Morteza_ProblemSet1

November 14, 2023

1 Problem set 1 - Ph.D. course in Household Finance

1.1 Morteza Aghajanzadeh

1.2 Problem Description

The problem requires us to simulate a life-cycle model of a household's consumption and savings decisions. The model has two periods: a working period and a retirement period. The household has access to a risky asset and faces uncertainty about future income. The goal is to find the optimal consumption and savings decisions for the household in each period. ### Note: For this code, I use library programmed by myself, Toolkit.py which has all the functions needed in the model.

1.3 Part 1

1.3.1 My Approach

I followed the following steps to solve the problem:

1. I solved the problem of HH for the **retirement** periods, agents solve the problem conditional on age.
2. I then solved the problem for the **working** periods, where again, agents solve the problem for each given year.
3. Now we solved the problem for two period, then we can simulate the model by using the calculated policy functions.
4. At the end, report the life-cycle profiles for different wealth percentile at the retirement and initial wealth

[ ]:
```
import Toolkit as tk
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[ ]:
```
########## Model Parameters ##########
  = 0.945 ## Discount factor
  = 2.0 ## Risk Aversion
start = 25 ## starting age
start += 1 ## first period of life
t_w = 40 ## working age
```

```
t_r = 35 ## retirement age
t_r -= 1 ## last period of life
T = t_w + t_r ## total periods
 = 0.6  ## replacement rate
Y_lower = 4.8
rw = 0.02 ## Interest rate when working
rr = 0.02 ## Interest rate when retired
######### Thresholds ##########
vmin = -1.e10
epsilon = tk.epsilon
```

1.4 Income Process Parameters

[ ]:
```
########## Income Process Parameters ##########
## Trend
g_t = pd.read_excel('Income_profile.xlsx')['Y'].to_numpy().flatten() ## Income profile
# trend_pension =   * g_t[t_w-1]

## Permanent Income Process
N_z = 15  ## number of states for the permanent income process
 _z = 0.0 ## mean of the permanent income process
 _z = 1 ## persistence of the permanent income process
 _  = 0.015   ## standard deviation of the permanent income process
Z, _z = tk.tauchenhussey(N_z,  _z,  _z,  _ )
Z = np.exp(Z)[0].reshape(N_z,1) ## permanent income process

## Transitory Income Process
N_  = 5 ## number of states for the transitory income process
 _  = 0.0 ## mean of the transitory income process
 _  = 0 ## iid process
 _  = 0.1  ## standard deviation of the transitory income process
 , _  = tk.tauchenhussey(N_ ,  _ ,  _ ,  _ )
  = np.exp( )[0].reshape(N_ ,1) ## transitory income process

  = np.kron( _z,  _ ) ## transition matrix for the income process

 .shape
```

[ ]: (75, 75)

1.5 Simulation Parameters

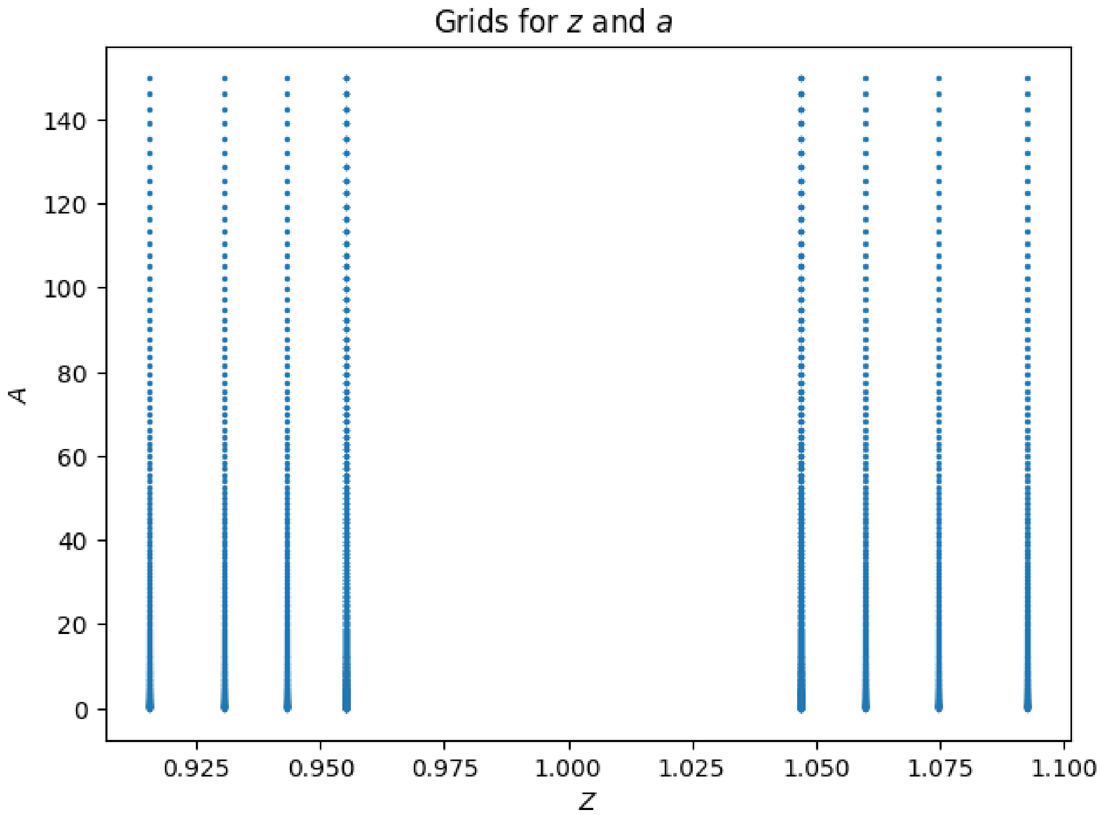
[ ]:
```
########## Simulation Parameters ##########
set_seed = 13990509
np.random.seed(set_seed)
N = 1000
```

```
 _A = 1.916
 _A = 2.129
A = np.random.normal( _A -  _A**2 / 2,  _A, N)
A = np.exp(A)
A[A < 0] = 0

 _z = 0.0
 _z = 0.015
Z0 = np.exp(np.random.normal( _z -  _z**2 / 2 ,  _z, N))
 _z = np.random.normal(- _z**2/2,  _z, (N,T))

 _  = 0.0
W0 = np.exp(np.random.normal( _  -  _ **2 / 2 ,  _ , N))
 _  = np.exp(np.random.normal( _ - _ **2,  _ , (N,T)))



######### Thresholds ##########
vmin = -1.e10
```

1.6 Creating Asset Grid

[ ]:
```
########## Asset Grid ##########
a_max = 150 ## upper bound of the grid for assets
  = 0  ## Borrowing Constraint
N_a = 200 ## number of grid points for assets
# a_grid = np.linspace( , a_max, N_a).reshape(N_a,1) ## linear grid for assets
a_grid = tk.discretize_assets_single_exp( , a_max, N_a).reshape(N_a,1) ##
 ↪double exponential grid for assets
```

[ ]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)

s_grid,a_grid_2 = np.meshgrid(Z,a_grid,indexing='ij')
ax.scatter(s_grid,a_grid_2,2)

ax.set_yscale('linear')
ax.set_xlabel('$Z$')
ax.set_ylabel('$A$')

fig.suptitle('Grids for $z$ and $a$')
fig.tight_layout(pad=0.5)
```



