problem set 4

September 16, 2020

[3]:
```
%%javascript
(function(on) {
const e=$( "<a>Setup failed</a>" );
const ns="js_jupyter_suppress_warnings";
var cssrules=$("#"+ns);
if(!cssrules.length) cssrules = $("<style id='"+ns+"' type='text/css'>div.
↪output_stderr { } </style>").appendTo("head");
e.click(function() {
    var s='Showing';
    cssrules.empty()
    if(on) {
        s='Hiding';
        cssrules.append("div.output_stderr, div[data-mime-type*='.stderr'] {␣
↪display:none; }");
    }
    e.text(s+' warnings (click to toggle)');
    on=!on;
}).click();
$(element).append(e);
})(true);
```

```
<IPython.core.display.Javascript object>
```

[1]:
```
import pandas as pd
import math
import numpy as np
import statsmodels.api as sm
from statsmodels import *
from patsy import dmatrices
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import fsolve
from numpy import *
%matplotlib inline
```

```
/srv/app/venv/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56:
FutureWarning: The pandas.core.datetools module is deprecated and will be
```

```
removed in a future version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

1 1 Production function: identification

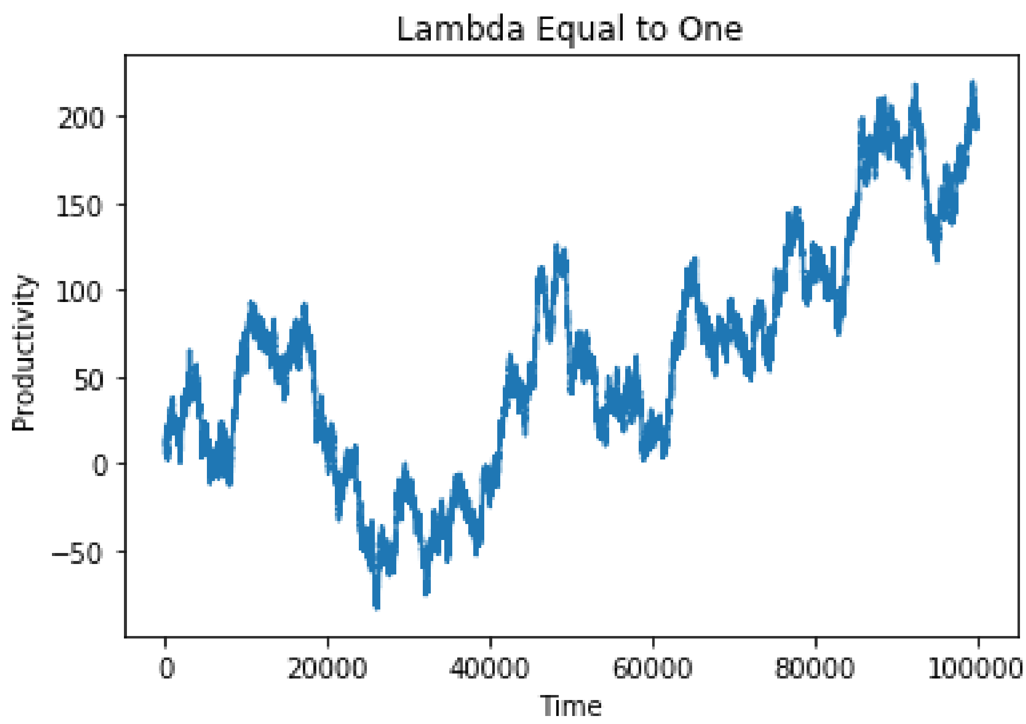
1.0.1 [a]

Assume that A is defined equal to a constant. Discuss this assumption; what does it imply about the distribution of productivity across firms when (i) $0 < \text{lambda} < 1$ and (ii) $\text{lambda} = 1$?

[46]:
```python
time = np.arange(0, 100000, 1)
production = np.arange(0, 100000, 1)
def markov_process(vals):
    a = 5
    u = a
    for i in vals:
        u = u + np.random.normal(0,0.7)
        vals[i] = u
markov_process(production)
```

[47]:
```python
lamb_eq_one = plt.plot(time, production)
plt.xlabel("Time")
plt.ylabel("Productivity")
plt.title("Lambda Equal to One")
```

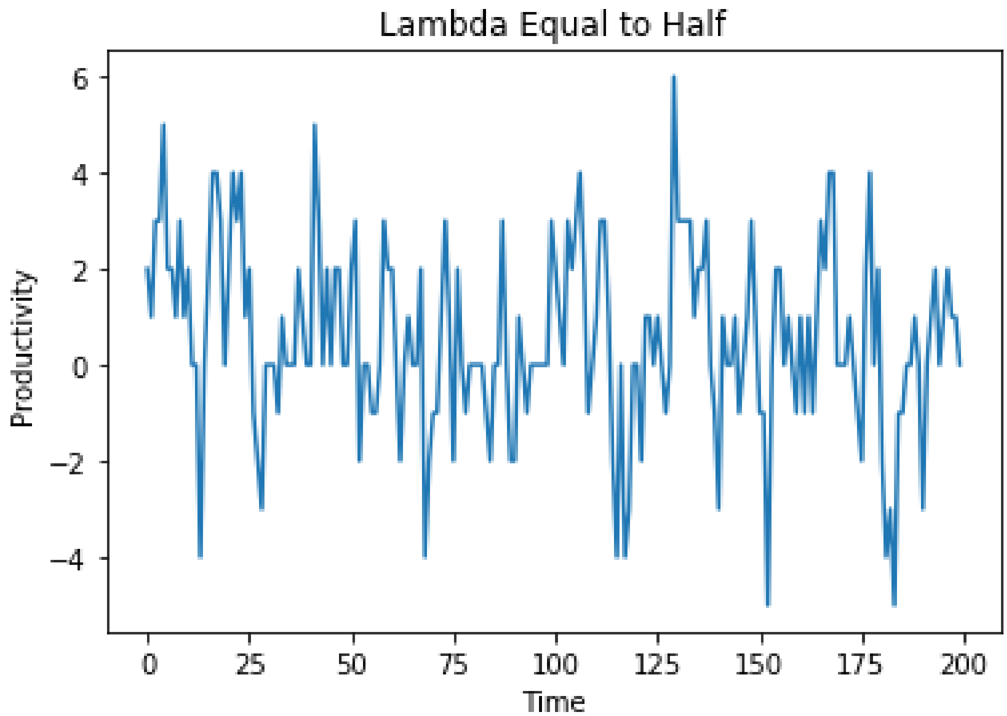
[47]:
```
Text(0.5, 1.0, 'Lambda Equal to One')
```



[48]:
```
time = np.arange(0, 200, 1)
production = np.arange(0, 200, 1)
def markov_process(vals):
    a = 5
    u = a
    for i in vals:
        u = 0.5*u + np.random.normal(0,2)
        vals[i] = u
markov_process(production)
```

[49]:
```
lamb_eq_half = plt.plot(time, production)
plt.xlabel("Time")
plt.ylabel("Productivity")
plt.title("Lambda Equal to Half")
```

[49]: Text(0.5, 1.0, 'Lambda Equal to Half')



Analysis: When lambda is equal to 1, the productivity for any given firm, i, is a random-walk during the time period T. Under this condition, the productivity for a firm is an time series integrable of order 1 (chapter one http://www.math.leidenuniv.nl/~avdvaart/timeseries/dictaat.pdf). The firms will have a massive deviation in their productivities according to this condition, as some firms will have productivity measures that tend towards infintity. On the other hand, when lambda is less than 1, the productivity for a firm i will be an integrable time series of order 0. These time series follow a finite distribution, and the deviation across firms will also be finite. These time series are said to be mean-reverting. For our purposes, when lambda is less than one, the productivity is contained in a finite interval, which will reduce variation of firm productivity. Intuitively, when lambda equals one, if U_i_t-1 was large, then U_i_t is will also be large, and each subsequent U_i_t will be large. On the other hand, large values of U_i_t-1 will eventually have no effect on the time-varying productivity for a period further in the series

1.0.2 [b]

We have,

$$\mathbb{E}[\rho(Z_{t-1}^t, \theta)|I_t] = \mathbb{E}[\eta(Z_t, \gamma) - \zeta - \lambda\eta(Z_{t-1}, \gamma)|I_t]$$

Given:

$$\theta_0$$

We have,

$$\eta(Z^t, \gamma) - \zeta - \lambda\eta(Z_{t-1}, \gamma) = \ln(Y_t) - \alpha\ln(K_t) - \beta\ln(L_t) - (1-\lambda)\kappa - \lambda(\ln(Y_{t-1}) - \alpha\ln(K_{t-1}) - \beta\ln(L_{t-1}))$$

Given:

$$\ln(Y_t) = \kappa + U_{i,t} + \alpha \ln(K_t) + \beta \ln(L_t)$$

Then:

$$\ln(Y_t)-\alpha\ln(K_t)-\beta\ln(L_t)-(1-\lambda)\kappa-\lambda(\ln(Y_{t-1})-\alpha\ln(K_{t-1})-\beta\ln(L_{t-1})) = \kappa+U_{i,t}-(1-\lambda)\kappa-\lambda(\kappa+U_{i,t-1}) = \lambda(U_{i,t-$$

So

$$\mathbb{E}[\rho(Z_{t-1}^t, \theta)|I_t] = \mathbb{E}[\epsilon_t|I_t] = 0$$

1.0.3 [c]

For T = 1, t = 0, 1

$$I_1 = (\ln(K_0^1), \ln(L_0^0), \ln(Y_0^0), \ln(U_0^0))'$$

Let,

$$V = (1, \ln(K_0), \ln(K_1), \ln(L_0), \ln(Y_0))'$$

We have, by the Law of Iterated Expectations:

