Solution and Simulation of Aiyagari-Bewley-Huggett-Imrohoglu model

Chinmaya Singh

April 8, 2024

1 Introduction

The **finite horizon** model named "Aiyagari-Bewley-Huggett-Imrohoroglu" (ABHI) is a model in which the representative household is the agent which seeks to optimize its discounted lifetime utility over a finite time periods. In every time period, it receives an income (exogenously determined by an AR(1) stochastic process) and has assets from the previous time period. The agent has to make the choice of selecting the consumption level and the assets to save for the next time period. The agent optimizes its choices in such a way that its discounted life time utility (i.e. the objective function) is maximized. Finding the solution of the model is equivalent to finding the policy function which optimizes the objective function. The mathematical problem is formally restated below and the choice problem is further simplified -

$$\max_{c_t, a_{t+1}} E \sum_{t=1}^{T} \beta^t ln(c_t)$$

subject to the constraints -

$$c_t + a_{t+1} = (1 + r)a_t + y_t$$

$$a_t \geq \underline{a}$$

$$\ln y_{t+1} = \rho \ln y_t + \epsilon_t$$

where

$$\epsilon_t \sim \mathcal{N}(0, \sigma^2)$$

First equation is called the budget constraint and the second equation is the liquiditiy constraint (which is the novelty of the ABHI-type models). Note that the budget constraint makes the choices of the current consumption c_t and the assets for the next period a_{t+1} inter-dependent. In other words, if the agent chooses one of the values the other can be found out using the budget constraint. In the computational model, the agent chooses the assets for the next period.

Sequential optimization problem is reformulated as a dynamic programming problem using the bellman equation -

$$V_t(a_t; y_t) = \max_{a_{t+1} \geq \underline{a}} \{\ln((1 + r)a_t + y_t - a_{t+1}) + \beta E[V_t(a_t)|y_t]\}$$

In the model, a_t and y_t are the state variables. Difference between the two is that y_t is exogenously given and random in nature (i.e. stochastic) whereas a_t is endogenously chosen by the agent. Since its a finite horizon model, the value function is time-variant depicted by the t-subscript in the value function. Informally stating, it also becomes compulsory to close the model from the end. Formally, use the given terminal condition $a_{T+1} = 0$ to determine $V_T(a_T; y_T) = \ln y_T + (1+r)a_T$.

Now all that remains is to take the equation to the computer to solve the optimization problem. However the computers are digital in nature and are able to handle only discrete values and the model variables (c_t, a_t, y_t) are of continuous nature. Note that time in the model is already discrete. Stochastic AR(1) process can be discretized using tauchen's method [2] and can be converted into a state transition probability matrix. [1] Instead of discretizing the space of y_t, $\ln y_t$- space has been discretized using the tauchen method (for code see appendix A). It has been made sure that initial income ($y_1 = 1$) is in the space be making sure that the number of points (arbitrarliy fixed) in the space is odd and the space is evenly and symmetrically distributed around the origin. Lower bound of a_t-space is determined by the value of $\underline{a}$ and the upper bound is fixed at sufficiently large value 100. Observe that it is unnecessary to discrete the c_t-space because the values of a_t and y_t can simply be plugged into the budget constraint to get c_t. Using the discrete space of $a_t \in A = [a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}, ..., a^{(A)}]$ (where A is the number of discrete points in a_t-discrete space), the policy function can be defined as follows -

$$g_t(a_t; y_t) = \arg\max_{a_{t+1} \in A} \{\ln((1+r)a_t + y_t - a_{t+1}) + \beta E[V_t(a_t)|y_t]\}$$

Since $a_{T+1} = 0$, the terminal condition pertaining to the policy function is $a_{T+1} = g_T(a_T; y_T) = 0$

Other important issue to address is the caclulation of $E[V_t(a_t)|y_t]$ term in bellman equation and policy equation. Given that we have transition probability matrix from tauchen method, $E[V_t(a_t)|y_t]$ can be easily calculated by a one-line python function -

```python
def expected_value_function(value_function, trans_matrix, y):
    return np.dot(value_function, trans_matrix[y,:])
```

It collapses the (3-D) value function accross the last dimension ($\ln y$-space) by taking a dot product from the relevant row ('y') of the probability transition matrix. The value of 'y' is deduced by the value of y_t in $E[V_t(a_t)|y_t]$ and the indexing of $\ln y$-space. Resulting value funciton matrix is only 2 dimensional (assets and time).

