Policy Function Iteration and Acceleration with MacQueen-Porteus Bounds

Seungyoon Jeong

March 18, 2021

Model

A simple Determinastic Growth Problem with following Bellman Equation

$$V(k) = \max_{k'} \log(Ak^{\alpha} + (1-\delta)k - k') + \beta V(k'), \text{ given } k_0$$

1) Standard Value Function Iteration

First, we need to construct a grid over capital around steady-state. Steady-state of capital can be obtain by solving $\alpha A k^{\alpha-1} + (1-\delta) = \beta^{-1}$. An easy way to construct grid is take a linspace between $0.5k_{ss}$ and $1.5k_{ss}$. I chose 1,001 grid points so that there are 1,000 spaces between the points. Then, with an choice of initial value $V_0(k)$, we can iterate the value functions through the Bellman operator. I chose $V_0(k)$, for simplicity. With more details, computational procedures are as follows: $\underline{\text{(1) With contructed grids, evaluate RHS of the Bellman equation over grid for capital today and capital tomorrow.}}$ $\underline{\text{(2) Solve the max operator by searching over the grids and records the max value and index for location on grids.}}$ $\underline{\text{(3) Obtain capital policy function and update } V_1(k)}$. $\underline{\text{(4) Iterate until the } V_{new}(k) \text{ and } V_{old}(k) \text{ are close enough}}$.

A Sanity Check

It is always good idea to have a way to check your code by solving a simple version of complicated problem. In this problem, if we set $\delta = 1$, our model collapse to a simple case, where we can solve the model with pencil. It is well-known that the optimal policy function from the simple Brock-Mirman model is

$$k_{t+1} = g_k(k_t) = \alpha\beta A k_t^{\alpha}$$

So, we can check our code.

Back to our complicated problem, I solved with VFI described above. Here is my parameterization

α	β	A	δ
0.33333	0.975	1.05	0.1

Table 1: Parameterization

And obtained policy function and value function is as follows:

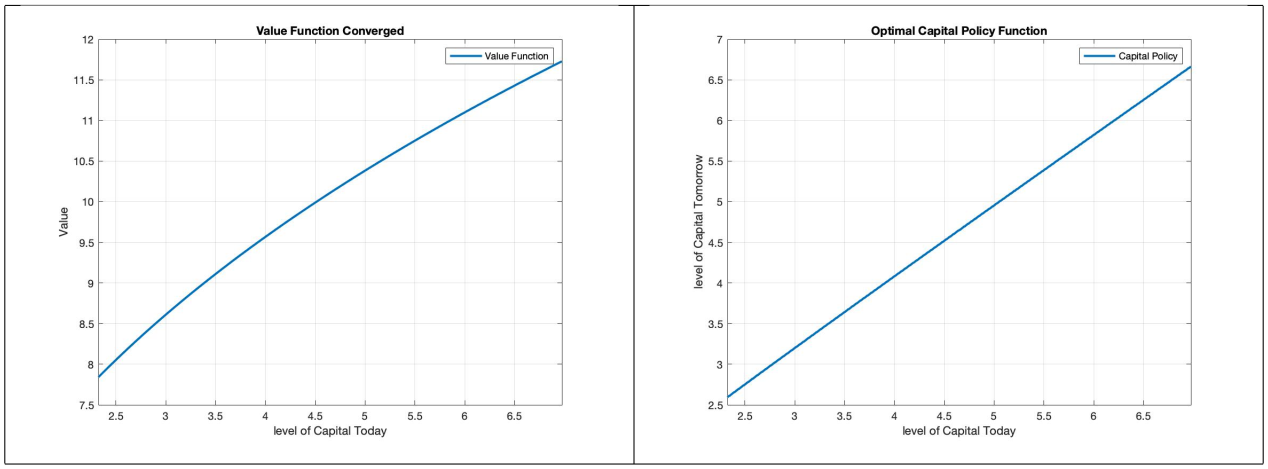

Figure 1: Converged Value function and Capital Policy function

2) Howard's Modified Policy Function Iteraton (MPI)

By its nature, VFI is reliable but it converges quite slowly (See the comparison result below) One of tricks speeding up the slow convergence is to evaluate RHS of Bellman equation bunch of times and then update policy function occasionally. Detailed procedures are as follows: (1) First, guess a initial policy function. I chose it same as the capital grid. (2) Over the capital grid, using the policy function($g_k^t(k)$) and evaluate RHS of Bellman equation $h = 1, 2, ..., m$ times. i.e.

$$V_t^{h+1}(k_i) = \log(Ak_i^\alpha + (1-\delta)k_i - g_k^t(k_i)) + \beta V_t^h(g_k^t(k_i))$$

(3) Then, update capital policy function, which is

$$g_k^{t+1}(k_i) = arg\max_j \log(Ak_i^\alpha + (1-\delta)k_i - k_j) + \beta V_t(k_j)$$

(4) Iterate these procedures until value functions are close enough.

