QUANT MACRO: PROBLEM SET 4

Mario Serrano

QUESTION 1. VALUE FUNCTION ITERATION

Consider a stationary economy populated by a large number of identical infinitely lived households that maximize:

$$E_0 \left\{ \sum_{t=0}^{\infty} \beta^t u(c, h_t) \right\}, \tag{1}$$

over consumption and leisure $u(c_t, 1-h_t) = \ln c_t - \kappa \frac{h_t^{1+\frac{1}{\nu}}}{1+\frac{1}{\nu}}$, subject to:

$$c_t + i_t = y_t \tag{2}$$

$$y_t = k_t^{1-\theta} \ (h_t)^{\theta} \tag{3}$$

$$i_t = k_{t+1} - (1-\delta)\, k_t \tag{4}$$

Set $\theta = .679, \beta = .988, \delta = .013$. Also, to start with, set $h_t = 1$, that is, labor is inelastically supplied. To compute the steady-state normalize output to one.

1. Pose the recursive formulation of the sequential problem without productivity shocks. Discretize the state space and the value function and solve for it under the computational variants listed below. In all these variants use the same initial guess for your value function.

a) Solve with brute force iterations of the value function. Plot your value function.

The recursive problem can be written as:

v(k) = max u(c) + Bv(k')
v(k) = max u(f(k) + (1-d)k - k') + Bv(k')

So, we need to deffine first the utility & the production function. Once we have them, we can construct a matrix in which we set the utility return of all possible combinations of todays & tomorrows capital. We need to do this cause our algorithm will be based on pick up the best of all these returns in each iteration.

First, we define the domain of k by discretizing it. We will choose as lower bond a value slightly above zero to avoid violate non-zero constrains of consumption and capital. For the upper bound, we pick up a value slightly above the steady state, since it is the convergence value. We will set a dimension of 200 points for the vector. We can compute the SS from the Euler equation:

1/B = (1-theta)*(h*z)^theta*k^(-theta) + 1 – d

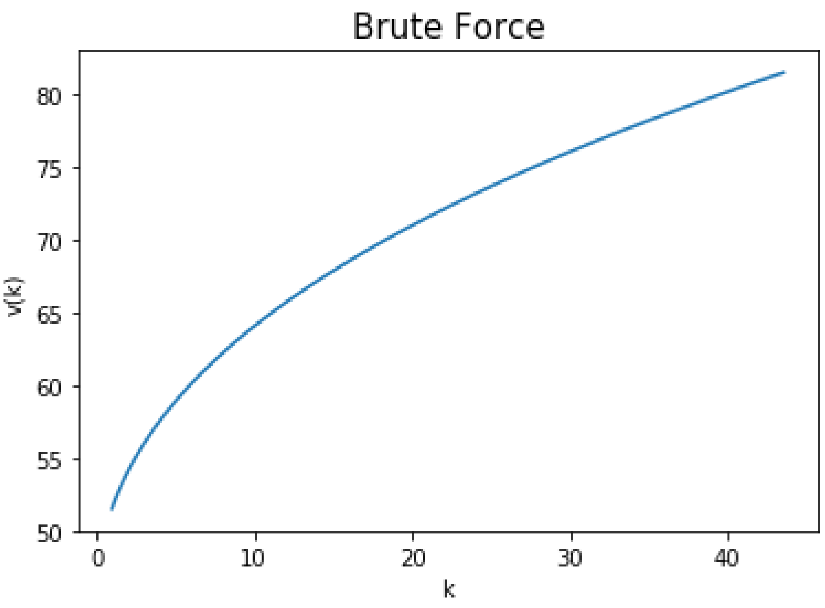
The SS obtained is: **k_SS = 42.552547163135195**

Once we have our grid for k. we define an initial guess for the value function solution to initialize iterations. Here we pick V0 = 0, that is a matrix of zeros of the specified dimension.