$$\mathbb{E}[\rho(Z_0^1, \theta_0) * V] = \mathbb{E}[\mathbb{E}[\rho(Z_0^1, \theta_0) * V|I_1]] = \mathbb{E}[\mathbb{E}[\rho(Z_0^1, \theta_0)|I_1]V] = \mathbb{E}[\mathbb{E}[\epsilon_t|I_1]V] = \mathbb{E}[0 * V] = \mathbb{E}[0] = 0$$

1.0.4 [d]

Let,

$$\theta = \theta_0$$

$$\rho(Z_{t-2}^t, \theta) = \eta(Z_t, \gamma) - \lambda\eta(Z_{t-1}, \gamma) - [\eta(Z_{t-1}, \gamma) - \lambda\eta(Z_{t-2}, \gamma)]$$

Given:

$$\ln(Y_t) = A_i + U_{i,t} + \alpha \ln(K_t) + \beta \ln(L_t)$$

We have:

$$A_i+U_{i,t}-\lambda(A_i+U_{i,t-1})-[A_i+U_{i,t-1}-\lambda(A_i+U_{t-2})] = A_i+\lambda(U_{i,t-1})+\epsilon_t-\lambda(A_i)-\lambda(U_{i,t-1})-A_i-\lambda(U_{i,t-2})+\epsilon_{t-1}+\lambda(A_i$$

Then,

$$\mathbb{E}[\rho(Z_{t-2}^t, \theta)|I_{t-1}] = \mathbb{E}[\epsilon_t - \epsilon_{t-1}|I_{t-1}] = 0$$

Assumptions: Here, we allow for the time-invariant component of productibity to vary for each firm. This assumption should allow for a better prediction of theta, as the permanent components of productivity should vary across firms if some firms are more efficient year after year in their production methods than others. It is more plausible, in my opinion, that firms vary in their abilites to produce in more ways than simply their capital and labor inputs, and some of these differences should be consistent over time.

1.0.5 [e]

For T = 2, t = 0, 1, 2

$$I_1 = (\ln(K_0^1), \ln(L_0^0), \ln(Y_0^0), \ln(U_0^0))'$$

Let,

$$V = (1, \ln(K_0), \ln(K_1), \ln(L_0), \ln(Y_0))'$$

We have, by the Law of Iterated Expectations:

$$\mathbb{E}[\rho(Z_0^2, \theta_0)*V] = \mathbb{E}[\mathbb{E}[\rho(Z_0^2, \theta_0)*V|I_1]] = \mathbb{E}[\mathbb{E}[\rho(Z_0^2, \theta_0)|I_1]V] = \mathbb{E}[\mathbb{E}[\epsilon_2 - \epsilon_1|I_1]V] = \mathbb{E}[0*V] = \mathbb{E}[0] = 0$$

2 2 Production function: estimation

The file semiconductor_firms.out contains several thousand firm-by-year observations for a sample of publicly traded semiconductor firms (NAICS 4-digit code 3344) drawn from the S&P Capital IQ - Compustat database. The following firm attributes, measured from 1998 to 2014 inclusive, are included: gvkey – Compustat firm identification code conm – firm name year – calendar year Y – total real sales by the firm -in millions of 2009 USD- K – capital stock (in millions of 2009 USD) L – employees (in thousands) M – materials expenditures (in millions of 2009 US dollars) VA - total real valued added by the firm (in millions of 2009 USD w - annual wage rate (in 2009 USD) i – real investment (in millions of 2009 USD) naics_4digits – NAICS four digit sector code for the firm This is the same dataset you used in Problem Set 1. For this assignment, keep only those observations corresponding to 2012 (t = 0), 2013 (t = 1) and 2014 (t = 2). Further only retain “complete cases”; that is firms with information on VA,K,L in all three periods. This will constitute our estimation sample. In what follows you may treated value added as output.

[5]:
```
scf = pd.read_csv("https://raw.githubusercontent.com/bryangraham/Ec_141/master/
↪Spring2020/Problem_Sets/semiconductor_firms.out",sep='\t', encoding='utf-8')
scf.set_index(['gvkey','year'], drop=False) #Set gvkey & year as indices
scf.head()
```

[5]:

	gvkey	year	gvkey.1	conm	year.1	Y	K	L	\
0	1056	1999	1056	AEROFLEX INC	1999	92.727756	51.519396	1.10	
1	1056	2000	1056	AEROFLEX INC	2000	122.685671	106.644832	1.18	
2	1056	2001	1056	AEROFLEX INC	2001	172.101069	111.645297	1.25	
3	1056	2002	1056	AEROFLEX INC	2002	156.510389	128.098419	2.03	
4	1056	2003	1056	AEROFLEX INC	2003	237.229156	146.030959	1.86	

	M	VA	w	i	naics_4digit
0	50.190881	42.536875	23840.047219	10.488479	3344
1	68.554502	54.131169	28345.375961	8.175873	3344
2	99.452777	72.648292	31764.418883	15.032798	3344
3	72.158871	84.351518	34059.627713	6.828937	3344
4	143.830432	93.398724	37331.598845	7.077533	3344

[6]:
```
dropped_scf = scf.drop(["naics_4digit", "gvkey", "gvkey.1", "year.1", "w", "M",
 ↪"i", "Y"], axis=1)
#dropped_scf.head()
```

[7]:
```
valid_years = dropped_scf[(dropped_scf["year"]>=2012) & (dropped_scf.VA > 0)]
grouped_firms = valid_years.groupby("conm").sum()
complete_cases = grouped_firms[(grouped_firms.year >= 6039)]
cases_series = complete_cases.reset_index()["conm"]
mutable_df = pd.merge(valid_years, complete_cases, on = ["conm"])
dropped_redundant_cols = mutable_df.drop(["year_y", "K_y", "L_y", "VA_y"],
 ↪axis=1)
final_df = dropped_redundant_cols.rename(columns = {"year_x": "year", "K_x":
 ↪"K", "L_x": "L", "VA_x": "VA"})
#final_df.head()
```

[8]:
```
# relevant_firms = final_df

