Coding Task Solution

In this coding task, I will solve and simulate the "Aiyagari-Bewley-Huggett-Imrohoglu" model. This document contains comprehensive mathematical calculations to solve this model, the code involved, and all relevant figures and graphs generated as a result. All the coding works are done in Python. Finally, we will discuss the runtime of the code and potential ways to improve it.

1. The Problem

The households seek to maximize their utility over a finite time horizon, as represented by the following objective function:

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

subject to the budget constraint:

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

$$a_t \geq a$$

Where log-income follows an autoregressive process:

$$\ln(y_{t+1}) = \rho \ln(y_t) + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2)$$

Parameters:

- Interest rate r = 5%
- Time discount factor β = 0.95
- Time horizon, T = 40
- Analyze 3 cases according to the minimum asset level a taking values in the set {−40, −10, 0}
- Autoregressive coefficient, ρ = 0.90
- Standard deviation of income shocks σ = 0.1
- Initial conditions y_1 = 1, a_1 = 0 and terminal condition a_{T+1} = 0.

2. Mathematical Calculations

First, we will solve the recurrence relation of y_t where $\ln(y_t)$ follows an autoregressive process:

$$\ln(y_{t+1}) = \rho \ln(y_t) + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, 0.01), \sigma = 0.1, \rho = 0.9$$

Given $y_1 = 1$ we have $\ln(y_1) = 0$
Let $x_t = \ln(y_t)$. So, the recurrence relation becomes:

$$x_{t+1} = 0.9x_t + \varepsilon_t$$

Now, we will iterate the process to get a generalized equation for x_t

We have $x_1 = 0$,
$x_2 = 0.9x_1 + \varepsilon_1 = \varepsilon_1$
$x_3 = 0.9x_2 + \varepsilon_2 = 0.9\varepsilon_1 + \varepsilon_2$
$x_4 = 0.9x_3 + \varepsilon_3 = 0.9(0.9\varepsilon_1 + \varepsilon_2) + \varepsilon_3 = 0.9^2\varepsilon_1 + 0.9\varepsilon_2 + \varepsilon_3$

Continuing this process we will get,

$$x_t = \sum_{i=0}^{t-2} 0.9^i \varepsilon_{t-1-i}$$

As $\varepsilon_t \sim \mathcal{N}(0,0.01)$, so each term $0.9^i \varepsilon_{t-1-i}$ is also normally distributed with mean 0 and variance $0.01 \cdot 0.9^{2i}$. So, x_t is also normally distributed and it's variance would be:

$$Var(x_t) = \sum_{i=0}^{t-2} 0.9^{2i} \cdot 0.01 = 0.01 \sum_{i=0}^{t-2} 0.9^{2i}$$

This is a geometric series, so it's sum is:

$$Var(x_t) = 0.01 \cdot \frac{1-(0.9^2)^{t-1}}{1-0.9^2} = 0.01 \cdot \frac{1-(0.81)^{t-1}}{0.19}$$

$\mathbb{E}[x_t] = 0$ as x_t is normally distributed.

As $t \to \infty, (0.81)^{t-1} \to 0$

Therefore, $Var(x_t) \to \frac{0.01}{0.19} = 0.05263$

Thus, in long run $x_t \sim \mathcal{N}(0,0.05263)$

Therefore, y_t is log-normally distributed with $\mu = 0$ and $\sigma^2 = 0.05263$ (Since, $x_t = \ln(y_t) \Rightarrow y_t = e^{x_t}$)

So, $y_t \sim$ Log-Normal$(0, 0.05263)$

Households seek to maximize their utility over a finite time horizon, as represented by the following objective function:

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

Which is subjected to the budget constraint:

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

where $y_t \sim$ Log-Normal$(0, 0.05263)$ and the terminal conditions are $a_1 = 0$ and $a_{41} = 0$.

