Quantitative Macroeconomics 2 PS1

Zhihang Liang, Tuba Şeker

Spring 2024

This report includes different approaches to implement the algorithm to solve for the steady-state of the Aiyagari model. We add in each step some techniques of the algorithm implementation and see how these techniques change the speed and the policy function. The results are summarized in figures in section . The figures are numbered and the numbers correspond to each step in the following.

1. Improve the guess for the value function. Start with a first guess implied by the constant policy $a' = a$. After updating the interest rate, use the value function of the previous guess for r as a new guess.

The initial guess of the value function is given by:

$$V^0(a, \epsilon) = \frac{u(c(a, \epsilon))}{1 - \beta} \tag{1}$$

where $c(a, \epsilon) = \epsilon \cdot wage + ra$.

This ensures the monotonicity of the initial guess for the value function and makes the guess closer to the true value function.

With this initial guess, the computation time decreases from 20.80 seconds to 17.71 seconds. This is not a huge improvement.

The results are summarized in figure 1. Though the policy function is not smooth, which indicates low precision, we can still make some comments about the intuition conveyed by the figure.

Firstly, the asset policy function for the highest-skilled workers is always above the 45-degree line, which indicates that workers with the highest level of earnings keep accumulating assets. This feature of over-saving is the mirror to the under-saving of low-skill workers. Low-skill workers have a higher propensity to consume: the low interest rate induced by the incomplete market with borrowing constraints makes low-skill workers more tempted to consume. One can see this in the value function: the curvature of the value function of the low-skill workers is sufficiently larger than that of the high-skill workers.

Secondly, the consumption policy function for the high skilled workers is not concave, which contrasts starkly with the complete market case. As asset holdings increase, the high-skilled workers have a higher marginal consumption level. This might be explained by the low interest rate and the nearly flat value function for these agents. Put differently, agents with high productivity shock will have a significant increase in consumption at the right tail. This is because the maximum level of asset holdings will bind and prevent these individuals from saving more; thus, increasing their consumption of productivity shock immediately.

Thirdly, the distribution is also interesting. We can see that for the low-skilled workers, there's a mass of workers that are borrowing constrained. Due to a positive mass of constrained agents in the market, the market exhibits inefficiency: those who have a large accumulation of assets cannot lend to constrained agents, and the fear of being borrowing constrained leads to over-saving. On the other hand, we can interpret this argument differently: the mass in the right tail of distribution might indicate that the chosen level of the maximum asset level is too low that it creates a concentration of assets over the top tail of the distribution. By increasing this threshold, we may allocate individuals who have higher savings motives to a wider distribution (see question 6).

2. Add the Howard step.

The Howard iteration suggests using the computed policy function in each step of the value function iteration to update the value function. Within the Howard iteration step, we sacrifice some stability in exchange for efficiency. For every 5 steps of the value function iteration, we implement the Howard iteration 20 times (in class, the suggested step is to implement the Howard iteration until it reaches some precision level).

Indeed, Howard iteration accelerates the computation to a large extent: the computation time reduces to 8.73 seconds. On the other hand, while we get efficiency for computation time, the output precision does not exhibit a significant change.

The code is presented below:

```
% Howard
if optHoward==1 && mod(iter,5)==1
     % Initialise variable
        dVV=1;
        v1how=tv;
        % Loop
            % V(k) = u(c(k)) + beta*V(k')
            % v1how = (c_VFI.^(1-mu) - 1) / (1-mu) + beta*prob(j,:)*v1how(tdecis);
        for k = 1:20
            for j = 1:N
                for i = 1:nkap
                    v1how(j,i)=(c_VFI(j,i)^(1-mu)-1)/(1-mu) + beta*prob(j,:)*v1how(:,tdecis(j,i));
                end
            end
        end
            %disp(dVV)
            tv=v1how;
    end
```

The result is presented in figure 2.

