$$g(x, y, A) = \log(\gamma_1 + \gamma_2 y) - (\alpha_1 + \alpha_2 x)$$

so

$$\Pr(\mathcal{W}_A) = 1 - \Phi\left(\frac{g(x, y, A)}{\sigma_\epsilon}\right) = \pi(x, y, A)$$

Expected terminal value is

$$E[V_A(x, y)] = y + [1 - \pi(x, y, A)](\gamma_1 + \gamma_2 y) + \pi(x, y, A) \left[\exp\{\alpha_1 + \alpha_2 x\} \underbrace{\frac{1 - \Phi\left(\frac{g(x, y, A) - \sigma_\epsilon^2}{\sigma_\epsilon}\right)}{\pi(x, y, A)} \exp\{\frac{1}{2}\sigma_\epsilon^2\}}_{E[e_A^\epsilon | \mathcal{W}^A]} \right]$$

This can be written as

$$E[V_A(x, y)] = y + [1 - \pi(x, y, A)](\gamma_1 + \gamma_2 y) + \exp\left\{\alpha_1 + \alpha_2 x + \frac{\sigma_\epsilon^2}{2}\right\} \left[1 - \Phi\left(\frac{g(x, y, A) - \sigma_\epsilon^2}{\sigma_\epsilon}\right)\right]$$

3.2 Other periods

This means

$$g(x, y, a) = \log \left(\gamma_1 + \gamma_2 y + \beta \overbrace{E[V_{a+1}^0(x, y)] - E[V_{a+1}^1(x, y)]}^{\Delta EV(x, y, a)} \right) - \alpha_1 - \alpha_2 x_a$$

and

$$\mathcal{W}_a = \{\epsilon_a \geq g(x, y, a)\}$$

so

$$\Pr(\mathcal{W}_a) = 1 - \Phi\left(\frac{g(x, y, a)}{\sigma_\epsilon}\right) = \pi(x, y, a)$$

and

$$E[V_a(x, y)] = y + [1 - \pi(x, y, a)] \{ \gamma_1 + \gamma_2 y + E[V_{a+1}(x, y)] \} \\ + \pi(x, y, a) E[V_{a+1}(x + 1, y)] + \exp \left\{ \alpha_1 + \alpha_2 x + \frac{\sigma_\epsilon^2}{2} \right\} \left[1 - \Phi \left(\frac{g(x, y, a) - \sigma_\epsilon^2}{\sigma_\epsilon} \right) \right]$$

Note that we could simply use the general definition for V_a and $g(x, y, a)$ and specify $V_{A+1} = 0$. This would be a bit neater (ie, for each agent, have $A + 1$ periods and just say $V_{A+1} = 0$... then start recursion at $a = A$).

4 Estimation

We have states $\Omega_{ia} = (x, y, a, \epsilon)_{ia}$ and control $p_{ia} \in \{0, 1\}$.

Immediately we can get parameters governing distribution of non-labor income from a kernel density estimation of observed y_i values. Or, since we know if we know the underlying distribution we just need μ_y and σ_y which can be estimated by $N^{-1} \sum_i y_i$ and $\widehat{SE}(y_i)$. Number of periods is irrelevant and are consistent as $N \rightarrow \infty$.

Remaining parameters are

$$\theta = \{\alpha_1, \alpha_2, \gamma_1, \gamma_2, \sigma_\epsilon^2, \sigma_\nu^2\}$$

We'll need functions(?) or matrices of(?)

$$V_a^0(x, y) \quad V_a^1(x, y) \quad g(x, y, a) \quad \pi(x, y, a) \quad w(x, \epsilon; \alpha_1, \alpha_2)$$

Note that $g(x, y, a)$ is a function of a because it has $\Delta E[V_{a+1}(x, y)]$

4.1 Identification

All parameters are identified through non-linearity in Probit except σ_v which is never identified.

4.2 Plan

1. We use the fact that in period A we have $\Delta E[V_{A+1}(x, y)] = 0.0 \forall x$
2. Run probit for working in A . Recover all structural parameters.
3. Estimate wages as

$$\log w = \underbrace{\alpha_1 + \alpha_2 x + \sigma_\epsilon \lambda(y, x, a)}_{E[\log w_a^* | \mathcal{W}_a]} + \nu$$

where

$$\sigma \lambda(x, y, a) = \sigma E \left[\frac{\epsilon}{\sigma} \middle| \frac{\epsilon}{\sigma} \geq \frac{g(x, y, a)}{\sigma_\epsilon} \right] = \frac{\phi(g/\sigma)}{1 - \Phi(g/\sigma)}$$

4. Compute $E[V_A(x, y)]$ and $E[V_A(x + 1, y)]$ for all relevant x given observed work history for person i at $A - 1$.
5. Use $\Delta E[V_A(x, y)]$ to repeat procedure from Step 2.

Each period you get a set of estimates of the structural parameters from the probit and a second set of estimates for wage function parameters and σ_ϵ . So you have A sets of