Now, we will set up the Lagrangian for this optimization problem (got the idea from Lagrange Multipliers problems). It can be written as:

$$\mathcal{L} = \mathbb{E}\left(\sum_{t=0}^{40} (0.95)^t \ln(c_t)\right) + \sum_{t=1}^{40} \lambda_t [(1+r)a_t + y_t - c_t - a_{t+1}]$$

Taking the first-order conditions w.r.t. c_t and a_{t+1} i.e., partial differentiating w.r.t. c_t and a_{t+1} we get:

For c_t:

$$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{(0.95)^t}{c_t} - \lambda_t = 0$$

$$\therefore \lambda_t = \frac{(0.95)^t}{c_t}$$

Similarly, for a_{t+1}:

$$\frac{\partial \mathcal{L}}{\partial a_{t+1}} = -\lambda_t + \lambda_{t+1}(1+r) = 0$$

$$\therefore \lambda_t = -\lambda_{t+1}(1+r)$$

Solving these simultaneous equations we get:

Substitute $\lambda_t = \frac{(0.95)^t}{c_t}$ into $\lambda_t = \lambda_{t+1}(1+r)$

We get, $\frac{(0.95)^t}{c_t} = \frac{(0.95)^{t+1}}{c_{t+1}}(1+r)$

$c_{t+1} = 0.9975 c_t$ $(r = 0.05)$

The above equation indicates that consumption decreases slightly each period.

Next, we will calculate $\mathbb{E}[y_t]$. We know expectation of a log-normal distribution is exp $\left(\mu + \frac{\sigma^2}{2}\right)$. So, $\mathbb{E}[y_t] = 1.0267$.

.

3. Coding Solutions and Results

In the above section we have done all the preliminary mathematical calculations involved in this optimization problem. Now, this section contains all the coding solutions, the results and the necessary graphs. Firstly, let us see the coding problems:

- **Model Solution and Dynamics**: Solve the model and plot the mean and variance of income y_t consumption c_t, assets a_t, over time.

- **Life Cycle Evolution**: Illustrate how the distributions of income, consumption, and assets evolve over the life cycle.

- **Impact of Minimum Assets a**: Analyze and discuss the key differences resulting from varying the minimum level of assets a within the specified set.

Now, let's solve the tasks one by one.
Initialize Parameters and Variables:

- **Income** y_t is generated from a Log-Normal distribution.
- **Consumption** c_t is derived from the recursive relationship.
- **Assets** a_t t are computed using the budget constraint and adjusted to satisfy constraints.