3. Update the measure using matrix computations instead of a simulation.

To compute the distribution of $\lambda(a,\epsilon)$, we use the matrix computation. We specify the matrix Q, such that the next-period distribution $\lambda^{'}(\cdot)$ can be obtained from $\lambda^{'} = Q\lambda$ where Q is built using policy function $a'(\alpha,\epsilon)$ and transition matrix of the shocks $\pi(\epsilon'|\epsilon)$. We start from any distribution λ, and the invariability of the result is ensured by the ergodicity. Intuitively, since we are in a stationary equilibrium, $\lambda(\alpha,\epsilon)$ should be the same today and tomorrow.

This also reduces the computation time to 1.782 seconds, and increases the precision of the computation of the steady-state distribution. The result is shown in figure 3.

4. Update the search for r using the implied interest rate (with slow weights!) rather than a simulation.

Instead of using bisection to search for the true interest rate r, we use the implied interest rate to update the interest rate in each step,

$$r_0 = 0.8r_0 + 0.2r_1;$$

where r_0 is the initial guess and r_1 is the interest rate computed from the marginal production of aggregate capital, where the aggregate capital is computed from the demand of heterogenous households. By giving higher weight to the initial guess, we aim to refrain from any large update of the bounds, which is a prevailing problem with the bisection method.

The computation time is reduced from 1.78 seconds to 1.13 seconds. The result is shown in figure 4. Again, the policy function is discontinuous, but the pattern is the same as in figure 1 to figure 3, which indicates the stability of the algorithm.

5. Update the algorithm to use a Golden search rather than a grid search.
(a) Create a function to linearly interpolate your value function between two asset points.
(b) Use a Golden algorithm to find the optimal a', and compute the new value function. Use the Golden code I've sent you.
(c) Adjust the matrix computation of the measure.

We use linear interpolation of asset values so that we can conduct off-grid maximization during value function iteration. The trade-off is between precision and efficiency. The code is presented below:

```
while test > 0.001
    iter = iter + 1;
    % Optimisation
    for j = 1:N
        for i = 1:nkap
        auxf = @(x) newV(x,i, j, wage, r0);
        upperK = min(maxkap, wage*s(j) + (1+r0)*kap(i) - 0.001);
        [aprime_VFI(j,i), tv(j,i)] = ...
                        golden(auxf,minkap,upperK);
            % We extract the nearest elements on the grid and the weights,
            [~,tdecis(j,i)]= min(abs(aprime_VFI(j,i) - kap));
        end
    end

    c_VFI = s'*wage + (1+r0)*kap - aprime_VFI;
    test=max(max(abs(tv-v))); % distance
    v=tv;
end
```

Here, the `newV` function is the interpolated function.

In order to implement the computation of distribution with the method presented in question 4, we record the nearest grid to the next-period capital. Thus, the computation of measure maintains on the grid.

The result is presented in figure 5. We can see in figure 5 that the policy function is much smoother, indicating a higher level of precision. However, the computation time increases substantially to 555 seconds.

Note that in this step, we gave up the Howard improvement step: there's little improvement and the computation time doesn't decrease. This suggests that the Howard acceleration in essence is unstable and needs to be treated with cautiousness.

6. Finally, revisit your asset grid. Implement a log-grid. Increase the upper bound to 50. What does it change? Do you want to increase the number of points for the asset-grid?

We revisit the asset grid and apply a log-grid which enables us to concentrate more points where the curvature of the value function is higher and results in higher accuracy. The number of grids increases from 403 to 415, and the range of value increases. The computation time is 869 seconds. Compared with step 5, the computation time is at the same level, but with a wider range of asset. We can see in figure 6 that the asset distribution doesn't admit any mass at the maximum capital anymore. This suggests better precision, as capital owners are allowed to accumulate more assets with a wider asset space, which is closer to the true model. Thus, we conclude that it's better to use the log-grid and enlarge the asset space, and there's no need to increase the number of grids.