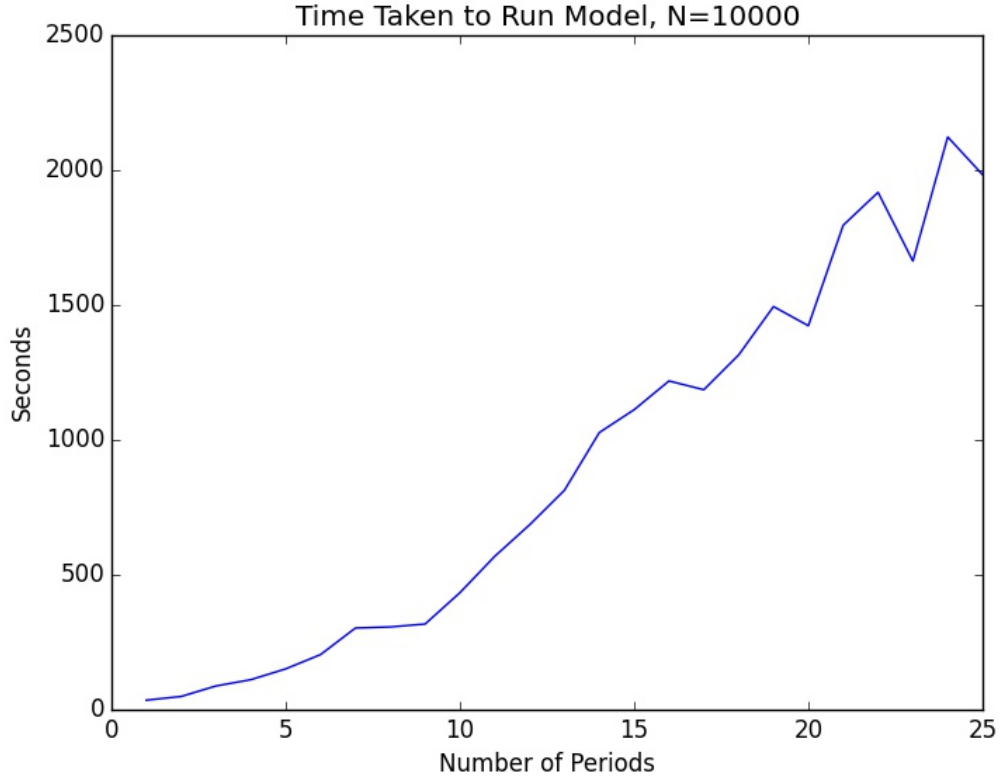
$$\hat{\theta}_{MLE} = \{\hat{\gamma}_1, \hat{\gamma}_2, \hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3, \hat{\sigma}_\epsilon\}$$

and A sets of

$$\hat{\beta}_{OLS} = \{\hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3, \hat{\sigma}_\epsilon\}$$

5 Results

First ran the model with $N = 10,000$ for 25 periods to get a sense of how long things took.



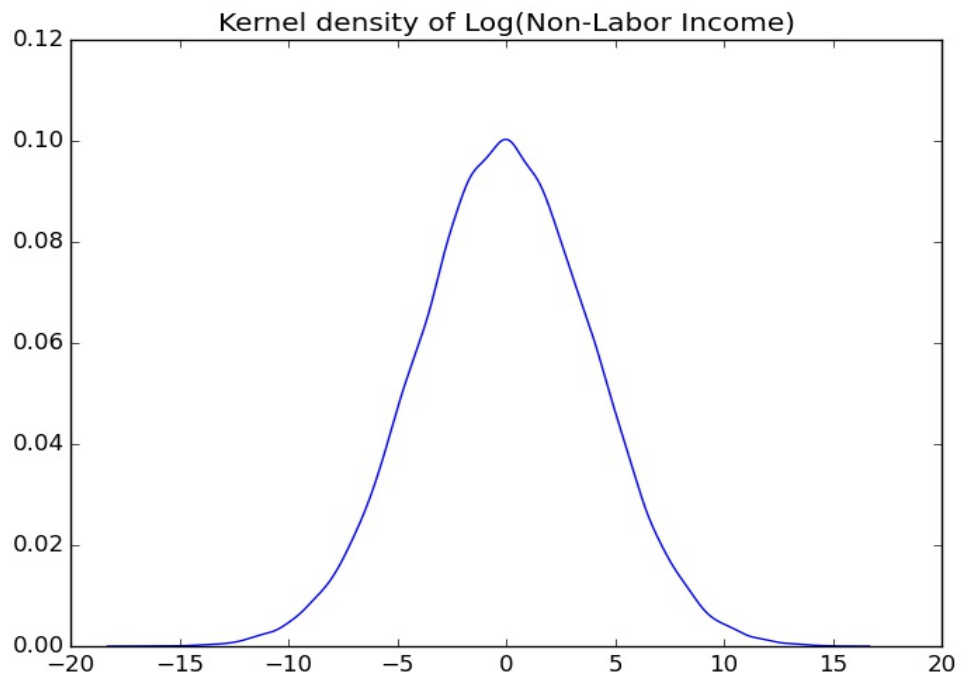
Then we upped $N = 35,000$ and ran for 11 periods to get more percision on our estimates. It took 62 minutes to run through all periods starting at $A = 11$, then $A = 10$, to $A = 1$. Estiamtes provided below. First a bit of background about our economy.