log_k = np.log(final_df.loc[:,"K"])
log_l = np.log(final_df.loc[:,"L"])
log_va = np.log(final_df.loc[:,"VA"])
final_df["LogK"] = log_k
final_df["LogL"] = log_l
final_df["LogVA"] = log_va

final_case = final_df
#""" Retain cases where Log of Capital, Labor, and Output are all greater than
 ↪or equal to 0 """

# filtered_firms = relevant_firms[(relevant_firms.LogK >= 0) & (relevant_firms.
 ↪LogL >= 0) & (relevant_firms.LogVA >= 0)]
```

2.1 [a] Construct a table of summary statistics for the estimation sample. How many firms are in the sample?

[8]:
```
print("There are {} firms in the sample".format(len(final_df)/3))
```

```
There are 108.0 firms in the sample
```

[9]:
```
final_df.describe([0.05, 0.25, .5, .75, .95])
```

[9]:

	year	K	L	VA	LogK	\
count	324.00000	324.000000	324.000000	324.000000	324.000000	
mean	2013.00000	1711.000030	7.471620	1009.205389	5.287482	
std	0.81776	7027.752378	19.345064	3254.850649	2.147350	
min	2012.00000	2.592858	0.026000	1.368806	0.952761	

```
5%     2012.00000      4.261548    0.086200      6.252376    1.449339
25%    2012.00000     52.603855    0.524250     38.196905    3.962788
50%    2013.00000    199.978529    1.678000    167.912837    5.298198
75%    2014.00000   1085.324020    5.682500    731.187247    6.989615
95%    2014.00000   4931.005000   31.002550   4304.193566    8.502903
max    2014.00000  72944.051984  177.000000  31566.168520   11.197448

             LogL       LogVA
count  324.000000  324.000000
mean     0.494894    5.055187
std      1.800023    2.005820
min     -3.649659    0.313939
5%      -2.451612    1.832961
25%     -0.645803    3.642736
50%      0.517517    5.123444
75%      1.737081    6.594610
95%      3.434069    8.367246
max      5.176150   10.359841
```

2.2 [b] Using the first two periods of data (i.e., t = 0; 1). Related changes in log output to changes in log inputs using OLS. Under what assumptions does this approach provide consistent capital and labor coefficients?

[21]:

```python
final_df.head()
```

[21]:

```
   year                    conm            K       L           VA      LogK  \
0  2012                AVX CORP  1953.683189  11.200  1088.912294  7.577472
1  2013                AVX CORP  1971.243363  10.800  1007.758263  7.586420
2  2014                AVX CORP  2033.063657  10.700   885.511269  7.617299
3  2012  ADVANCED MICRO DEVICES  2279.734654  10.340   873.793897  7.731814
4  2013  ADVANCED MICRO DEVICES  2302.573526  10.671   917.149964  7.741783