Figures

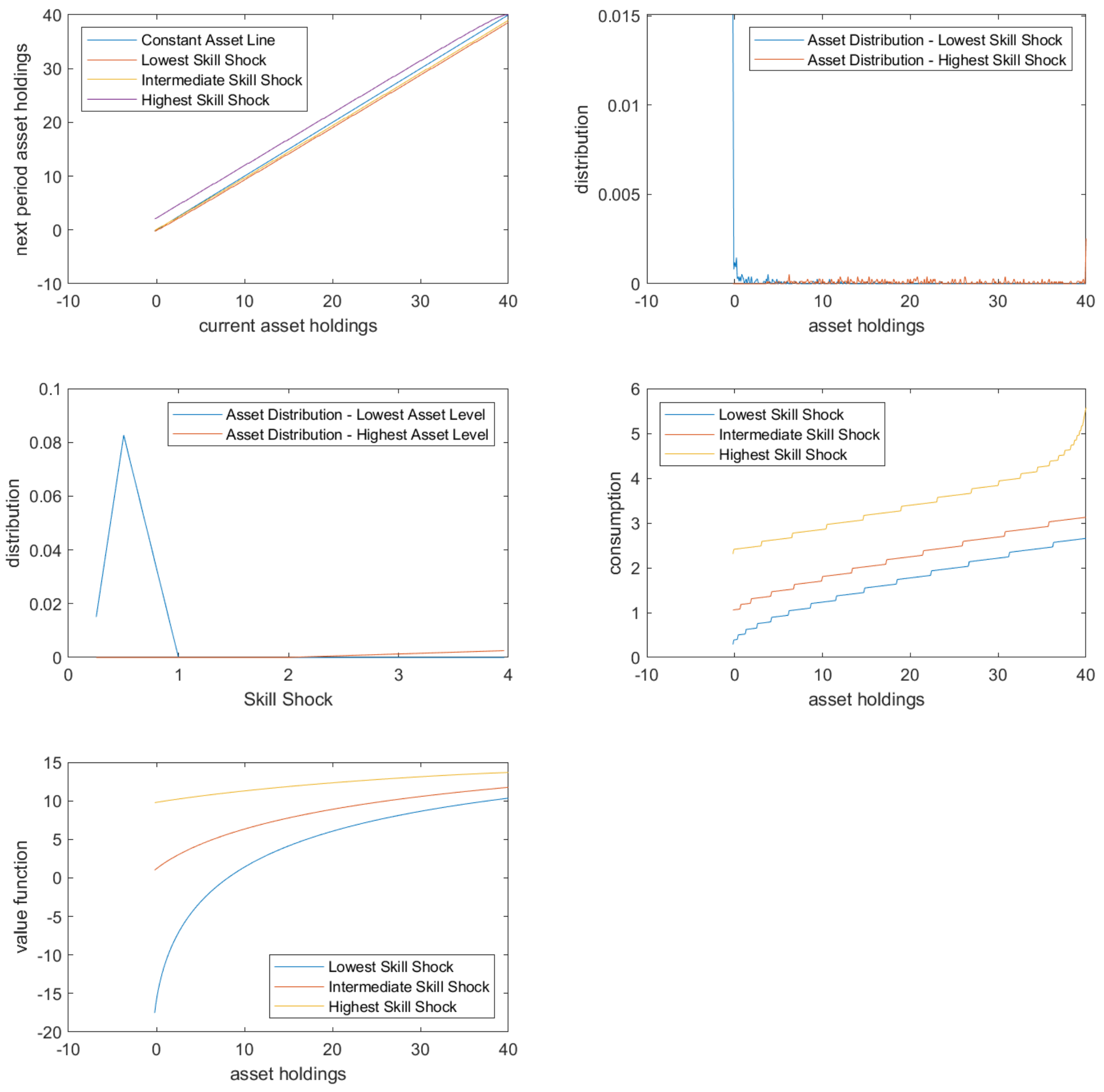

Figure 1: Aiyagari V1