Listing 1: Functions used

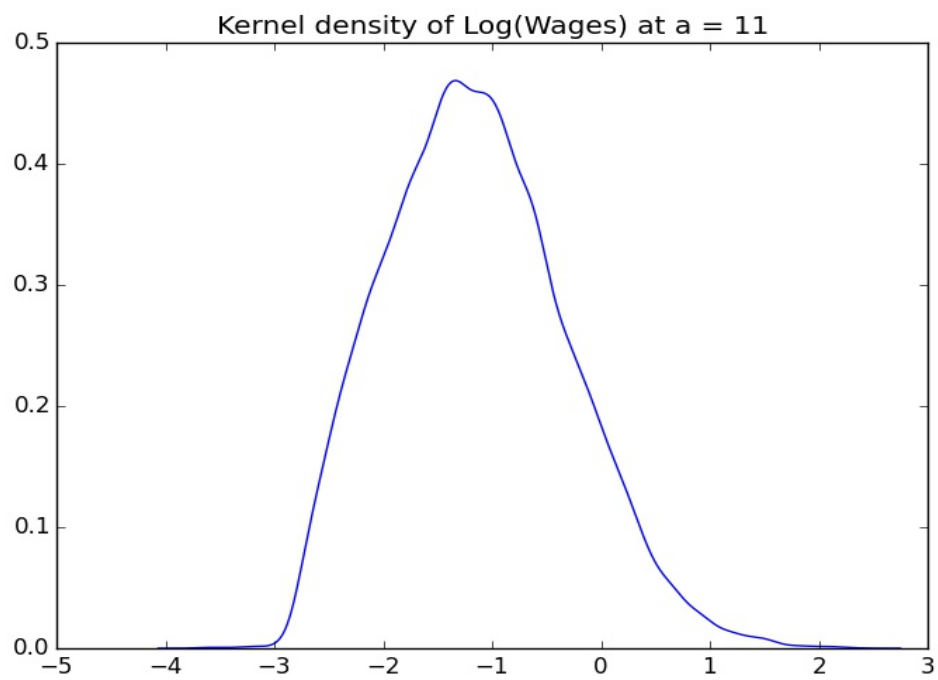
```
#####
# Specify Model Parameters
#####
# A      = 11      # Number of Periods
γ_1     = 0.300    # Leisure Coefficient
γ_2     = 0.500    # Consumption-Leisure Interaction Coefficient
δ       = 0.1      # Discount Rate
N       = 35000    # Number of Individuals
σ_e     = 1.000    # Standard Error of Wage Shock
σ_v     = 0.100    # Standard Error of Measurement Error
α_1     = 5.000    # Wage Function Parameter
α_2     = 0.500    # Wage Function Parameter
α_3     = -0.100   # wage function parameter
μ_y     = 0.0      # Mean of non-labor income
σ_y     = 2.0      # std dev of non-labor income
# yL = 0          # Minimum Non-Labor Income
# yH = 1000       # Top Non-Labor Income

θ_real = [γ_1; γ_2; α_1; α_2; α_3; σ_e]
β_real = [α_1; α_2; α_3; σ_e]
```

