Quantitative Macro
Problem Set 3

Piotr Królak

26 October 2020

Question 1. Value function iteration

Let us start with rewriting the problem:

$$V(k_t) = \max_{c_t, i_t, k_{t+1}, h_t} \sum_{t=0}^{\infty} \beta^t u(c_t, h_t) \tag{1}$$

s.t

$$k_t^{1-\theta}(h_t)^\theta + k_t(1-\delta) - k_{t+1} = c_t \tag{2}$$

Then we can write the Belmann equation:

$$V(k_t) = \max_{c_t, i_t, k_{t+1}, h_t} u(c_t, h_t) + V(k_{t+1}) \tag{3}$$

Plugging in c_t from second equation, starting with $h_t = 1$:

$$V(k_t) = \max_{k_{t+1}} u(k_t^{1-\theta}(h_t)^\theta + k_t(1-\delta) - k_{t+1}, 1) + V(k_{t+1}) \tag{4}$$

such that:

$$k_{t+1} \in [0, k_t^{1-\theta}(h_t)^\theta + k_t(1-\delta)] \tag{5}$$

Based on this I will use value function iteration algorithm to find $V(k_t)$. In the first question labour is exogenously set to one, so I consider the utility function: $u(c_t) = lnc_t$ - with $\kappa = 0$.

a) Brute force method

Let us use the basic VFI algorithm without any time accelerating methods. Before doing this, let us compute steady state value for k, which could be useful for choosing appropriate grid. Using equation (3) from my previous problem set (it is an Euler equation for consumption) we obtain a steady state equation:

$$1 = \beta[(1-\theta)k_{t+1}^{-\theta} + 1 - \delta] \tag{6}$$

After some reshuffling:

$$k = \left[\frac{\frac{1}{\beta} + \delta - 1}{1-\theta}\right]^{-\frac{1}{\theta}} \tag{7}$$

Which is equal to 42.55.
So, I follow steps presented during the lecture. I have decided to make an evenly spaced grid with 300 points, beginning at 0.2 and ending at 60 and $\epsilon = 0.1$. Such parametrization enables comparing results between different methods and not waiting too long for the results. Following steps from lecture I start with a initial guess that is a vector of zeros, then I compute the M matrix. After that I check the positivity constraint for consumption. If it is violated I assign -999999 to the corresponding matrix element. Then, I compute matrix χ and get the value function in the next iteration. I repeat that until the distance (euclidean in my case) is bigger then the epsilon. I have decided to
After the 418 iterations and 117.584671 seconds the value function converges. The value function is increasing, concave function that may be seen on the graph below.

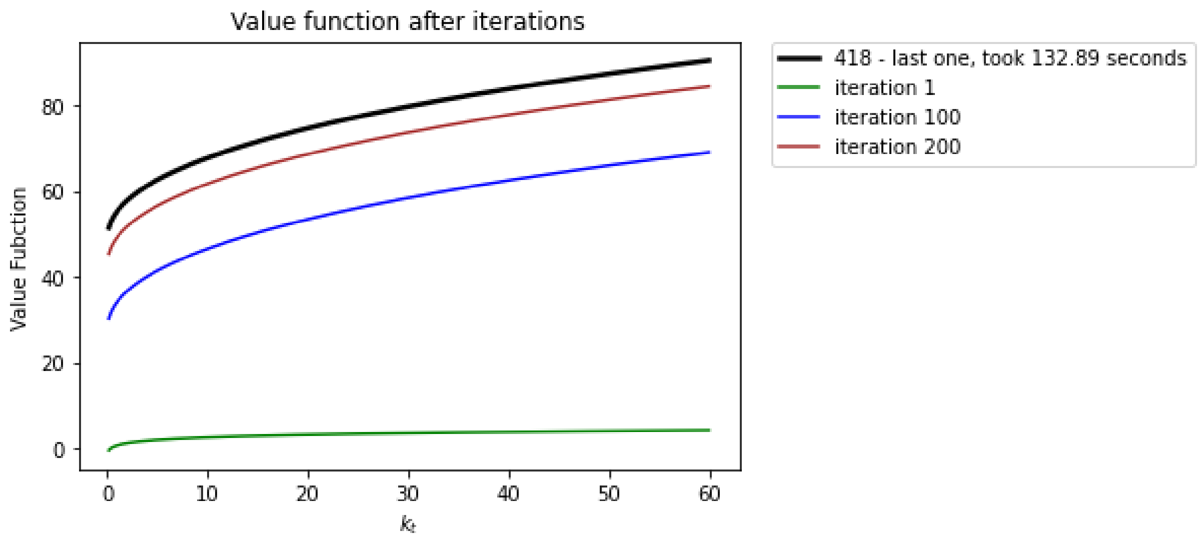


Figure 1: Value function after 1, 100, 200 iteration

One can notice that after each iteration the value function moves upwards, the more iterations we make this shift is smaller and smaller until the convergence.

