Quantitative Macroeconomics 2 PS3

Zhihang Liang, Tuba Şeker

Spring 2024

Exercise 6. We start from Exercise 5 of Problem Set 2.

1. **Start from Exercise 5, question 2. Use Golden on a' and VFI to compute value functions, but compute the function $\hat{n}(a, \varepsilon, a')$ using Newton rather than a bisection method. Which one is faster?**

In the Newton method, we start from the policy function which prices are given in the previous iteration rather than found from bounds. Thus, the price jump will be smaller and smaller, and Newton will be a faster algorithm unless we start from far away to the true solution or if the solution is not well-defined.

In this problem, we need to find n^* such that $f(n) \approx f(n_0) + f'(n_0)(n - n_0) = 0$. However, there is a lower bound and an upper bound for labor n. Newton's method might deliver a guess that's out of bounds. Considering the fact that the FOC for labor is monotonic, we combine the Newton method and the bisection method. We first update the new guess in each iteration by using the standard formula of the Newton method:

$$n_{\text{new}} = n - \frac{f(n)}{f'(n)} \tag{1}$$

Then, we check if n_{new} is out of bound or not. If n_{new} doesn't belong to (n_l, n_u), we update

$$n_{\text{new}} = \frac{(n_l + n_u)}{2} \tag{2}$$

Next, we update n_l and n_u. If $f(n_{new}) < 0$, we update $n_l = n_{new}$. If $f(n_{new}) > 0$, we update $n_u = n_{new}$. We initialize n_l by using the fact that consumption is non-negative:

```
nlow = (kap-(1+r0)*kap')/(wage*s(j)) ;
nlow = max(nlow,0);
```

The complete code for this problem is given below:

```
% n(a,a', epsilon)
for j=1:N
    % Initialise variables
        % Bounds
        nlow = (kap-(1+r0)*kap')/(wage*s(j)) ;
        nlow = max(nlow,0);
        nup = ones(nkap, nkap);
        nlow(nlow>nup) = 1-1e-3;
        % Loop criterium
        maxtestn = 1;
        n0 = squeeze(nopt(:,:,j));
    % Loop: we combine newton method and bisection method.
    % bisection method is robust in the sense that it will not create a negative consumption
    % monotonicity of the focn function ensures that bisection works.
    % Newton is faster but not robust.
    while (maxtestn>0.00001)
```

```
        foc = focn( s(j),n0, wage, r0);
        first_dev = first_dev_focn(s(j),n0, wage, r0);
        n1 = n0 - foc./first_dev;
        % if n1 out of the bound,
        n1(n1<nlow | n1>nup ) = 0.5*nlow(n1<nlow | n1>nup) + 0.5*nup(n1<nlow|n1>nup);
        foc1 = focn(s(j),n1,wage,r0);
        nlow(foc1<0 & n1<=nup & n1>=nlow) = n1(foc1<0 & n1<=nup & n1>=nlow);
        nup(foc1>=0 & n1<=nup & n1>=nlow) = n1(foc1>=0 & n1<=nup & n1>=nlow);
        maxtestn = max(abs(foc1));
        n0 = n1;
    end
    nopt(:,:,j)= n1 ;
    % Rows: a. Columns: a'. 3rd dimension: current productivity.
end
```

Another notable technical detail is that we declare `nopt` as a global variable. Thus, when we update the guess for interest rate r, the code recalculate $nopt(a, a', \epsilon)$ starting from the solution of the last loop, which accelerate the calculation to a great extent. Indeed, the code takes 25.60 seconds, while previously, it was 34 seconds.

2. **Now, replace the Golden on a' by a Golden on c. Be very careful with the bounds on a'. Does it work better? Faster? Do you still need a root finder to find n ?**

Replacing the Golden on a' by a Golden on c allows us to skip the initial optimization over the labor decision, as we can deduce current labor from the first-order condition. So, we no longer need to interpolate labor supply in the Golden algorithm.

On the other hand, this time problem is harder since bounds in consumption should be set such that a' stays on the grid. Thus, we put a penalty on the bounds of a' in the Golden algorithm such that

```
if max(a1) > kap(end)
    f1(a1>kap(end)) = -1000 + a1(a1>kap(end));
    a1(a1>kap(end)) = kap(end);
elseif min(a1) < kap(1)
    f1(a1<kap(1)) = -1000 + a1(a1<kap(1));
    a1(a1<kap(1)) = kap(1);
end
```

Additionally, we need to impose bounds on consumption ex-ante in the Aiyagari file so that a' stays on the grid:

```
% Golden search bounds
        cmin_gd = 1e-2*ones(1,N*nkap);
        cmax_gd = prodwage_gd + (1+r0)*rkap_gd + b; % BC with a'=-b and n=1 as maximum
```

Now, the code takes 14.705562 seconds, which is slightly a good improvement.