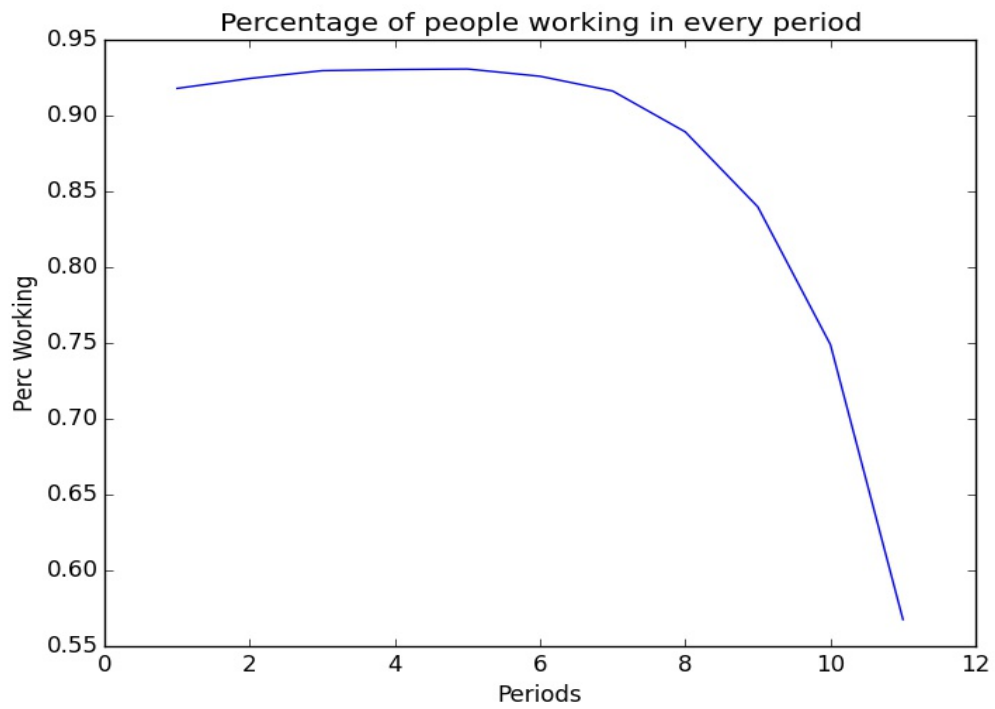
5.1 Non-labor income



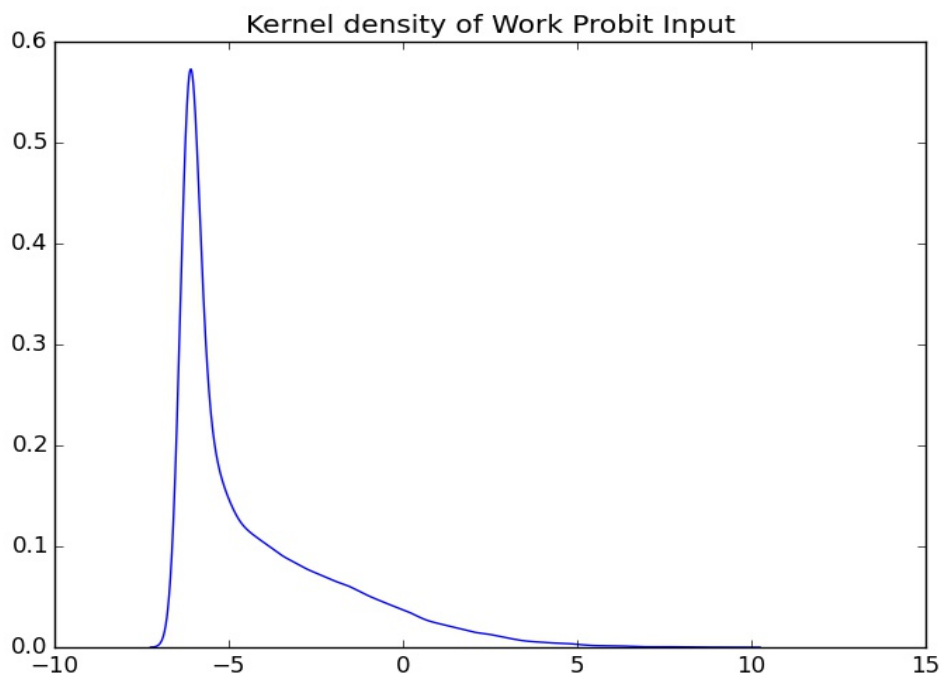
5.2 Wage Income at Period 11



5.3 Percentage working every period



5.4 Input in the Probit for decision to work



5.5 Sample Output

Listing 2: Functions used

```

Eval 3: value = 3938.49124
Eval 4: value = 3393.73034
Eval 5: value = 2335.27842
Eval 10: value = 2823.92179
Eval 20: value = 2381.45825
Eval 30: value = 2366.84353
Eval 40: value = 1.7165000000000002e54
Eval 50: value = 2335.01217
Eval 75: value = 2334.88533
Eval 100: value = 2334.88218
Eval 125: value = 2334.88033
Eval 150: value = 2334.88032
Eval 175: value = 2334.88031
Eval 200: value = 2334.88031
Eval 250: value = 2334.88031
Eval 300: value = 3018.2841
Eval 350: value = 2334.89949
Eval 400: value = 2334.88031
Eval 450: value = 2334.88031
Eval 500: value = 2343.89512
Eval 600: value = 2334.88031
Eval 700: value = 2335.3637
Eval 800: value = 2334.88031
Eval 900: value = 2334.88488
Eval 1000: value = 2334.88031
Results of Optimization Algorithm
* Algorithm: Nelder-Mead
* Starting Point:
  [0.616155778922203,0.508501689628141,5.0377041439308625,1.2182100632738138,1.198399291904949,1.0062741405941
* Minimum:
  [0.6161618008355569,0.5085065629459316,5.037713726302649,1.5515365722879093,1.5317258009190446,1.00627413594
* Value of Function at Minimum: 2334.880305
* Iterations: 91
* Convergence: true
  * |x - x'| < NaN: false
  * |f(x) - f(x')| / |f(x)| < 1.0e-12: true
  * |g(x)| < NaN: false
  * Exceeded Maximum Number of Iterations: false
* Objective Function Calls: 182
* Gradient Call: 0elapsed time: 3731.772961383 seconds
There were 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 1:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at  $\theta$  true: 2334.88

MLE Parameters:
      TRUE    ESTIMATED
 $\gamma_1$     0.3      0.616
 $\gamma_2$     0.5      0.509
 $\alpha_1$    5.0      5.038
 $\alpha_2$    0.5      1.552

```

```

α_3      -0.1    1.532
σ_e      1.0     1.006

OLS Parameters:
      TRUE    ESTIMATED
α_1      5.0     5.399
α_2      0.5     NaN
α_3      -0.1    NaN
σ_e      1.0     1.027
There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 2:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 1902.961

MLE Parameters:
      TRUE    ESTIMATED
γ_1      0.3     1.129
γ_2      0.5     0.514
α_1      5.0     5.009
α_2      0.5     0.517
α_3      -0.1    -0.092
σ_e      1.0     0.984

OLS Parameters:
      TRUE    ESTIMATED
α_1      5.0     5.604
α_2      0.5     NaN
α_3      -0.1    NaN
σ_e      1.0     0.95
There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 3:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 1687.306

MLE Parameters:
      TRUE    ESTIMATED
γ_1      0.3     -0.0
γ_2      0.5     0.503

```


α_1	5.0	4.983
α_2	0.5	0.608
α_3	-0.1	-0.141
σ_e	1.0	0.996

OLS Parameters:

	TRUE	ESTIMATED
α_1	5.0	5.589
α_2	0.5	-0.004
α_3	-0.1	0.004
σ_e	1.0	1.02

There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 4:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 1732.686

MLE Parameters:

	TRUE	ESTIMATED
γ_1	0.3	-0.0
γ_2	0.5	0.512
α_1	5.0	5.014
α_2	0.5	0.49
α_3	-0.1	-0.102
σ_e	1.0	0.993

OLS Parameters:

	TRUE	ESTIMATED
α_1	5.0	5.464
α_2	0.5	-0.09
α_3	-0.1	0.022
σ_e	1.0	0.971

There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 5:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 1853.577

MLE Parameters:

	TRUE	ESTIMATED
--	------	-----------

γ_1	0.3	2.266
γ_2	0.5	0.531
α_1	5.0	5.11
α_2	0.5	0.457
α_3	-0.1	-0.095
σ_e	1.0	0.963

OLS Parameters:

	TRUE	ESTIMATED
α_1	5.0	4.993
α_2	0.5	-0.03
α_3	-0.1	0.009
σ_e	1.0	0.98

There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked **in** period 6:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ **true**: 2222.365

MLE Parameters:

	TRUE	ESTIMATED
γ_1	0.3	0.146
γ_2	0.5	0.157
α_1	5.0	3.811
α_2	0.5	0.46
α_3	-0.1	-0.09
σ_e	1.0	1.002

OLS Parameters:

	TRUE	ESTIMATED
α_1	5.0	4.312
α_2	0.5	0.043
α_3	-0.1	-0.005
σ_e	1.0	1.014

There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked **in** period 7:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ **true**: 2950.578

```

MLE Parameters:
      TRUE  ESTIMATED
γ_1      0.3    0.677
γ_2      0.5    0.499
α_1      5.0    5.004
α_2      0.5    0.521
α_3     -0.1   -0.103
σ_e      1.0    0.977

OLS Parameters:
      TRUE  ESTIMATED
α_1      5.0    3.698
α_2      0.5   -0.026
α_3     -0.1    0.002
σ_e      1.0    0.963
There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 8:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 4320.6

MLE Parameters:
      TRUE  ESTIMATED
γ_1      0.3   -0.0
γ_2      0.5    0.435
α_1      5.0    4.817
α_2      0.5    0.532
α_3     -0.1   -0.104
σ_e      1.0    1.004

OLS Parameters:
      TRUE  ESTIMATED
α_1      5.0    2.576
α_2      0.5    0.046
α_3     -0.1   -0.006
σ_e      1.0    0.966
There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 9:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 6327.785

```

```

MLE Parameters:
      TRUE      ESTIMATED
γ_1      0.3      0.243
γ_2      0.5      0.507
α_1      5.0      5.038
α_2      0.5      0.496
α_3      -0.1     -0.1
σ_e      1.0      1.009

OLS Parameters:
      TRUE      ESTIMATED
α_1      5.0      1.452
α_2      0.5      -0.03
α_3      -0.1     0.003
σ_e      1.0      1.0
There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 10:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 9500.981

MLE Parameters:
      TRUE      ESTIMATED
γ_1      0.3      0.094
γ_2      0.5      0.148
α_1      5.0      3.97
α_2      0.5      0.444
α_3      -0.1     -0.096
σ_e      1.0      0.979

OLS Parameters:
      TRUE      ESTIMATED
α_1      5.0      0.168
α_2      0.5      -0.03
α_3      -0.1     0.001
σ_e      1.0      0.981
There where 35000 workings and 11 periods
It took 3731.773 seconds to run

Percentage that worked in period 11:
[0.92,0.92,0.93,0.93,0.93,0.93,0.92,0.89,0.84,0.75,0.57]

LLN value at θ true: 16365.872

MLE Parameters:

```

	TRUE	ESTIMATED
γ_1	0.3	0.373
γ_2	0.5	0.587
α_1	5.0	5.142
α_2	0.5	0.491
α_3	-0.1	-0.099
σ_e	1.0	0.985

OLS Parameters:

	TRUE	ESTIMATED
α_1	5.0	-1.598
α_2	0.5	-0.015
α_3	-0.1	0.001
σ_e	1.0	1.023

NumPeriods: Time Taken:
11 3731.772961383

5.6 Main Code

Listing 3: Functions used

```
# Set Working Directory
cd("C:/Users/nick/skydrive/projects/laborecon/ps8")
# Call Packages
using DataFrames
using Optim
using Distributions
using PyPlot
using KernelDensity
pwd()

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

A_min = 11
A_max = 11
time_taken = zeros(A_max)

for A in A_min:A_max

    tic()

    #####
    # Specify Model Parameters
    #####
    # A      = 11          # Number of Periods
     $\gamma_1$  = 0.300          # Leisure Coefficient
     $\gamma_2$  = 0.500          # Consumption-Leisure Interaction Coefficient
     $\delta$    = 0.1            # Discount Rate
    N      = 35000         # Number of Individuals
     $\sigma_e$  = 1.000         # Standard Error of Wage Shock
     $\sigma_v$  = 0.100         # Standard Error of Measurement Error
     $\alpha_1$  = 5.000         # Wage Function Parameter
     $\alpha_2$  = 0.500         # Wage Function Parameter
```

```

α_3 = -0.100      # wage function parameter
μ_y = 0.0         # Mean of non-labor income
σ_y = 2.0         # std dev of non-labor income
# yL = 0          # Minimum Non-Labor Income
# yH = 1000       # Top Non-Labor Income

θ_real = [γ_1; γ_2; α_1; α_2; α_3; σ_e]
β_real = [α_1; α_2; α_3; σ_e]
# srand(12345)

β = 1/(1+δ)

#####
# Start Dataset
#####

Y_n1 = exp(rand(Normal(μ_y,σ_y^2),N))
# Y_n1 = rand(Uniform(yL,yH),N)
df = DataFrame(
    ID = squeeze(kron([1:N],int(ones(A,1))),2), # ID
    A = repmat([1:A],N), # Indicate Period
    Y = squeeze(kron(Y_n1,ones(A,1)),2), # Non-Labor Income
    e = rand(Normal(0,σ_e^2),N*A), # Wage Shock
    v = rand(Normal(0,σ_v^2),N*A) # Measurement Error
)
head(df)

# generate empty value function
# pre-allocation makes code cleaner later
for tt in A+1:-1:1
    for x in 0:tt
        df[symbol("EV_$(tt)_x$(x)")] = 0.0 # what are the $$'s? can i get some? is that a
            different pkg?
        df[symbol("V_$(tt)_x$(x)")] = 0.0
        df[symbol("w_$(tt)_x$(x)")] = 0.0
        df[symbol("V_$(tt)_x$(x)_no")] = 0.0
        df[symbol("V_$(tt)_x$(x)_yes")] = 0.0
        df[symbol("p_$(tt)_x$(x)")] = 0.0
    end
end
head(df)

# fill in values
for tt in A:-1:1
    for x in 0:tt
        println("$tt and $x")

        # Convenience
        X_a = int(x.*ones(N)) # give everyone X = x at a
        E_a = (df[:e])[df[:A].==tt] # errors this period
        y = df[:Y][df[:A].== tt] # N-vec of non-labor income at a

        ##### WAGES
        w_a_x = symbol("w_$(tt)_x$(x)") # wage[ A= a, X = x]
        w_a = symbol("w_$(tt)") # observed wage at a

        ##### VALUES
        # generated before
        EV_a1_x = symbol("EV_$(tt+1)_x$(x)") # of being at a+1 with x (didn't work)
        EV_a1_x1 = symbol("EV_$(tt+1)_x$(x+1)") # being at a+1 with x+1 (did work)

        # generated here
        V_a_x = symbol("V_$(tt)_x$(x)") # being at a with x
        EV_a_x = symbol("EV_$(tt)_x$(x)") # exp of being at a with x
        V_a_x_no = symbol("V_$(tt)_x$(x)_no") # of working this period
        V_a_x_yes = symbol("V_$(tt)_x$(x)_yes") # not working this period

        ##### POLICY FUNCITON