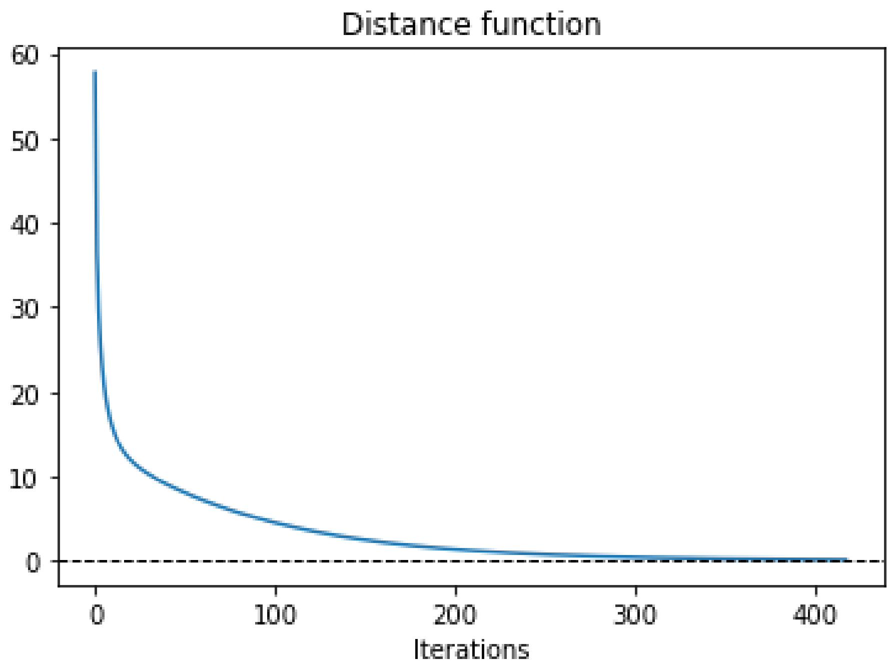


Figure 2: Distance function for brute force method

It could be well seen on the distance function plot. In the first iterations the value function changes a lot, and after the 200 iteration (still more than a half to go) the changes are negligible.

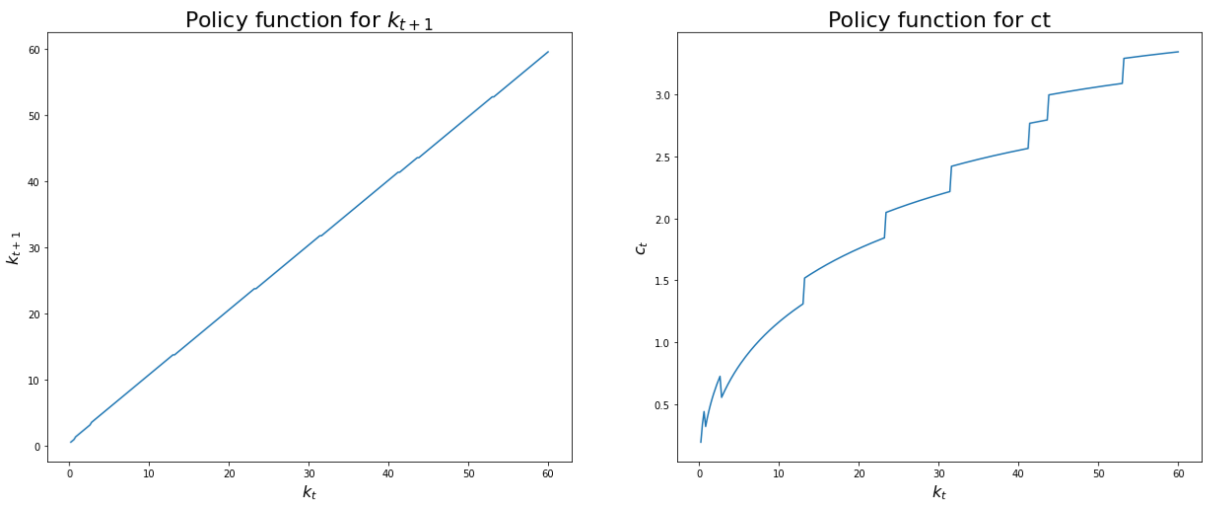
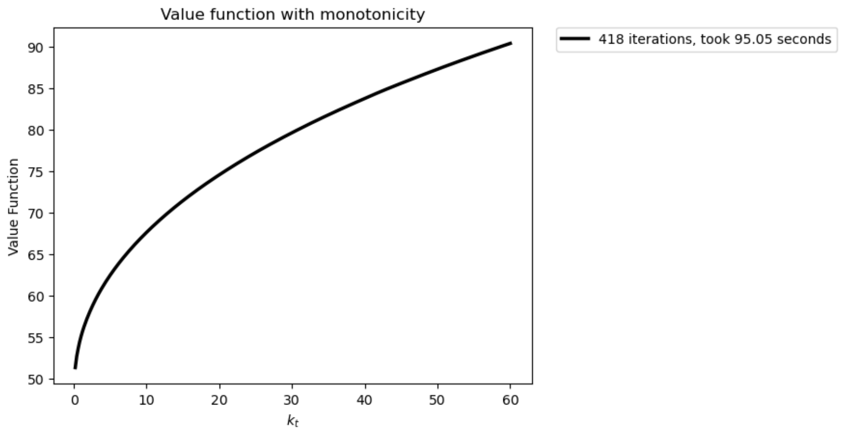


Figure 3: Policy function for capital (on the left) and consumption (on the right)