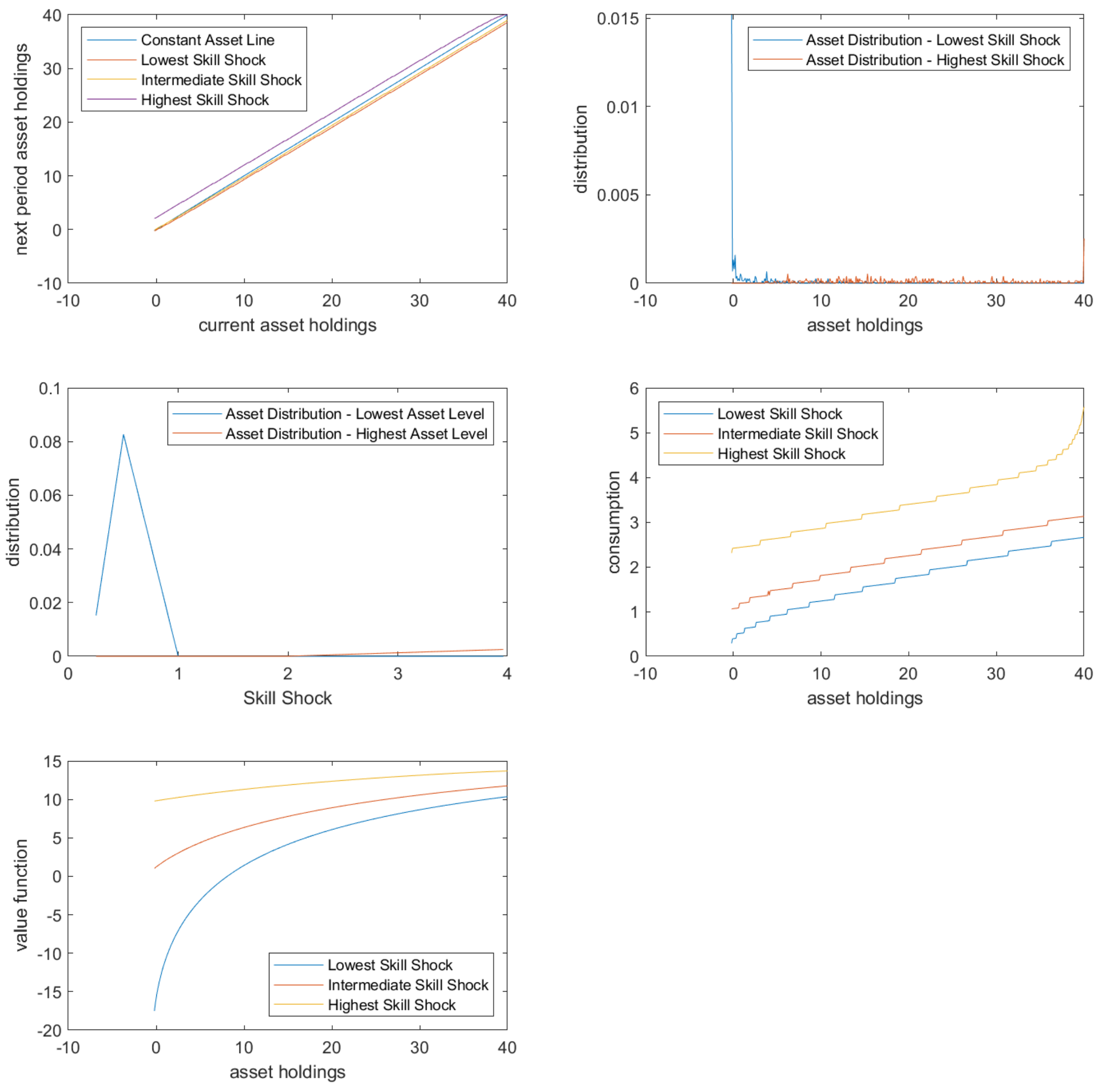

Figure 2: Aiyagari V2