The python code for calculating the value and policy function by solving the bellman equation backwards in time is in appendix B.

2 Model Solution and Dynamics

Solving the model and plotting the mean and variance of income y_t , consumption c_t, assets a_t, over time

I simulate the model 5000 times (sufficiently large and arbitrarily fixed) to find the mean and the variance of the income, consumption and assets. To make sure that the number is sufficiently large, I compare the results when I run the model 10000 and found no noticeable difference in the plots of the mean and the variance. Python function for the simulation can be found in appendix C. Evolution of mean and variance of the variables (case $\underline{a} = 10$) is illustrated as follows -

[1] I have used pseudocode in [1] for reference

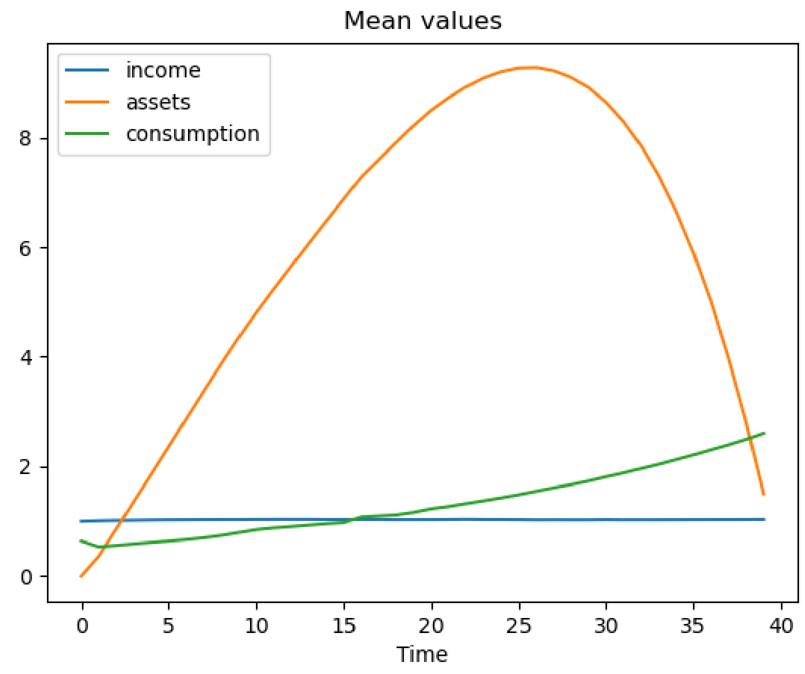
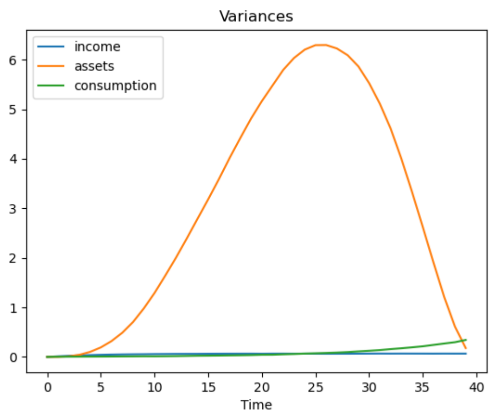
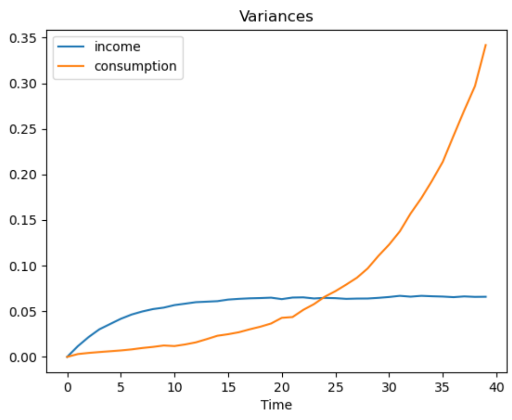