Exercise 7: Deterministic transition. Take the most efficient of your codes with exogenous labor supply. We now assume that there is aggregate productivity A_t such that the production function is

$$F(K_t, N_t) = A_t N_t^{\alpha} K_t^{1-\alpha}.$$

1. **In $t = 0$, we are in steady-state, where A_t is fixed equal to 1 and the measure is stationary. Suddenly and unexpectedly, in $t = 1$, productivity jumps up to $A_t = 1.01$. Then, productivity goes back to steady-state with some persistence $\rho_a = 0.9$:**

$$A_t = (1 - \rho_a) + \rho_a A_{t-1} \forall t \geq 2$$

1. **Compute the equilibrium path for the interest rate. Plot the aggregate paths for capital, wages, and interest rates.**

As a result of a productivity shock, interest rates and wages increase (Figure 3). The initial interest rate jump comes from the fact that positive technology shock increases the return to capital so that firm become willing to pay for capital. Following that, capital stock follows an increase as well. This increase in capital stock reduces its return over time; thus, the interest rate decreases gradually and falls below its steady-state levels for certain periods. Then, all variables go back to their steady-state levels.

Notably, our algorithm compute the backward iteration using the idea of endogenous grid method instead of using the value function. The idea is that we use the FOC to compute backwards the policy functions for consumption and capital in each period instead of optimizing the value function. This makes the code more efficient, as we don't need to compute optimization in each iteration.

Our code is presented below:

```
for t = T-[1:(T-1)]

    % Implied consumption and savings
    auxC = reshape(c_t(:,t+1),nkap,N)';
    c_imp = ( beta*(1+r_t(t)) * prob * auxC.^(-mu) ).^(1/-mu);  % Euler Equation
    a_imp = ( kap + c_imp - w_t(t)*s' )/(1+r_t(t));             % Budget Constraint

    % Policy function: savings (by interpolation)
    for j=1:N
        auxAprime(j,:) = linInterp(kap,a_imp(j,:),kap);
    end
    % Bounds
    auxAprime(auxAprime<kap(1))=kap(1);
    auxAprime(auxAprime>kap(end))=kap(end);

    % Policy functions
        % Savings
        aprime_t(:,t) = reshape(auxAprime',N*nkap,1);
        clear auxAprime
        % Consumption
        c_t(:,t) = w_t(t)*s_vec + (1+r_t(t))*kap_vec - aprime_t(:,t);
end
```

2. **Compute the welfare gains for each household, using consumption equivalents. Who are the households who benefit most?**

Intuitively, consumption equivalence means the necessary level of change in consumption so that an agent becomes indifferent between the initial baseline state and under the new policy/shock.

From Figure 4, we observe that as asset holdings increase, consumption equivalents decrease. The reason is that as wealth increases, the marginal utility will decrease and thus, get less and less benefit from a change/increase in additional units of consumption as asset holdings increase.

On the other hand, regarding the question of who benefits the most, it's the most productive workers who benefit the most in the form of higher wages, since the labor is exogenous and the same for each type of household. However, notice that the consumption equivalence is high for the low-productivity agents who borrows. This is because they are initially so poor that any increase in consumption level will increase their welfare substantially.

3. **Can you compute the gains in terms of wealth?**

The idea of gains in terms of wealth is that we want to compute the additional capital that agents need so that they can enjoy the same level of welfare, with and without the shock.

Formally, the welfare gain $\Delta(a, \epsilon)$ is defined such that

$$v_0(a, \epsilon) = v^*(a + \Delta(a, \epsilon), \epsilon), \tag{3}$$

where v_0 is the value function after the technology shock, and v^* is the function in the steady-state.