       LogL     LogVA
0  2.415914  6.992935
1  2.379546  6.915484
2  2.370244  6.786165
3  2.336020  6.772845
4  2.367530  6.821271
```

[22]:

```python
df2012 = final_df[final_df.year == 2012]
df2013 = final_df[final_df.year == 2013]
merged1213 = pd.merge(df2013, df2012, on="conm")
merged1213["delta_output"] = merged1213["LogVA_x"] - merged1213["LogVA_y"]
merged1213["delta_capital"] = merged1213["LogK_x"] - merged1213["LogK_y"]
merged1213["delta_labor"] = merged1213["LogL_x"] - merged1213["LogL_y"]
```

```
# y, X = dmatrices("delta_output ~ delta_capital + delta_labor",␣
↪data=merged1213, return_type = "dataframe")
# mod = sm.OLS(y, X)
# res = mod.fit()

# res.params


mod = sm.OLS(merged1213[["delta_output"]], merged1213[["delta_capital",␣
↪"delta_labor"]])
res = mod.fit()
res.params
```

[22]:
```
delta_capital   -0.220633
delta_labor      0.899177
dtype: float64
```

Assumptions: Our assumption under OLS is that capital and labor inputs satisfy no correlation between the the inputs and the noise/residuals of our regression. However, the noise of this regression will have a U_i_t-1 term, which will surely be correlated with delta inputs as firms make input choices after observing U_i_t

2.3 [c] Consider the model outlined in part [a] to [c] of question 1 above. Write a computer program that implements Algorithm 1.

[51]:
```
### This cell is for testing purposes, please ignore

### x denotes 2013, y denotes 2012

merged1213["eta_y"] = merged1213["LogVA_y"] - 0.2*merged1213["LogK_y"] - 0.
↪9*merged1213["LogL_y"]
merged1213["eta_x"] = merged1213["LogVA_x"] - 0.2*merged1213["LogK_x"] - 0.
↪9*merged1213["LogL_x"]

y, X = dmatrices("eta_x ~ eta_y", data=merged1213, return_type = "dataframe")
mod = sm.OLS(y, X)
res = mod.fit()

cee = res.params[0]
gamma = res.params[1]

moment_vector = matrix(array([[0, 0, 0, 0, 0]])).transpose()


for ind, row in merged1213.iterrows():
```

```
    arr = array([
        [1, row["LogK_y"], row["LogK_x"], row["LogL_y"], row["LogVA_y"]]]
        )
    m = matrix(arr).transpose()

    moment_vector = ((row["eta_x"] - cee - (gamma*row["eta_y"])) * m) +␣
↪moment_vector

#print(moment_vector)
sample_moment_vector = moment_vector / len(final_df) / 3
#print(sample_moment_vector)
sample_moment =  sample_moment_vector.transpose() * sample_moment_vector
sample_moment.item(0)
```

[51]: 5.006674928941445e-06

[10]:
```
def olley_pakes(df):
    alphas = np.arange(0.05, 0.96, 0.01)
    betas = np.arange(0.05, 0.96, 0.01)
    minimum = float("inf")
    optimal_alpha = 10
    optimal_beta = 10
    optimal_cee = 10
    optimal_gamma = 10
    d2012 = df[df.year == 2012]
    d2013 = df[df.year == 2013]
    m1213 = pd.merge(d2013, d2012, on="conm")
    for alpha in alphas:
        for beta in betas:
            ###x denotes 2013, y denotes 2012

            m1213["eta_y"] = m1213["LogVA_y"] - alpha*m1213["LogK_y"] -␣
↪beta*m1213["LogL_y"]
            m1213["eta_x"] = m1213["LogVA_x"] - alpha*m1213["LogK_x"] -␣
↪beta*m1213["LogL_x"]

            y, X = dmatrices("eta_x ~ eta_y", data=m1213, return_type =␣
↪"dataframe")
            mod = sm.OLS(y, X)
            res = mod.fit()

            cee = res.params[0]
            gamma = res.params[1]

            moment_vector = matrix(array([[0, 0, 0, 0, 0]])).transpose()
            for ind, row in m1213.iterrows():
                arr = array([
```

```
                    [1, row["LogK_y"], row["LogK_x"], row["LogL_y"],␣
↪row["LogVA_y"]]]
                    )
                m = matrix(arr).transpose()
                moment_vector = ((row["eta_x"] - cee - (gamma*row["eta_y"])) *␣
↪m) + moment_vector

            sample_moment_vector = moment_vector / len(final_df) / 3
            sample_moment = sample_moment_vector.transpose() *␣
↪sample_moment_vector
            #print(sample_moment.item(0))
            if sample_moment.item(0) < minimum:
                minimum = sample_moment.item(0)
                optimal_alpha = alpha
                optimal_beta = beta
                optimal_cee = cee
                optimal_gamma = gamma

    return optimal_alpha, optimal_beta, optimal_cee, optimal_gamma, minimum
```

[11]:
```
alpha, beta, cee, gamma, minimum = olley_pakes(final_df)
```

[12]:
```
print("Our estimation of theta yields alpha: {0}, beta: {1}, zeta: {2},  and␣
↪gamma: {3}".format(alpha, beta, cee, gamma))
```