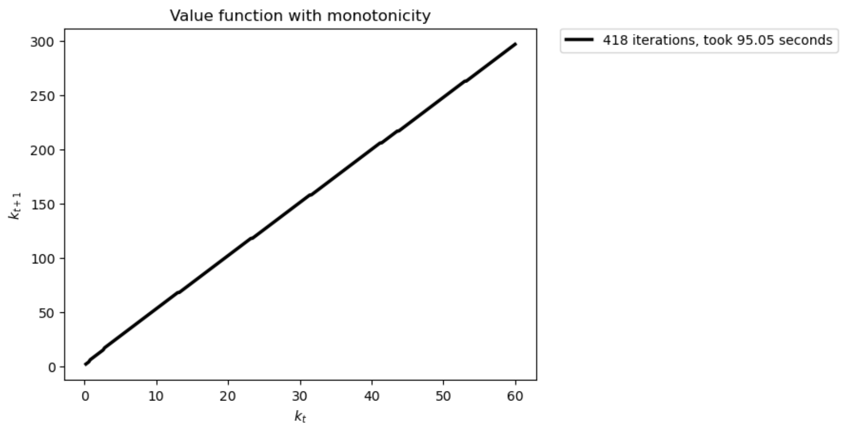
Let us look at the policy function for a moment. For the capital in the next period it resembles a 45-degree line, but it is slightly below it. For the consumption is a bit messy. It looks similar to a concave function, but has some ramps. Probably, if we choose a denser grid the policy function will be smoother, but that costs additional time, due to curse of dimensionality.

b) Speeding up the algorithm - monotonicity

Let us start with speeding up the algorithm using monotonicity of the policy function property. It uses the fact that for $k_j > k_i$ then $g(k_j) > g(k_i)$. So, while iterating we obtain for each i the decision rule and then for $i+1$ we compute elements of the M matrix only for k_j that are bigger than the $g(k_i)$. Obtained are following results:



(a) Value function



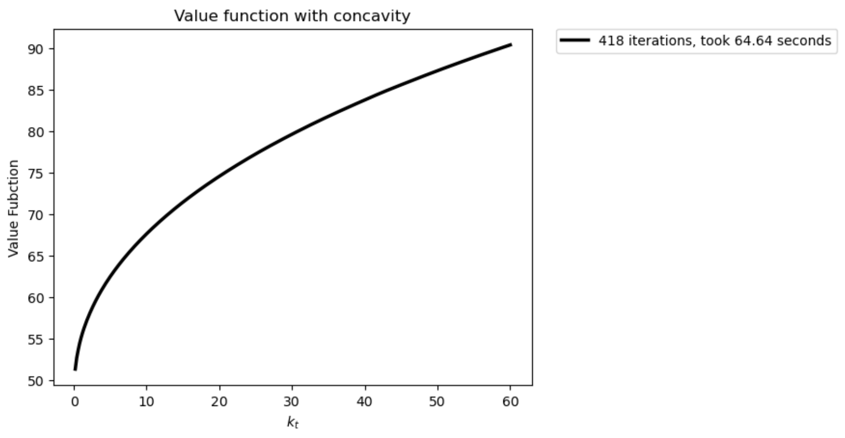
(b) Decision rule for next period capital

Figure 4: Value function iteration with monotonicity

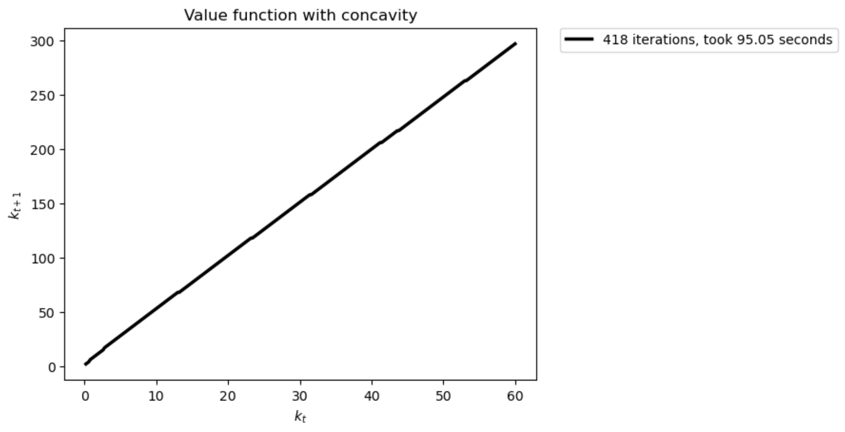
The change in code accelerates the algorithm. The 418 iterations take 95.05 seconds, which is about 20 percent less. The value function and policy function is same as in the baseline "brute force" method. It is reasonable, because we speed up the algorithm by not computing some parts of the matrix, were we state that there are no solutions to the maximization problems.

c) Speeding up the algorithm - concavity

Now let us use the property that the maximand of the i-th row in the matrix is concave in k_j. So, while we are iterating over $j's$ the computed elements start to decrease we break the loop and report the optimal values.



(a) Value function



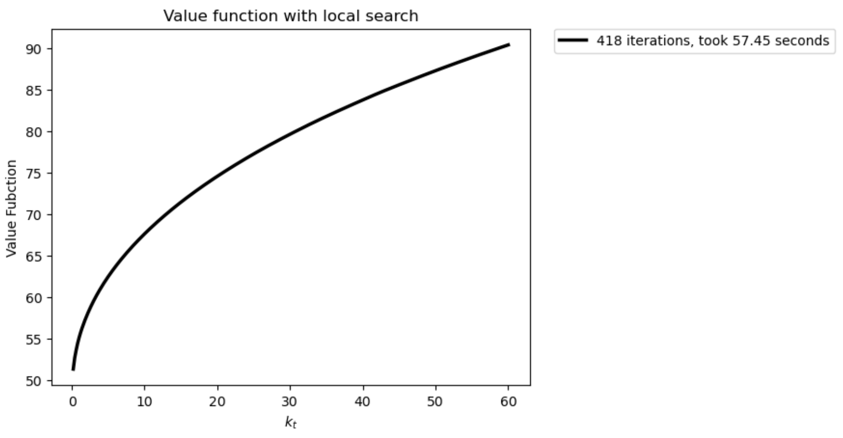
(b) Decision rule for next period capital