However, when $a + \Delta$ is out of the capital grid we define, we don't like the idea of extrapolating the value function and we would like to restrict everything inside the capital grids we give. Typically, this happens when a equals the highest level of capital in our grid. In this case, we redefine $\Delta(a, \epsilon)$:

$$v_0(a - \Delta(a, \epsilon)) = v^*(a, \epsilon) \tag{4}$$

The absolute level of wealth gains is presented in figure 5. The average level of the gains in wealth is 0.14. We also compute the average wealth gains in percentage, which is 20.11%. But this number is less interpretable, as there are agents with negative wealth. This large number is driven by the fact that a lot of high-skilled workers who own negative wealth begin to accumulate wealth thanks to the positive productivity shock.

Exercise 8. Take the most efficient of your codes with endogenous labor supply of Exercise 6. We add a government, which finances some spending G with a HSV tax function on labor income only. We assume $\tau = 0.18$.

1. **Find λ and B such that in equilibrium, G/Y and average hours are around 10% and 0.3 , respectively. What is the implied value of r ? How large is aggregate welfare?**

To find λ, we create an outer loop for interest rate and an inner loop for average tax rate λ such that the government budget constraint holds by this formula,

$$\lambda^{implied} = \frac{\int w\epsilon n d\hat{\lambda}(a, \epsilon) - G}{\int (w\epsilon n)^{1-\tau} d\hat{\lambda}(a, \epsilon)} \tag{5}$$

For the results, refer to the Table 1.

2. **Now, fix G and preference parameters, and find the new λ when $\tau = 0.30$. What is the implied value of r ? How large is aggregate welfare? How large is capital?**

The comparative statistics are presented in table 1. We observe that the welfare increases after a progressivity increase. This result is driven due to our calibration. On the other hand, we observe a decrease in the average tax rate after a progressivity increase. This might be related to the fact that lower hours worked might result in lower income; thus lower tax paid by the households and a decrease in the average tax rate(see equation 5). Also, the aggregate capital decreases due to lower income, thus a lower level of savings. Labor drops due to less incentive to work. People save less. Due to the decrease in capital, interest rate rises correspondingly.

The policy function and steady-state distribution are presented in figure 6 and figure 7.

Table 1: Steady State Statistics

	$\tau = 0.18$	$\tau = 0.30$
λ	0.72	0.63
G/Y	0.1032	0.1082
Aver. hours worked	0.2924	0.2818
Agg. welfare	-74.87	-74.52
Agg. capital	1.53	1.4282
Interest Rate	0.0358	0.0398

Exercise 9: Deterministic transition. Take the code of exercise 8. We want to analyze the transition from question 8 a to 8 b.

- **This question is very hard, don't be surprised if the code is very slow, reduce the number of points and loosen convergence criteria if needed; do your best! Compute the transition from question 8a to 8b. Why is this exercise significantly harder than exercise 7? What could we do to speed up the codes?**

In this problem, we need to calculate the sequence of $\{r_t, \lambda_t\}_{t=0}^{T}$ such that the government budget constraint holds. The complication in this problem comes from updating 2 objects $\{r_t, \lambda_t\}_{t=0}^{T}$ at once. One way is creating a loop for $\{r_t\}_{t=0}^{T}$, an additional inner loop for $\{\lambda_t\}_{t=0}^{T}$ but this method doesn't generate the desired result.

In our implementation, this method cannot deliver convergence to the new steady state! This lack of robustness comes from the fact that when we create a loop for $\{r_t\}_{t=0}^{T}$ and $\{\lambda_t\}_{t=0}^{T}$, we didn't impose the terminal condition to be satisfied: that is, we can't force $r_T = r^{**}$ and $\lambda_T = \lambda^{**}$ during iterations.

We switch to updating these two paths together instead of separately. The code is presented below:

```
while testr > tolr & testlambda> tollambda
    % Implied variables
    [k_imp,prodN_imp, lambda_imp]  =
    deterministicseq9(r_t,lambda_t,T, v, v_post, dist, dist_post,kk_post, false);
    r_imp   = alpha * A .* max(0.001,k_imp).^(alpha-1) .* prodN_imp.^(1-alpha) - delta;
    % Update test and guess
    testlambda = max(abs(lambda_t-lambda_imp));
    lambda_t = 0.7* lambda_t + 0.3 * lambda_imp;
    testr   = max(abs(r_t-r_imp));
    r_t = 0.95 * r_t + 0.05 * r_imp;
end
```

The transition figure we calculated is presented in figure 8. Though the capital falls as expected, aggregate labor increases during transition, which is not at all correct, despite the convergence back to the steady state.