```
Our estimation of theta yields alpha: 0.22000000000000003, beta:
0.8600000000000002, zeta: 0.44781505373805375,  and gamma: 0.8576808044190785
```

2.3.1 [d] Explain why (3) should be small when the estimated theta is approximately equal to the population theta. More generally give a verbal justification for the estimation procedure with reference to your theoretical analysis in the first part of the problem set. Can you think of any modifications you might like to make to your procedure? Speculate on any advantages or disadvantages of these modifications.

Explaination: if our esimated theta is approximately equal to the population values for theta, then the values for theta should satisfy the moment restriction we defined earlier. In reference to our theoretical model, we approximate the 5 x 1 moment vector by using the sample expected value of Z^t_t-1 function times the vector containing 1, LogK_0, .. LogY_0. If this vector is indeed close to 0 as the moment restriction in our theoretical model implies it will be for the population theta, then our moment vector, transpose, times itself, should yield a value of 0. Because our theoretical model satisfies the moment restriction if theta is equal to the population theta, we can take guesses for the values of theta and choose values for the capital and labor elasticities, zeta, and gamma such that (3) is satisfied. If these values satisfy (3), or are at least close, then our estimate for theta should be approximately equal to the population theta, with some standard errors, as we are computing the sample expected value. Modifying the procedure to consider more values for alpha and beta could provide a more accurate estimation for theta, but this will increase the runtime of the algorithm significantly. A different approach could be to attempt to minimize (3) using a gradient descent algorithm, where irrelvant values for alpha and beta are discarded, but I have not studied gradient descent and do not know the potential drawbacks.

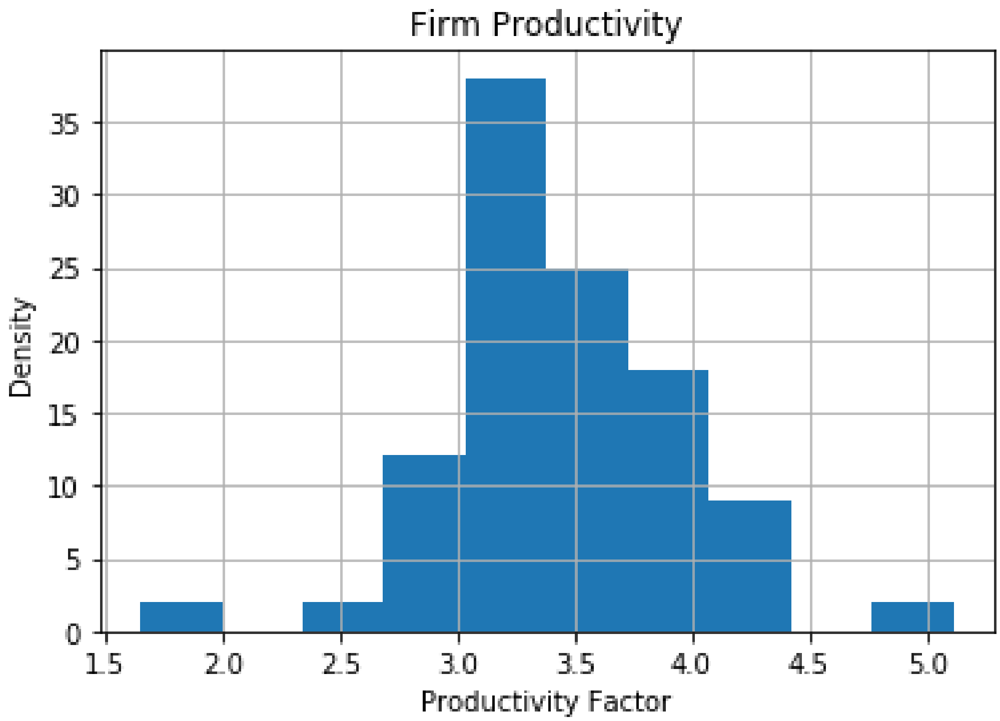
2.3.2 [e] Compute the esimated A + U_i . Plot a histogram of A + U_i. Compare your analysis with the productivity analysis you undertook in Problem Set 1. Compute the average, standard deviation and 5th, 25th, 50th, 75th and 95th percentiles of the sample distribution of A + U_i

[13]:
```
df13 = final_df[final_df.year == 2013]
df13["Firm_Productivity"] = df13["LogVA"] - alpha*df13["LogK"] -␣
 ↪beta*df13["LogL"]
histogram_2013 = df13["Firm_Productivity"].hist()
histogram_2013.set_title("Firm Productivity")
histogram_2013.set_xlabel("Productivity Factor")
histogram_2013.set_ylabel("Density")
```

```
/srv/app/venv/lib/python3.6/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

[13]:
```
Text(0, 0.5, 'Density')
```



[14]:
```
df13.drop(["year", "K", "L", "VA", "LogK", "LogL", "LogVA"], axis=1).
 ↪describe([0.05, 0.25, .5, .75, .95])
```

[14]:
```
       Firm_Productivity
count         108.000000
mean            3.441975
std             0.523972
min             1.649056
5%              2.771000
25%             3.127065
50%             3.365889
75%             3.756030
95%             4.247088
max             5.111669
```

Comparison to problem set 1: In our analysis of firm productivity in problem set one, we found that semiconductor firms had a distribution of productivies that was centered at a higher productivity, with a larger maximum value of productivity. Our estimation of alpha and beta may have been insufficient to provide accurate estimations of TFPR. After calculating the elasticies for each firm, we simply took the median elasticity for our first model, but this approach will surely not satisfy the moment restriction in our current theoretical model. Under our current model, we find a distribution of productivities that has a significantly smaller center and maximum.

2.3.3 [f] To construct standard errors for your estimate of the population theta you will use the the bootstrap procedure described in Algorithm 2. Set B = 1000 (or more!). Report your estimation results (with bootstrap standard errors) in an easy-to-read table.

To conserve time and avoid runtime errors as a result of overloading the memory capacity I set B = 100

[35]:
```
observations = [[0, 0, 0, 0]]*100

for i in np.arange(100):

    sample1213 = merged1213.sample(n=len(merged1213), replace = True)

    def modified_olley_pakes(df):
        alphas = np.arange(0.05, 0.96, 0.01)
        betas = np.arange(0.05, 0.96, 0.01)
        minimum = float("inf")
        optimal_alpha = 10
        optimal_beta = 10
        optimal_cee = 10
        optimal_gamma = 10

        for alpha in alphas:
            for beta in betas:
                ###x denotes 2013, y denotes 2012

                df["eta_y"] = df["LogVA_y"] - alpha*df["LogK_y"] -␣
↪beta*df["LogL_y"]
                df["eta_x"] = df["LogVA_x"] - alpha*df["LogK_x"] -␣
↪beta*df["LogL_x"]

                y, X = dmatrices("eta_x ~ eta_y", data=df, return_type =␣
↪"dataframe")
                mod = sm.OLS(y, X)
                res = mod.fit()

                cee = res.params[0]
                gamma = res.params[1]

                moment_vector = matrix(array([[0, 0, 0, 0, 0]])).transpose()
                for ind, row in df.iterrows():
                    arr = array([
                        [1, row["LogK_y"], row["LogK_x"], row["LogL_y"],␣
↪row["LogVA_y"]]]
                        )
                    m = matrix(arr).transpose()
```

```
                    moment_vector = ((row["eta_x"] - cee -␣
↪(gamma*row["eta_y"])) * m) + moment_vector

                sample_moment_vector = moment_vector / len(final_df) / 3
                sample_moment = sample_moment_vector.transpose() *␣
↪sample_moment_vector
                #print(sample_moment.item(0))
                if sample_moment.item(0) < minimum:
                    minimum = sample_moment.item(0)
                    optimal_alpha = alpha
                    optimal_beta = beta
                    optimal_cee = cee
                    optimal_gamma = gamma

        return optimal_alpha, optimal_beta, optimal_cee, optimal_gamma, minimum

    print(i, sep="",end="")

    alpha, beta, cee, gamma, minimum = modified_olley_pakes(sample1213)

    observations[i] = [alpha, beta, cee, gamma]
```

```
0123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899
```