Figure 5: Value function iteration with concavity

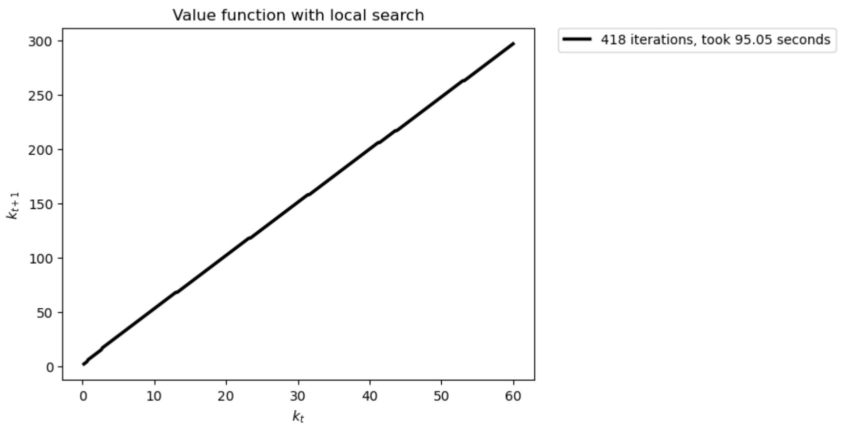
The results are more than satisfactory. The elapsed time is equal to 64.65 seconds, which is a decrease of almost 50 %. Again, the value function and policy function are same as in the baseline.

d) Speeding up the algorithm - local search

Now, let us use the continuity of the decision rule property. It states that the decision rule is continuous, hence if we have $g(k_i)$ then $g(k_{i+1})$ is in the small neighbourhood, let us assume 10 gridpoints to the left and right. Let us see the results.



(a) Value function



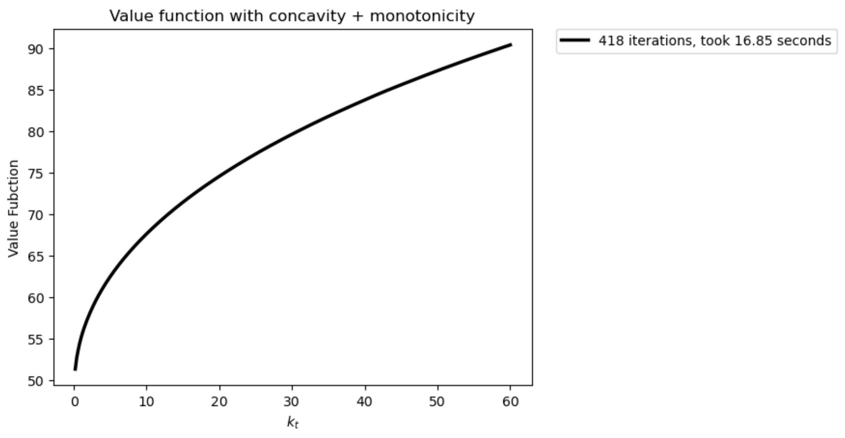
(b) Decision rule for next period capital

Figure 6: Value function iteration with local search

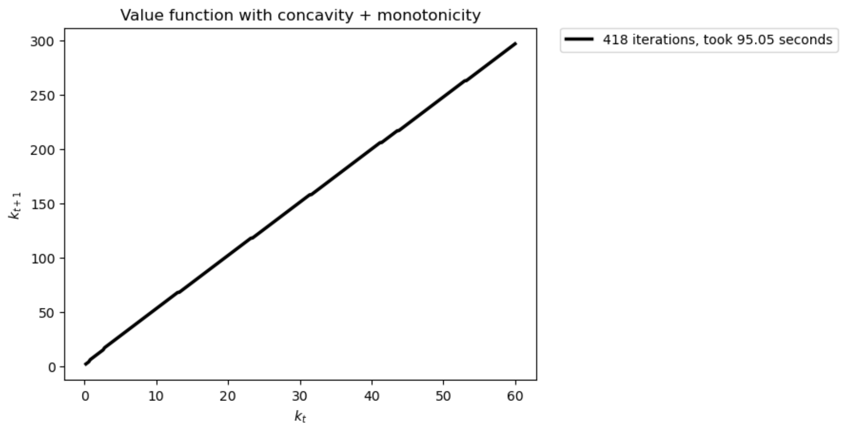
The time savings is even better, one can save more than a half. The elapsed time is equal to 57.45 seconds. Once again the value function and policies function are the same.

3) Speeding up the algorithm - concavity + monotonicty

Here let us combine previously mentioned methods that use monotonicty and concavity properties.



(a) Value function



(b) Decision rule for next period capital

Figure 7: Value function iteration with concavity + monotonicity

For this combination of methods the results are the best, the time elapsed id equal to 16.85 seconds. Again, the value function and policy function are the same as in the baseline model.

f, g) Howard's policy iteration

We are asked here to apply a guessed policy function into the VFI algorithm in order to save time. I decided to apply the guess in three cases: 1, 50, 100 iteration. My guess is: $k_j = k_i + 0.2$, one can notice that for the highest i we end up outside the grid, so we assign $K_{300} = 60$. I choose this policy function, because in the previous points we obtained an almost linear one that was shifted a bit upwards.
Let us present the results on the graph that shows the difference between value function from the baseline algorithm and that applying guessed decision rule at three different iterations.