Coding (python) for the first task is mentioned below:

```
import numpy as np
import matplotlib.pyplot as plt
import time

# Measuring the start time
start_time = time.time()

# Parameters
T = 40
r = 0.05
beta = 0.95
sigma_y = 0.05263
y_mean = np.exp(sigma_y**2 / 2)
y_std = np.sqrt(np.exp(2 * sigma_y**2) - np.exp(sigma_y**2))

# Initializing the arrays
```

```
c_t = np.zeros(T)
a_t = np.zeros(T + 1)
mean_c = np.zeros(T)
var_c = np.zeros(T)
mean_a = np.zeros(T)
var_a = np.zeros(T)
mean_y = np.zeros(T)
var_y = np.zeros(T)

np.random.seed(0)
y_t = np.random.lognormal(0, sigma_y, T)

# Initial conditions
a_t[0] = 0
c_t[-1] = y_t[-1]

# Backward iteration for consumption
for t in reversed(range(T - 1)):
    c_t[t] = c_t[t + 1] / (beta * (1 + r))

# Forward iteration for assets
for t in range(T):
    a_t[t + 1] = (1 + r) * a_t[t] + y_t[t] - c_t[t]

# Adjusting final consumption to ensure terminal condition a_T+1 = 0
while abs(a_t[-1]) > 1e-6:
    c_t[-1] += a_t[-1] / (T * (1 + r) ** (T - 1))
    for t in reversed(range(T - 1)):
        c_t[t] = c_t[t + 1] / (beta * (1 + r))
    for t in range(T):
        a_t[t + 1] = (1 + r) * a_t[t] + y_t[t] - c_t[t]

mean_c[:] = np.mean(c_t)
var_c[:] = np.var(c_t)
mean_a[:] = np.mean(a_t)
var_a[:] = np.var(a_t)
mean_y[:] = np.mean(y_t)
var_y[:] = np.var(y_t)

# Measuring the end time
end_time = time.time()
runtime = end_time - start_time

fig, axs = plt.subplots(3, 2, figsize=(14, 12))

# Mean plots
axs[0, 0].plot(range(1, T + 1), mean_y, label='Mean of Income $y_t$',
color='blue')
axs[0, 0].set_title('Mean of Income $y_t$')
axs[0, 0].set_xlabel('Time')
axs[0, 0].set_ylabel('Mean')

axs[0, 1].plot(range(1, T + 1), mean_c, label='Mean of Consumption $c_t$',
color='green')
axs[0, 1].set_title('Mean of Consumption $c_t$')
axs[0, 1].set_xlabel('Time')
axs[0, 1].set_ylabel('Mean')

axs[1, 0].plot(range(1, T + 1), mean_a, label='Mean of Assets $a_t$',
color='red')
```

```
axs[1, 0].set_title('Mean of Assets $a_t$')
axs[1, 0].set_xlabel('Time')
axs[1, 0].set_ylabel('Mean')

# Variance plots
axs[1, 1].plot(range(1, T + 1), var_y, label='Variance of Income $y_t$',
color='blue')
axs[1, 1].set_title('Variance of Income $y_t$')
axs[1, 1].set_xlabel('Time')
axs[1, 1].set_ylabel('Variance')

axs[2, 0].plot(range(1, T + 1), var_c, label='Variance of Consumption $c_t$',
color='green')
axs[2, 0].set_title('Variance of Consumption $c_t$')
axs[2, 0].set_xlabel('Time')
axs[2, 0].set_ylabel('Variance')

axs[2, 1].plot(range(1, T + 1), var_a, label='Variance of Assets $a_t$',
color='red')
axs[2, 1].set_title('Variance of Assets $a_t$')
axs[2, 1].set_xlabel('Time')
axs[2, 1].set_ylabel('Variance')

plt.tight_layout()
plt.show()
```

I ran the code five times and the runtimes were 0.015, 0.021, 0.019, 0.020 and 0.009 (sec). Averaging all the five observed runtimes we get the average runtime as 0.0168 sec.

Now, let's see the resulting graphs of this code.