1.7 Retirement Problem

In the retirement period, there is no uncertainty about the future so the HH's problem is

$$\max_{A_t} \quad u(X_t - A_t) + \beta V(A_{t+1})$$

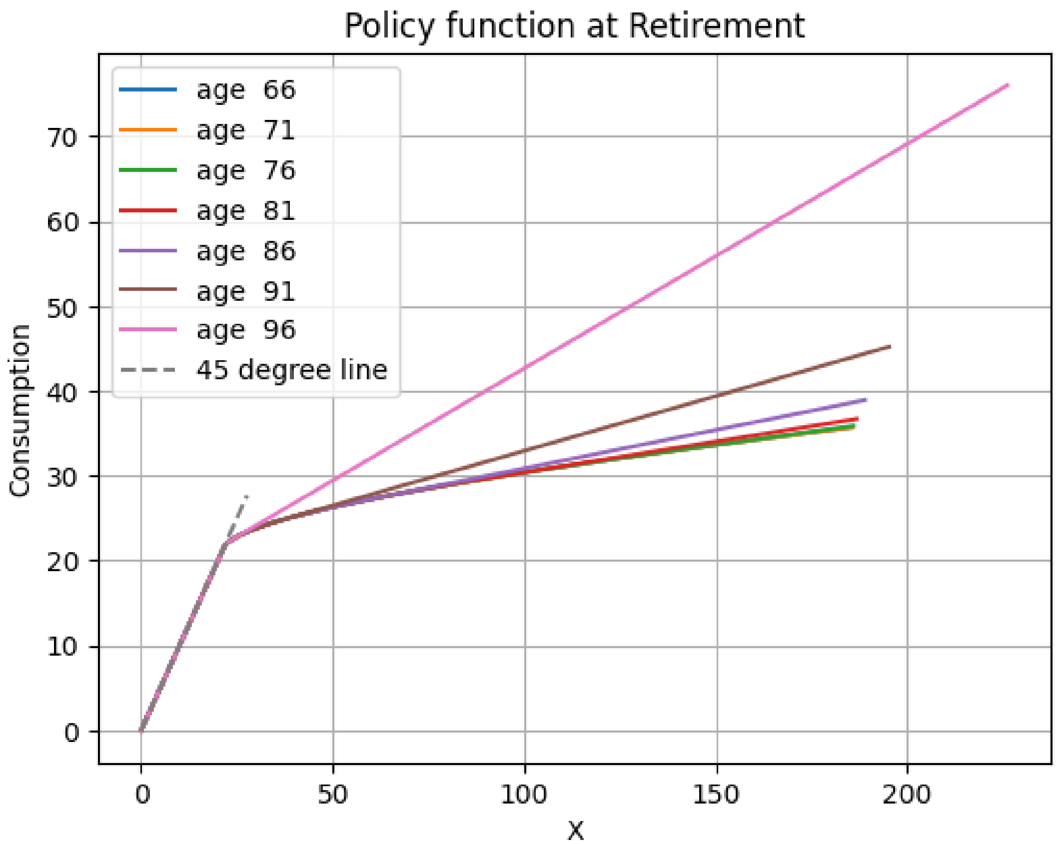
```
[ ]: Vr, Cr, Xr = tk.retirement(N_a,a_grid, rr,  ,  , t_r, t_w, g_t,  ,vmin)
     # Vr is the value function for retirement
     # Cr is the consumption function for retirement
     print(Vr.shape, Cr.shape, Xr.shape)
```

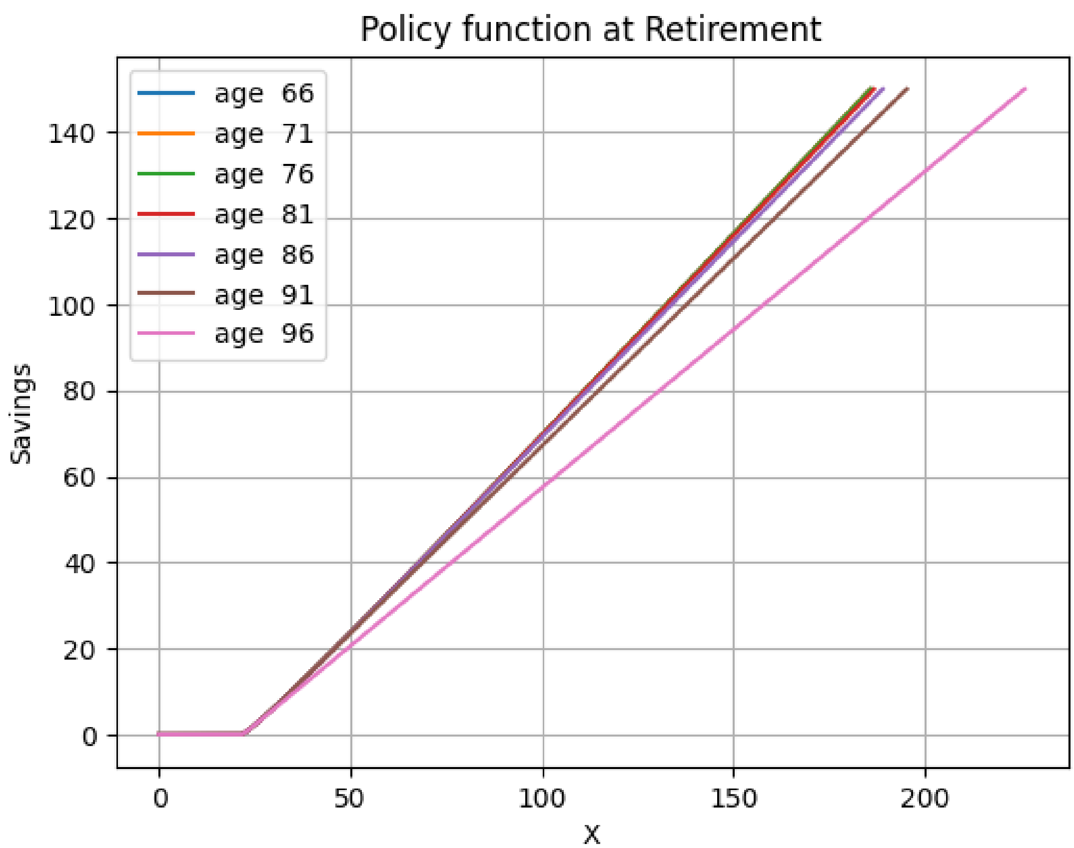
```
(201, 34) (201, 34) (201, 34)
```

```
c:\Github\Household-Finance\Part I - Life cycle consumption savings
models\Problem set 1\Toolkit.py:98: RuntimeWarning: divide by zero encountered
in reciprocal
  Vr[1:,-1:] = Cr[1:,-1:]**(1- )/(1- )
```

```
[ ]: tk.plot_policy_over_ages(Xr,Cr,t_r,'Retirement', N_a, start, t_w,line45=True,␣
     ↪yaxis='Consumption')
     Sr = Xr - Cr
```

```
tk.plot_policy_over_ages(Xr,Sr,t_r,'Retirement', N_a, start, t_w,line45=False,
    yaxis='Savings')
```





