Monetary_Code_Hasan_Cetin

February 21, 2023

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import quantecon as qe
import pandas as pd
%matplotlib inline
```

0.1 Partial Default

ECON 8701 HW2

Hasan Cetin Github Link

$$V(z,a)=\max_{a',d\in[0,1]} u\left(z\left(1-\gamma_0 d^{\gamma_1}\right)-a(1-d)+q\left(a',z\right)\left[a'-R\kappa da\right]\right)+\beta EV\left(z',a'\right)$$

The first order condition for d gives

$$d=\left(\frac{\left(1-R\kappa q\left(a',z\right)\right)a}{z\gamma_0\gamma_1}\right)^{\frac{1}{\gamma_1-1}}$$

Sketch of Algorithm 1. Set up grids for $a=[0,0.5\bar{z}]$.

2. Start with a guess of $q\left(a',z\right)=1/R$ and $V\left(z',a'\right)=u\left(z'\right)/(1-\beta)$
3. Evaluate default decision $d\left(z,a,a'\right)$ given (3) and guess $q\left(a',z\right)$. Clamp it to be bounded between 0 and 1 .
4. Maximize the objective by choosing a' given $q\left(a',z\right),V\left(z',a'\right),d\left(z,a,a'\right)$

   $$V(z,a)=\max_{a'} u\left(z\left(1-\gamma_0 d^{\gamma_1}\right)-a\left(1-d\left(z,a,a'\right)\right)+q\left(a',z\right)\left[a'-R\kappa d\left(z,a,a'\right)a\right]\right)+\beta EV\left(z',a'\right)$$

5. Recover decision rule decision rule $\mathbf{a}'(z,a)$ that results from the maximization.
6. Update price function using by applying the decision rule $a'(z,a)$ such that $\mathbf{d}(z,a)=d\left(z,a,\mathbf{a}'(z,a)\right)$ and

$$q\left(a',z\right)=\frac{1}{R}\left(1-\mathbf{d}(z,a)+R\kappa\mathbf{d}(z,a)Eq\left(\mathbf{a}''\left(z',a'\right),z'\right)\right)$$

7. Check for convergence of the value function and the price function. Go back to step 3, and continue until convergence.

[ ]:
```
 =0.954
 _0=[0.05, 0.15, 0.5]
 _1=2
R=1.04
 =0.875
 =0.0052
 =0.7
 _utility = 2

a_grid_size=90
z_grid_size= 5
a_min=0

tol = 10e-5
dist = 10e5
iteration = 0
max_iteration = 10e5
```

[ ]:
```
#Income grid and Transition matrix
Markov = qe.markov.tauchen( ,   , 0, 3, z_grid_size)
Π = Markov.P    #Transition probability
z_grid = np.exp(Markov.state_values) #z_grid
```

[ ]:
```
#Debt grid
a_max = 1/2 * np.max(z_grid)
a_grid = np.linspace(a_min,a_max,a_grid_size)  #a grid
```

[ ]:
```
np.mean(z_grid)
```

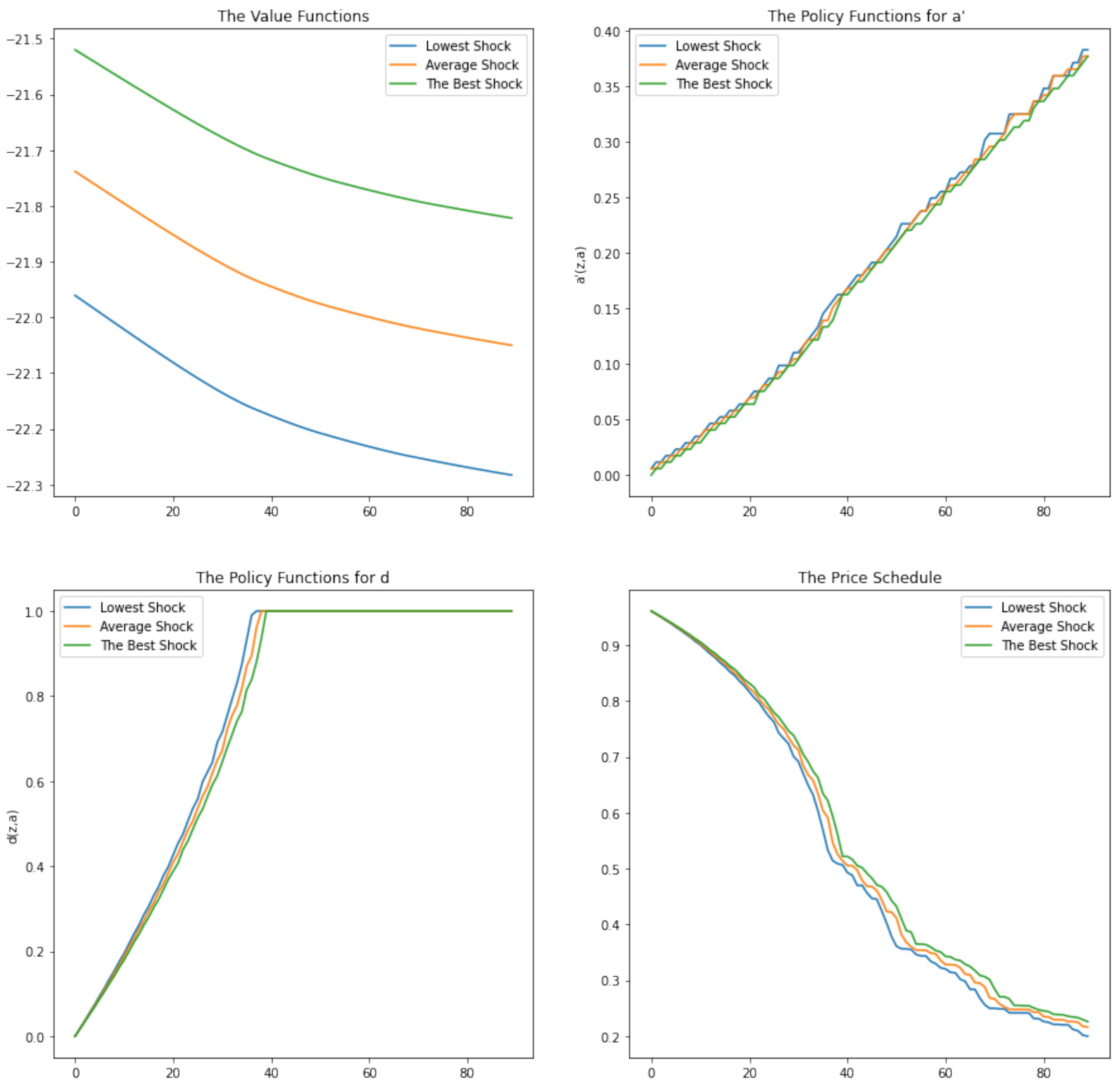
[ ]:
```
1.0002596030927233
```

[ ]:
```
debt_policies_indices = []
debt_policies = []
default_policies = []
value_functions = []
prices = []
iterations = []
distances = []

