Quantitative Macroeconomics Homework 4

Sebastian A. Roy*

October 2019

*In collaboration with Jakub Bławat, Zuzanna Brzóska-Klimek, and Szymon Wieczorek

Exercise 1

We are given a representative household problem of a following form:

$$\max E\left\{\sum_{t=0}^{\infty}\beta^t u(c_t, h_t)\right\} \tag{1}$$

$$u(c_t, h_t) = \ln c_t - \kappa\frac{h_t^{1+\frac{1}{\nu}}}{1+\frac{1}{\nu}} \tag{2}$$

$$c_t + i_t = y_t \tag{3}$$

$$y_t = k_t^{1-\theta}h_t^{\theta} \tag{4}$$

$$i_t = k_{t+1} - (1-\delta)k_t \tag{5}$$

1

Recursive formulation of the problem given above means putting down a Bellman equation that allows to optimise using dynamic programming. One should note that, since labour supply is fixed ($h_t = h$), the **only** state variable in this problem is capital level k_t. Therefore, Bellman equation looks as following:

$$V_t(k_t) = \left(\max\left(\ln\left(k_t^{1-\theta}h_t^{\theta} - k_{t+1} + (1-\delta)k_t\right) - \kappa\frac{h_t^{1+\frac{1}{\nu}}}{1+\frac{1}{\nu}}\right) + V_{t+1}(k_{t+1})\right) \tag{6}$$

As a means of state space discretisation, capital grid $k\_grid = (5.0, 5.5, 6.0, \ldots, 55.0)$ is used. The grid consists of 101 elements.

Initial guess for value function is a vector of zeros.

a) Brute force VFI

Method	no. of iterations	time elapsed [s]
Brute force VFI	564	0.53

Table 1: Brute force VFI

Table 1 presents details of the baseline VFI method. Figure 1 depicts obtained, optimal values of the value function.

Obviously, standard, baseline VFI is a robust method of solving recursive, Bellman-type problems. It is also simple to implement in the majority of computational software. Using built-in numpy vector piece-wise computation method makes it also quick.

b) VFI with monotonicity of the decision rule

Method	no. of iterations	time elapsed [s]
VFI & decision rule monotonicity	563	3.13

Table 2: VFI with monotonicity of the decision rule

Table 2 shows details of the improved VFI algorithm. As one can see, despite almost the same number of iterations, piece-wise computations in loops make this method slower, as identifying exact positions of χ matrix elements at which we stop further calculations prevents us from using quick numpy vector methods.

c) VFI with value function concavity

Method	no. of iterations	time elapsed [s]
VFI & value function concavity	322	12.28

Table 3: VFI with value function concavity

Including value function concavity in the VFI algorithm lowers the overall number of iterations by almost one half (see Table 3). However, due to higher number of operations in the loops, once again we can see that this improvement eventually slows the algorithm down.

Moreover, as one can read in Figure 3, this improvement makes the value function more hoarse (discontinuities are way higher).

d) VFI with monotonicity of the decision rule and with value function concavity

Method	no. of iterations	time elapsed [s]
VFI & monotonicity & concavity	379	0.47

Table 4: VFI with decision rule monotonicity and value function concavity

As one can see in Table 4, combining two improvements from the previous point actually works miracles. Number of iterations needed is relatively low and elapsed time is lower from half a second. Moreover, as depicted in Figure 5, this time the value function obtained is much more smooth than in the previous case.

e) VFI with local search

Local search method proves inefficient in our case. Summary of the VFI with local search (under assumption that optimal tomorrow's capital lies not further

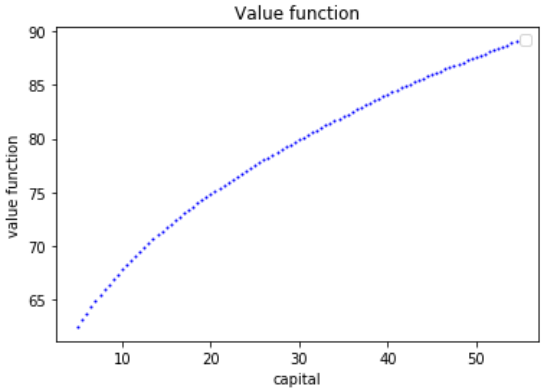


Figure 1: Representative agent model value function

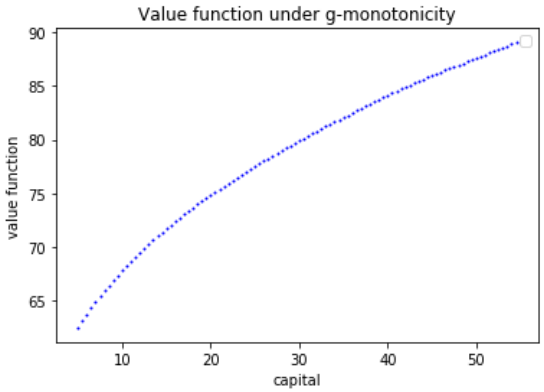


Figure 2: Representative agent model value function with optimal policy rule monotonicity

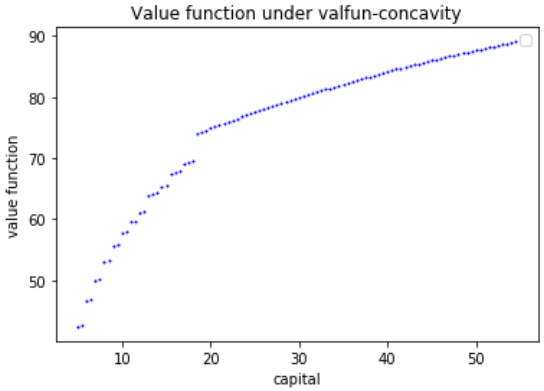


Figure 3: Representative agent model value function with value function concavity