Figure 1: Evolution of the mean of the variables



(a) Evolution of the variance of the variables



(b) Evolution of the variance (magnified view)

Looking at the graph having mean values, it is evident that purchasing and selling of assets help the agent smoothen out its consumption.

3 Life Cycle Evolution

Illustrating how the distributions of income, consumption, and assets evolve over the life cycle

Few sample runs for different cases have been simulated using the solved model and are illustrated in the appendix D

4 Impact of Minimum Assets $\underline{a}$

Analyzing and discussing the key differences resulting from varying the minimum level of assets $\underline{a}$ within the specified set.

5 Efficiency

Efficiency of any computer program is measured by the amount of space on RAM it uses and and its actual runtime. Computer scientists frequently use big-O notation ($O(.)$) to specify theoretical runtime of algorithms and/or pseudocodes. Denoting number of discrete points in a_t-space and $\ln y_t$-space as A and Y respectively and T as number of time periods, runtime of implemented computational solution is $O(TYA^2)$. For the values of $T = 40$, $Y = 11$, and $A = 500$, runtime of the code on my PC is approximately 348 seconds.

The runtime of the code can be decreased by many methods using both programming techniques and numerical techniques. For example, a graphics processor can be employed alongwith a python library (e.g. PyTorch or Tensorflow or numba) to parallelize vector operations on matrices. Mathematical properties of value function and policy function can also be exploited to reduce the search area while finding the maximum of the bellman equation. Two most common properties used are the monotonicity of the policy function and the concavity of the value function.

A Code Snippet : Tauchen Method

The function discretizes an AR(1) process of the form

$$x_{t+1} = \rho x_t + \epsilon_t$$

where

$$\epsilon_t \sim \mathcal{N}(0, \sigma^2)$$

. The arguments of the function n, pho, and sigma are number of discrete points in x-space, ρ in the AR(1) process and the variance of the error term.

```
def tauchen(n, pho, sigma):
    sigma_z = sigma/( np.sqrt( 1 - pow(pho,2) ) )
    z_max = +3*sigma_z
    z_min = -3*sigma_z
    z = [-1]*n
    for i in range(0,n):
        z[i] = z_min + (z_max - z_min)*(i)/(n - 1)
    z_mid = [-1]*(n-1)
    for i in range(0,n-1):
        z_mid[i] = (z[i+1] + z[i])/2
    tra_prob_matrix = np.zeros( shape = (n , n) )
    for i in range(0,n):
        for j in range(0,n):
            if(j == 0):
                tra_prob_matrix[i][j] =
                scipy.stats.norm.cdf((z_mid[j] - pho*z[i])/sigma)
            elif(j == n - 1):
                tra_prob_matrix[i][j] = 1 -
                scipy.stats.norm.cdf((z_mid[j-1] - pho*z[i])/sigma)
            else:
                tra_prob_matrix[i][j] =
                scipy.stats.norm.cdf((z_mid[j] - pho*z[i])/sigma) -
                scipy.stats.norm.cdf((z_mid[j-1] - pho*z[i])/sigma)
    disp_tra_prob_matrix = copy.deepcopy(tra_prob_matrix)
    for i in range(0,n):
        for j in range(0,n):
            disp_tra_prob_matrix[i][j] = round(tra_prob_matrix[i][j],3)
    return (np.array(z), tra_prob_matrix, disp_tra_prob_matrix)
```