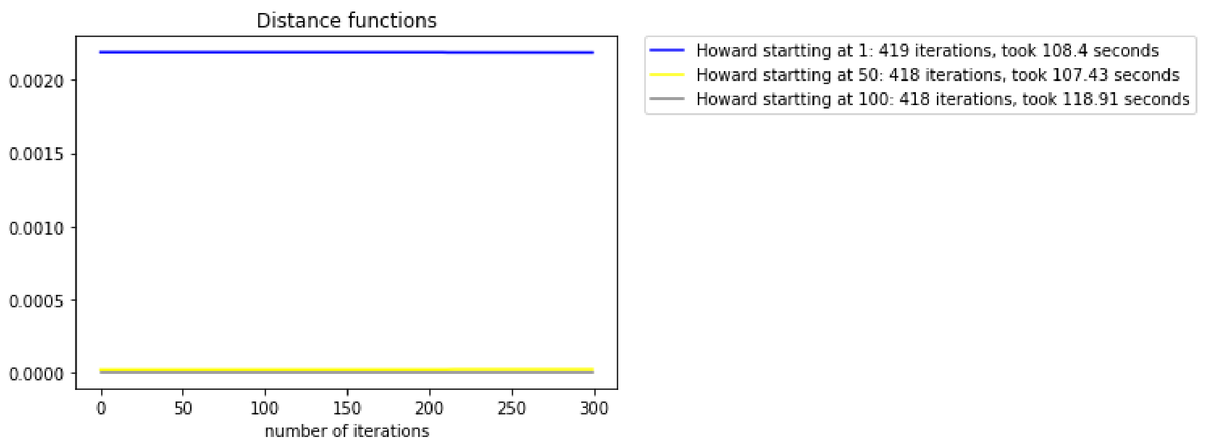


Figure 8: Differences between value functions

The value functions are in two cases the same and in one almost the same as in the baseline algorithm. When applying the guessed for the first iteration we may gain some seconds (108.4 sec) as well as for the 50th iteration (107.43). But, for the iteration number 100, time elapsed is a bit higher (118.90).
Now the reassessment of policy function will be considered. After some periods (n_h) policy function is updated using the guess, which is the solution to the baseline model. Now let us update the guessed policy function every n_h iterations.

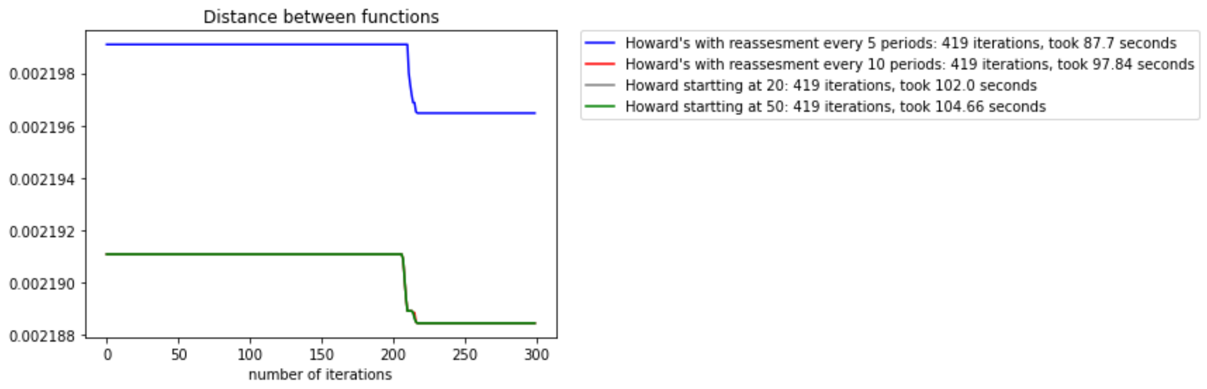


Figure 9: Differences between value functions

The differences between reassessing every 10, 20, 50 periods to the baseline algorithm are the same and represented by the green line. Whereas for reassessing every 5 periods the discrepancy is a bit higher, but still very small. For every method here the time was gained. The higher the frequency the bigger time gains (full results are in the table below).

Method	Iterations	Time elapsed [sec]
Brute Force	418	117.584671
Monotonicity	418	95.052025
Concavity	418	64.635093
Local search	418	57.453489
Mon. and Conc.	418	16.854801
Howard's at $s = 1$	419	108.398318
Howard's at $s = 50$	418	107.431446
Howard's at $s = 100$	418	118.905709
Howard's with reassessment every 5 steps	419	87.702306
Howard's with reassessment every 10 steps	419	97.382108
Howard's with reassessment every 20 steps	419	102.004901
Howard's with reassessment every 50 steps	419	104.660918

Table 1: Performance of the VFI

2. Continuous choice for an endogenous labour

In the second exercise we are asked to implement endogenous labour choice with a continuous method. For the clearer notation I change θ in the production function to α.

$$\max_{c_t, i_t, k_{t+1}, h_t} \sum_{t=0}^{\infty} \beta^t u(c_t, h_t) \tag{8}$$

s.t

$$k_t^{1-\alpha}(h_t)^{\alpha} - c_t + k_t(1-\delta) - k_{t+1} = 0 \tag{9}$$

Then we set up a Lagrangian :

$$\mathcal{L} = \sum_{t=0}^{\infty} \beta^t u(c_t, h_t) + \sum_{t=0}^{\infty} \lambda_t [k_t^{1-\theta}(h_t)^{\theta} - c_t + k_t(1-\delta) - k_{t+1}]$$