1.8 Working problem

HH's problem is

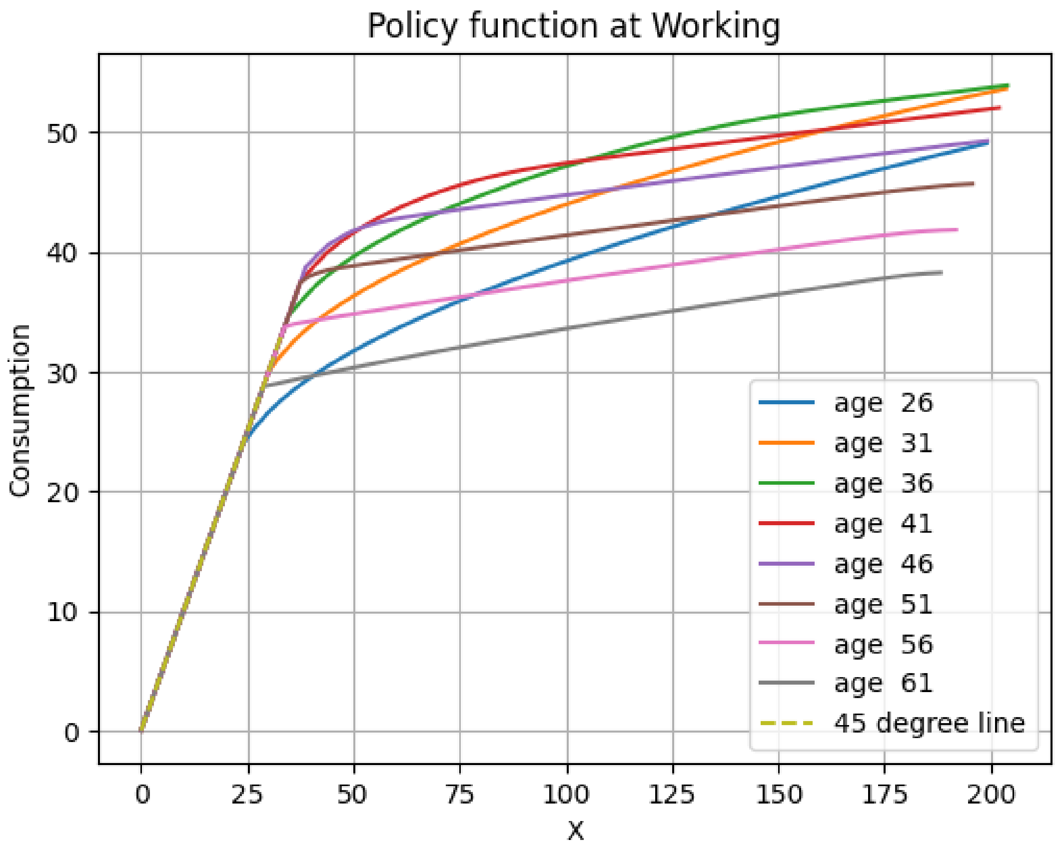
$$\max_{A_t} \quad u(X_t - A_t) + \beta E[V(A_{t+1})]$$

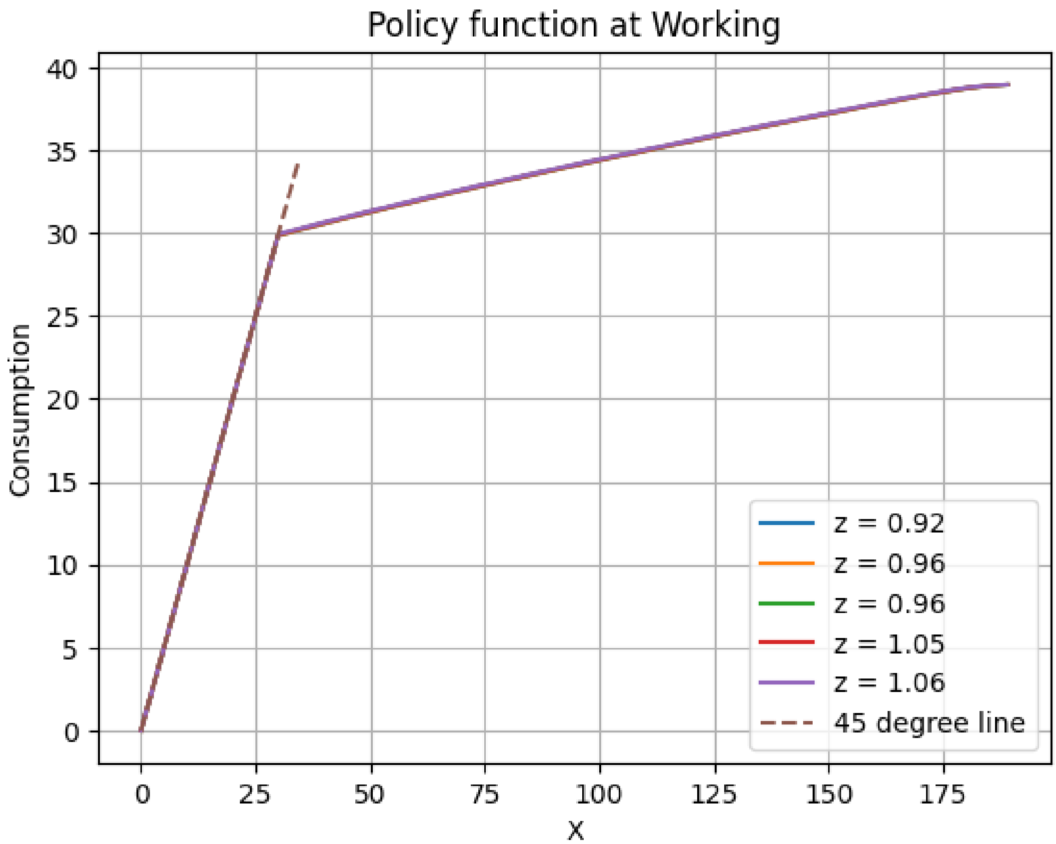
[ ]:
```
Vw, Cw, Xw = tk.working(N_z, N_ , N_a, a_grid, Z,  ,  , rw, rr, Xr, Cr, Vr,  ,  ,␣
  ↪t_w, g_t, ,vmin)
# Vw is the value function for working
# Cw is the consumption function for working
print(Vw.shape, Cw.shape, Xw.shape)
```

```
(3015, 40) (3015, 40) (3015, 40)
```