```

```

p_a_x      = symbol("p_$(tt)_x$(x)") # actual policy function

# Add Wages to Dataset
# True Wage
df[w_a_x][df[:A]      .==tt] = wage_eqn(theta_real,X_a,E_a)

# value of not working
df[V_a_x_no][df[:A]      .==tt] = leisure_value_t(theta_real, tt) + (df[EV_a1_x])[df[:A]
].==tt]
# value of working
df[V_a_x_yes][df[:A]      .==tt] = y + wage_eqn(theta_real, X_a,E_a) + df[EV_a1_x1][df[:A]
].==tt]

# Choose between and record
df[p_a_x][df[:A]      .==tt] = df[V_a_x_no][df[:A].==tt] .< df[V_a_x_yes][df[:A]
].==tt]
(df[V_a_x])[df[p_a_x] .==false] = (df[V_a_x_no])[df[p_a_x].==false]
(df[V_a_x])[df[p_a_x] .==true] = (df[V_a_x_yes])[df[p_a_x].==true]

# calculate expected value of being at current period
# Correct expectations for selection
# Weight values by probability of occurrence (cond'l on e_{ia})
Pi = Pi_work(theta_real, X_a,tt)
(df[EV_a_x])[df[:A].==tt] =
  (1-Pi).*( leisure_value_t(theta_real, tt)
  + beta*df[EV_a1_x][df[:A].==tt] )
  + Pi.*( y + beta*df[EV_a1_x1][df[:A].==tt] )
  + exp(.5*sigma_e^2)*wage_eqn(theta_real,X_a, zeros(N)).*
  ( 1 - normcdf( (g(theta_real,X_a,tt) - sigma_e^2)./sigma_e ) )

end
end
head(df)

DATA = DataFrame(
  ID = squeeze(kron([1:N],int(ones(A,1))),2), # ID
  A = repmat([1:A],N), # Indicate Period
  Y = squeeze(kron(Y_n1,ones(A,1)),2), # Non-Labor Income
)
# generate empty observed data set
for tt in 1:A
  DATA[symbol("P_$(tt)")] = 0.0 # do they work in period a
  DATA[symbol("W_$(tt)")] = NaN # wage in period a
  DATA[symbol("X_$(tt)")] = 0.0 # experience in period a
end
head(DATA)

for jj in 1:A

  P_a = symbol("P_$(jj)") # observed decision
  W_a = symbol("W_$(jj)")
  X_a = symbol("X_$(jj)")
  X_a1 = symbol("X_$(jj+1)")

  X_vec = convert(Array{Float64},(DATA[X_a])[df[:A].== jj])
  for x in 0:jj
    # println("$jj,$x")
    tP_a = (df[symbol("p_$(jj)_x$(x)"])[ df[:A].==jj ])
    tP = DATA[P_a][df[:A].== jj]
    tP[x==X_vec] = tP_a[x==X_vec]
    DATA[P_a][df[:A].== jj] = tP

    WORKED = DATA[P_a][df[:A].== jj]
    tW_a = (df[symbol("w_$(jj)_x$(x)"])[ df[:A].==jj ])
    # don't forget measurement error, + v
    tW = DATA[W_a][df[:A].== jj] + df[:v][df[:A].== jj]

```

```

        tw[WORKED.==true] = tw_a[WORKED.==true]
        DATA[W_a][df[:A].== jj] = tw
    end

    if jj < A
        (DATA[X_a1])[df[:A].== jj+1] = X_vec + (DATA[P_a][df[:A].== jj])
    end
end
# DATA

# percentage that work in period A
perc_a = zeros(A)
for a in 1:A
    perc_a[a] = sum( DATA[symbol("P_$(a)")] [DATA[:A].== a ] )/N
end

#####
##### Estimation
#####

theta = theta_real

ntheta = length(theta_real)
nbeta = length(beta_real)

theta_MLE = zeros(A,ntheta)
beta_ols = zeros(A,nbeta)
Sigma_w = zeros(A)
Sigma_ols = zeros(nbeta,nbeta,A)

for tt in A+1:-1:1
    DATA[symbol("EV_$(tt)_x")] = 0.0
    DATA[symbol("EV_$(tt)_x1")] = 0.0
end

for tt in A:-1:1 # for every period, starting at A and working backward

    initials = ones(ntheta)
    initials = theta

    probit_opt = []
    global count = 1
    for i =1:5
        probit_opt = optimize(probit_wrapper,vec(initials),autodiff = true,
            ftol=1e-12,
            iterations = 3000)
        initials = probit_opt.minimum
    end
    show(probit_opt)

    theta_MLE[tt,:] = probit_opt.minimum

# Second stage

theta_hat = probit_opt.minimum

W_a = DATA[symbol("W_$(tt)")] [DATA[:A].==tt]
X_a = DATA[symbol("X_$(tt)")] [DATA[:A].==tt]
P_a = DATA[symbol("P_$(tt)")] [DATA[:A].==tt]

W_a = W_a[P_a.== true]
X_a = X_a[P_a.== true]