for gamma in  _0:
    print("\n gamma_0 is: ", gamma, "\n")

    #Initializing price and value functions
    V_p = np.ones((z_grid_size, a_grid_size))
    for i in range(np.shape(V_p)[0]):
        V_p[i,:] = (z_grid[i]**(1- _utility)/(1- _utility) )  /(1- )


    V_row = np.ones(a_grid_size)

    V = np.zeros((z_grid_size, a_grid_size)) # Value function

    def_pol = np.zeros((z_grid_size, a_grid_size, a_grid_size)) #default policy

    d_hat = np.empty_like(V_p) #default policy given prices

    q = np.ones((z_grid_size, a_grid_size))* (1/R) #Initial guess for prices
    q_p = q.copy()

    a_policy = np.zeros((z_grid_size, a_grid_size)) #To store optimal policy␣
↪function
    a_policy_index = np.zeros((z_grid_size, a_grid_size))

    while dist > tol and iteration < max_iteration:
        for i_z in range(z_grid_size):
            z = z_grid[i_z]
            for i_a in range(a_grid_size):
                a = a_grid[i_a]
                for i_a_p in range(a_grid_size):
                    a_p = a_grid[i_a_p]

                    def_val = (((1 - (R *   * q[i_z,i_a_p]))* a)/(z * gamma *␣
↪ _1))**(1/( _1 - 1))
                    if def_val >= 1:
                        d = 1
                    elif def_val <= 0:
                        d = 0
                    else:
                        d = def_val

                    c = z*(1-gamma* (d ** _1)) - a*(1-d) + q[i_z, i_a_p] * (a_p␣
↪- R* *d*a)

                    if c <= 0:
                        util = -10e3
                    else:
                        util = (c ** (1 -  _utility)) / (1 -  _utility)

                    V_row[i_a_p] = util +   * np.dot(Π[i_z,:], V_p[:, i_a_p])

                V[i_z,i_a] = np.nanmax(V_row)
                a_policy_index[i_z,i_a] = np.nanargmax(V_row)
                a_policy[i_z,i_a] = a_grid[int(a_policy_index[i_z,i_a])]

                def_val = (((1 - (R *   *␣
↪q[i_z,int(a_policy_index[i_z,i_a])]))* a_grid[i_a])/(z_grid[i_z] * gamma *␣
↪ _1))**(1/( _1 - 1))
                if def_val >= 1:
                    d = 1
                elif def_val <= 0:
                    d = 0
                else:
                    d = def_val

                d_hat[i_z,i_a] = d

        for i_a_p in range(a_grid_size):
            q_p_p = np.array([q[0, int(a_policy_index[0, i_a_p])], q[1,␣
↪int(a_policy_index[1, i_a_p])], q[2, int(a_policy_index[2, i_a_p])], q[3,␣
↪int(a_policy_index[3, i_a_p])], q[4, int(a_policy_index[4, i_a_p])]])
            for i_z in range(z_grid_size):
                z = z_grid[i_z]

                q_p[i_z,i_a_p] = (1/R) * np.matmul(Π[i_z, :].T, (1 - d_hat[:,␣
↪i_a_p]) + R *   * d_hat[:, i_a_p] * q_p_p)