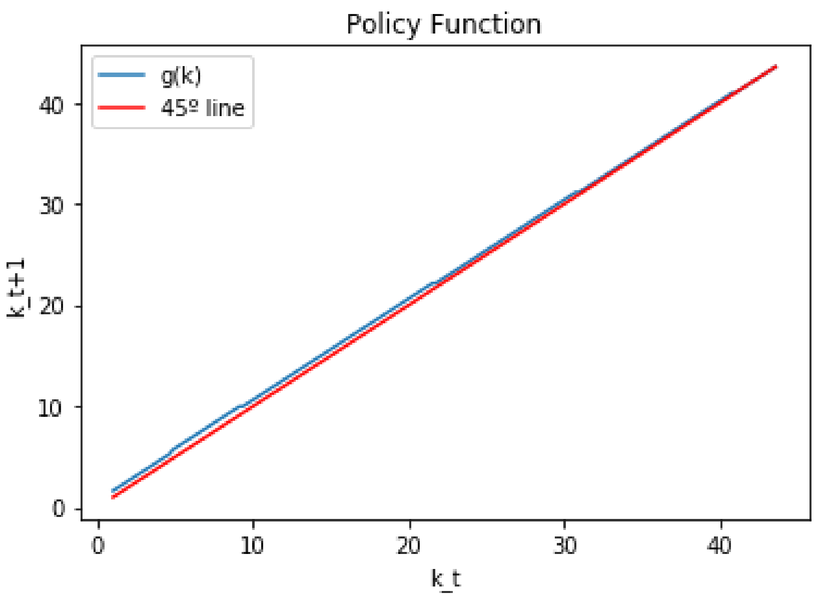
Third, we will define our M matrix, which collects the utility return of all possible combinations of todays & tomorrows capital. Besides, we make sure that we do not get any solution for which consumption is zero by adding a constrain.
Fourth, we define a method that delivers the higher value function of all possible combination, taking as input the value of next period. It also delivers the position in the matrix of this maximum. The method defines a matrix X which collects all values of V, and select for each todays k the maximum of them.

Finally, we define a method that takes the previous function as input, and iterate until the difference between the last two obtained values for V is less than a specified error. This will mean that we have reached the SS. The method delivers the vector solution V and the policy function k_t1 = g(k_t)

Plotting vector solution in front of the domain of k gives:



And comparing the policy function path with the actual equilibria:



We also compute the execution time, and obtain:
Time - Brute Force: 7.6983307 s

b) Iterations of the value function taking into account monotonicity of the optimal decision rule.

Now, we take advantage of the monotonicity of the policy rule to reduce the number of iterations. This means that the optimal decision rule increases in K. Therefore:

$$\text{if } kj > ki \text{ then } g(kj) >= g(ki).$$

We will do this by modifying matrix X, to take into account only those values of tomorrows k that are equal or bigger than the optimal decision of the past iteration.

We obtain the same graph for the solution and a lower time of execution:

Time - Monotonicity: 5.1337242 s

c) Iterations of the value function taking into account concavity of the value function.

Now, we take advantage of the concavity of the value function to reduce the number of iterations. That is:

$$\text{if } Mij + BVj > Mij+1 + BVj+1, \text{ then } Mij + BVj > Mij+2 + BVj+2.$$

We impose this condition, in order not take into account all possible combination as before. We obtain the same graph for the solution and a lower time of execution than in the brute force case, but higher that imposing monotonicity:

Time - Concavity: 6.6225001 s

d) Iterations of the value function taking into account local search on the decision rule.

In this case we take advantage of the fact that the optimal decision rule is continuous. This means that it cannot be possible that the function has jumps. Therefore, we can stablish a small grid for the solution in the next iteration, around the solution of the actual iteration. In this case, I also take into account monotonicity, and I did not consider lower possible solutions than the previous optimal decision. Doing this, and setting as a maximum the two next positions in the grid, we reduce considerably the number of iterations.

We obtain the same results in the graph than in the other parts, and a considerably lower execution time:

Time - Local Grid: 1.0390613 s

e) Iterations of the value function taking into account both concavity of the value function and monotonicity of the decision rule

We take into account now these two conditions simultaneously. Although it was expected a lower time of execution, the time obtained is superior. This has to be an error in the code, but I cannot find it.

Time - Monotonicity+Concavity: 16.530113 s

2. Redo item 1 adding a labor choice that is continuous. For this, set kapa = 5,24 and v = 2,0.

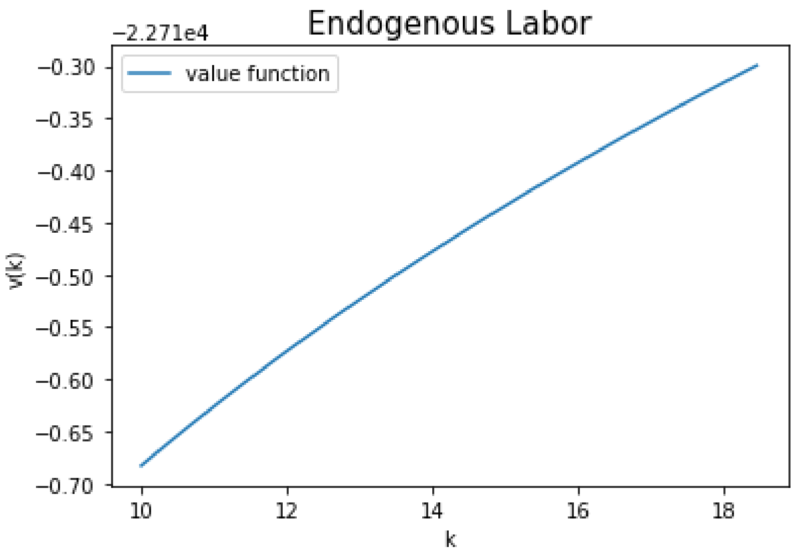
Now, we introduce labor choice as a decision variable. This do not change our problem so much, but now we have to consider a matrix of three dimensions and include labor choice in the utility function. The new SS is reached now in:

k_SS = 12.297703877378268

h_SS = 0.2890004170657163

Besides, since now there is an extra dimension, the number of iterations increases a lot and the time of execution explodes. For this reason, I change slightly the code, in order to avoid the loop for the last dimension. Also, I used the quantecon package recommended in class.

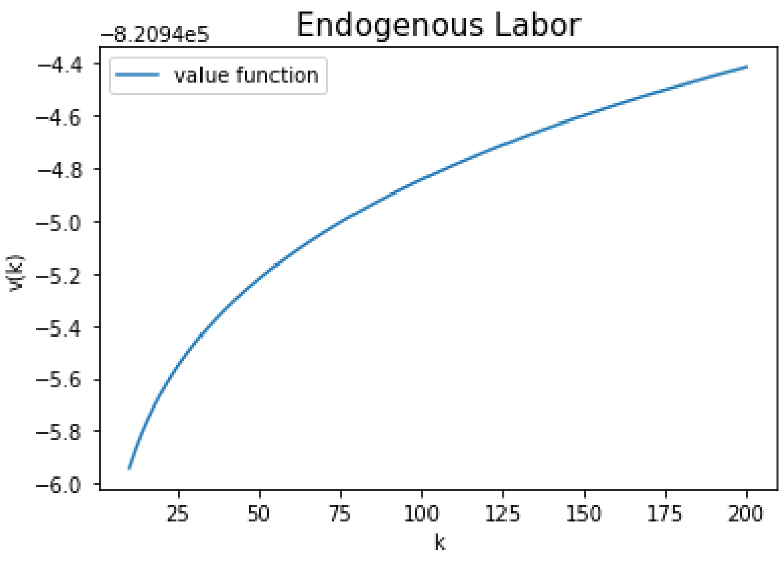
However, the results obtained are strange, cause the value of V is negative. And the convergence seems not be reached at k_SS. Again, must be a failure in the code.



The execution time obtained:

Time - Endogenous Labor: 78.9984271 s

Extending the grid of k until 200 we obtain:

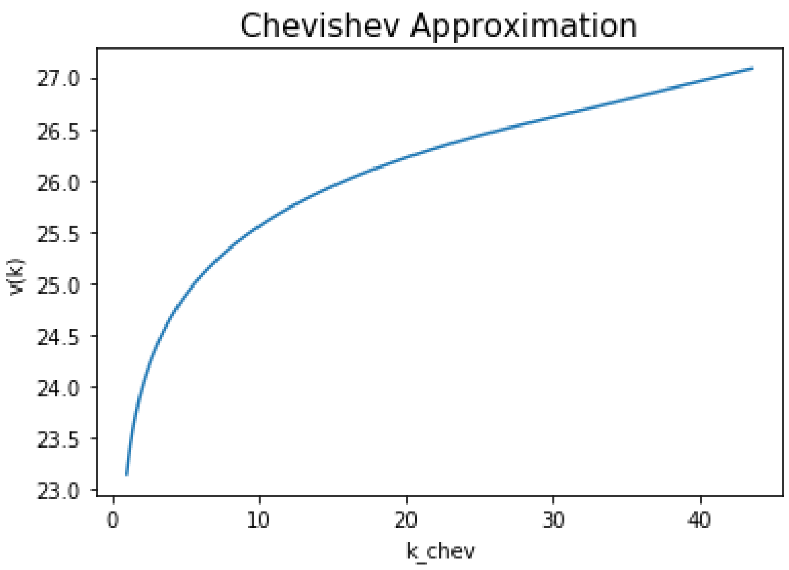


3.Redo item 1 using a Chebyshev regression algorithm to approximate the value function. Compare your results.

Now, we will use the Chebyshev regression to approximate the value function. We will define methods to compute chevysev roots, chevyshev polinomials and its coefficients. Once we did that, we will define a new computation method for V taking into account these Chevyshev approximations.

I found that the time to do this as it is in my code is extremely large, so I defined a higher tolerance. Even with this modification, the time to compute it is much higher than in the previous exercises.

The graph obtained is:



The time of execution is:

Time - Chevishev Approximation: 100.926023 s