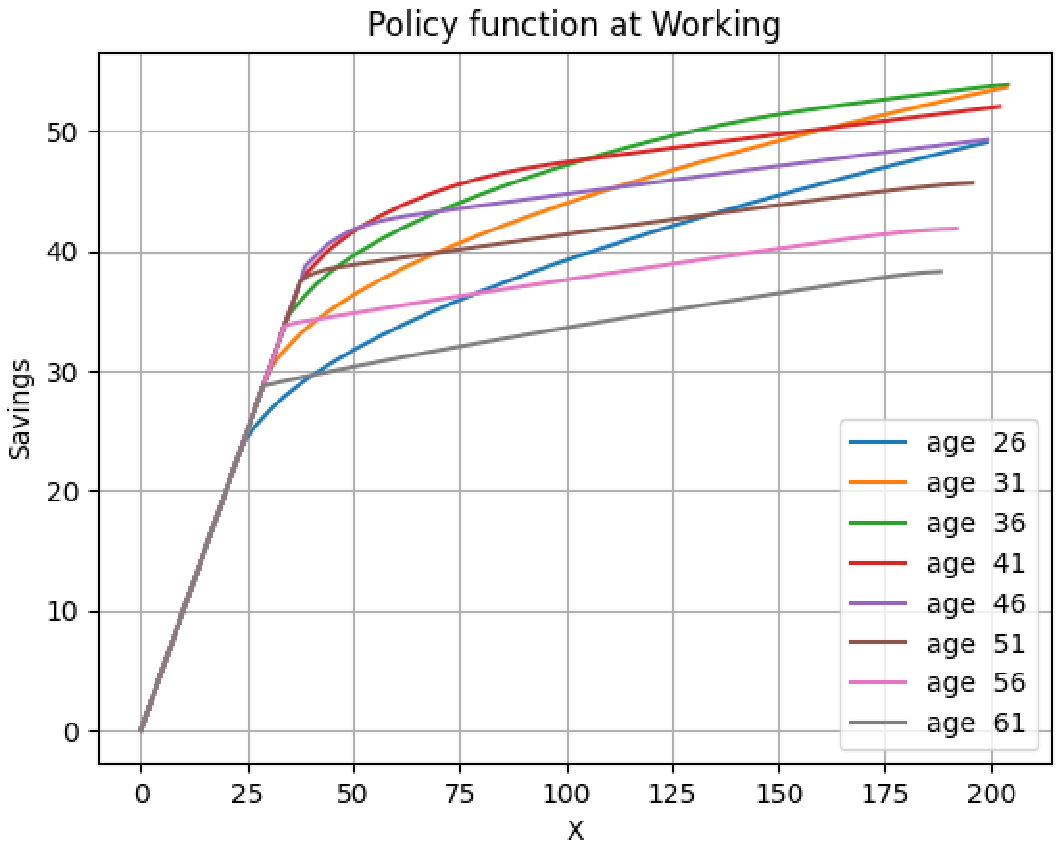
[ ]:
```
tk.plot_policy_over_ages(Xw,Cw,t_w,'Working', N_a, start, 0,line45=True,␣
  ↪yaxis='Consumption')
tk.
  ↪plot_policy_over_states(Xw,Cw,Z,20,t_w,start,N_z,N_a,line45=True,yaxis='Consumption')
Sw = Xw - Cw
```

```
tk.plot_policy_over_ages(Xw,Cw,t_w,'Working', N_a, start, 0,line45=False,
    yaxis='Savings')
```





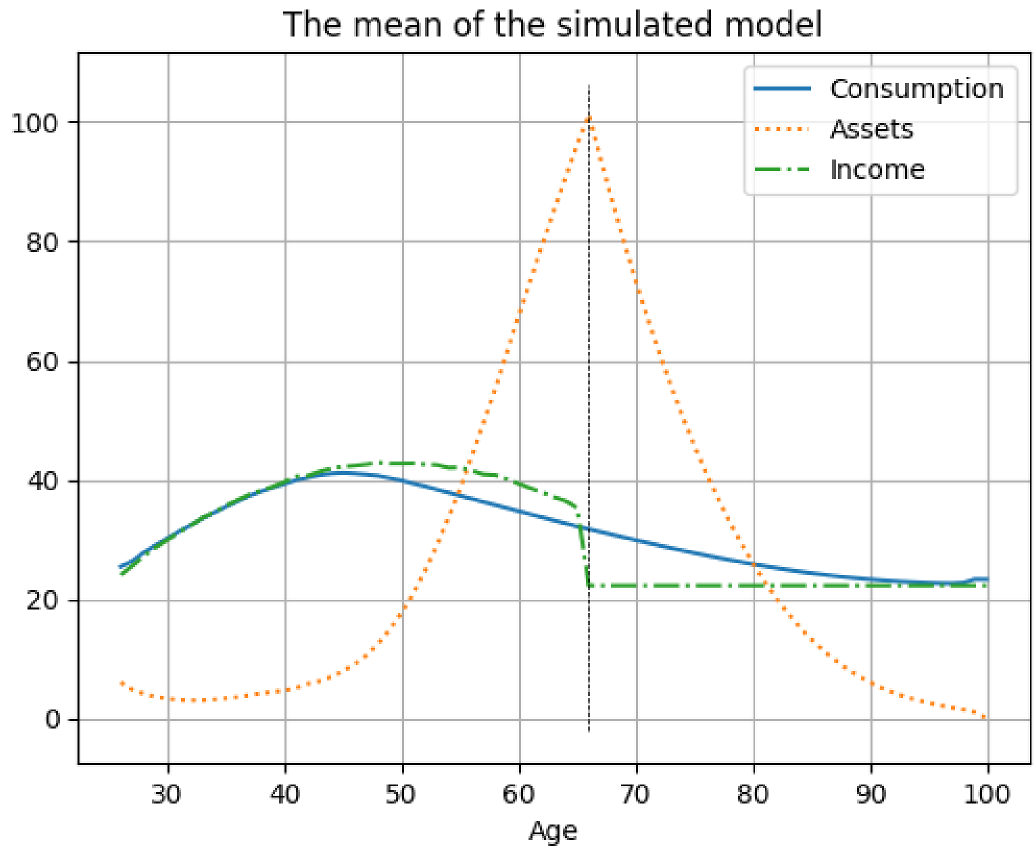
Policy function at Working
Consumption
X
0
5
10
15
20
25
30
35
40
0
25
50
75
100
125
150
175
z = 0.92
z = 0.96
z = 0.96
z = 1.05
z = 1.06
45 degree line



1.9 Simulation

[ ]:
```
A_sim, Z_sim,  _sim, income_sim, Zi_sim, X_sim, C_sim =  tk.simulate_model(T,
 rw, rr, Xw, Cw, Y_lower, t_w, t_r, g_t,  _z, N, N_a, Xr, Cr, Z,  ,  _z,  _ , A,
 Z0,start)
```

```
Simulated model for working age
Simulated model for retirement age
```