        dist1 = np.max(np.abs(V-V_p))
        dist2 = np.max(np.abs(q-q_p))
        dist = np.max((dist1,dist2))

        q= q_p.copy()
        V_p = V.copy()

        iteration += 1

        if iteration % 50 == 0:
            print("iteration: ", iteration)
            print("dist: ", dist)

    debt_policies_indices.append(a_policy_index)
    debt_policies.append(a_policy)
    default_policies.append(d_hat)
    value_functions.append(V)
    prices.append(q)
    iterations.append(iteration)
    distances.append(dist)

    dist = 10e10
    iteration = 0
```

```
 gamma_0 is:  0.05

iteration:  50
dist:  0.0019066660073931985
iteration:  100
dist:  0.00017786052215384984

 gamma_0 is:  0.15

iteration:  50
dist:  0.0020796123102293507
iteration:  100
dist:  0.00019520035265330193

 gamma_0 is:  0.5

iteration:  50
dist:  0.0022430682595953044
iteration:  100
dist:  0.0002513633280365468
```

[ ]:
```
fig, ax = plt.subplots(2,2, figsize=(15,15))
name_list = ["Lowest Shock", 1,"Average Shock",  2,"The Best Shock"]
for i in [0,z_grid_size//2,-1]:
    fig.suptitle("For $\gamma_0 = 0.05$", fontsize=16 )
    ax[0,0].plot(value_functions[0][i,:], label = name_list[i])
    ax[0,0].set_title("The Value Functions")
    ax[0,0].legend()

    ax[0,1].plot(debt_policies[0][i,:],label=name_list[i])
    ax[0,1].set_title("The Policy Functions for a' ")
    ax[0,1].set_ylabel("a'(z,a)")
    ax[0,1].legend()

    ax[1,0].plot(default_policies[0][i,:],label=name_list[i])
    ax[1,0].set_title("The Policy Functions for d")
    ax[1,0].set_ylabel("d(z,a)")
    ax[1,0].legend()

    ax[1,1].plot(prices[0][i,:],label=name_list[i])
    ax[1,1].set_title("The Price Schedule")
    ax[1,1].legend()
```

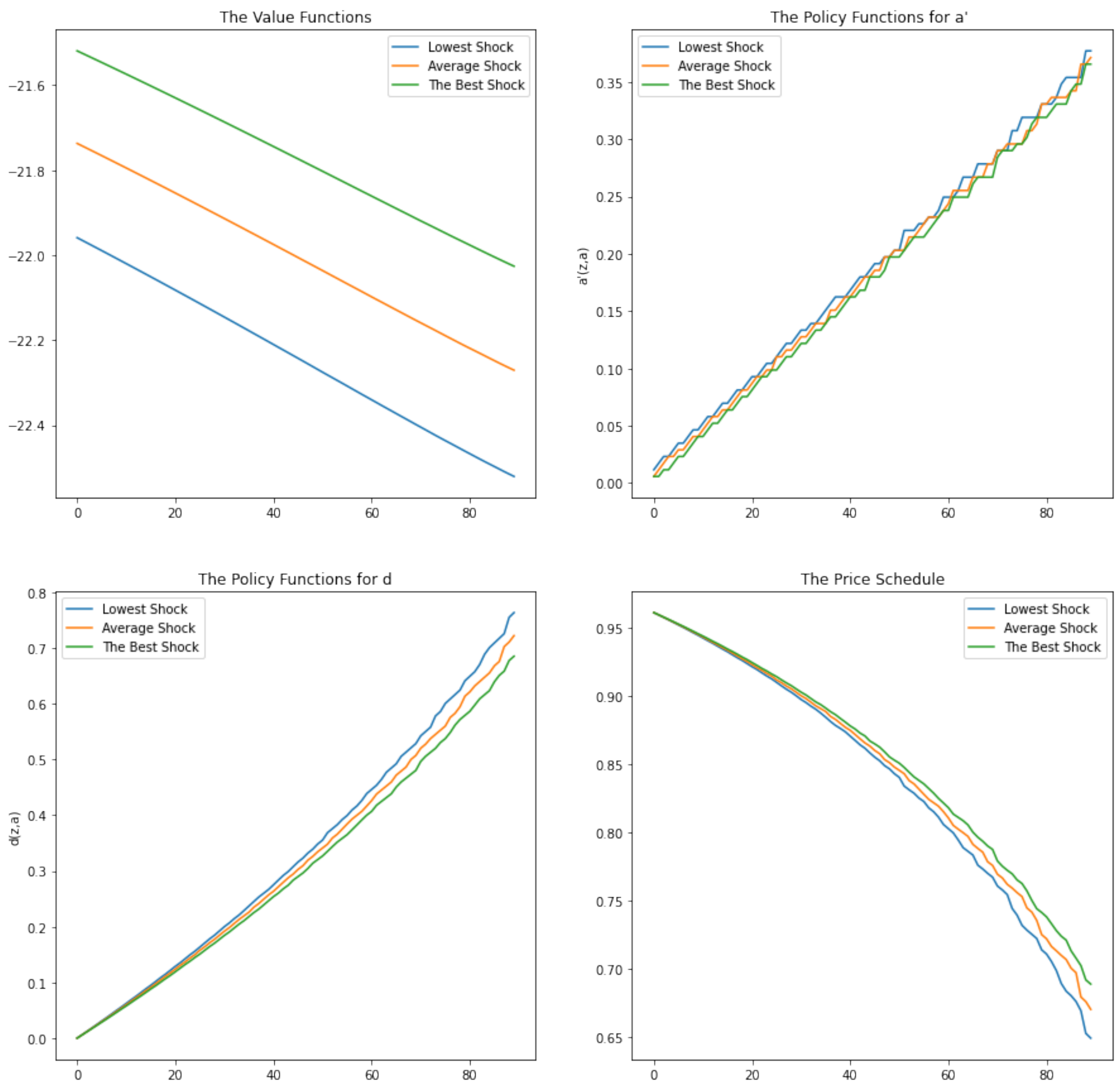
For $\gamma_0 = 0.05$