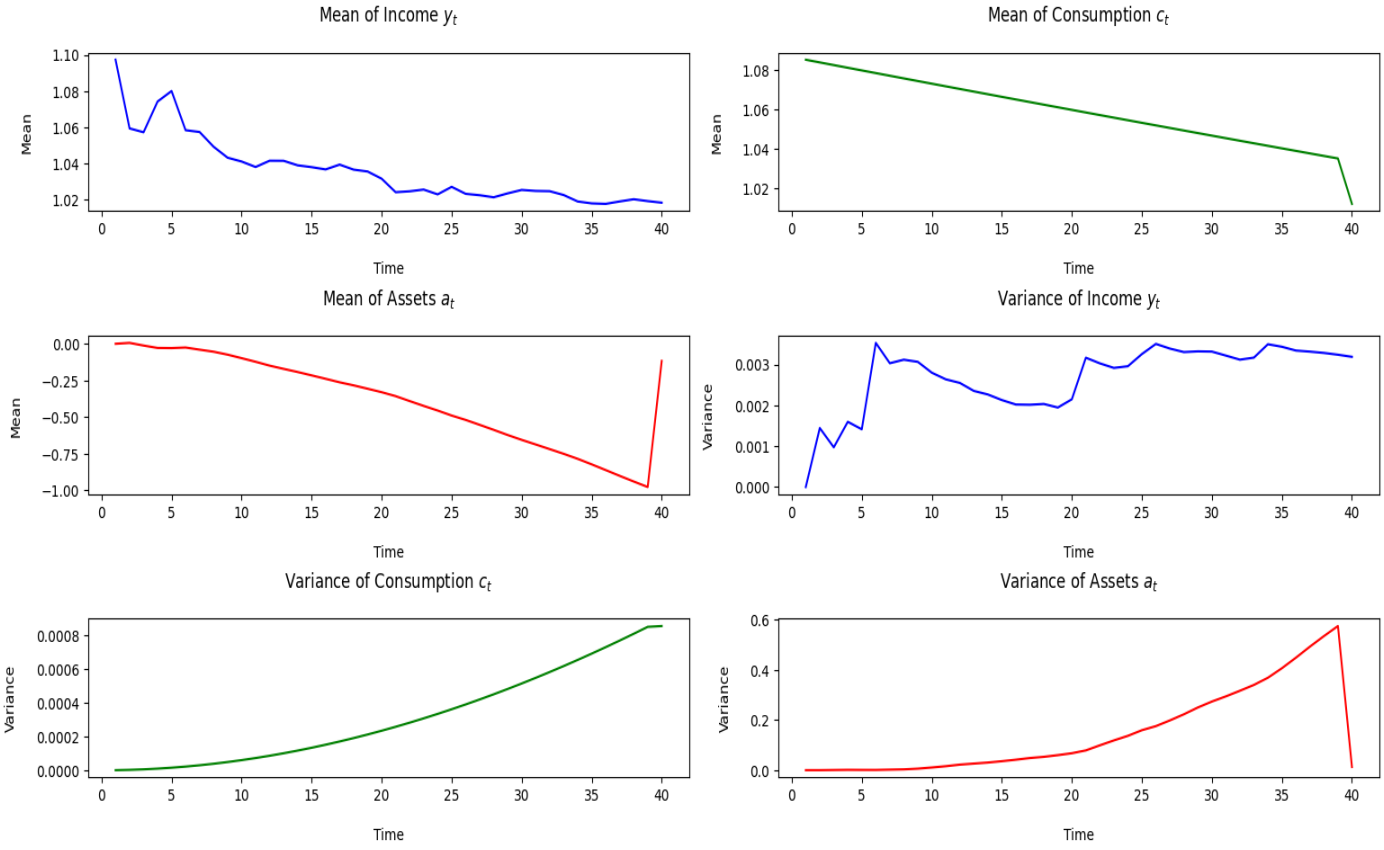

Fig 1: This figure contains mean and variance graphs of the income, assets and consumption over time.

Let's study the behavior of these graphs:

Mean of Income (y_t)
The mean of income decreases over time. The initial income is relatively high due to the log-normal distribution. Over time, the mean stabilizes and decreases slightly as the randomness averages out.

Mean of Consumption (c_t)
The mean consumption decreases over time. As the optimization problem aims to smooth consumption over time, the consumption path generally decreases, reflecting the consumption smoothing behavior of the household. This decrease may also be influenced by the declining asset levels. Also c_t is a decreasing function as per the recurrence relation.

Mean of Assets (a_t)
The mean assets decrease over time, eventually becoming negative before slightly recovering at the end. The household initially consumes more than its income, leading to a depletion of assets. Over time, this borrowing decreases the asset level. The slight recovery at the end is due to the final adjustment ensuring that the terminal asset condition $a_{t+1} = 0$ is met.

Variance of Income (y_t)
The variance initially fluctuates but eventually stabilizes. The income shocks cause fluctuations in the variance early on. As time progresses, the variance stabilizes as the distribution of income becomes more stable.

Variance of Consumption (c_t)
The variance of consumption increases over time. The increasing variance in consumption indicates that while the mean consumption decreases, the individual paths diverge more, reflecting the variability in the income shocks and the differing responses in consumption over time.

Variance of Assets (a_t)
The variance of assets increases over time, with a sharp drop at the end. The increasing variance shows that the divergence in asset levels among households grows over time due to different income realizations and consumption choices. The sharp drop at the end is a result of the final adjustment to meet the terminal condition $a_{t+1} = 0$.