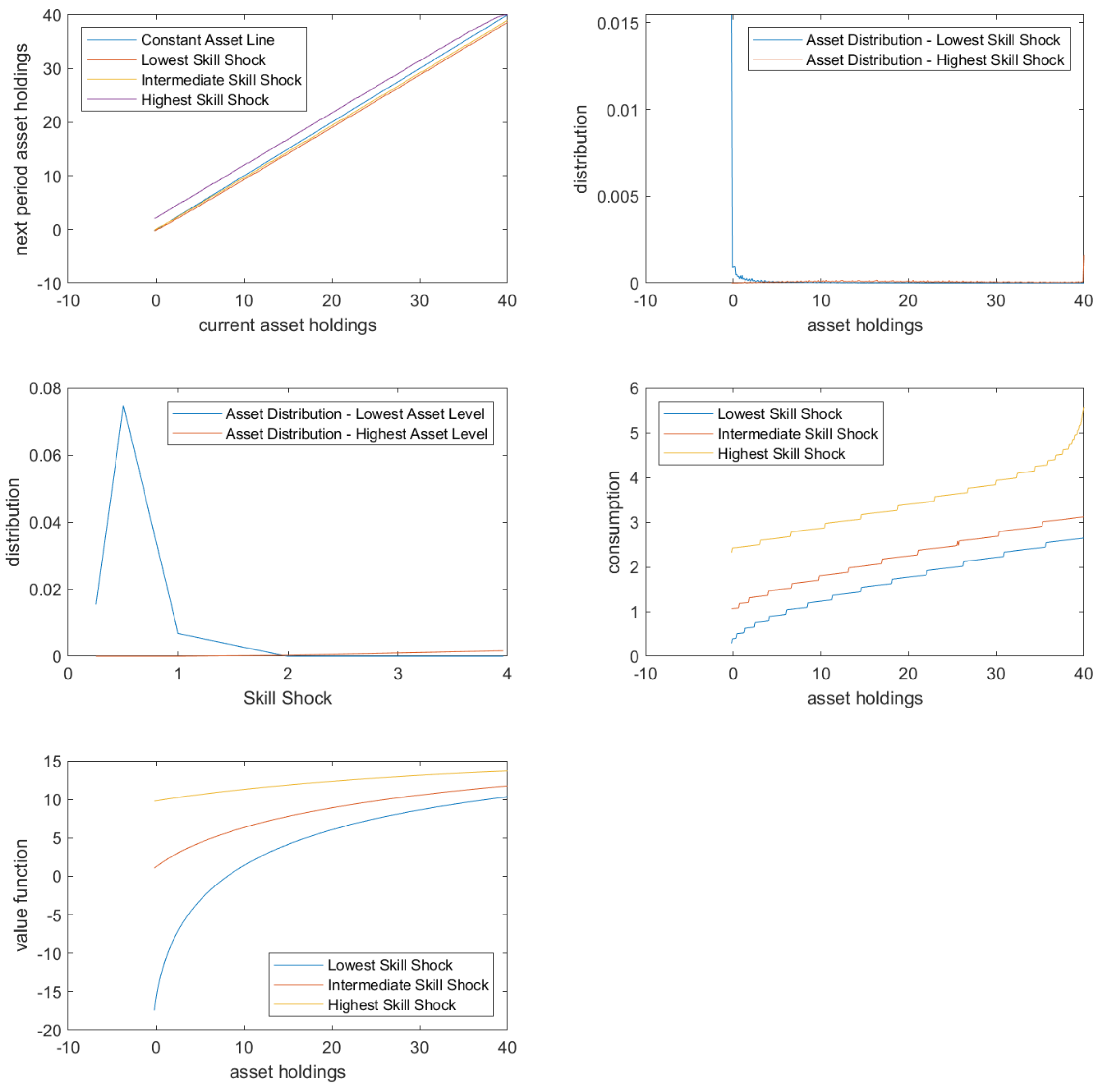

Figure 3: Aiyagari V3