```



```

Y_mat = log(W_a)

X_mat = [ones(int(sum(P_a))) X_a X_a.^2 λ(θ_hat)[P_a==true]]

if det(X_mat'*X_mat) > 0.0
    (β_ols[tt,:], Σ_w[tt], Σ_ols[:, :, tt]) = least_sq(X_mat, Y_mat)
else
    X_mat = [ones(int(sum(P_a))) λ(θ_hat)[P_a==true]]
    (test, ~, ~) = least_sq(X_mat, Y_mat)
    β_ols[tt,:] = [test[1] NaN NaN test[2]]
end

# Calculate EV_a_x and EV_a_x1 (current period)
## assumes value for next period already exists
EV_hat(θ_hat, tt)

end

time_taken[A] = toc()

θ_names = ["γ_1" "γ_2" "α_1" "α_2" "α_3" "σ_e"]
β_names = ["α_1" "α_2" "α_3" "σ_e"]

for tt in 1:A
    println("There were $N workings and $A periods")
    println("It took $(round(time_taken[A],3)) seconds to run" )

    println("\n Percentage that worked in period $tt:")
    println("$ (round(perc_a,2)) \n")
    println("\n LLN value at θ true: $(round(probit_LL(θ_MLE[tt,:][:]),3)) \n \n")

    println("\n MLE Parameters: \n \t TRUE \t ESTIMATED")
    for ii in 1:ntheta
        println("")
        println("$θ_names[ii] \t $(round(θ_real[ii],3)) \t $(round(θ_MLE[tt,ii],3))")
    end

    println("\n OLS Parameters: \n \t TRUE \t ESTIMATED")
    for ii in 1:nbeta
        println("")
        println("$β_names[ii] \t $(round(β_real[ii],3)) \t $(round(β_ols[tt,ii],3))")
    end

end

end # end of loop over A

for ii in 1:N
    temp = 1:A
    println("NumPeriods: Time Taken:")
    println("$ (temp[ii]) \t \t $(time_taken[ii])")
end

fig5 = figure
fig5 = plot([1:11], time_taken)
fig5 = title("Time Taken to Run Model, N=$(N)")
fig5 = xlabel("Number of Periods")
fig5 = ylabel("Seconds")
savefig("./plots/time_taken.jpg")

# # map g function

```

```

# k = kde(probit_input(theta_real))

# fig2 = figure()
# fig2 = plot(k)
# fig2 = title("Kernel density of Work Probit Input")
# savefig("./plots/KdenX.jpg")

# k_y = kde(log(df[:,Y][df[:,A].== 1]))
# fig3 = figure()
# fig3 = plot(k_y)
# fig3 = title("Kernel density of Log(Non-Labor Income)")
# savefig("./plots/Yden_normal.jpg")

# fig4 = figure()
# fig4 = plot([1:A],perc_a)
# fig4 = xlabel("Periods")
# fig4 = ylabel("Perc Working")
# fig4 = title("Percentage of people working in every period")
# savefig("./plots/perc_working.jpg")

```

5.7 Functions

Listing 4: Functions used

```

function leisure_value_t(
    theta::Array{Float64},
    a::Int64)

    p_vec = unpackparams(theta)
    y_1 = p_vec["y_1"]
    y_2 = p_vec["y_2"]

    # calls df but for :Y and :A same as DATA
    y_1 + (1 + y_2).*(df[:,Y])[df[:,A] .== a]
end

function wage_eqn(
    theta::Array{Float64},
    X_a::Union{Array{Int64}, Int64, DataArray},
    e ::Union{Array{Float64}, Float64, DataArray}
)

    p_vec = unpackparams(theta)
    alpha_1 = p_vec["alpha_1"]
    alpha_2 = p_vec["alpha_2"]
    alpha_3 = p_vec["alpha_3"]

    exp( alpha_1 + alpha_2.*X_a + alpha_3.*(X_a.^2) + e )
end

function obs_wage_eqn(
    theta::Array{Float64},
    X_a::Union{Array{Int64}, Int64, DataArray},
    e ::Union{Array{Float64}, Float64, DataArray},
    v ::Union{Array{Float64}, Float64, DataArray}
)

    exp( log(wage_eqn(theta,X_a,e)) + v )
end

function g(
    theta::Array{Float64},
    X_a::Union{Array{Int64}, Int64, DataArray},

```

```

a :: Int64)

x      = unique(X_a)
EV_a1_x = symbol("EV_$(a+1)_x$(x[1])")
EV_a1_x1 = symbol("EV_$(a+1)_x$(x[1]+1)")
EV_1     = (df[EV_a1_x1])[df[:A] .== a]
EV_0     = (df[EV_a1_x])[df[:A] .== a]
y        = (df[:Y])[df[:A] .== a]

log(
    leisure_value_t(θ,a)
    - y + β*(EV_0 - EV_1) )
- wage_eqn(θ,X_a,zeros(N))
end

function π_work(
    θ::Array{Float64},
    X_a::Union(Array{Int64}, Int64, DataArray),
    a ::Int64
)

p_vec = unpackparams(θ)
o_e = p_vec["σ_e"]

1 - normcdf( g(θ,X_a,a)./σ_e )
end

function E_g_fun(
    θ::Array{Float64},
    X_a::Union(Array{Int64}, DataArray),
    tt::Int64)

p_vec = unpackparams(θ)
γ_1 = p_vec["γ_1"]
γ_2 = p_vec["γ_2"]
α_1 = p_vec["α_1"]
α_2 = p_vec["α_2"]
α_3 = p_vec["α_3"]
σ_e = p_vec["σ_e"]

X_a      = DATA[symbol("X_$(tt)")] [DATA[:A].==tt]
X_a_p1   = X_a + 1
y        = DATA[:Y][DATA[:A].== tt] # N-vec of non-labor income at a
# will update entry for current period
EV_a_x   = symbol("EV_$(tt)_x") # exp of being at a with x
EV_a_x1  = symbol("EV_$(tt)_x1") # exp of being at a with x+1
# use next period's in calculation
EV_a1_x  = symbol("EV_$(tt)_x") # exp of being at a with x
EV_a1_x1 = symbol("EV_$(tt)_x1") # exp of being at a with x+1
EV_a1_1  = (DATA[EV_a_x1])[DATA[:A] .== tt]
EV_a1_0  = (DATA[EV_a_x])[DATA[:A] .== tt]

log_term = leisure_value_t(θ,tt) - y + β*(EV_a1_0 - EV_a1_1)
log_term[log_term .<= 0] = eps()
g_fun = log( log_term ) - wage_eqn(θ,X_a,zeros(N))

end

function EV_hat(
    θ::Array{Float64},
    tt::Int64
)

p_vec = unpackparams(θ)
γ_1 = p_vec["γ_1"]
γ_2 = p_vec["γ_2"]
α_1 = p_vec["α_1"]