```
[ ]: fig, ax = plt.subplots(2,2, figsize=(15,15))
     name_list = ["Lowest Shock", 1,"Average Shock",  2,"The Best Shock"]
     for i in [0,2,4]:
         fig.suptitle("For $\gamma_0 = 0.15$", fontsize=16 )
         ax[0,0].plot(value_functions[1][i,:], label = name_list[i])
         ax[0,0].set_title("The Value Functions")
         ax[0,0].legend()

         ax[0,1].plot(debt_policies[1][i,:],label=name_list[i])
         ax[0,1].set_title("The Policy Functions for a'  ")
         ax[0,1].set_ylabel("a'(z,a)")
```

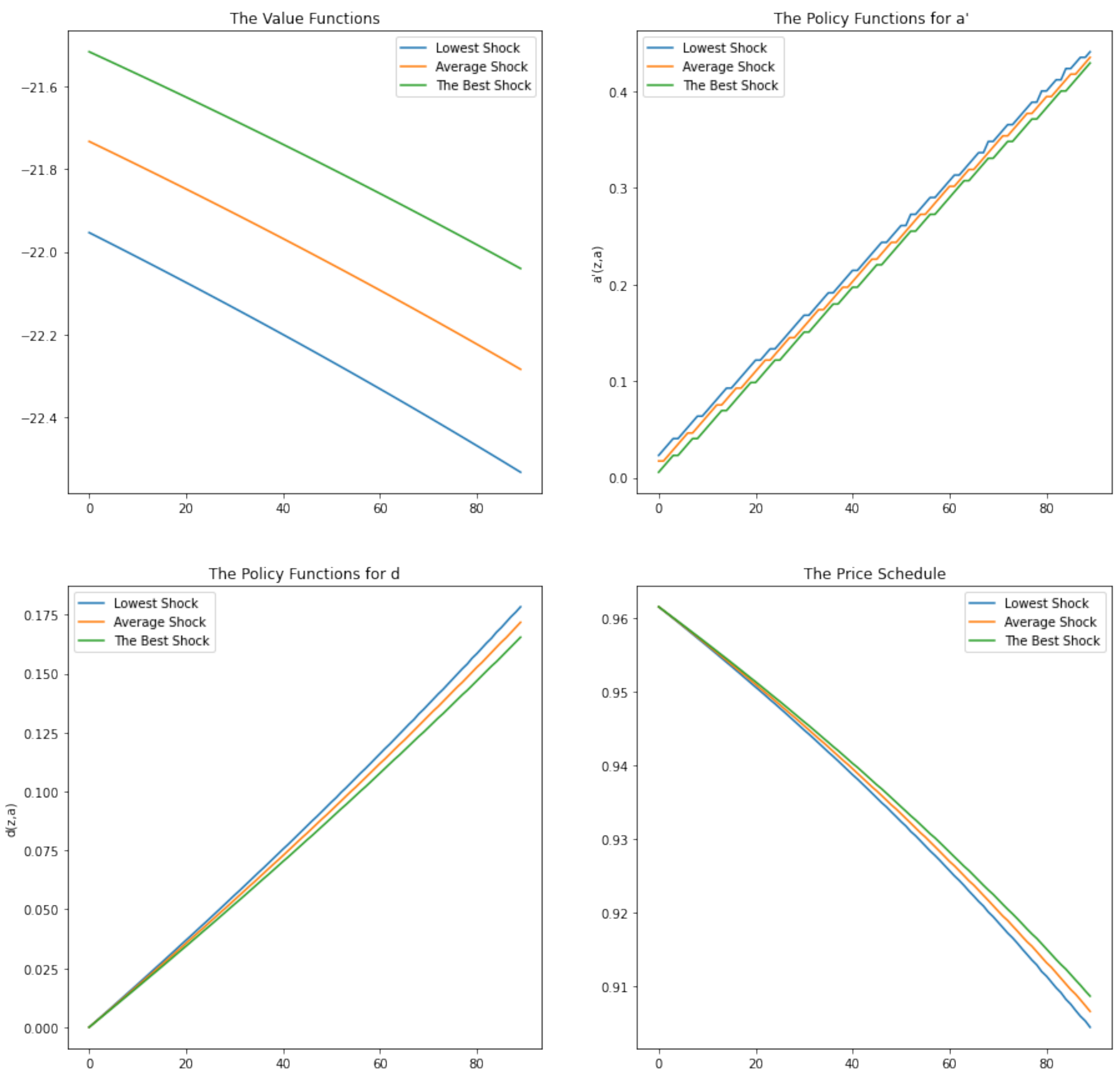
```
    ax[0,1].legend()