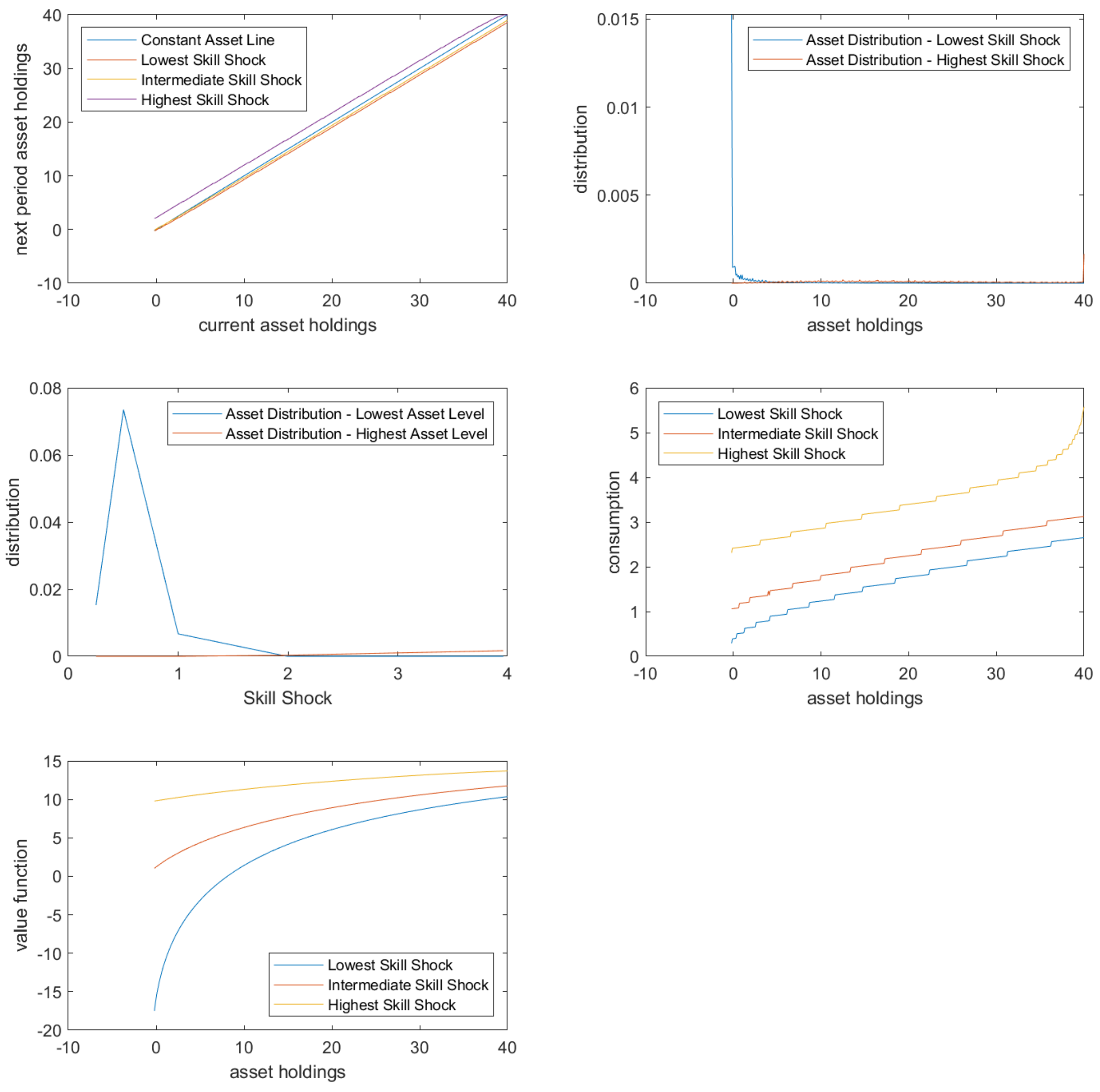

Figure 4: Aiyagari V4