[36]:
```
boot_alph = []
boot_beta = []
boot_zeta = []
boot_gamma = []
for i in np.arange(100):
    boot_alph = boot_alph + [observations[i][0]]
    boot_beta += [observations[i][1]]
    boot_zeta += [observations[i][2]]
    boot_gamma += [observations[i][3]]
```

[37]:
```
bootstraped_theta = pd.DataFrame({"alpha": boot_alph, "beta": boot_beta, "zeta":
↪ boot_zeta, "gamma": boot_gamma })
```

The statistics for each element of theta are shown below. Let the ”std” row denote the standard errors for each component of theta

[38]:
```
bootstraped_theta.describe()
```

[38]:

	alpha	beta	zeta	gamma
count	100.000000	100.00000	100.000000	100.000000
mean	0.243200	0.70670	0.416580	0.857261
std	0.148955	0.28577	0.298434	0.094786
min	0.050000	0.05000	-0.243161	0.624785
25%	0.157500	0.68000	0.265443	0.801234
50%	0.235000	0.81000	0.427173	0.843750
75%	0.310000	0.91250	0.579918	0.906175
max	0.950000	0.95000	1.121676	1.069773

Bootstrap results are displayed below

[39]:
```
bootstraped_theta.head()
```

[39]:

	alpha	beta	zeta	gamma
0	0.36	0.66	0.208990	0.906632
1	0.05	0.15	-0.059536	1.002357
2	0.14	0.92	1.060980	0.715716
3	0.28	0.75	0.417036	0.861335
4	0.16	0.18	-0.082707	1.012056

2.3.4 [g] Try to construct an estimation and inference procedure along the lines of the one outlined above, but this time appropriate for the model outlined in parts [d] and [e] of part 1 of the problem set. Carefully implement and describe your procedure. Repeat parts [e] and [f] above with your new procedure's coefficient and productivity estimates.

[9]:
```
d2 = final_df[final_df.year == 2012]
d3 = final_df[final_df.year == 2013]
d4 = final_df[final_df.year == 2014]
firstmerge = pd.merge(d3, d2, on="conm")
merged = pd.merge(d4, firstmerge, on='conm')
merged.head()
```

[9]:

	year	conm	K	L	VA	LogK	\
0	2014	AVX CORP	2033.063657	10.700	885.511269	7.617299	
1	2014	ADVANCED MICRO DEVICES	1834.114746	9.687	906.884503	7.514317	
2	2014	SKYWORKS SOLUTIONS INC	853.776069	5.550	1065.548237	6.749669	
3	2014	ANALOG DEVICES	2707.300485	9.600	1692.727551	7.903707	
4	2014	CTS CORP	301.806152	2.948	258.185766	5.709785	

	LogL	LogVA	year_x	K_x	…	LogK_x	LogL_x	\
0	2.370244	6.786165	2013	1971.243363	…	7.586420	2.379546	
1	2.270785	6.810015	2013	2302.573526	…	7.741783	2.367530	

```
2  1.713798  6.971245    2013    783.360844    …     6.663593  1.558145
3  2.261763  7.434096    2013   2630.856151    …     7.875065  2.208274
4  1.081127  5.553679    2013    385.274194    …     5.953955  1.070898

    LogVA_x  year_y          K_y     L_y         VA_y    LogK_y    LogL_y  \
0  6.915484    2012  1953.683189  11.200  1088.912294  7.577472  2.415914
1  6.821271    2012  2279.734654  10.340   873.793897  7.731814  2.336020
2  6.641923    2012   712.519183   4.700   684.488444  6.568807  1.547563
3  7.308923    2012  2523.114071   9.200  1542.455180  7.833249  2.219203
4  5.458762    2012   392.192080   4.264   287.635584  5.971752  1.450208

    LogVA_y
0  6.992935
1  6.772845
2  6.528672
3  7.341131
4  5.661694

[5 rows x 22 columns]
```

To recover gamma, we must adjust our regression.

$$\rho(Z_{t-2}^t, \theta) = \eta(Z_t, \gamma) - \lambda\eta(Z_{t-1}, \gamma) - [\eta(Z_{t-1}, \gamma) - \lambda\eta(Z_{t-2}, \gamma)] = \epsilon_t - \epsilon_{t-1}$$

Which implies,

$$\eta(Z_t, \gamma) = (1 + \lambda)\eta(Z_{t-1}, \gamma) - \lambda\eta(Z_{t-2}, \gamma) + \epsilon_t - \epsilon_{t-1}$$

However, I found this variation of the equation to be most useful for our regression:

$$\eta(Z_t, \gamma) - \eta(Z_{t-1}, \gamma) = \lambda(\eta(Z_{t-1}, \gamma) - \eta(Z_{t-2}, \gamma)) + \epsilon_t - \epsilon_{t-1}$$

So, if we regress the difference of eta at t=2 and t=1 onto the difference of eta at t=1 and t=0, we will recover gamma as the cofficient. After recovering gamma, we may construct our sample moment vector and the resulting moment restriction using the model in part 1 questions D and E. The epsilon terms are the residuals of our regression, so we need not regress onto a constant. After finding the 5 x 1 sample moment vector, we should expect condition (3) outlined in algorithm one to apply, and we will attempt to minimize (3) by trying different combinations of alpha and beta. The alpha, beta, and gamma that minimize (3) will be the components of our sampled theta. We will repeat this process by bootstraping as before and we will record the results