    ax[1,0].plot(default_policies[1][i,:],label=name_list[i])
    ax[1,0].set_title("The Policy Functions for d")
    ax[1,0].set_ylabel("d(z,a)")
    ax[1,0].legend()

    ax[1,1].plot(prices[1][i,:],label=name_list[i])
    ax[1,1].set_title("The Price Schedule")
    ax[1,1].legend()
```

For $\gamma_0 = 0.15$



```
[ ]: fig, ax = plt.subplots(2,2, figsize=(15,15))
     name_list = ["Lowest Shock", 1,"Average Shock",  2,"The Best Shock"]
     for i in [0,2,4]:
         fig.suptitle("For $\gamma_0 = 0.50$", fontsize=16 )
         ax[0,0].plot(value_functions[2][i,:], label = name_list[i])
         ax[0,0].set_title("The Value Functions")
         ax[0,0].legend()

         ax[0,1].plot(debt_policies[2][i,:],label=name_list[i])
         ax[0,1].set_title("The Policy Functions for a' ")
         ax[0,1].set_ylabel("a'(z,a)")
         ax[0,1].legend()

         ax[1,0].plot(default_policies[2][i,:],label=name_list[i])
         ax[1,0].set_title("The Policy Functions for d")
         ax[1,0].set_ylabel("d(z,a)")
         ax[1,0].legend()

         ax[1,1].plot(prices[2][i,:],label=name_list[i])
         ax[1,1].set_title("The Price Schedule")
         ax[1,1].legend()
```

For $\gamma_0 = 0.50$

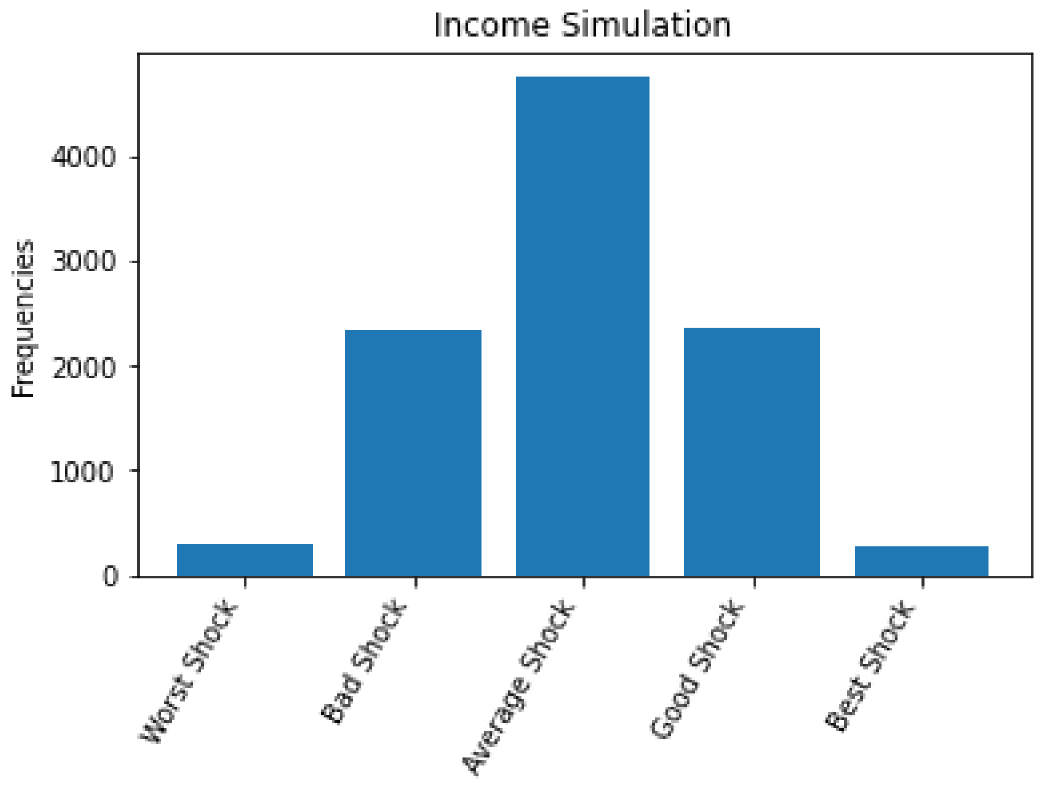


0.2 Simulation

Let's first simulate the income shocks:

```
[ ]: T = 10000 #Time period

     #Simulation of income shocks
     mc = qe.MarkovChain(Π, z_grid)
     z_sim_indices = mc.simulate_indices(T, init = np.searchsorted(z_grid, np.
      ↪mean(z_grid))  #We assumed that
```