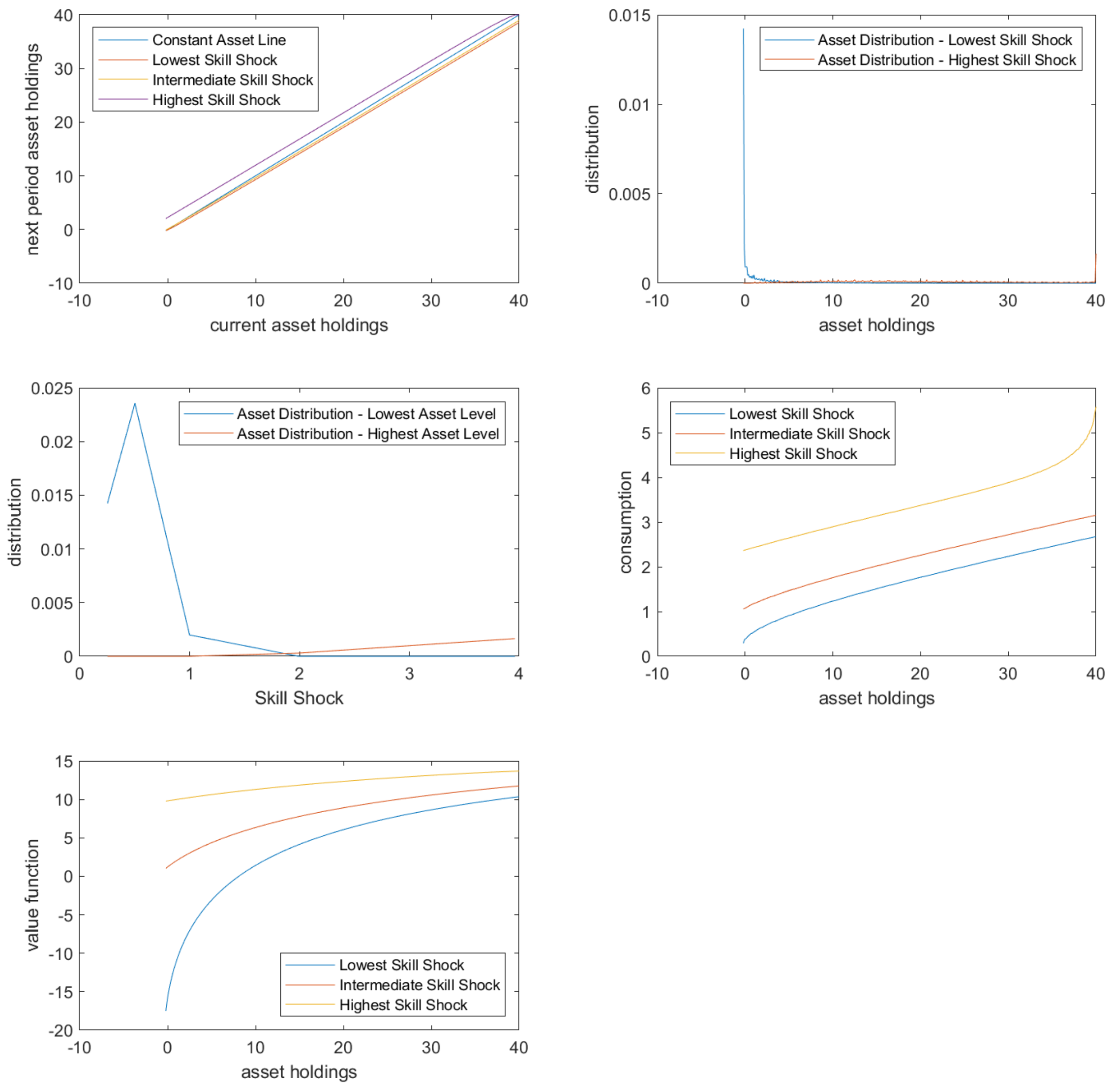

Figure 5: Aiyagari V5