[12]:
```python
def final_olley_pakes(df):
        alphas = np.arange(0.05, 0.96, 0.01)
        betas = np.arange(0.05, 0.96, 0.01)
        minimum = float("inf")
        optimal_alpha = 10
        optimal_beta = 10
        optimal_gamma = 10
```

```
        for alpha in alphas:
            for beta in betas:
                ### x denotes 2013, y denotes 2012
                df["eta_y"] = df["LogVA_y"] - alpha*df["LogK_y"] -␣
↪beta*df["LogL_y"]
                df["eta_x"] = df["LogVA_x"] - alpha*df["LogK_x"] -␣
↪beta*df["LogL_x"]
                df["eta"] = df["LogVA"] - alpha*df["LogK"] - beta*df["LogL"]
                df["delta_eta_x"] = df["eta"] - df["eta_x"]
                df["delta_eta_y"] = df["eta_x"] - df["eta_y"]

                #y, X = dmatrices("eta ~ eta_x + eta_y", data=df, return_type =␣
↪"dataframe")


                #mod = sm.OLS(y, X)
                mod = sm.OLS(df[["delta_eta_x"]], df[["delta_eta_y"]])
                res = mod.fit()



                gamma = res.params[0]

                moment_vector = matrix(array([[0, 0, 0, 0, 0]])).transpose()
                for ind, row in df.iterrows():
                    arr = array([
                        [1, row["LogK_y"], row["LogK_x"], row["LogL_y"],␣
↪row["LogVA_y"]]]
                        )
                    m = matrix(arr).transpose()
                    moment_vector = ((row["eta"] - (gamma*row["eta_x"]) -␣
↪(row["eta_x"] - (gamma*row["eta_y"])))* m) + moment_vector

                sample_moment_vector = moment_vector / len(final_df) / 3
                sample_moment = sample_moment_vector.transpose() *␣
↪sample_moment_vector
                if sample_moment.item(0) < minimum:
                    minimum = sample_moment.item(0)
                    optimal_alpha = alpha
                    optimal_beta = beta
                    #optimal_cee = cee
                    optimal_gamma = gamma

        return optimal_alpha, optimal_beta, optimal_gamma, minimum
```

[13]:
```
sec_alp, sec_beta, sec_gam, sec_m = final_olley_pakes(merged)
```

[14]:
```
print(sec_m)
```

```
0.00035990703439118594
```

[15]:
```
print("Our estimation of theta yields alpha: {0}, beta: {1}, gamma: {2}".
 ↪format(sec_alp, sec_beta, sec_gam))
```

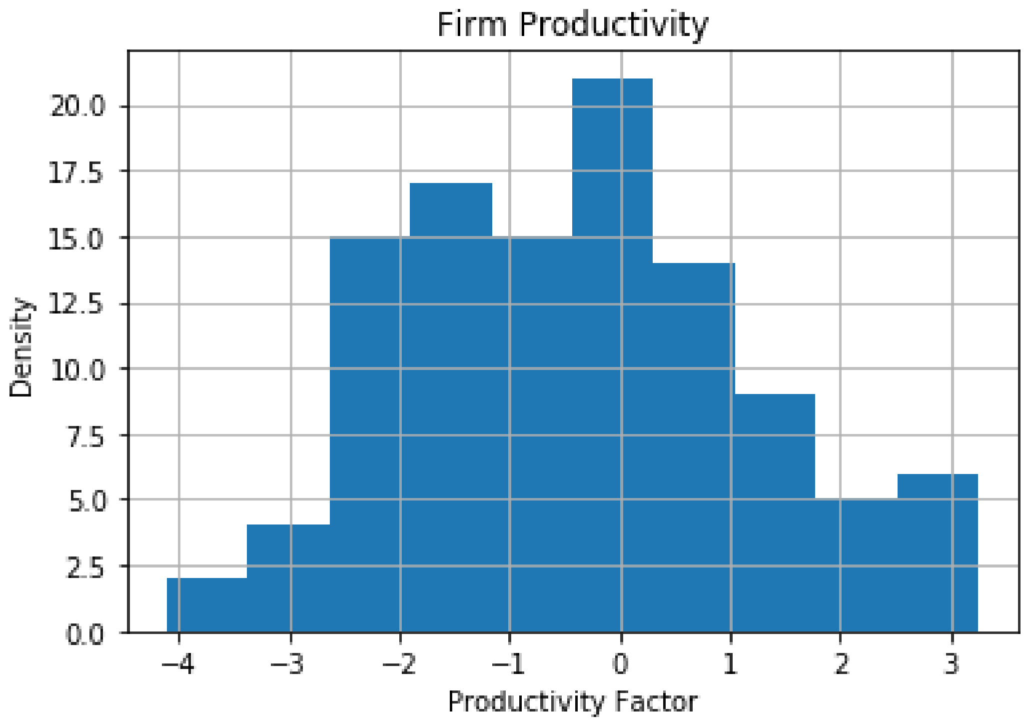
```
Our estimation of theta yields alpha: 0.9500000000000002, beta:
0.8200000000000002, gamma: -0.022632948298932174
```

[16]:
```
df14 = final_df[final_df.year == 2014]
df14["Firm_Productivity"] = df14["LogVA"] - sec_alp*df14["LogK"] -␣
 ↪sec_beta*df14["LogL"]
histogram_2014 = df14["Firm_Productivity"].hist()
histogram_2014.set_title("Firm Productivity")
histogram_2014.set_xlabel("Productivity Factor")
histogram_2014.set_ylabel("Density")
```

```
/srv/app/venv/lib/python3.6/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

[16]:
```
Text(0, 0.5, 'Density')
```



[17]:
```
df14.drop(["year", "K", "L", "VA", "LogK", "LogL", "LogVA"], axis=1).
 ↪describe([0.05, 0.25, .5, .75, .95])
```

[17]:
```
       Firm_Productivity
count         108.000000
mean           -0.420493
std             1.595876
min            -4.107152
5%             -2.712846
25%            -1.695168
50%            -0.400887
75%             0.591344
95%             2.640592
max             3.254729
```

To conserve time and avoid runtime errors as a result of overloading the memory capacity I set B = 50

[10]:
```
#### Bootstrap