Now, let us calculate the FOCs, given that the

$$u(c_t) = ln(c_t) :$$

$$\frac{\partial \mathcal{L}}{\partial c_t} : \frac{\beta^t}{c_t} = \lambda_t \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial k_{t+1}} : \lambda_{t+1} = \lambda_t[(h_t)^\alpha (1-\alpha) k_{t+1}^{-\alpha} + 1 - \delta] \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial h_t} : \beta^t \kappa h_t^{\frac{1}{v}} = \lambda_t[(zh)^\alpha (1-\alpha) k_{t+1}^{-\alpha} + 1 - \delta] \tag{12}$$

Substituting for λ_t, we obtain the following equation:

$$\frac{c_t \kappa}{\alpha k_t^{1-\alpha}} = h_t^{\alpha - 1 - \frac{1}{v}} \tag{13}$$

The result above will be used to compute optimal labour given that optimal consumption was calculated before and optimal consumption is already calculated.
Here, I follow approach presented in Fernandez-Vilaverde, Ramirez, Schorfheide SOLUTION AND ESTIMATION METHODS FOR DSGE MODELS (2016) chapter 5.6.
I start with creating 51 evenly-spaced knots between $k = 0.2$ and $k = 60$. Because, in the lecture notes it is not specified what happens for the first and last spline, I assume that $\phi_0(k)$ is a line that connects peak of value 0 for k_0 and k_1. Similarly, the $\phi_m(k)$ starts at k_{m-1} and has a peak at k_m - just like in the previously mentioned book in chapter 5.4, figure 8. Then I use B^1 splines to as the basis functions, creating 51 of them. This way, number of knots is equal to number of unknown θ, hence it would not create any problem when looking for solutions of θ (Step 5 from lecture notes). I create a splines for both value function and decision rule for labour. My initial guesses are such that all thetas are equal to one (for the labour it means that $h_t = 1$).
In the step 3, I plug the decision rule for the capital for each knot:

$$gk^s(k_i) = \arg\max_{k'} u([\tilde{h_t}(k_t)]^\alpha k_t^{1-\alpha} + (1-\delta)k_i - k') + \beta \tilde{V}^s(g^s(k_i) \tag{14}$$

where, $h_t\tilde{(k_t)}$ is an approximated decision rule for labour and $gk^s(k_i)$ is a decision rule for capital in the next period. In addition k' must be within the grid and such that the consumption is positive. To solve this equation I use trust-constraint method of function minimze ftom the Python's scipy.optimize module
Once we obtain the decision rule for capital, it is plugged-in to the value function. Next the optimal consumption is computed:

$$gc^s(k_i) = [\tilde{h_t}(k_t)]^\alpha k_t^{1-\alpha} + (1-\delta)k_i - gk^s(k_i) \tag{15}$$

where, $gc^s(k_i)$ is a decision rule for consumption. Finally the decision rule for labour is updated using equations 13 and 15.

Next step is solving for coefficients of basis function given the updated values of a function (Step 5 from the lecture notes). To do this a function fsolve is used, that bases on the Powell's method. After that I compute the distance between thetas, after trials and errors I decided to assign $\epsilon = 0.01$ for labour decision rule and $\epsilon = 3$ for Value function, the convergence is very, very slow.

Let us see the results.

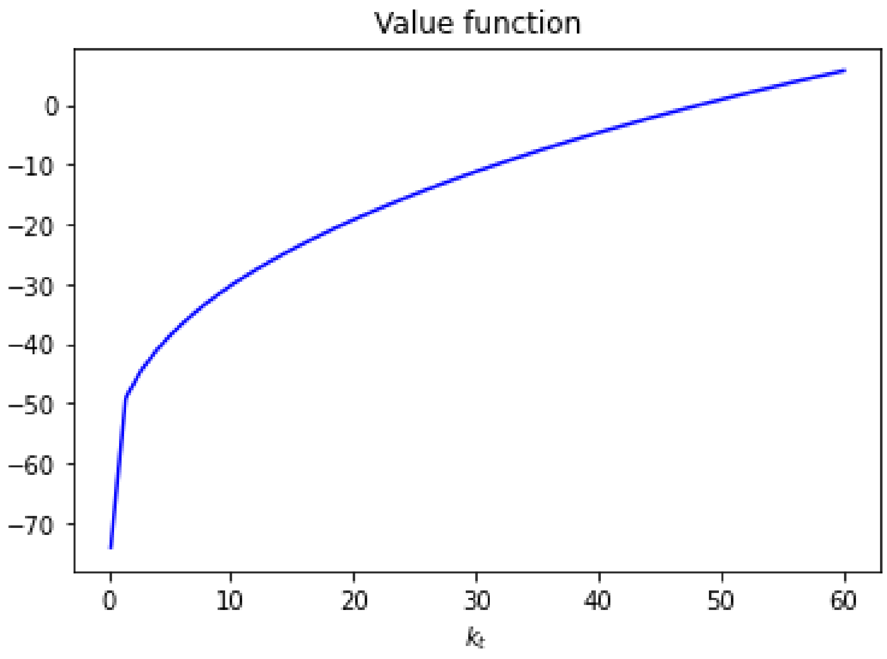


Figure 10: Value function with continuous labour choice

After the 393.162515 seconds and 60 iterations we obtain a value function that is increasing, has a concave shape with a very steep first part up to $k_t = 2$.