B Code Snippet : Model solution

```
### Defining parameters (given)
r     = 0.05
beta  = 0.99
T     = 40
a_bar = [-40,-10,0]
pho   = 0.9
sigma = 0.1

### User-defined parameters

a_space_start  = a_bar[0]
a_space_end    =  50

a_space_points = 500
y_space_points = 5 # Make sure that it's odd number

###### Computational solution of the model

### Discretizing the y_space and the stochastic process of income
(y_space, trans_matrix, dtrans_matrix) = tauchen(y_space_points, pho,
                                                 sigma)

# Index of midpoint of y_space
y_space_mid = int( (y_space_points - 1)/2 )
# y_space[y_space_mid] should be zero if y_space_points is odd

### Discretizing the a_space
a_space = np.array(np.linspace(a_space_start,a_space_end,a_space_points))

### Defining the time space;
### note that it is already discrete in the model
t_space = np.array(np.linspace(1,T,T))

### Initializing the value and policy functions
```

```
value_function   = np.random.normal(size=
(a_space.shape[0],t_space.shape[0],y_space.shape[0]))
state_policy_function = np.random.normal(size=
(a_space.shape[0],t_space.shape[0],y_space.shape[0]))


### Terminal condition dictates that :-
### state-policy function at T = 40 is 0
for i in range(0, y_space.shape[0]):
    for j in range(0, a_space.shape[0]):
        state_policy_function[j,T-1,i] = 0

### value-function at T = 40 is ln(y_T + (1+r)*a_T)
for i in range(0, y_space.shape[0]):
    for j in range(0, a_space.shape[0]):
        if( np.exp(y_space[i]) + (1 + r)*a_space[j] > 0):
            value_function[j,T-1,i] = np.log( np.exp(y_space[i]) +
            (1 + r)*a_space[j] )
        else:
            value_function[j,T-1,i] = -inf


### Iterating through time backwards
for t in range(t_space.shape[0]-2,-1,-1):

    print( "Value function for time period : " + str(t_space[t])
+ " computed" )

    ### Updating the value function
    for i in range(0,y_space.shape[0]):

        ### Calculating expected value function using tauchen's method
        exp_val  = expected_value_function(
        value_function, trans_matrix, i)

        for j in range(0,a_space.shape[0]):

            temp_values = []
            for k in range(0,a_space.shape[0]):
                if((1+r)*a_space[j]+np.exp(y_space[i])-a_space[k]>0):
                    temp = np.log( (1+r)*a_space[j] +
                    np.exp(y_space[i]) - a_space[k] ) +
                    beta*exp_val[k,t+1]
                    temp_values.append(temp)
                else:
                    temp_values.append(-inf)

            temp_values = np.array(temp_values)

            if( np.nanmax(temp_values) == -inf):
                value_function[j][t][i] = -inf
                state_policy_function[j][t][i] = np.nan
            else:
                ### Updating the value function
                ### with the value which maximizes the bellman equation
                value_function[j][t][i] = np.nanmax(temp_values)
                ### Updating the policy function
                ind = np.nanargmax(temp_values)
                state_policy_function[j][t][i] = a_space[ind]
```