```
                                                                      #the␣
  ↪simulation started from the middle shock


frequencies = [np.sum(z_sim_indices == 0),np.sum(z_sim_indices == 1),np.
  ↪sum(z_sim_indices == 2),np.sum(z_sim_indices == 3),np.sum(z_sim_indices ==␣
  ↪4)]
shocks = ["Worst Shock","Bad Shock","Average Shock","Good Shock","Best Shock"]

fig,ax = plt.subplots()
fig.autofmt_xdate(rotation=60)
ax.bar(shocks, frequencies)
ax.set_title("Income Simulation")
ax.set_ylabel("Frequencies")

plt.show()
```



[ ]:
```
a_sim_indices = [np.zeros(T),np.zeros(T),np.zeros(T)]
a_sim = [np.zeros(T),np.zeros(T),np.zeros(T)]
d_sim = [np.zeros(T-1),np.zeros(T-1),np.zeros(T-1)]
y_sim = [np.zeros(T),np.zeros(T),np.zeros(T)]
```

```
#Starting with debt
a_sim_indices[0][0] = 0
a_sim_indices[1][0] = 0
a_sim_indices[2][0] = 0

a_sim[0][0] = a_grid[0]
a_sim[1][0] = a_grid[0]
a_sim[2][0] = a_grid[0]

t = 0
```

```
[ ]: a_sim_indices[2]
```

```
[ ]: array([0., 0., 0., …, 0., 0., 0.])
```

```
[ ]: for i in range(3):
         t = 0
         while t < T-1:
             a_idx = int(a_sim_indices[i][t])
             z_idx = int(z_sim_indices[t])

             d_sim[i][t] = default_policies[i][z_idx, a_idx]
             a_sim_indices[i][t+1] = debt_policies_indices[i][z_idx, a_idx]
             a_sim[i][t+1] = a_grid[int(a_sim_indices[i][t+1])]

             y_sim[i][t] = z_grid[z_idx] * (1-  _0[i] * (d_sim[i][t]**( _1)))
             t = t + 1
```

```
[ ]: #Default Simulations
     default_times_l = np.sum(d_sim[0] != 0)
     default_times_m = np.sum(d_sim[1] != 0)
     default_times_h = np.sum(d_sim[2] != 0)

     d_mean_l = np.mean(d_sim[0][2000:])
     d_max_l = np.max(d_sim[0][2000:])
     d_min_l = np.min(d_sim[0][2000:])
     d_std_l = np.std(d_sim[0][2000:])

     d_mean_m = np.mean(d_sim[1][2000:])
     d_max_m = np.max(d_sim[1][2000:])
     d_min_m = np.min(d_sim[1][2000:])
     d_std_m = np.std(d_sim[1][2000:])

     d_mean_h = np.mean(d_sim[2][2000:])
     d_max_h = np.max(d_sim[2][2000:])
```

```
d_min_h = np.min(d_sim[2][2000:])
d_std_h = np.std(d_sim[2][2000:])

default_low = {" _0 ": _0[0], "default mean": d_mean_l, "default max":d_max_l,␣
 ↪"default min": d_min_l, "default std":d_std_l, "default phase":␣
 ↪default_times_l/T}
default_middle = {" _0 ": _0[1], "default mean": d_mean_m, "default max":
 ↪d_max_m, "default min": d_min_m, "default std":d_std_m, "default phase":␣
 ↪default_times_m/T}
default_high = {" _0 ": _0[2], "default mean": d_mean_h, "default max":d_max_h,␣
 ↪"default min": d_min_h, "default std":d_std_h, "default phase":␣
 ↪default_times_h/T}


#Debt Simulations
a_mean_l = np.mean(a_sim[0][2000:])
a_max_l = np.max(a_sim[0][2000:])
a_min_l = np.min(a_sim[0][2000:])
a_std_l = np.std(a_sim[0][2000:])

a_mean_m = np.mean(a_sim[1][2000:])
a_max_m = np.max(a_sim[1][2000:])
a_min_m = np.min(a_sim[1][2000:])
a_std_m = np.std(a_sim[1][2000:])

a_mean_h = np.mean(a_sim[2][2000:])
a_max_h = np.max(a_sim[2][2000:])
a_min_h = np.min(a_sim[2][2000:])
a_std_h = np.std(a_sim[2][2000:])

debt_low = {" _0 ": _0[0], "debt mean": a_mean_l, "debt max":a_max_l, "debt min":
 ↪ a_min_l, "debt std":a_std_l}
debt_middle = {" _0 ": _0[1], "debt mean": a_mean_m, "debt max":a_max_m, "debt␣
 ↪min": a_min_m, "debt std":a_std_m}
debt_high = {" _0 ": _0[2], "debt mean": a_mean_h, "debt max":a_max_h, "debt␣
 ↪min": a_min_h, "debt std":a_std_h}