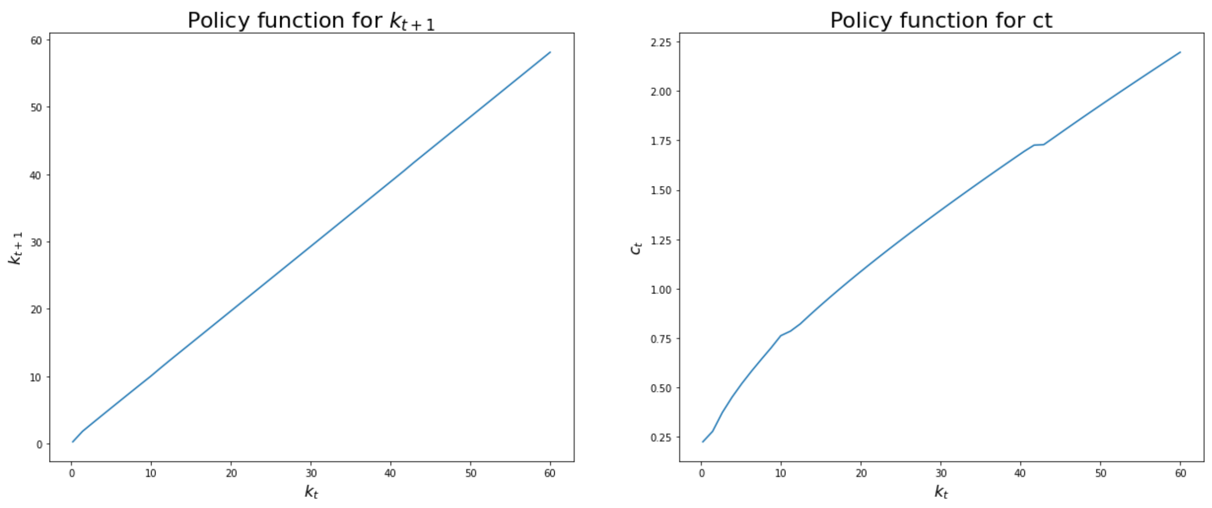


Figure 11: Decision rules for capital next period and consumption with continuous labour choice

Here there unsurprisingly the policy function for capital next period is very close to the 45-degree line, which is the same result as for the Value Function Iteration. Whereas, the policy function for consumption is more smooth compared to the baseline model (does not have any 'kinks'). Also the optimal levels of consumption are slightly below the baseline model for the higher k's.

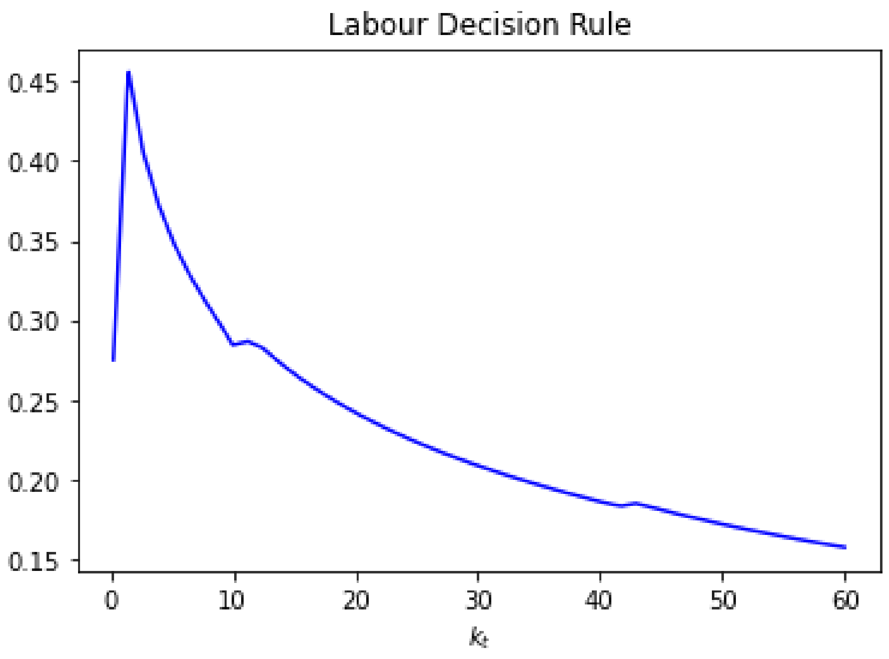


Figure 12: Decision rules for labour with continuous labour choice

Here, one can notice an interesting pattern. For the low values of capital (up to 1.46) agents want to increase the supplied labour even up to 0.45. It is reasonable, the output is small, because of the low capital level, hence they choose to work more to enhance it, which has the effect on the savings and consumption. However, when the capital level is relatively big, agents decide to work less then before.

3. Chebyshev Regression Algorithm

Here we are asked to implement. To do this we follow the algorithm for continuous methods from the lecture, but with 3 changes. Let us briefly go through the steps. I set here $\kappa = 0$, to compare the results to the baseline methods.