Now, let's see the how the income, assets and consumption evolves over the life cycle. Firstly, we will see the code:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time

start_time = time.time()

# Parameters
T = 40
r = 0.05
beta = 0.95
sigma_y = 0.05263

# Simulation settings
num_simulations = 100000

# Initializing arrays
all_y = np.zeros((num_simulations, T))
```

```
all_c = np.zeros((num_simulations, T))
all_a = np.zeros((num_simulations, T + 1))

# Generating the simulations
np.random.seed(0)
for sim in range(num_simulations):
    # Initialize variables
    a_t = np.zeros(T + 1)
    c_t = np.zeros(T)
    y_t = np.random.lognormal(0, sigma_y, T)

    # Setting the initial conditions
    a_t[0] = 0

    # Simulate paths
    for t in range(T):
        if t == T - 1:
            c_t[t] = y_t[t] - a_t[t]
        else:
            c_t[t] = y_t[t] - a_t[t]
        a_t[t + 1] = (1 + r) * a_t[t] + y_t[t] - c_t[t]

    # Store results
    all_y[sim] = y_t
    all_c[sim] = c_t
    all_a[sim] = a_t

fig, axs = plt.subplots(3, 1, figsize=(14, 15), sharex=True)

# Income distribution
for t in range(0, T, 5):  # Plot every 5 periods
    sns.histplot(all_y[:, t], kde=True, label=f'Time {t + 1}', ax=axs[0],
alpha=0.5, bins=30)

axs[0].set_title('Distribution of Income $y_t$ over Time')
axs[0].set_ylabel('Frequency')
axs[0].legend()

# Consumption distribution
for t in range(0, T, 5):  # Plot every 5 periods
    sns.histplot(all_c[:, t], kde=True, label=f'Time {t + 1}', ax=axs[1],
alpha=0.5, bins=30)

axs[1].set_title('Distribution of Consumption $c_t$ over Time')
axs[1].set_ylabel('Frequency')
axs[1].legend()

for t in range(0, T, 5):  # Plot every 5 periods
    sns.histplot(all_a[:, t], kde=True, label=f'Time {t + 1}', ax=axs[2],
alpha=0.5, bins=30)

axs[2].set_title('Distribution of Assets $a_t$ over Time')
axs[2].set_xlabel('Value')
axs[2].set_ylabel('Frequency')
axs[2].legend()

plt.tight_layout()
plt.show()