We tried to identify what's the issue. We tried two different backward iteration algorithms, in `deterministicseq9_1.m` and `deterministicseq9_2.m`. We get exactly the same results, which suggests that our backward iteration algorithm is correct. The two algorithms are presented in the appendix.

We argue that our forward iteration that computes the distribution is correct as well. We apply the same code in exercise 7 and it delivers the desirable result in figure 3.

Also, we tried to to first fix the guess for $\{r_t\}$, then guess $\{\lambda_t\}$. Still, we don't have the desirable result.

- **If you have done some progress with question 9a. How large is aggregate welfare at the moment of the tax reform, compared to the long-run steady state? Why?**

No. We don't think we have the correct result.

1 Figure

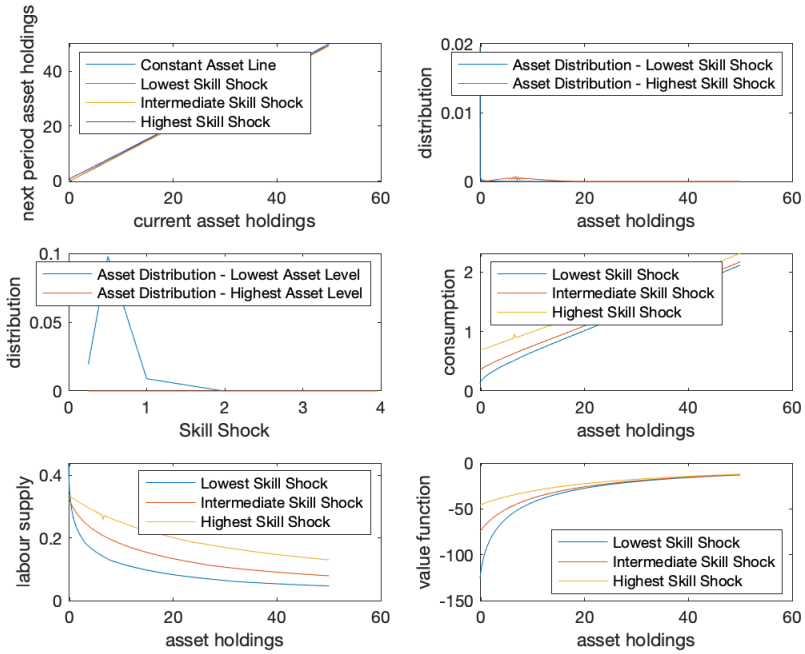

Figure 1: Question 6.1

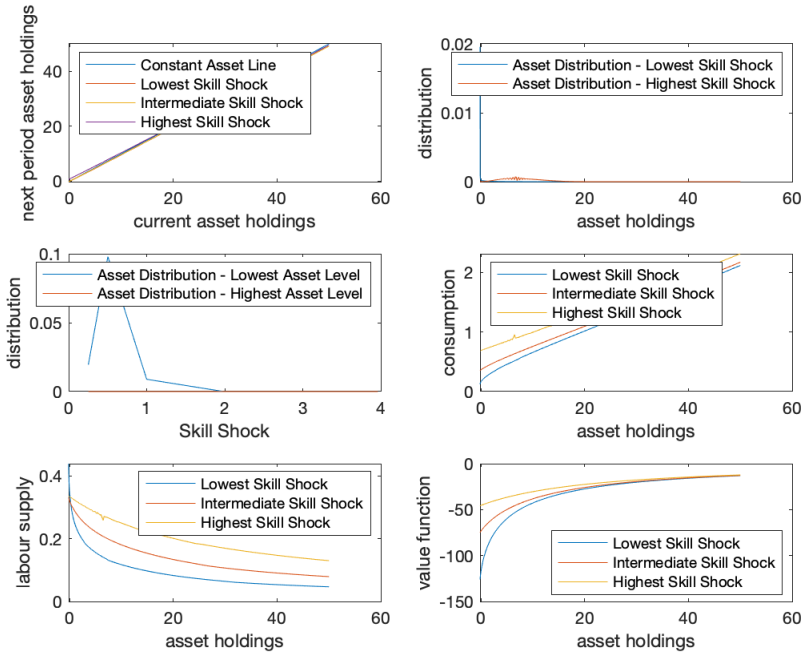

Figure 2: Question 6.2

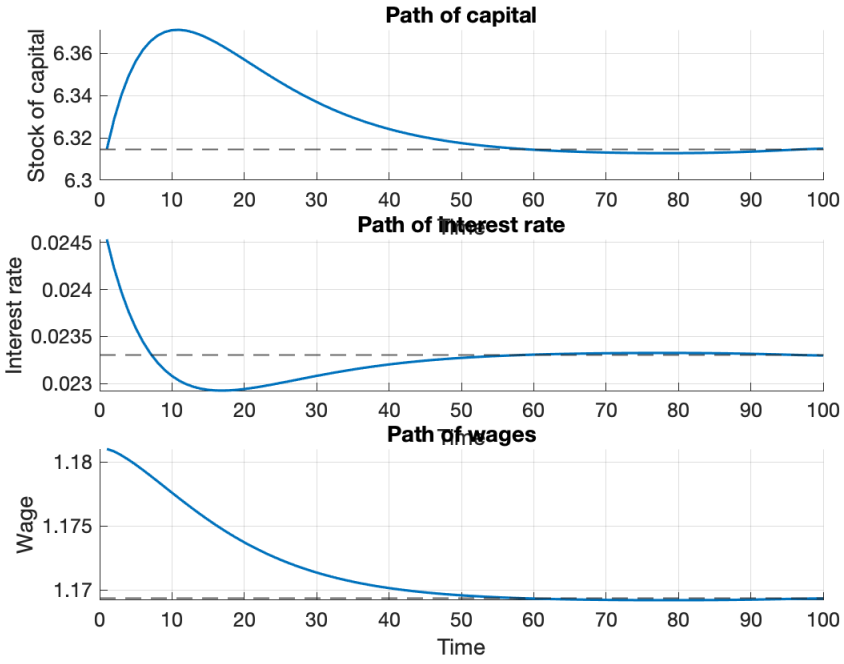