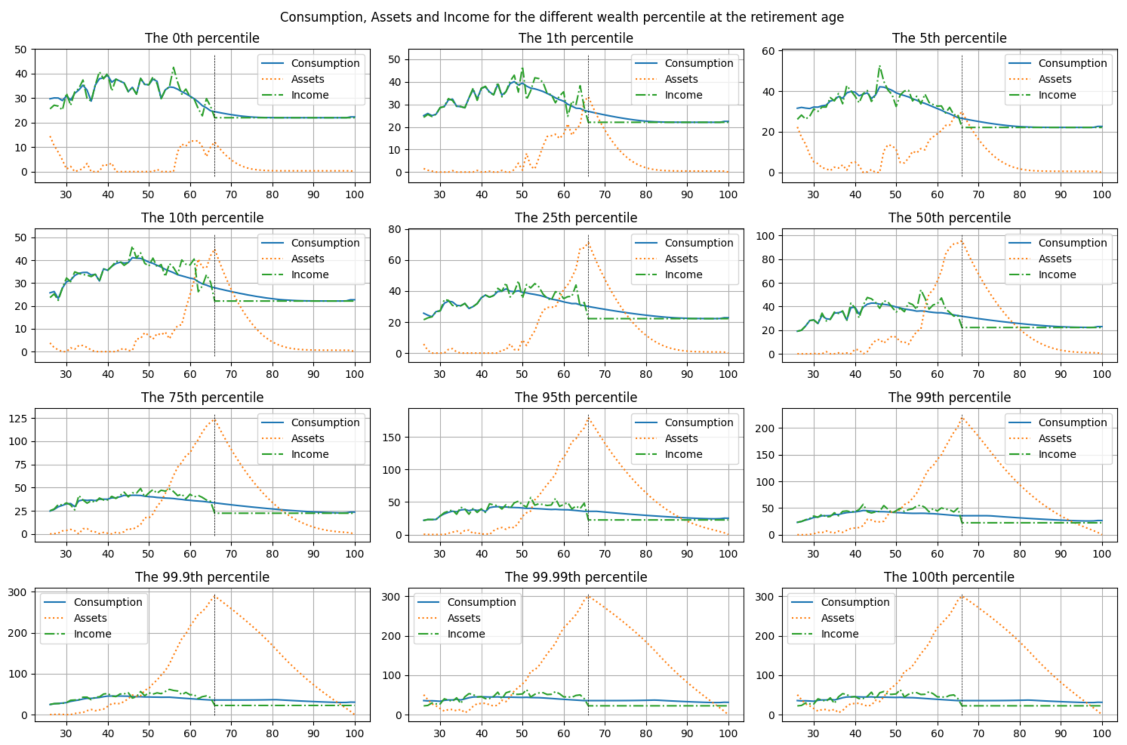


1.10 Plot the individual person

```
[ ]: def plot_specific_person(A_index,ax=False):
         if ax != False:
             plt.sca(ax)
         C_mean = C_sim[A_index,:]
         plt.plot(range(start,start + T+1),C_mean)
         A_mean = A_sim[A_index,:]
         plt.plot(range(start,start + T+1),A_mean,linestyle=':')
         I_mean = income_sim[A_index,:]
         plt.plot(range(start,start + T+1),I_mean,linestyle='-.')
         # plt.plot(range(start,start + T+1),np.zeros(T+1),linestyle='--')
         max_value = max(C_mean.max(),A_mean.max(),I_mean.max())
         plt.plot([start + t_w]*100,np.linspace(-2,max_value +␣
     ↪5,100),linestyle='--',color='black',linewidth=0.5)
         plt.legend(['Consumption','Assets','Income'])
         plt.grid()
```

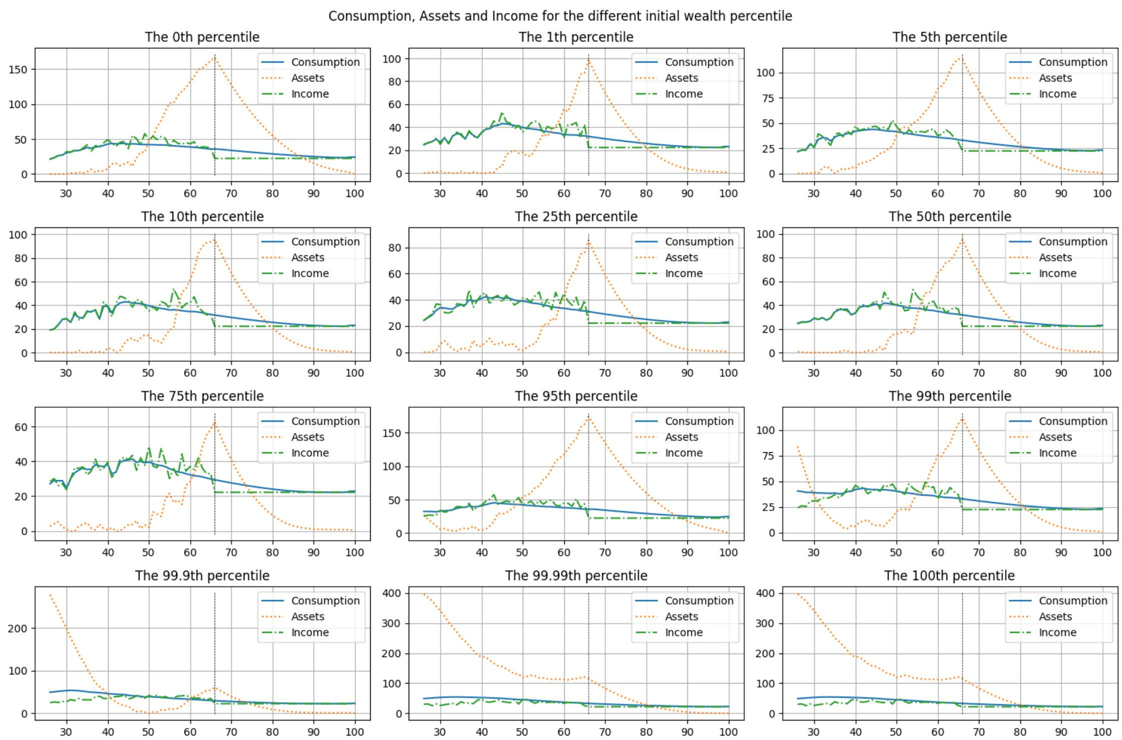
1.10.1 Individual Life-Cycle results

[ ]:
```
percentiles = [0,1,5,10,25,50,75,95,99,99.9,99.99,100]
fig, axs = plt.subplots(int(len(percentiles)/3), 3, figsize=(15, 10))
axs = axs.flatten()
counter = 0
for i, ax in zip(percentiles, axs):
    tempt = np.abs(A_sim[:, start + t_w] - np.percentile(A_sim[:, start +␣
 ↪t_w],i))
    A_index = np.where(tempt == tempt.min())[0][0]
    plot_specific_person(A_index,ax=ax)
    # plt.show()
    ax.set_title('The {}th percentile'.format(i))

fig.suptitle('Consumption, Assets and Income for the different wealth␣
 ↪percentile at the retirement age')
plt.tight_layout()
plt.show()
```



[ ]:
```
percentiles = [0,1,5,10,25,50,75,95,99,99.9,99.99,100]
fig, axs = plt.subplots(int(len(percentiles)/3), 3, figsize=(15, 10))
axs = axs.flatten()
```