observations_two = [[0, 0, 0]]*50

for i in np.arange(50):
```

```
sample14 = merged.sample(n=len(merged), replace = True)


def final_olley_pakes(df):
    alphas = np.arange(0.05, 0.96, 0.01)
    betas = np.arange(0.05, 0.96, 0.01)
    minimum = float("inf")
    optimal_alpha = 10
    optimal_beta = 10
    optimal_gamma = 10

    for alpha in alphas:
        for beta in betas:
            ### x denotes 2013, y denotes 2012
            df["eta_y"] = df["LogVA_y"] - alpha*df["LogK_y"] -␣
↪beta*df["LogL_y"]
            df["eta_x"] = df["LogVA_x"] - alpha*df["LogK_x"] -␣
↪beta*df["LogL_x"]
            df["eta"] = df["LogVA"] - alpha*df["LogK"] - beta*df["LogL"]
            df["delta_eta_x"] = df["eta"] - df["eta_x"]
            df["delta_eta_y"] = df["eta_x"] - df["eta_y"]

            #y, X = dmatrices("eta ~ eta_x + eta_y", data=df, return_type =␣
↪"dataframe")


            #mod = sm.OLS(y, X)
            mod = sm.OLS(df[["delta_eta_x"]], df[["delta_eta_y"]])
            res = mod.fit()



            gamma = res.params[0]

            moment_vector = matrix(array([[0, 0, 0, 0, 0]])).transpose()
            for ind, row in df.iterrows():
                arr = array([
                    [1, row["LogK_y"], row["LogK_x"], row["LogL_y"],␣
↪row["LogVA_y"]]]
                    )
                m = matrix(arr).transpose()
                moment_vector = ((row["eta"] - (gamma*row["eta_x"]) -␣
↪(row["eta_x"] - (gamma*row["eta_y"])))* m) + moment_vector

            sample_moment_vector = moment_vector / len(final_df) / 3
```

```
            sample_moment = sample_moment_vector.transpose() *␣
↪sample_moment_vector
            if sample_moment.item(0) < minimum:
                minimum = sample_moment.item(0)
                optimal_alpha = alpha
                optimal_beta = beta
                #optimal_cee = cee
                optimal_gamma = gamma

        return optimal_alpha, optimal_beta, optimal_gamma, minimum

    aa, bb, gg, mm = final_olley_pakes(sample14)
    print(mm, sep=" ",end=" ")
    observations_two[i] = [aa, bb, gg]
```

```
0.0003084865047897285 0.0015131205375413593 4.064936026735289e-05
7.63697094934115e-05 0.00066464328262401 7 0.0002776700357998488 6
0.00017868227092406538 0.0002322721117178823 0.001206955667652194
0.0017438594324126551 9.98664997402104e-05 0.0007939393960991834
0.00011313106648735491 0.00023875022033292902 0.0006331219916087879
0.0004134214115838068 0.0001657270016627225 0.0019040709136944286
0.00010998518639096162 0.0002942725645335712 0.00032500408737076497
0.0002585337595251297 0.0002847202687993967 6.293687818772815e-05
0.0006110978776469175 7.049689047773659e-05 7.80075358619324e-05
0.0001507218542596501 8.431847283630941e-05 0.0002031486102925567
0.00010801921675152539 0.0005954442689160408 0.0013415736942208802
0.0002944894486434328 0.0005642920195821104 7.314038799259426e-05
0.0001865351170507656 0.00010922356507317104 0.0010250487870217058
0.0011997348525836812 0.0003400879569640087 0.00034553516138629783
0.0008684639294791542 0.001458535472312506 0.0006184418464476566
0.0004522592546493084 0.0003178313062475942 0.0023546506262300078
0.003678953504400617 0.0002325465642958469 8
```

[11]:
```
### Printed the minimums above to check how my values looked and keep track of␣
↪progress
```

[12]:
```
boot_alph_t = []
boot_beta_t = []
boot_gamma_t = []
for i in np.arange(50):
    boot_alph_t = boot_alph_t + [observations_two[i][0]]
    boot_beta_t += [observations_two[i][1]]
    boot_gamma_t += [observations_two[i][2]]
```

[13]:
```
bootstraped_theta_t = pd.DataFrame({"alpha": boot_alph_t, "beta": boot_beta_t,␣
↪"gamma": boot_gamma_t })
```

[14]:
```
bootstraped_theta_t.describe()
```

[14]:

	alpha	beta	gamma
count	50.000000	50.000000	50.000000
mean	0.755200	0.583000	0.041708
std	0.290047	0.405665	0.138026
min	0.050000	0.050000	-0.223005
25%	0.685000	0.050000	-0.056781
50%	0.950000	0.850000	0.038581
75%	0.950000	0.950000	0.136858
max	0.950000	0.950000	0.453310

[15]:
```
bootstraped_theta_t.head()
```

[15]:

	alpha	beta	gamma
0	0.71	0.95	0.210361
1	0.95	0.05	0.023332
2	0.53	0.05	0.040179
3	0.43	0.05	-0.064402
4	0.95	0.20	0.000758

2.3.5 [h] What have you learned about the distribution of productivity across large U.S. semiconductor firms? What else are you interested in learning? What data/methods might help you do so?

In our first model, I found that the distribution of productivity across semiconductor firms appeared to be a normal distribution with some degree of skew-ness. In the second model, I found that the firm productivities were similarly distributed, but the second model found almost half of firm productivities in 2014 to be negative. I am uncertain if my second model is correct, as I am not convinced that this many firms would have productivity factors that reflect lower output after considering their capital and labor expendatures. I am more convinced of the first model, which describes a plausible situation where firms varied in productivity but did not demonstrate productivities that reflected overall negative revenue during the period. However, if done correctly, I think the second model best captures firm productivity, as the time-invariant component of productivity should vary from firm to firm in my opinion. The second model reflects this condition, and demonstrates a wider distribution in firm productivies which I find both plausible and interesting. I am interested in the evolution of productivity and understanding the Markov Process. I will have to read on the theory of these processes, but after reading I hope to apply these statistical methods elsewhere as I think they may be useful in applications of time series theory

[ ]:
```

```