Figure 3: Question 7: Deterministic Path

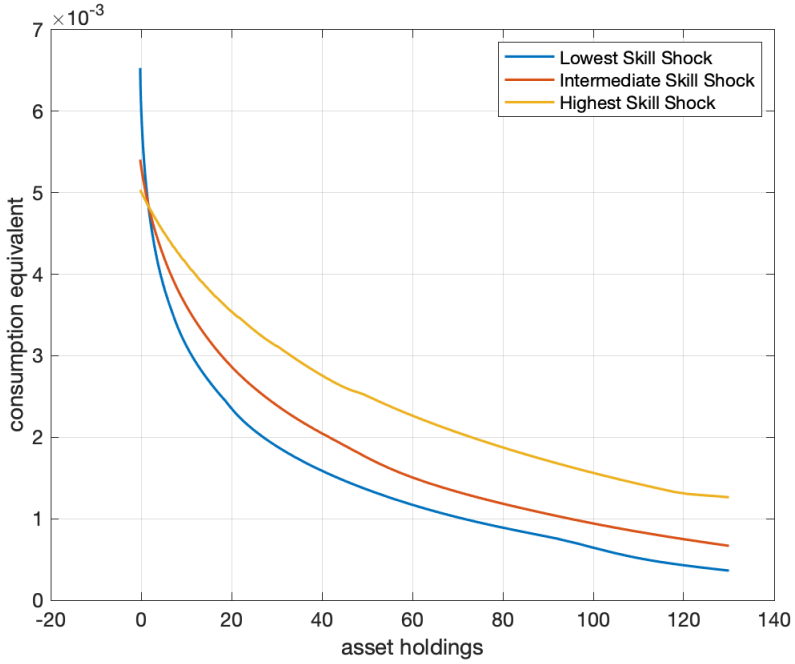

Figure 4: Question 7: Consumption Equivalence

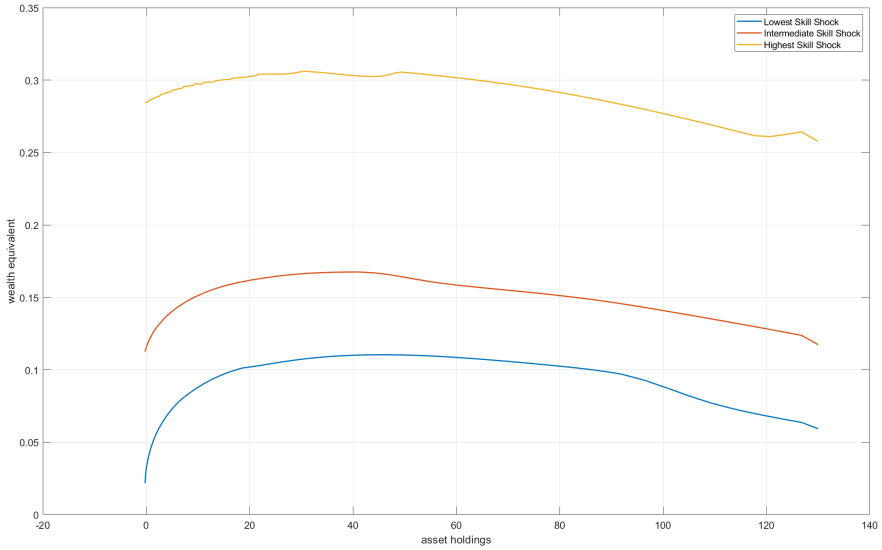

Figure 5: Question 7: Wealth Gain

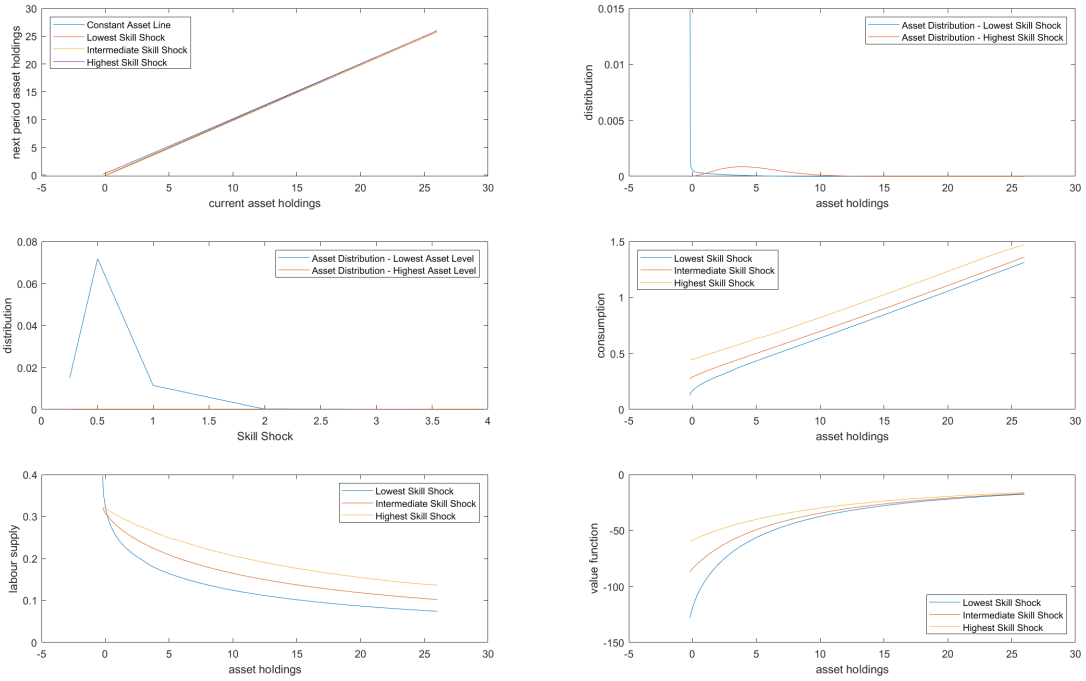

Figure 6: Question 8: $\tau = 0.15$

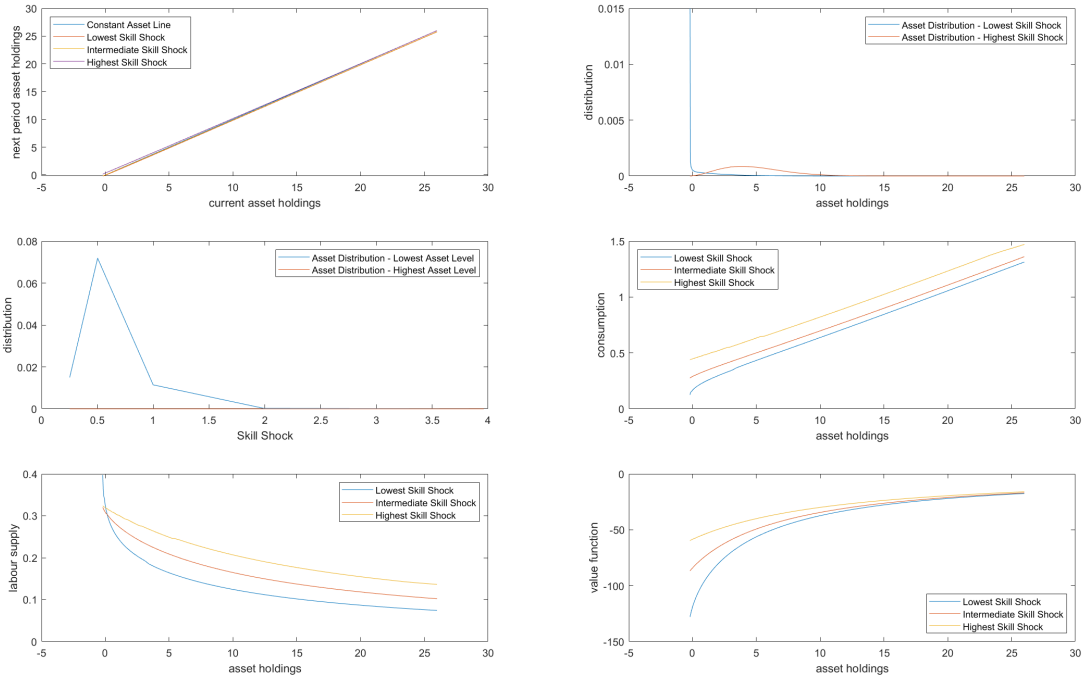

Figure 7: Question 8: $\tau = 0.3$

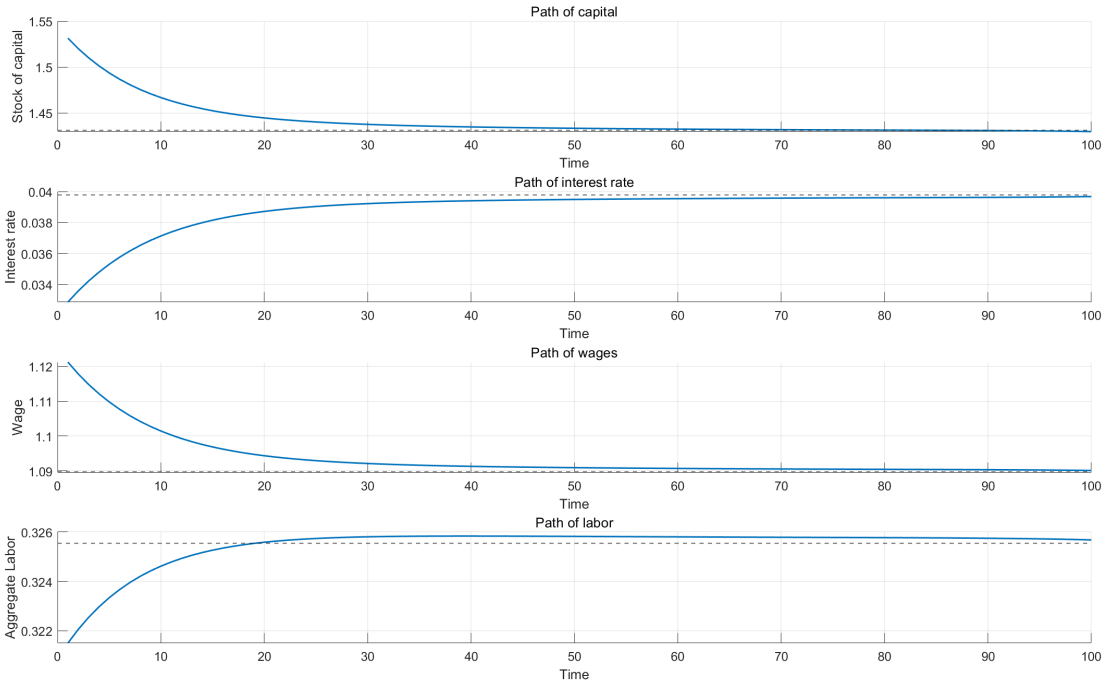

Figure 8: Question 9: Transition Path

2 Appendix

In the appendix, we attach two versions of backward iteration to get the policy function at each period during the transition. Firstly, we exploit the FOCs and use the endogenous grid method. Secondly, we maximize the value function over the current period's consumption with linear interpolation. Both methods deliver the same result. One can simply change line 238 in our `main9_1.m` by replacing the function `deterministicseq9` with `deterministicseq9_2` to switch from the first method to the second method.