We start with choosing nodes, here comes the first modification, we use Chebyshev's nodes. Altogether 60 nodes were chosen on the interval $[0.2, 60]$. Then I choose a basis functions to be Chebyshev's polynomials (second modification) up to degree 10. The initial guess for the vector of coefficients θ to be the vector of ones. Given that, for each node a decision rule for capital is found using the trust-constraint method of Python's minimize function from scipy.optimize module. When they are solved an optimal consumption level is calculated (using equation 9) and the value function is updated. After the process is repeated for all nodes, every value of value function is known, so using Chebyshev's regression we solve for thetas. Then, recently solved coefficients are compared with the previous ones and if the distance is above 0.09 the iteration is repeated. After 203 iterations and 1607.32 seconds we obtain the following results:

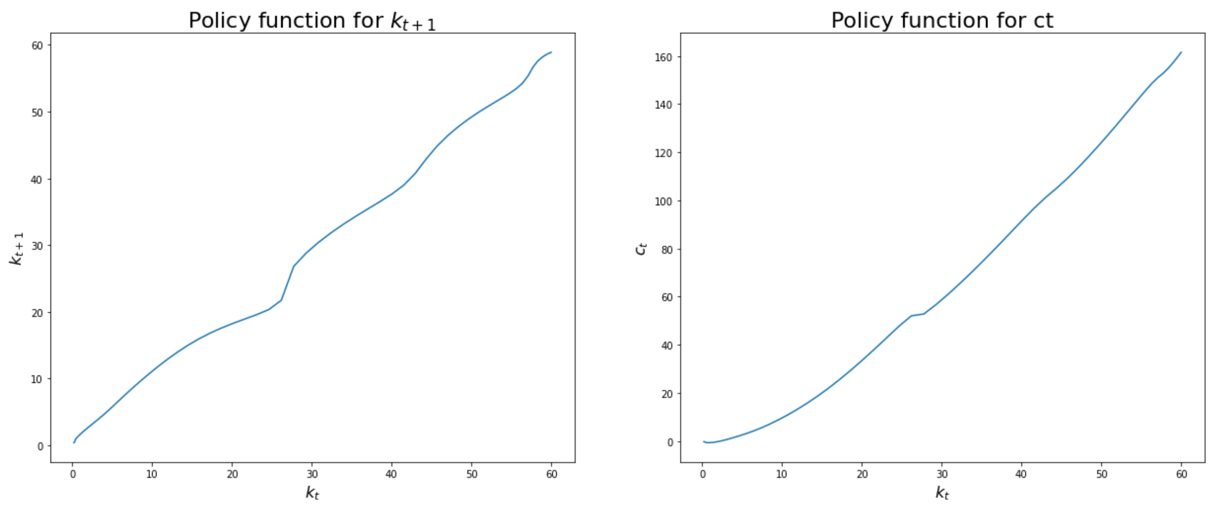


Figure 13: Decision rules for capital next period and consumption

The policy function for the capital in the next period looks similar to those from the previous methods - it is linear and very close to the 45-degree line. However, one may notice some 'perturbations' $K_t = 27, 42, 55$. When it comes to the decision rule for capital, here are obtained different results than previously. The function is linear and starts with some features of convex functions.

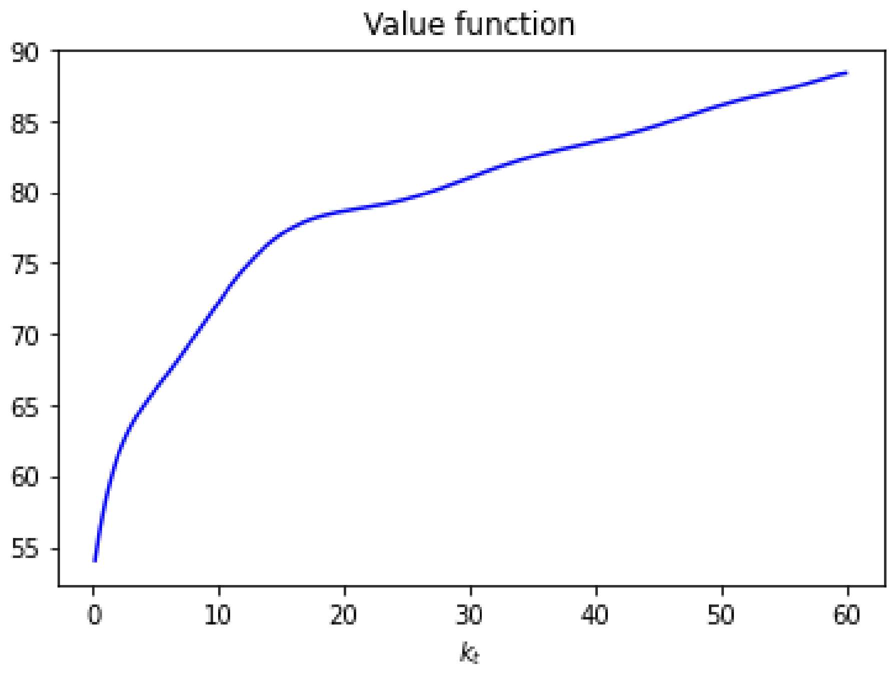


Figure 14: Value function

The value function presents similar features as previously: concave shape, monotonicity, similar values. It is more similar to those from the exercise 2, with a sharp increase for the low capital values. Also, the previously mentioned perturbations are visible. It could be the case, that due the lower density of nodes in the middle they appear and hence we use polynomials of high degree there is a tendency of appearance of some small fluctuations. At the end I provide a table with the specification of my laptop.

Component	Parameters
System	Windows 10
CPU	Intel Core i3-8130U 2.20 GHz
RAM memory	20 (19.8) GB

Table 2: Laptop's parameters