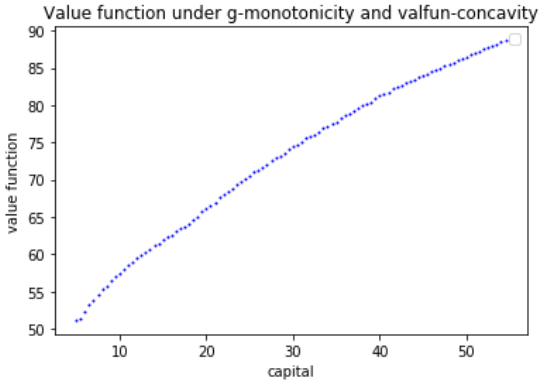


Figure 4: Representative agent model value function with decision rule monotonicity and value function concavity

than 5 elements from today's capital) are presented in Table 5. They seem correct unless we analyse the value function obtained under local search (see Figure 5).

Method	no. of iterations	time elapsed [s]
VFI & local	564	0.88

Table 5: VFI with local search

This value function differs from the previous results, which might imply that our locality bound is too tight. However, one finds out that the value function converges to its correct form only if the locality bound is not less than 51 (which, with a capital grid of length 100, means that our local search is transformed into brute force VFI).

This can easily be explained by looking at the details of the brute force VFI. E.g. in the second iteration, we can find out that the optimal tomorrow's capital for the highest capital level possible today lies in the middle of the grid (formally, $g(k_2 = k\_grid[99]) = k\_grid[50]$). Furthermore, in the first iteration policy function is a vector of zeros. Therefore local search cannot work properly.

f) Howard method

Method	policy function iteration start	no. of iterations	time elapsed [s]
VFI & Howard	$\emptyset$	564	0.36
VFI & Howard	50	564	0.47
VFI & Howard	100	564	0.46
VFI & Howard	200	564	0.44

Table 6: VFI with Howard method and different starting points for policy iteration

VFI results with Howard improvement and initial policy guess being a vector of zeros are presented in Table 6. As one can see, introducing policy iterations does not affect the total number of iterations, while slowing down the procedure.

Final results are the same under any specification, as depicted in Figure 6.

g) Howard method with regular policy improvements

As one could have expected from the f) result, regular updates of the policy function do not affect total number of iterations, while slowing down the computations. Value functions are exactly the same as in Figure 6 and are not shown here.

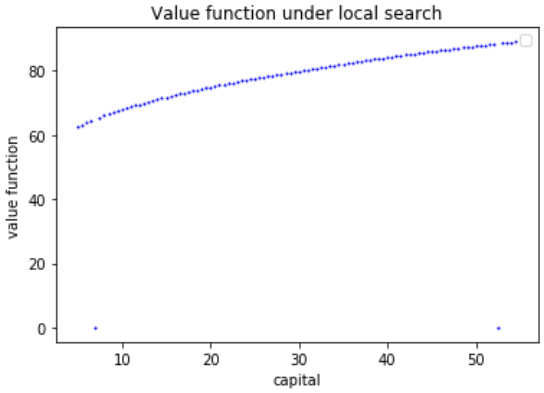
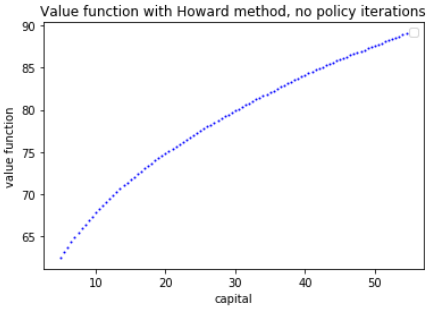
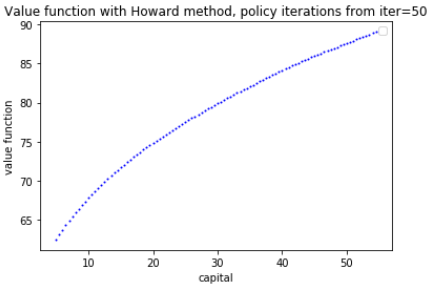


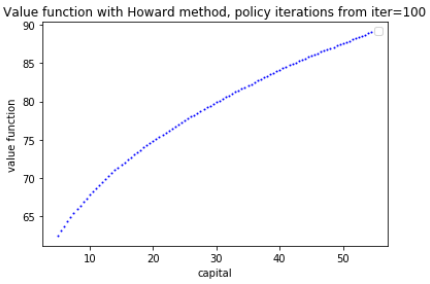
Figure 5: Representative agent model value function with local search



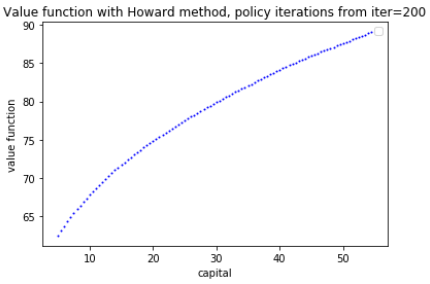
(a) No policy function iterations



(b) Policy function iterations starting at the 50 iteration



(c) Policy function iterations starting at the 100 iteration



(d) Policy function iterations starting at the 200 iteration

Figure 6: Value function under different specifications of the Howard method

Method	no. of iterations between policy updates	no. of iterations	time elapsed [s]
VFI & Howard	5	564	0.40
VFI & Howard	10	564	0.40
VFI & Howard	20	564	0.39
VFI & Howard	50	564	0.39

Table 7: VFI with Howard method and different frequency of policy updates