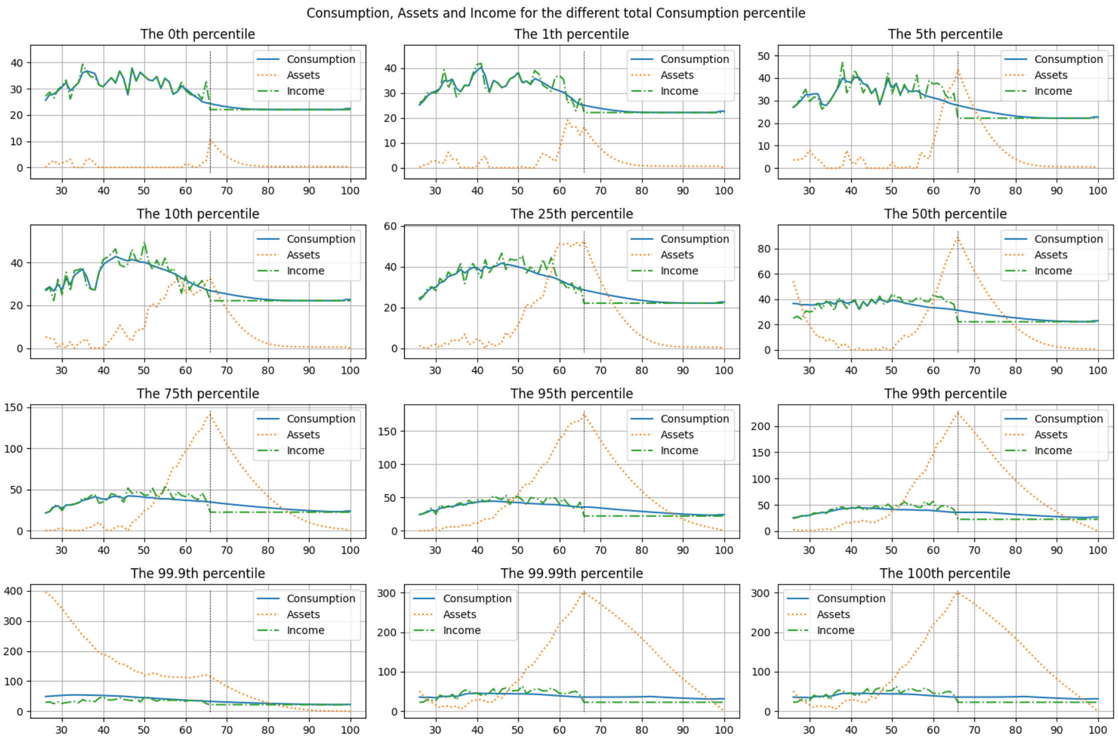
```
for i, ax in zip(percentiles, axs):
    tempt = np.abs(A_sim[:, 0] - np.percentile(A_sim[:, 0],i))
    A_index = np.where(tempt == tempt.min())[0][0]
    plot_specific_person(A_index,ax=ax)
    ax.set_title('The {}th percentile'.format(i))

fig.suptitle('Consumption, Assets and Income for the different initial wealth␣
 ↪percentile ')
plt.tight_layout()
plt.show()
```



[ ]:

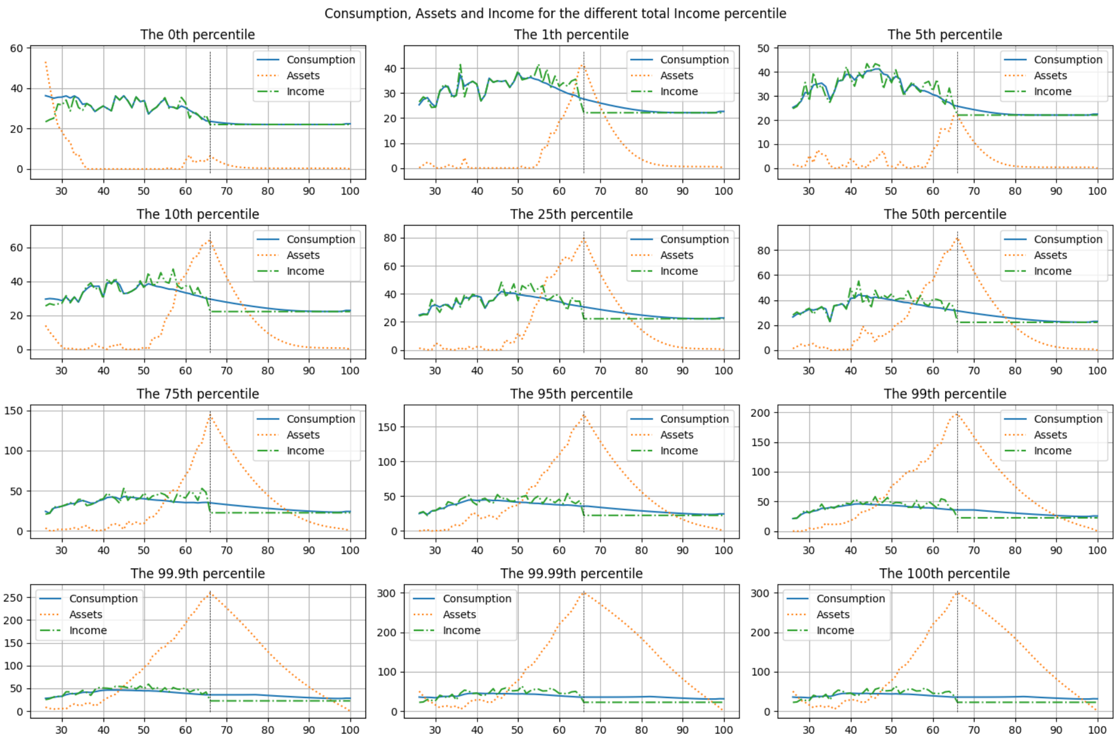
```
percentiles = [0,1,5,10,25,50,75,95,99,99.9,99.99,100]
fig, axs = plt.subplots(int(len(percentiles)/3), 3, figsize=(15, 10))
axs = axs.flatten()

tempt_1 = C_sim.sum(axis=1)

for i, ax in zip(percentiles, axs):
    tempt = np.abs(tempt_1[:] - np.percentile(tempt_1,i))
    A_index = np.where(tempt == tempt.min())[0][0]
    plot_specific_person(A_index,ax=ax)
    ax.set_title('The {}th percentile'.format(i))
```

```
fig.suptitle('Consumption, Assets and Income for the different total Consumption percentile ')
plt.tight_layout()
plt.show()
```



[ ]:

```
percentiles = [0,1,5,10,25,50,75,95,99,99.9,99.99,100]
fig, axs = plt.subplots(int(len(percentiles)/3), 3, figsize=(15, 10))
axs = axs.flatten()

tempt_1 = income_sim.sum(axis=1)

for i, ax in zip(percentiles, axs):
    tempt = np.abs(tempt_1[:] - np.percentile(tempt_1,i))
    A_index = np.where(tempt == tempt.min())[0][0]
    plot_specific_person(A_index,ax=ax)
    ax.set_title('The {}th percentile'.format(i))

fig.suptitle('Consumption, Assets and Income for the different total Income percentile ')
plt.tight_layout()
plt.show()
```



1.11 Part 2

1.12 My Approach:

1. I define a new state variable **r** which is following the

$$R_t = \exp(\ln R^f + \mu + \varepsilon_t)$$

2. I define a function that solve the problem for given X and C.
3. Now it is the time to solve the problem for the HH during the retirement. As you know, problem for the retirement period is simpler, due to the fact that we only have one state variable
4. I need to solve the model for the working period too, which I **could not!**
5. I put a lot of time, but it is still ongoing.