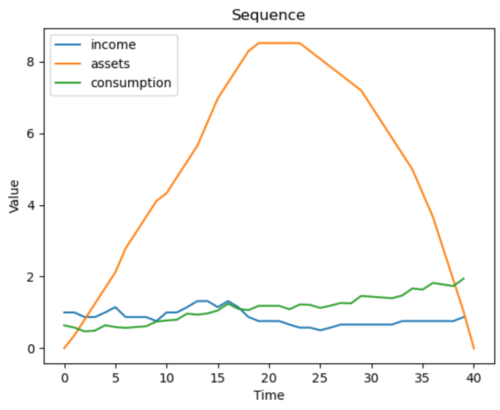
C Code Snippet : Model simulation

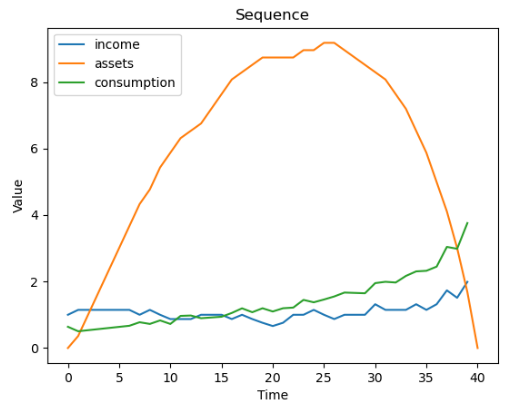
```
### Simulation (single run) of the model
### (with discretized stochastic process)
def simulation_run(state_policy_function,T,y_space_points,
            y_space,trans_matrix):

    y1,a1,a_last = 1,0,0

    y_space_mid = int( (y_space_points - 1)/2 )
    # y_space[y_space_mid] should be zero if y_space_points is odd
    y_space_indices = np.arange(0,y_space_points,1)

    log_income      = [-10]*T
    log_income[0] = y_space[y_space_mid]

    ### Generating a sequence of log(income) - exogenous
    for i in range(1,T):
        y_curr = np.where(np.isclose(y_space, log_income[i-1] ))[0][0]
        y_next = scipy.stats.rv_discrete(
        values=(y_space_indices,trans_matrix[y_curr,:])
        ).rvs(size=1)[0]
        log_income[i] = y_space[y_next]


    income = np.exp(log_income)

    ### Initializing list for storing asset sequence
    a = [-100]*(T+1)
    a[0] = a1

    ### Initializing list for storing consumption sequence
    c = [-1]*T

    ### Model run
    for i in range(0,T):

        x = closest_index_1d(y_space, log_income[i])
```

```
            y = closest_index_1d(a_space, a[i])
            a[i+1] = state_policy_function[y][i][x]
            c[i]    =  (1 + r)*a[i] + income[i] - a[i+1]

    return (income, a, c)
```

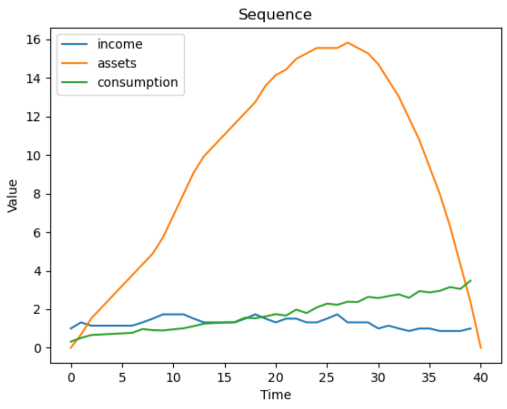
D Illustrations of sample runs

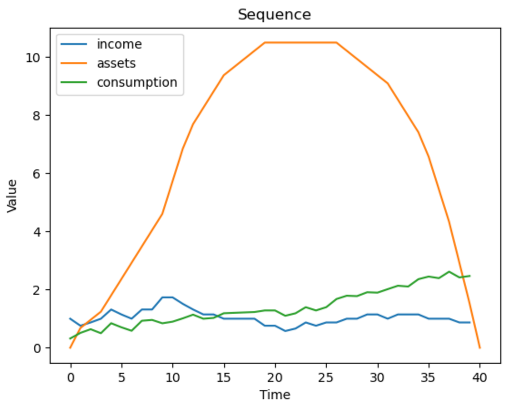
D.1 Case : $\underline{a}$ = -10





D.2 Case : $\underline{a}$ = -40





References

[1] Raul Santaeulalia-Llopis. “Value Function Methods (slides)”. In: (2018). URL: http://r-santaeulalia.net/pdfs/QM-Value_Function_Methods.pdf.

[2] George Tauchen. “Finite state markov-chain approximations to univariate and vector autoregressions”. In: *Economics Letters* 20.2 (1986), pp. 177–181. ISSN: 0165-1765. DOI: https://doi.org/10.1016/0165-1765(86)90168-0. URL: https://www.sciencedirect.com/science/article/pii/0165176586901680.