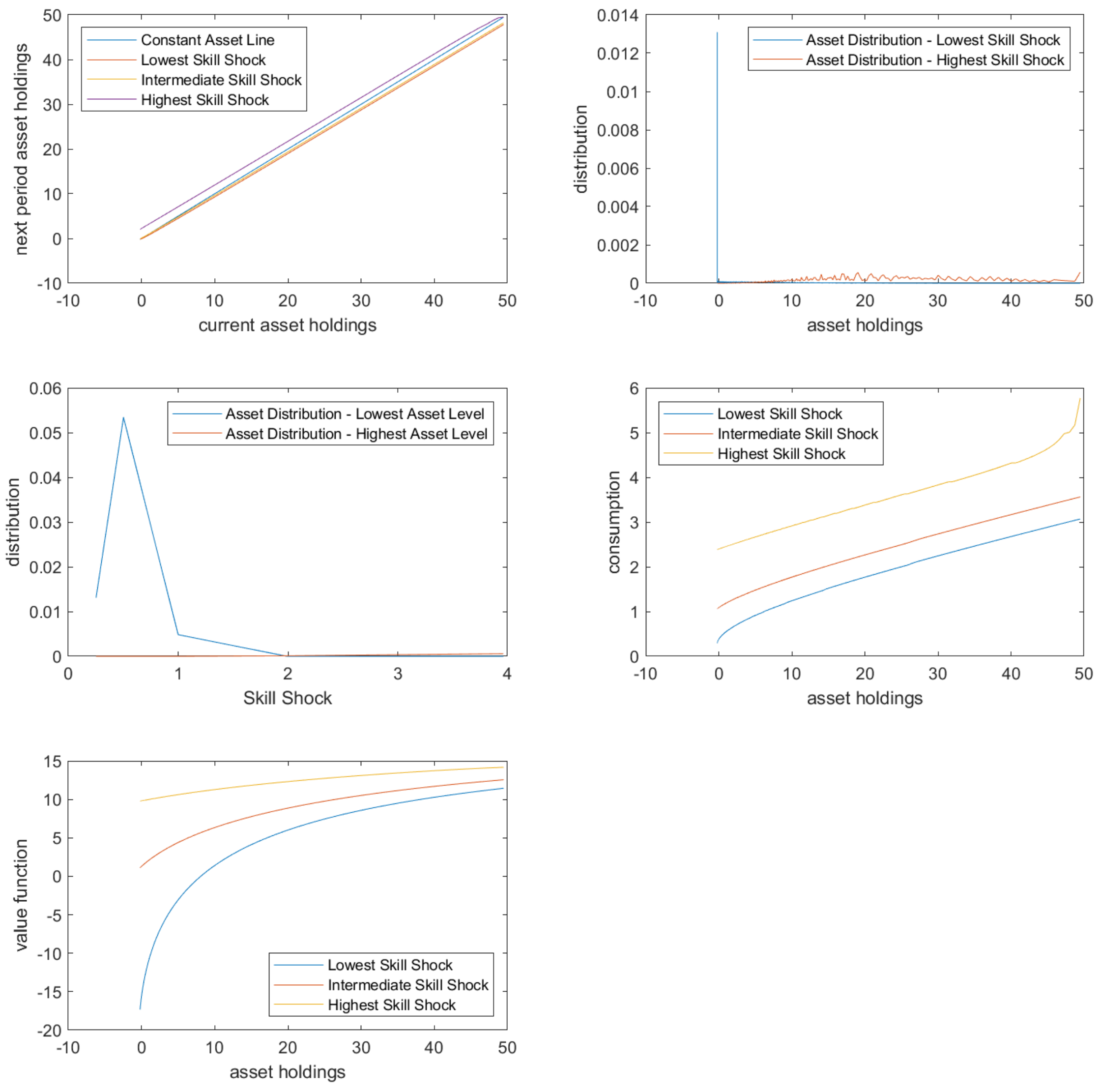

Figure 6: Aiyagari V6