#Income Simulations
y_mean_l = np.mean(y_sim[0][2000:-1])
y_max_l = np.max(y_sim[0][2000:-1])
y_min_l = np.min(y_sim[0][2000:-1])
y_std_l = np.std(y_sim[0][2000:-1])

y_mean_m = np.mean(y_sim[1][2000:-1])
y_max_m = np.max(y_sim[1][2000:-1])
y_min_m = np.min(y_sim[1][2000:-1])
```

```
y_std_m = np.std(y_sim[1][2000:-1])

y_mean_h = np.mean(y_sim[2][2000:-1])
y_max_h = np.max(y_sim[2][2000:-1])
y_min_h = np.min(y_sim[2][2000:-1])
y_std_h = np.std(y_sim[2][2000:-1])

income_low = {" _0 ": _0[0], "income mean": y_mean_l, "income max": y_max_l,␣
 ↪"income min": y_min_l, "income std":y_std_l}
income_middle = {" _0 ": _0[1], "income mean": y_mean_m, "income max":y_max_m,␣
 ↪"income min": y_min_m, "income std":y_std_m}
income_high = {" _0 ": _0[2], "income mean": y_mean_h, "income max":y_max_h,␣
 ↪"income min": y_min_h, "income std":y_std_h}

income_list = [income_low, income_middle, income_high]
debt_list = [debt_low, debt_middle, debt_high]
default_list = [default_low, default_middle, default_high]

df_income = pd.DataFrame(income_list)
df_debt = pd.DataFrame(debt_list)
df_default = pd.DataFrame(default_list)
```

[ ]:
```
df_income
```

[ ]:
```
     _0   income mean  income max  income min  income std
0  0.05      1.000282    1.032733    0.968224    0.013507
1  0.15      1.000214    1.032728    0.968093    0.013520
2  0.50      0.999983    1.032675    0.967562    0.013587
```

[ ]:
```
df_debt
```

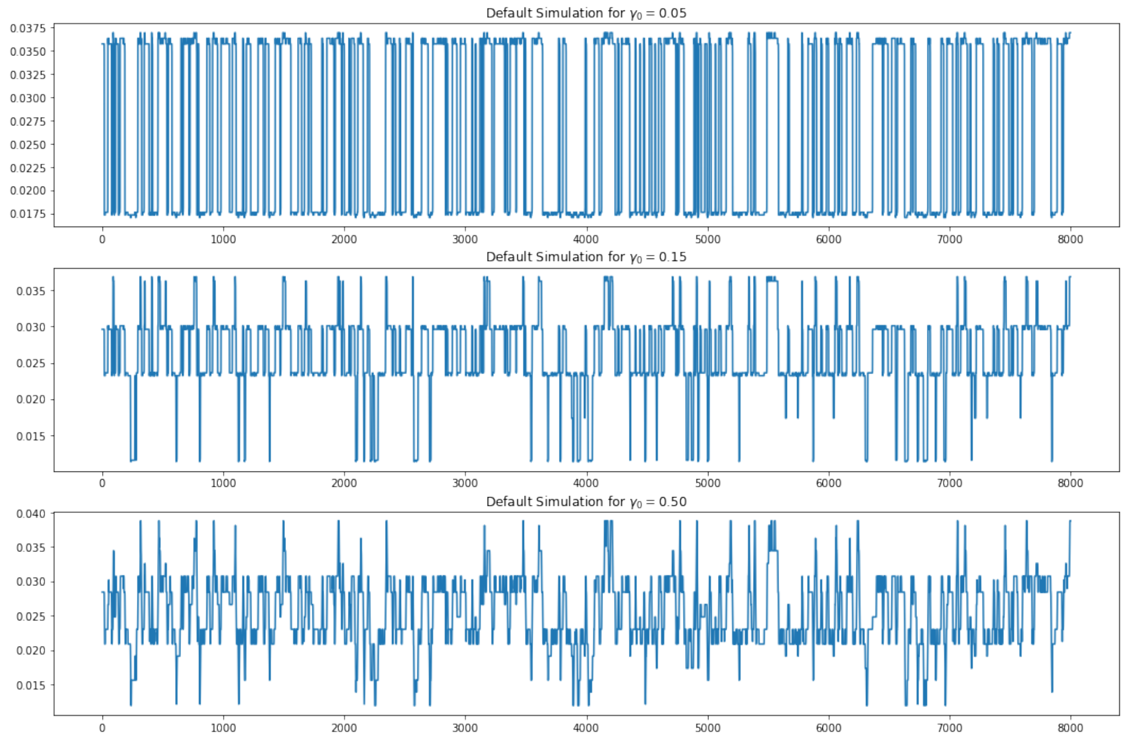
[ ]:
```
     _0   debt mean  debt max  debt min  debt std
0  0.05    0.008601  0.011604  0.005802  0.002899
1  0.15    0.025648  0.034812  0.011604  0.005036
2  0.50    0.083430  0.121841  0.040614  0.015259
```

[ ]:
```
df_default
```

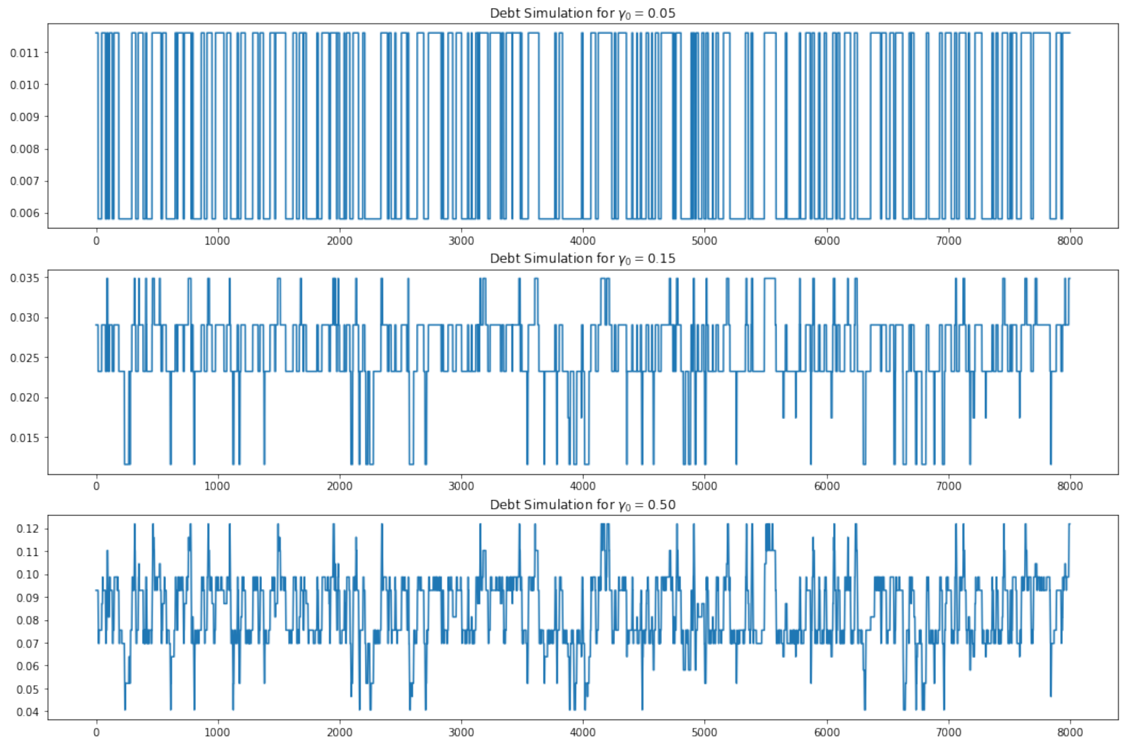
[ ]:
```
     _0   default mean  default max  default min  default std  default phase
0  0.05       0.026417     0.036901     0.017058     0.009264         0.9998
1  0.15       0.026190     0.036901     0.011326     0.005446         0.9998
2  0.50       0.025544     0.038799     0.011897     0.005012         0.9998
```

[ ]:
```
fig, ax = plt.subplots(3,1,figsize=(18,12))