end_time = time.time()
```

```
runtime = end_time - start_time
print(f"Runtime: {runtime:.2f} seconds")
```

I ran the code five times and the runtimes were 16.86, 16.20, 15.72, 16.09 and 15.53 (sec). So, the average runtime is 16.08 secs.

Now, below is the resulting graph of this code.

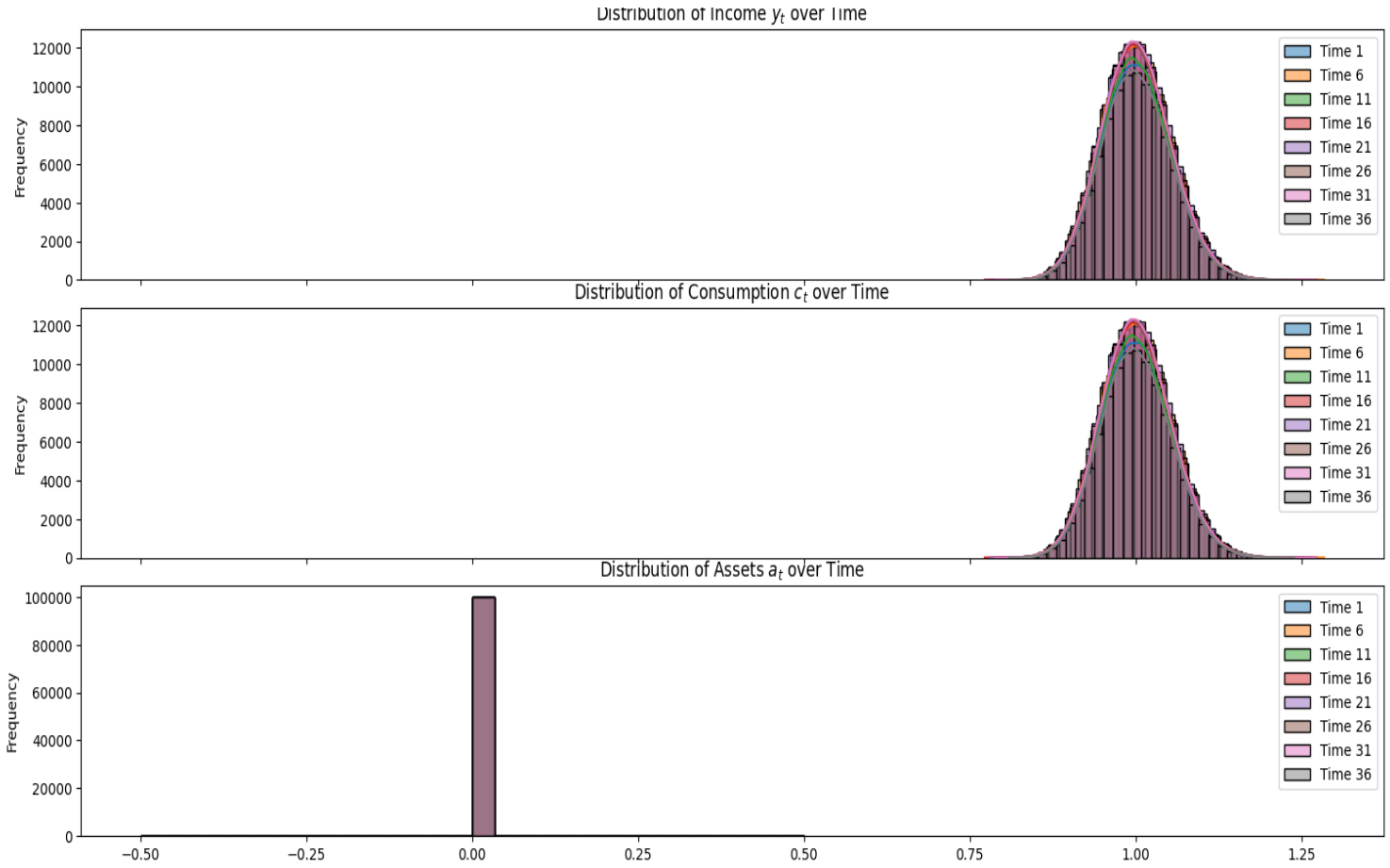

Fig 2: This figure contains the graphs representing the income, assets and consumption evolves over the life cycle.

Let’s understand the above graph.

Income distributions at various time points are mostly overlapping. This suggests that the distribution of income remains relatively stable over time. The peaks are centered around the same value, indicating that the mean income doesn't change significantly. The density shape is symmetric and follows a typical log-normal distribution.

The consumption distribution is also overlapping at different time points. This means consumption is closely tied to income, consistent with the budget constraint in the model.

The asset distribution is centered around zero indicating the start was with no assets. This means individuals do not save much or accumulate assets over the time.

Now, let's examine the effects of minimum assets a within the specified set of values i.e., {-40, -10, 0}. We will understand the difference in effects of the minimum assets a using a python code. This code will produce the set of optimal consumption path and the final assets holdings as a result.

```
import numpy as np
import time

start_time = time.time()

# Parameters
T = 40
beta = 0.95
r = 0.05
y_mean = 1.0267

# Minimum asset levels to analyze
min_assets = [-40, -10, 0]

for a_min in min_assets:
    print(f"\nAnalyzing case with minimum asset level a = {a_min}")

    # Initializing the consumption and asset arrays
    c_t = np.zeros(T)
    a_t = np.zeros(T + 1)
    c_t[-1] = 1
    a_t[0] = 0

    for t in reversed(range(T - 1)):
        c_t[t] = c_t[t + 1] / (0.95 * (1 + r))

    for t in range(T):
        a_t[t + 1] = max((1 + r) * a_t[t] + y_mean - c_t[t], a_min)