Backward Iteration: Endogenous Grid Method

```
% Computing backwards until initial period
for t = T-[1:(T-1)]

    % Implied consumption and savings
    auxC = reshape(c_t(:,t+1),nkap,N)';
    c_imp = ( beta*(1+r_t(t)) * prob * (auxC.^(-mu)) ).^(-1/mu);  % Euler Equation doesn't change
    c_imp = reshape(c_imp',N*nkap,1);
    l_imp = (1/B.*lambda_t(t).*(1-tau2) .*(w_t(t).*s_vec).^(1-tau2) .* c_imp.^(-mu) ).^(1/(phi+tau2));
    a_imp = ( kap_vec + c_imp - lambda_t(t).* (w_t(t).*s_vec .*l_imp).^(1-tau2))./(1+r_t(t));


    % Policy function: savings (by interpolation)
    a_imp = reshape(a_imp,nkap,N)';
    c_imp = reshape(c_imp,nkap,N)';
    for j=1:N
        auxAprime(j,:) = interp1(a_imp(j,:),kap,kap,'linear','extrap');
    end
    % Bounds
    auxAprime(auxAprime<kap(1))=kap(1);
    auxAprime(auxAprime>kap(end))=kap(end);


    % Policy functions
        % Savings
        aprime_t(:,t) = reshape(auxAprime',N*nkap,1);
        clear auxAprime
        % Consumption & labor
nlow = ( max(  (aprime_t(:,t)-(1+r_t(t)).*kap_vec)./lambda_t(t),0  )).^(1/(1-tau2))./(w_t(t).*s_vec) ;
        nlow = max(nlow,0);
        nup = ones(N*nkap,1);
        nlow(nlow>nup) = 1-1e-5;
        % Loop criterium
        maxtestn = 1;
        n0 = 0.3*ones(N*nkap,1);
        prodwage = w_t(t).* s_vec;
    % Loop: we combine newton method and bisection method.
    % bisection method is robust in the sense that it will not create a negative consumption
    % monotonicity of the focn function ensures that bisection works.
    % Newton is faster but not robust.
    while (maxtestn>0.0001)
        foc = focn(n0,kap_vec,aprime_t(:,t), lambda_t(t), prodwage, r_t(t));
        first_dev = first_dev_focn(n0,kap_vec,aprime_t(:,t), lambda_t(t), prodwage, r_t(t));
        n1 = n0 - foc./first_dev;
        % if n1 out of the bound,
        n1(n1<nlow | n1>nup ) = 0.5*nlow(n1<nlow | n1>nup) + 0.5*nup(n1<nlow|n1>nup);
        foc1 = focn(n1,kap_vec,aprime_t(:,t), lambda_t(t), prodwage, r_t(t));
        nlow(foc1<0 & n1<=nup & n1>=nlow) = n1(foc1<0 & n1<=nup & n1>=nlow);
        nup(foc1>=0 & n1<=nup & n1>=nlow) = n1(foc1>=0 & n1<=nup & n1>=nlow);
```

```
        maxtestn = max(abs(foc1));
        n0 = n1;
    end
l_t(:,t) = n0;
c_t(:,t) = (1+r_t(t))*kap_vec + lambda_t(t).*( w_t(t).*s_vec.*l_t(:,t)).^(1-tau2) -
aprime_t(:,t); %c_t needs to be adjusted

end
```

Backward Iteration: Value Function Maximization

```
% Computing backwards until initial period
for t = T-[1:(T-1)]
    % Probability of transition to s'

    % Additional variables
     prodwage_gd = w_t(t).*s(st_gd(:,1));
     rkap_gd = (1+r_t(t))*kap(st_gd(:,2));

     % Optimization
        % Golden search bounds
            cmin_gd = 1e-2*ones(1,N*nkap);
            cmax_gd = lambda_t(t)*prodwage_gd.^(1-tau2) + rkap_gd - kap(1); % BC with a'=-b and n=1
test = 10;

    v_T = reshape(v_t(:,t+1)',nkap,N);
    v_T = v_T';
    [c_t(:,t), l_t(:,t), aprime_t(:,t), tv, indU(:,t)] =
    golden9(prodwage_gd,rkap_gd,cmin_gd,cmax_gd,prob_gd*v_T,lambda_t(t), tau2);

    v_t(:,t) = tv;

end
```