```

```

α_2 = p_vec["α_2"]
α_3 = p_vec["α_3"]
σ_e = p_vec["σ_e"]

X_a      = DATA[symbol("X_$(tt)")] [DATA[:A].==tt]
X_a_p1   = X_a + 1
y        = DATA[:Y][DATA[:A].== tt] # N-vec of non-labor income at a
# will update entry for current period
EV_a_x   = symbol("EV_$(tt)_x") # exp of being at a with x
EV_a_x1  = symbol("EV_$(tt)_x1") # exp of being at a with x+1
# use next period's in calculation
EV_a1_x  = symbol("EV_$(tt)_x") # exp of being at a with x
EV_a1_x1 = symbol("EV_$(tt)_x1") # exp of being at a with x+1
EV_a1_1  = (DATA[EV_a_x1])[DATA[:A] .== tt]
EV_a1_0  = (DATA[EV_a_x])[DATA[:A] .== tt]

# EV if not working in a
Π = 1 - normcdf( E_g_fun(θ,X_a,tt)./σ_e )
(DATA[EV_a_x])[DATA[:A].==tt] =
    (1-Π).*( leisure_value_t(θ,tt)
    + β*DATA[EV_a1_x][DATA[:A].==tt] )
    + Π.*( y + β*DATA[EV_a1_x1][df[:A].==tt] )
    + exp(.5*σ_e^2)*wage_eqn(θ,X_a, zeros(N)).*
    ( 1 - normcdf( (E_g_fun(θ,X_a,tt) - σ_e^2)/σ_e ) )

# EV if working in a
Π = 1 - normcdf( E_g_fun(θ,X_a,tt)./σ_e )
(DATA[EV_a_x1])[DATA[:A].==tt] =
    (1-Π).*( leisure_value_t(θ,tt)
    + β*DATA[EV_a1_x][DATA[:A].==tt] )
    + Π.*( y + β*DATA[EV_a1_x1][df[:A].==tt] )
    + exp(.5*σ_e^2)*wage_eqn(θ,X_a_p1, zeros(N)).*
    ( 1 - normcdf( (E_g_fun(θ,X_a_p1,tt) - σ_e^2)/σ_e ) )
end

function unpackparams(θ::Array{Float64})
    d = minimum(size(θ))
    θ = squeeze(θ,d)
    γ_1 = θ[1]
    γ_2 = θ[2]
    α_1 = θ[3]
    α_2 = θ[4]
    α_3 = θ[5]
    σ_e = θ[6]

    return [
        "γ_1" => γ_1,
        "γ_2" => γ_2,
        "α_1" => α_1,
        "α_2" => α_2,
        "α_3" => α_3,
        "σ_e" => σ_e
    ]
end

function least_sq(
    X::Union{Array,DataArray,Float64},
    Y::Union{Array,DataArray,Float64};
    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(1)
return β, sigma_hat, VCV
end

function λ(t)
    normpdf( probit_input(t) )./(1-normcdf(probit_input(t)))
end

```

5.8 Probit Functions

Listing 5: Functions used

```

## 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

function probit_wrapper(θ::Array{Float64})
    probit_LL(θ)
end

function probit_LL(θ::Vector{Float64})
    P = DATA[symbol("P_$(tt)")] [DATA[:A].== tt]

    g_over_sig = normcdf( probit_input(θ) )
    out = P.*log( 1 - g_over_sig ) + (1-P).*log(g_over_sig)

    # clean output
    out[isnan(out).==1] = - 1e50
    out = - sum( out )

    countPlus!(out)
    return out
end

function probit_input(θ::Array{Float64})
    p_vec = unpackparams(θ)
    y_1 = p_vec["y_1"]
    y_2 = p_vec["y_2"]

```

```

    alpha_1 = p_vec["alpha_1"]
    alpha_2 = p_vec["alpha_2"]
    alpha_3 = p_vec["alpha_3"]
    sigma_e = p_vec["sigma_e"]

    Y_a = DATA[:Y][DATA[:A].== tt]
    X_a = DATA[symbol("X_$(tt)")][DATA[:A].==tt]
    EV_x = DATA[symbol("EV_$(tt+1)_x")][DATA[:A].==tt]
    EV_x1 = DATA[symbol("EV_$(tt+1)_x1")][DATA[:A].==tt]

    term = [ones(N) Y_a]*[gamma_1; gamma_2 ] + beta*(EV_x - EV_x1)

    term[term .<= 0] = NaN

    g_over_sig = (log(term) - [ones(N) X_a X_a.^2]*[alpha_1; alpha_2;alpha_3])./sigma_e

    return g_over_sig
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
end

```

```
    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
```