I did $m = 5, 20, 100, 500$ and compared it to the results from VFI.

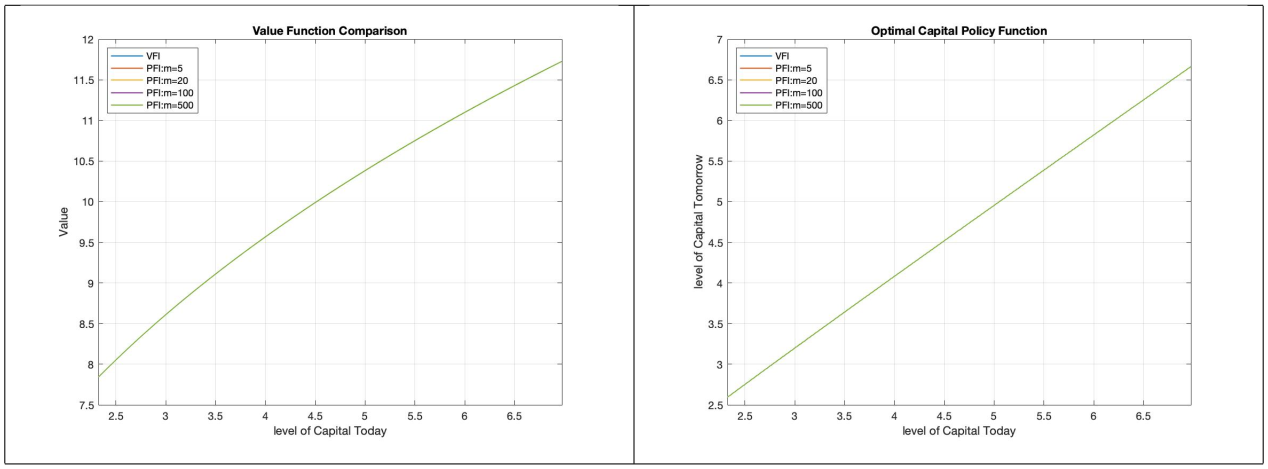

Table 2: Comparision of Value function and Capital Policy Function

It looks quite indistinguishable. Please refer the attached *result.xlsx* file. There are difference at 1E-9 among the VFI, PFI:m=5, PFI:m=20. PFI:m=100 and PFI:m=500 are the same at the level of Matlab's machine epsilon.

Next, here I compared those in terms of number of iteration for convergence to be achieved and computing time. I used *Matlab*. Note: nubmer of the capital grid is 1,001 same as before.

Program	**# of Iteration**	**Computing time**
VFI	910 iterations	115.53 sec
PFI:m=5	166 iterations	20.97 sec
PFI:m=20	49 iterations	6.43 sec
PFI:m=100	20 iterations	3.63 sec
PFI:m=500	20 iterations	4.39 sec

Table 3: Computing Performance Comparison

As seen in above, there are improved performances from PFI. It is way better than VFI, and efficient. However, one interesting thing is that there is no improvement when I increased $m = 100$ to 500. Actually, it slowed down the computing time. My intution is, once m gets high enough, there is no benefit to obtain by iterating more. So, after that level, it just fruitlessly iterates.

3) Acceleration with MacQueen-Porteus Bounds

An additional method to speed up the PFI is to choose another stoppting rule. MacQueen (1966) and Porteus (1971) provided a bound for discounted sequential decision problem. Roughly that is, If $a \leq V_n - V_{n-1} \leq b$, then for $\bar{x} \in X$, $a\beta/(1-\beta) \leq V^*(\bar{x}) - T^n V_0(\bar{x}) \leq b\beta/(1-\beta)$. If we apply this bounds, we can sharpen our stopping rule, and thus accelerates our computing speed.

Here is the result I got. I intetionally have the $m = 5$ so that I can see the speed improvement clearly. I applied the MQP in two ways; (1) evaluate it in every iterations, and (2) every 5 iterations. Below is what I got.

Program	**# of Iteration**	**Computing time**
VFI	910 iterations	115.53 sec
PFI:m=5	166 iterations	20.97 sec
PFI:m=5 with MQP (every)	87 iterations	16.04 sec
PFI:m=5 with MQP (5)	90 iterations	16.44 sec

Table 4: Computing Performance Comparison

As seen from the table, there is a big improvement by adapting MacQueen-Porteus bounds. The number of iteration is halved, and it is 5 seconds faster. However I couldn't observe hugh improvement whether I evaluate the new stopping rule in every iterations or every 5 iterations.