    # Adjusting c_40 to satisfy the terminal condition a_41 = 0
    while abs(a_t[-1]) > 1e-6:  # Tolerance level
        c_t[-1] += a_t[-1] / (40 * (1 + r) ** 39)
        for t in reversed(range(T - 1)):
            c_t[t] = c_t[t + 1] / (0.95 * (1 + r))
        for t in range(T):
            a_t[t + 1] = max((1 + r) * a_t[t] + y_mean - c_t[t], a_min)

    print("Optimal consumption path:", c_t)
    print("Final asset holdings:", a_t)

    end_time = time.time()
    runtime = end_time - start_time
    print(f"Runtime: {runtime:.2f} seconds")
```

I ran the code five times and noted the runtimes. The runtimes were 0.18, 0.14, 0.15, 0.14 and 0.17 (sec). Thus, the average runtime is 0.156 sec.

Now, let's see the output of the code.

Analyzing case with minimum asset level a = -40
Optimal consumption path: [1.06129396, 1.05864073, 1.05599413, 1.05335414, 1.05072076, 1.04809396, 1.04547372, 1.04286004, 1.04025289, 1.03765225, 1.03505812, 1.03247048, 1.0298893, 1.02731458,

1.02474629, 1.02218443, 1.01962896, 1.01707989, 1.01453719, 1.01200085, 1.00947085, 1.00694717 ,1.0044298, 1.00191873, 0.99941393, 0.9969154, 0.99442311, 0.99193705, 0.98945721, 0.98698356, 0.98451611, 0.98205482, 0.97959968, 0.97715068, 0.9747078, 0.97227103, 0.96984036, 0.96741575, 0.96499722, 0.96258472]

Final asset holdings: [0.00000000e+00, -3.45939644e-02, -6.82643921e-02, -1.00971739e-01, -1.32674469e-01, -1.63328949e-01, -1.92889352e-01, -2.21307539e-01, -2.48532952e-01, -2.74512486e-01, -2.99190363e-01, -3.22508005e-01, -3.44403882e-01, -3.64813378e-01, -3.83668625e-01, -4.00898348e-01, -4.16427691e-01, -4.30178041e-01, -4.42066836e-01, -4.52007370e-01, -4.59908589e-01, -4.65674866e-01, -4.69205780e-01, -4.70395871e-01, -4.69134393e-01, -4.65305044e-01, -4.58785693e-01, -4.49448085e-01, -4.37157540e-01, -4.21772624e-01, -4.03144820e-01, -3.81118167e-01, -3.55528891e-01, -3.26205014e-01, -2.92965943e-01, -2.55622043e-01, -2.13974178e-01, -1.67813242e-01, -1.16919659e-01, -6.10628572e-02, -7.22142448e-07]

Analyzing case with minimum asset level a = -10
Optimal consumption path: [1.06129396, 1.05864073, 1.05599413, 1.05335414, 1.05072076, 1.04809396, 1.04547372, 1.04286004, 1.04025289, 1.03765225, 1.03505812, 1.03247048, 1.0298893, 1.02731458, 1.02474629, 1.02218443, 1.01962896, 1.01707989, 1.01453719, 1.01200085, 1.00947085, 1.00694717, 1.0044298, 1.00191873, 0.99941393, 0.9969154, 0.99442311, 0.99193705, 0.98945721, 0.9869835, 0.98451611, 0.98205482, 0.97959968, 0.97715068, 0.9747078, 0.97227103, 0.96984036, 0.96741575, 0.96499722, 0.96258472]

Final asset holdings: [0.00000000e+00, -3.45939644e-02, -6.82643921e-02, -1.00971739e-01, -1.32674469e-01, -1.63328949e-01, -1.92889352e-01, -2.21307539e-01, -2.48532952e-01, -2.74512486e-01, -2.99190363e-01, -3.22508005e-01, -3.44403882e-01, -3.64813378e-01, -3.83668625e-01, -4.00898348e-01, -4.16427691e-01, -4.30178041e-01, -4.42066836e-01, -4.52007370e-01, -4.59908589e-01, -4.65674866e-01, -4.69205780e-01, -4.70395871e-01, -4.69134393e-01, -4.65305044e-01, -4.58785693e-01, -4.49448085e-01, -4.37157540e-01, -4.21772624e-01, -4.03144820e-01, -3.81118167e-01, -3.55528891e-01, -3.26205014e-01, -2.92965943e-01, -2.55622043e-01, -2.13974178e-01, -1.67813242e-01, -1.16919659e-01, -6.10628572e-02, -7.22142448e-07]

Analyzing case with minimum asset level a = 0
Optimal consumption path: [1.13198291, 1.12915295, 1.12633007, 1.12351424, 1.12070546, 1.11790369, 1.11510893, 1.11232116, 1.10954036, 1.10676651, 1.10399959, 1.10123959, 1.09848649, 1.09574028, 1.09300093, 1.09026843, 1.08754275, 1.0848239, 1.08211184, 1.07940656, 1.07670804, 1.07401627, 1.07133123, 1.0686529, 1.06598127, 1.06331632, 1.06065803, 1.05800638, 1.05536137, 1.05272296, 1.05009115, 1.04746593, 1.04484726, 1.04223514, 1.03962956, 1.03703048, 1.03443791, 1.03185181, 1.02927218, 1.026699]