ax[0].plot(d_sim[0][2000:])
ax[0].set_title("Default Simulation for $\gamma_0 = 0.05$")
```

```
ax[1].plot(d_sim[1][2000:])
ax[1].set_title("Default Simulation for $\gamma_0 = 0.15$")
ax[2].plot(d_sim[2][2000:])
ax[2].set_title("Default Simulation for $\gamma_0 = 0.50$")
plt.show()
```



[ ]:
```
fig, ax = plt.subplots(3,1,figsize=(18,12))

ax[0].plot(a_sim[0][2000:])
ax[0].set_title("Debt Simulation for $\gamma_0 = 0.05$")
ax[1].plot(a_sim[1][2000:])
ax[1].set_title("Debt Simulation for $\gamma_0 = 0.15$")
ax[2].plot(a_sim[2][2000:])
ax[2].set_title("Debt Simulation for $\gamma_0 = 0.50$")
plt.show()
```



0.3 Discussion

First of all, we can see from the value function graphs that the value function is decreasing over the debt holding. I did three cases: Small, average and big default income punishments (by changing γ_0). And for lower income punishment, government finds it optimal to fully default on higher debts. That makes the price of bonds decrease a lot on high debt levels (for low income punishment case). The default and bond demand is increasing in debts as well, i.e. if government has higher debt levels, he wants to default more and borrow more.

Let's try to explain the simulation results. I start the simulation from 0 debt, and for graphs and tables, I ignored first 2000 simulations. We see from graphs that the partial default rate oscillates around 2.6% and debt holding oscillates around 1% for low income punishment case. As I increase the income punishment, partial default mean decreases and debt holding increases. The reason of partial default decrease is easy to understand: because it is now more costly. But why do we see higher debt rates? My intuition is that there are two ways for the government to smooth his consumption and finance his consumption: 1) Borrow to repay tomorrow; 2) Partial default on your debt today. When partial default becomes more costly, the borrowing option become more attractive for the government and that's why we see the increase in demand for borrowing. One interesting result is that most of the time, the government stays in the "default" zone.

The main issue of this model is having low debt/GDP ratio. We can increase this by making the default more costly but then if we try to match the debt/GDP ratio with the data, we will have so high default rate that the partial default become much lower.