1.12.1 Conclusion:

The main problem is the fact that the *alpha* function is not concave, and we have **multiple solutions**!

[ ]:
```
Z,  ,  _ ,  _z,  , A, Z0,  _z,  _ ,a_grid = tk.initialize(T,N_z,  _z,  _z,␣
 ↪ _ ,N_ , _ , _ , _ ,N_a,a_max, ,N, _A, _A, _z)

## Shock in the interest rate
N_r = 5 ## number of states for the interest rate process
  = 0.04
```

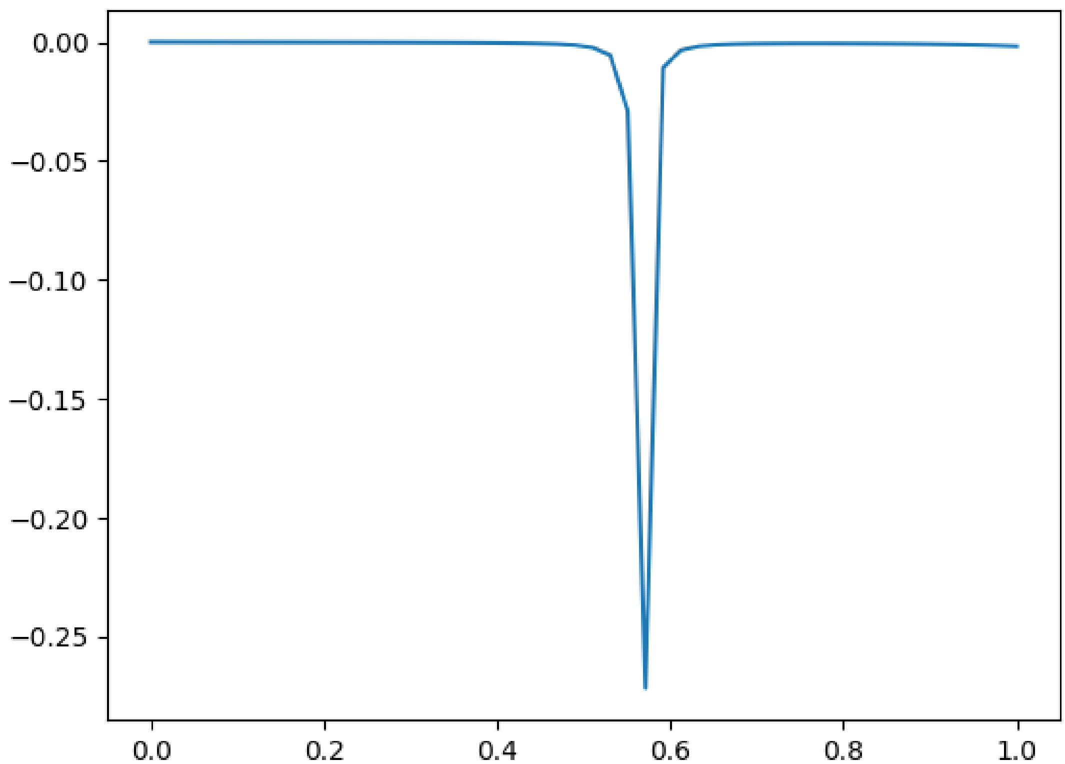
```
r_f = 0.02
 _r = np.log(1+r_f) +
 _r = 0.18
r, _r = tk.tauchenhussey(N_r,  _r, 0,  _r)
r = np.exp(r)[0].reshape(N_r,1) ## transitory income process
print(r)
r = r - 1

 _r = 0.18
 _r = np.exp(np.random.normal(- _r**2,  _r, (N,T)))

N_  = 5
 _grid = np.linspace(0,1,N_ )
  = np.kron(np.kron( _r,  _z),  _ )
```

```
[[0.70767015]
 [0.8757767 ]
 [1.06162699]
 [1.28691694]
 [1.59262316]]
```

[ ]:
```
x = Xr[-5,5]
c = Cr[-5,5]
a = a_grid[-5]
  = 0.5
r_state = 0
r_index = r_state
z_state = 0
pension =    * g_t[t_w-1]
income = pension
def alpha( ,x,c,a,r_index, _r,pension):
    A_t = x - c
    Y_t1 = pension
    X_t1 = Y_t1 +  A_t* (1 + r_f +   * (r - r_f)).T
    c_t1 = X_t1 - a
    # c_t1 = np.interp(c_t1,Xr[:,t+1], Cr[:,t+1])
    M_t1 =   * c_t1 ** (- )
    # E =  (M_t1 * (r - r_f).T) @  _r[i,:].T
    E =  (M_t1 * (r - r_f).T) @  _r[r_index,:].T
    return E[0]

def alpha( ,x,c,a,r_index, _r,income):
    A_t = x - c
    Y_t1 = income
    X_t1 = Y_t1 +  A_t* (1 + r_f +   * (r - r_f)).T
    c_t1 = X_t1 - a
    M_t1 =   * c_t1 ** (- )
```

```
    E =  (M_t1 * (r - r_f).T) @  _r[r_index,:].T
    return E[0]

test_a = np.linspace(0,1,50)
res = []
for i in test_a:
    # res.append(alpha_stocastic_income(i,x,c,a,r_state,z_state, _r, _z,income))
    res.append(alpha(i,x,c,a,r_state, _r,pension))
plt.plot(test_a,res)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x2e227fbe7d0>]
```