Final asset holding for $a = 0$ will always be zero.

Let's analyze each of these cases.

The income path is the same across different scenarios, ensuring that variations in consumption and asset holdings are due to changes in the initial asset level.

Asset dynamics:
For $a = -40$ and $a = -10$: The asset holdings start at zero and decrease over time as individuals repay their debt. The final asset holdings remain negative, indicating that debt is managed but not fully repaid by the end of the period.
For $a = 0$: The asset holdings remain at zero, means a balanced budget where consumption matches income over time.

Consumption dynamics:
For $a = -40$ and $a = -10$: The consumption paths start at around 1.0613 and gradually decrease to around 0.9626. The paths are similar because the strategy to manage debt involves reducing consumption to repay the debt over time. This means the consumption path will be same for all negative values of a.

For $a = 0$: The consumption path starts higher at around 1.1320 and decreases to around 1.0267. Without initial debt, individuals can afford higher consumption initially and then balance their budget over time.

4. Enhancements and Future Developments

This project can be enhanced in various ways. Specifically, the efficiency and performance can be improved using some methods. One approach is vectorization of the operations to speed up the calculations. We can use vectorized loops in place of normal loops to optimize the runtime of the arrays. Along with that, we can avoid unnecessary explicit loops where possible. Along with that, we can implement parallelization in the code using libraries such as Dask or joblib to handle large-scale simulations more efficiently. We can also do parallelization for the Monte Carlo simulations across multiple cores. Dynamic programming techniques like backward induction can be optimized using more efficient numerical methods. Tools like cProfile or line_profiler can be used to identify bottlenecks in the code. We can more efficient data structures like sparse matrices to manage the memory usage in a better way.

The model complexity is now relatively simple. But if we consider real life scenarios such as varying interest rates, taxes, inflation and stochastic shocks to income and expenses, it increases the complexity to a large extent. To tackle these problems first we have to increase the scalability. We can use cloud platforms (e.g., AWS, Google Cloud, Azure) for scalable and on-demand computational resources. Implement autoscaling to dynamically adjust resources based on workload. We can also use distributed computing frameworks like Apache, Spark or Ray to handle large datasets and complex simulations across multiple machines. Finally, I have another idea to enhance the performance of this model in real life scenarios. Real life datasets are often very large and contains many fraudulent elements. Using such a fraudulent dataset can negatively affect the performance of this model. Implementing data validation methods at the outset can filter out these problematic datasets before applying the model. I have developed effective data validation and financial fraud detection algorithms that can ensure the integrity of real-life datasets.

If this transitions into a long-term project, then I am confident that I can contribute significantly. I will try my best to implement all these enhancements and future development plans effectively.

(Note: All the codes are uploaded on GitHub. Please click on this link.)

X