[ ]:
```
from scipy.optimize import fsolve
def alpha( ,x,c,a,r_index, _r,income):
    A_t = x - c
    Y_t1 = income
    X_t1 = Y_t1 +  A_t* (1 + r_f +   * (r - r_f)).T
    c_t1 = X_t1 - a
    M_t1 =   * c_t1 ** (- )
    E =  (M_t1 * (r - r_f).T) @  _r[r_index,:].T
    return E[0]
```

```
def solve_alpha(x,c,a,r_index, _r,income):
    alpha_0 , alpha_1 =␣
↪alpha(0,x,c,a,r_index, _r,income),alpha(1,x,c,a,r_index, _r,income)
    if alpha_0 * alpha_1 > 0:
        if alpha_0 > 0: return 1
        else: return 0
    else:
         = fsolve(alpha,0.5,args = (x,c,a,r_index, _r,income))[0]
        if  >1: return 1
        else: return 

def solve_over_array(X,C,grid,r_index, _r,income):
     p = np.zeros((N_a,1))
    for n_a in range(0,N_a):
         p[n_a] = solve_alpha(X[n_a],C[n_a],grid[n_a],r_index, _r,income)
    return  p
    #  p = np.zeros((N_a,1))
    # for n_a in range(0,N_a):
    #      p[n_a] = solve_alpha(Xp[n_a],Cp[n_a],a_grid[n_a],r_index = i, _r =␣
↪ _r,pension = pension)


def retirement_with_asset(r, _r, , ,g_t,t_w,t_r,N_a,a_grid,vmin):
    Vr = np.zeros((N_r * (N_a+1), t_r))
    Cr = np.zeros((N_r * (N_a+1), t_r))
    Xr = np.zeros((N_r * (N_a+1), t_r))
     r = np.zeros((N_r * (N_a+1), t_r))

    # Set the last period
    for i in range(N_r):
        Vr[i*(N_a+1),:] = vmin
        index = range(i*(N_a+1)+1,(i+1)*(N_a+1))
        Xr[index,-1:] = a_grid
        Cr[index,-1:] = Xr[index,-1:]
        Vr[index,-1:] = Cr[index,-1:]**(1- )/(1- )

    # backward iteration
    for t in range(t_r-1,0, -1):
        t -= 1
        # I have to think more about the pension income
        pension =   * g_t[t_w-1]
        Cp = np.zeros((N_a,N_r))
        Vp = np.zeros((N_a,N_r))
        for i in range(N_r):
            Xp = a_grid * (1 + r[i]) + pension ## cash-on-hand tomorrow
            index = range(i*(N_a+1)+1,(i+1)*(N_a+1))
```

```
            Cp[:,i:i+1] = np.interp(Xp,Xr[index,t+1], Cr[index,t+1]) #␣
↪interpolate consumption
            Vp[:,i:i+1] = np.interp(Xp,Xr[index,t+1], Vr[index,t+1]) #␣
↪interpolate consumption
            for n_a in range(0,N_a):
                index = i*(n_a+1)+1
                r[index,t:t+1] = solve_alpha(Xp[n_a],Cp[n_a,i:
↪i+1],a_grid[n_a],r_index = i, _r =  _r,income = pension)
        dVp = Cp ** (- )
        EV =   * np.dot(dVp, _r.T)
        for i in range(N_r):
            index = range(i*(N_a+1)+1,(i+1)*(N_a+1))
            dV =   * np.dot(dVp, _r[i,:].T) * (1 + r[i])
            Cr[index,t:t+1] = dV.reshape(dV.shape[0],1) ** (-1/ )
            Xr[index,t:t+1] = Cr[index,t:t+1] + a_grid
    return Cr,Xr,Vr, r
Cr,Xr,Vr, r = retirement_with_asset(r, _r, , , ,g_t,t_w,t_r,N_a,a_grid,vmin)
```

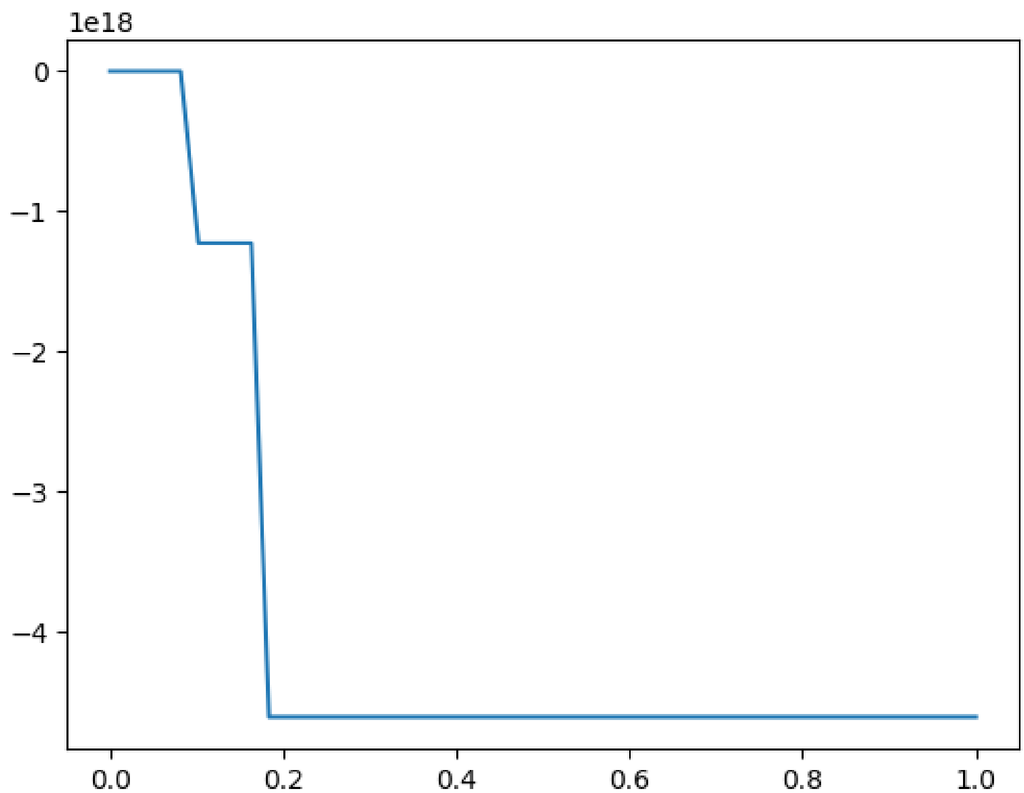
```
C:\Users\SE.5203\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz
5n2kfra8p0\LocalCache\local-packages\Python310\site-
packages\scipy\optimize\_minpack_py.py:177: RuntimeWarning: The iteration is not
making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
C:\Users\SE.5203\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz
5n2kfra8p0\LocalCache\local-packages\Python310\site-
packages\scipy\optimize\_minpack_py.py:177: RuntimeWarning: The iteration is not
making good progress, as measured by the
  improvement from the last five Jacobian evaluations.
  warnings.warn(msg, RuntimeWarning)
C:\Users\SE.5203\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz
5n2kfra8p0\LocalCache\local-packages\Python310\site-
packages\scipy\optimize\_minpack_py.py:177: RuntimeWarning: The number of calls
to function has reached maxfev = 400.
  warnings.warn(msg, RuntimeWarning)
```

[ ]:

```
x = Xr[-1,5]
c = Cr[-1,5]
a = a_grid[-1]
  = 0.5
r_state = 0
z_state = 0
income = Z
def alpha_stocastic_income( ,x,c,a,r_state,z_state, _r, _z,income):
    A_t = x - c
    Y_t1 = income
    X_t1 = Y_t1 +  A_t* (1 + r_f +   * (r - r_f)).T
```

```
    c_t1 = X_t1 - a
    c_t1[c_t1<0] = epsilon
    M_t1 =   * c_t1 ** (- )
    r_repeated = np.kron(r,np.ones((1,N_z))).T
    E =  ((M_t1 * (r_repeated - r_f)) @  _r[r_state,:]).T @  _z[z_state,:]
    return E

test_a = np.linspace(0,1,50)
res = []
for i in test_a:
    res.append(alpha_stocastic_income(i,x,c,a,r_state,z_state, _r, _z,income))
plt.plot(test_a,res)
fsolve(alpha,1,args = (x,c,a,r_state, _r,pension))
```

[ ]: array([0.6971274])



[ ]:

[ ]: