Homework 3 *Quantitative Macroeconomics*

Arnau Pagès López

Thursday 29 October 2020

IMPORTANT NOTES ABOUT THE CODE

The code is divided by many sections. Each exercise begins with a section that parametrizes the model, creates grids and initial guess for that exercise. After that common part, each method has it's own subsection. Hence, before running a concrete method you need to parametrize it, running it's respective common part. In addition, I don't know why, sometimes, in exercises 1 and 2, when you have runned a method before and run the following method, it runs like in less than a second. If that happens, try to run it once again and it will work. Sorry for the inconvenience. An exception to this is question 2 d) where some problem makes it run super fast always. But the rest work as expected. Regarding to exercise 3 Chebychev method runs fast because is fast by definition.

Question 1.Value function iteration

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

$$E_0\Bigg[\sum_{t=0}^{\infty}\beta^t u(c_t,h_t)\Bigg] \tag{1}$$

over consumption and leisure $u(c_t,1-h_t) = lnc_t - \kappa\frac{h_t^{1+\frac{1}{\nu}}}{1+\frac{1}{\nu}}$ **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 $\theta = 0.679$, $\beta = 0.988$, $\delta = 0.013$**.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 compuitational variants listed below. In all this variants use the same initial guess for your value function.

Note that since for the moment labor supply is not a choice variable $h = 1$, the recursive formulation of previous problem is given by:

$$V(k) = \max_{c\geq 0\ \ k'}\Big[u(c,1) + \beta V(k')\Big] \tag{5}$$

subject to the recursive formulation version of the constraints (2) (3) (4) where I also already assume that $h = 1$:

$$c + i = y \tag{6}$$

$$y = k^{1-\theta} \tag{7}$$

$$i = k' - (1-\delta)k \tag{8}$$

Next, note that all this equations can be conveniently expressed as just one:

$$c = k^{1-\theta} + (1-\delta)k - k' \tag{9}$$

And using this in (5) we can rewrite this constrained recursive formulation of the problem as an unconstrained one:

$$V(k) = \max_{k' \in [0, k^{1-\theta}+(1-\delta)k]} \left[u(k^{1-\theta} + (1-\delta)k - k', 1) + \beta V(k') \right] \tag{10}$$

where notice that an additional advantage of this formulation is that now we just have one choice variable:k', i.e., c is endogenously determined by the decision of k'. Notice that actually, given the initial capital stock (the initial value for the state variable), k' decision rule actually determines everything in the model, in the sense that the decision of k' not only determines, as I've said, the consumption c, but also determines the state of the following iteration, i.e. k' chosen in current iteration is the state k of the following iteration. From here it's straightforward to see that decision on k' also determines investment, output, welfare, etc. Hence apart from making the problem become unconstrained, the substitution that I've done also allows us to work with only one choice variable (obviously this happens because we are assuming inelastic labor supply, when we will assume it to be elastic things will become more complicated). Also I just wanted to comment that the bounds for k' in the above equation follow from the fact that capital stock cannot be negative (lower bound) and that consumption cannot be negative (upper bound).
Using the utility specification that is given in the statement the problem is:

$$V(k) = \max_{k' \in [0, k^{1-\theta}+(1-\delta)k]} \left[ln(k^{1-\theta} + (1-\delta)k - k') - \frac{\tilde{\kappa}}{1+\frac{1}{\nu}} + \beta V(k') \right] \tag{11}$$

where I just use the tilde notation ($\tilde{\kappa}$) to distinguish the kappa parameter from the notation for capital stock.
However notice that the term:$-\frac{\tilde{\kappa}}{1+\frac{1}{\nu}}$ is just a constant, so that we can monotonically transform the utility by simply omitting it. The reason to do so is that (I don't know why) it was giving lots of problems in the code, so doing so I avoid this issues preserving preference order. Notice that this therm will appear later in item 2, but there, since labor supply will be elastic, it will not be a constant, and hence, obviously, I will not omit it. Hence, the equation that I will be solving in Python code in this item 1 will actually be:

$$V(k) = \max_{k' \in [0, k^{1-\theta}+(1-\delta)k]} \left[ln(k^{1-\theta} + (1-\delta)k - k') + \beta V(k') \right] \tag{12}$$

Notice that when we say solve in this environment, we mean to find (or rather approximate) a value function $V(k)$ that solves the Bellman equation and it's associated (approximations) for optimal policy functions or decision rules $g_c(k)$ and $g_{k'}(k)$.

However, before to start solving for it, we need to compute the capital stock at the steady state $k*$. This will allow me to have a good maximum value of the grid of capital when I discretize the state space. Since I already solved for the steady state step by step in Homework 2 in a very similar (not identical) environment, and it's not the main goal of this Homework 3, I will solve it skipping several steps. Notice that the Euler equation associated to the sequential formulation of the problem is:

$$u'(c_t, 1) = \beta u'(c_{t+1}, 1)[1 - \delta + (1-\theta)k_{t+1}^{-\theta}] \tag{13}$$

At the setady state (13) is:

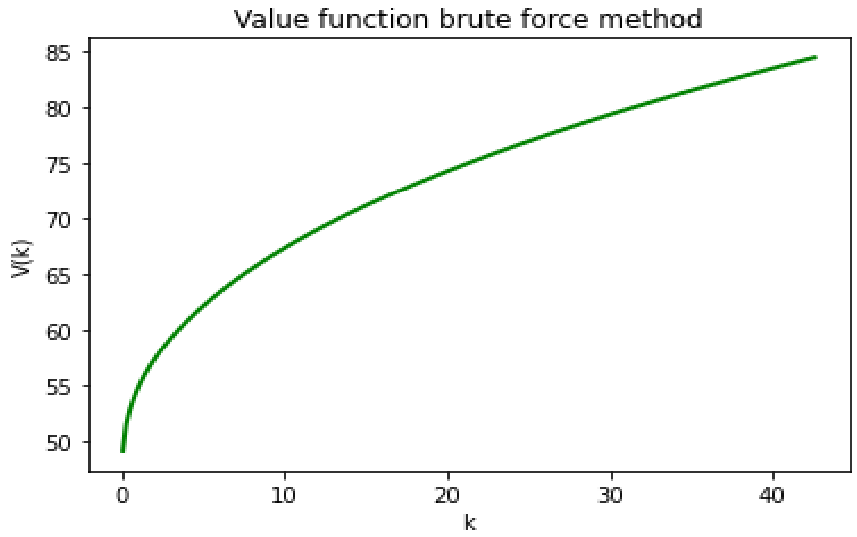
$$1 = \beta[1 - \delta + (1-\theta)k^{*-\theta}] \tag{14}$$

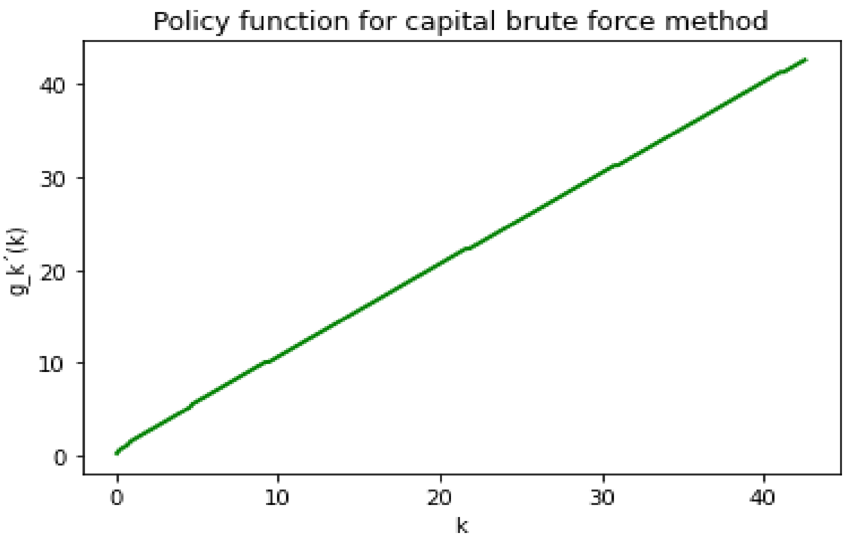
And finally from here we can obtain an explicit expression for k^*:

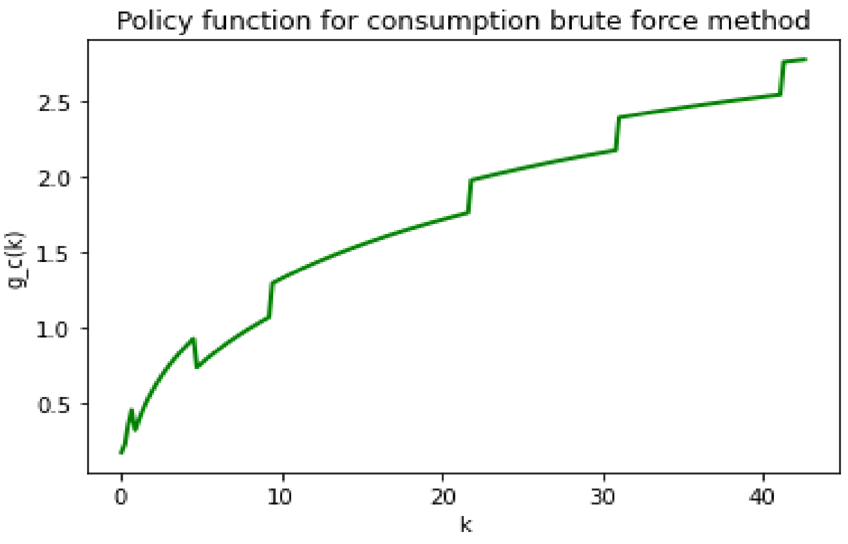
$$k^* = \left[\frac{\frac{1}{\beta} + \delta - 1}{1-\theta}\right]^{\frac{-1}{\theta}} \tag{15}$$

Now I have all the elements I need to solve in Python using the different methods suggested below. For all the methods in item 1 I use a grid for capital of 200 elements, an initial guess of 0 for the value function and a tolerance error of 0.01.

(a)Solve with brute force iterations of the value function. **Plot your value function.**
Brute force iterations took 372 iterations and 51.71 seconds. Results are presented below.

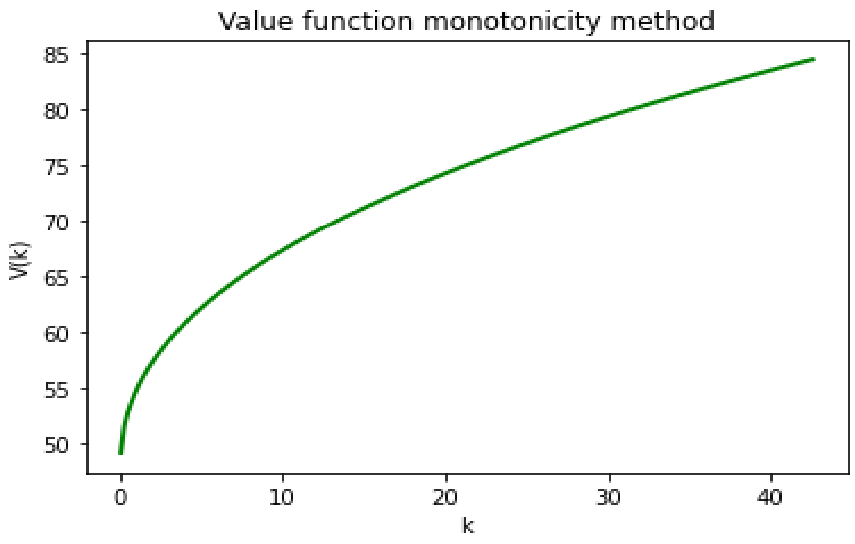


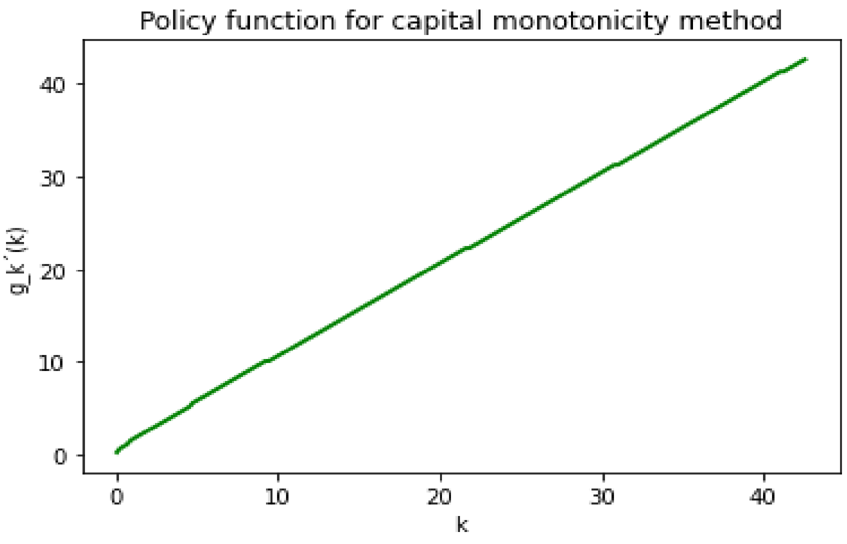


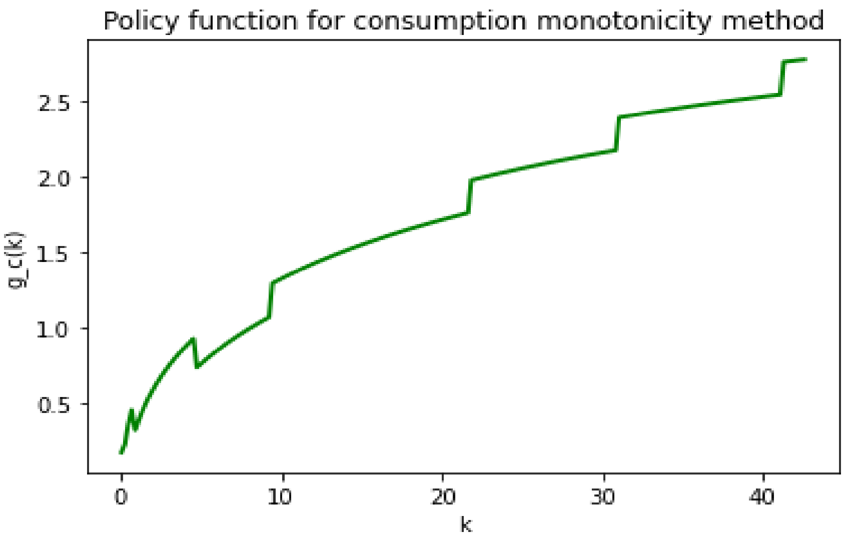


(b)Iterations of the value function taking into account monotonicity of the optimal decision rule.

Iterations taking into account monotonicity of decision rule took 372 iterations and 38.86 seconds. Results are presented below.

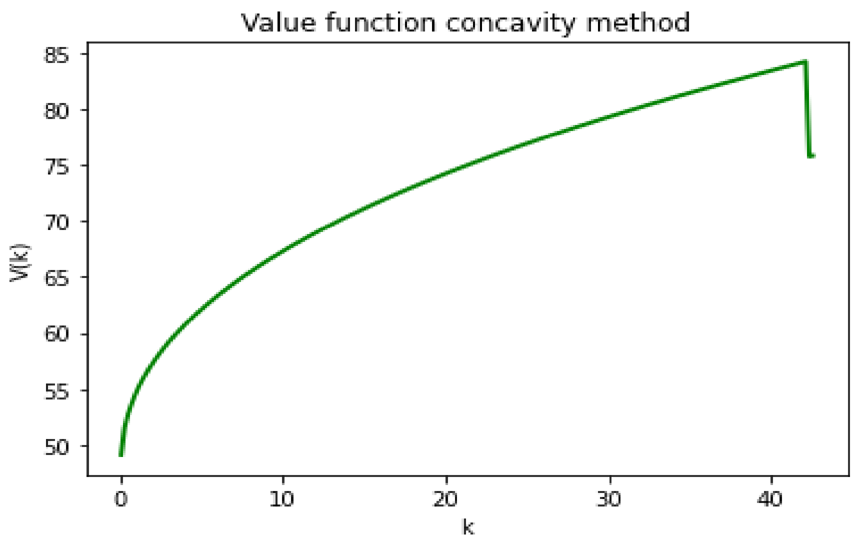

Value function monotonicity method
V(k)
k

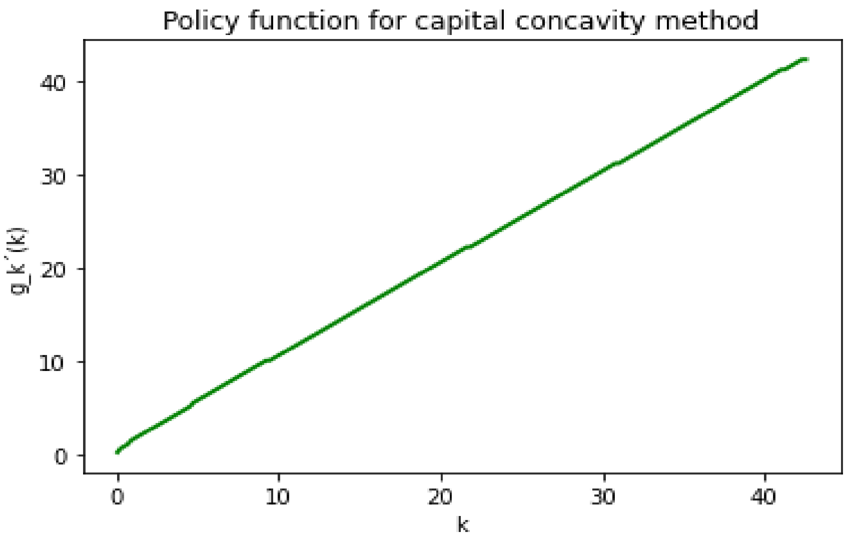

Policy function for capital monotonicity method
g_k´(k)
k

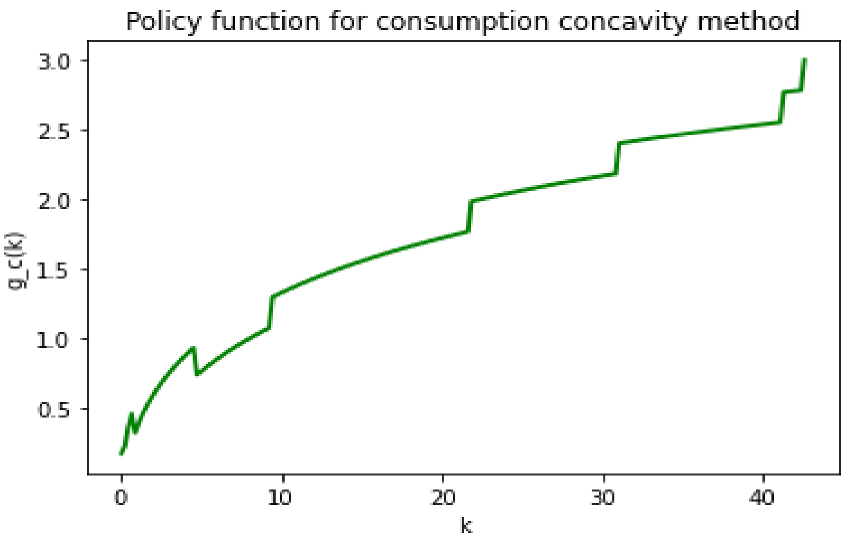

Policy function for consumption monotonicity method
g_c(k)
k

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

Iterations taking into account concavity of value function took 372 iterations and 32.33 seconds. Results are presented below.

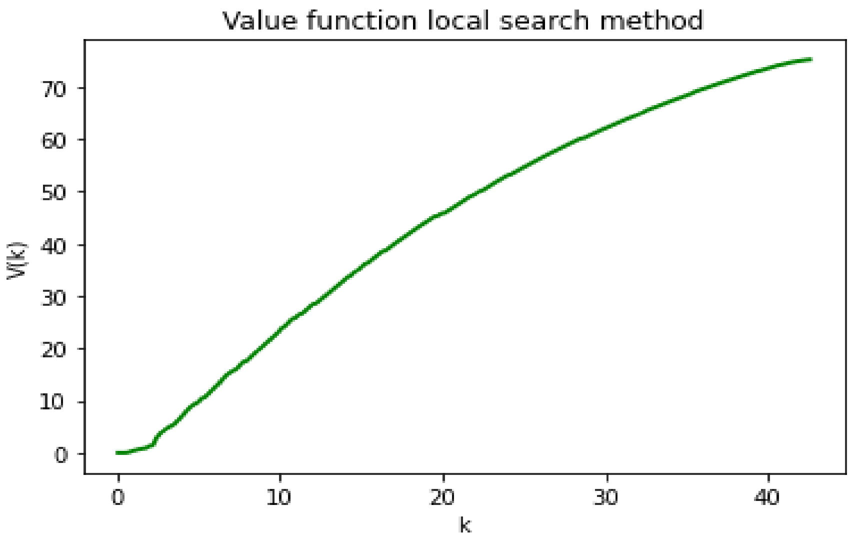


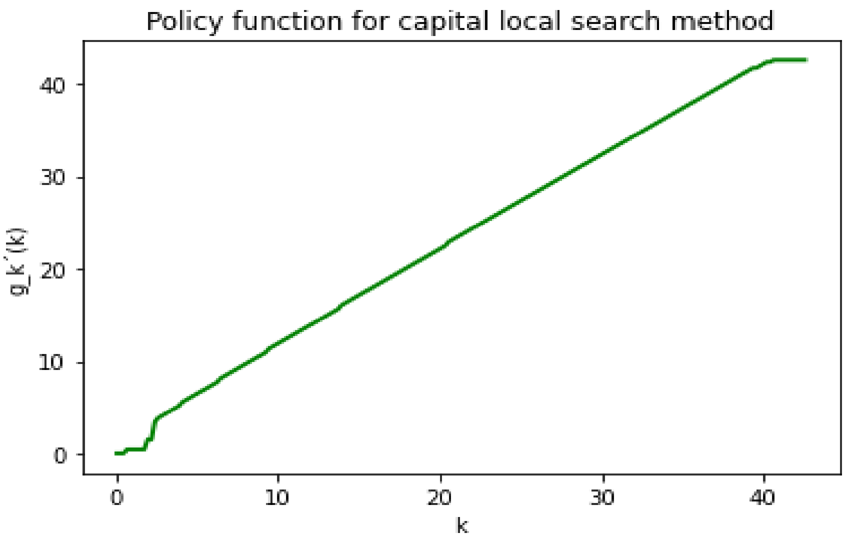


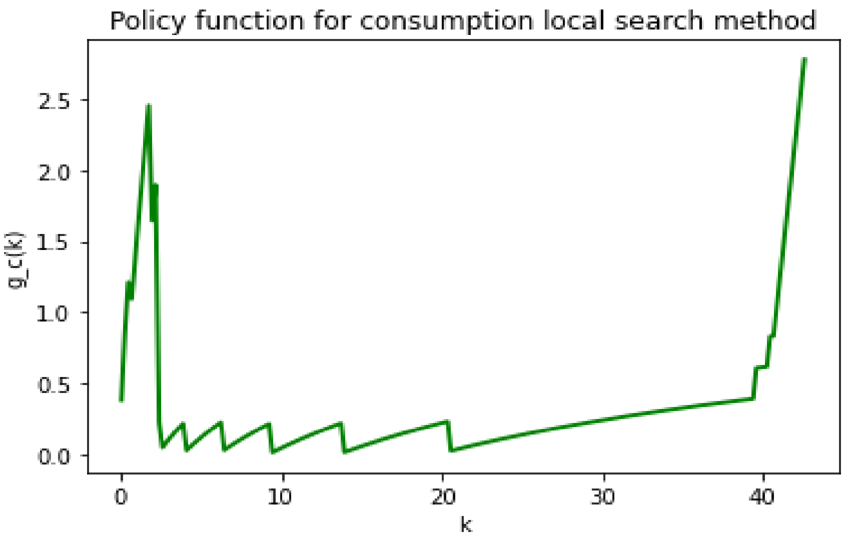


(d)Iterations of the value function taking into account local search on the decision rule.

Iterations taking into account local search on the decision rule took 200 iterations and 31.36 seconds. Results are presented below.

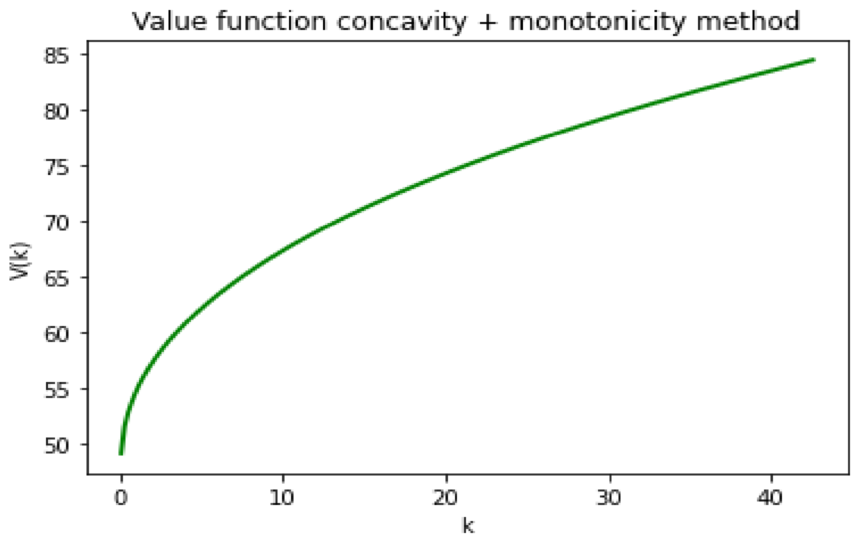


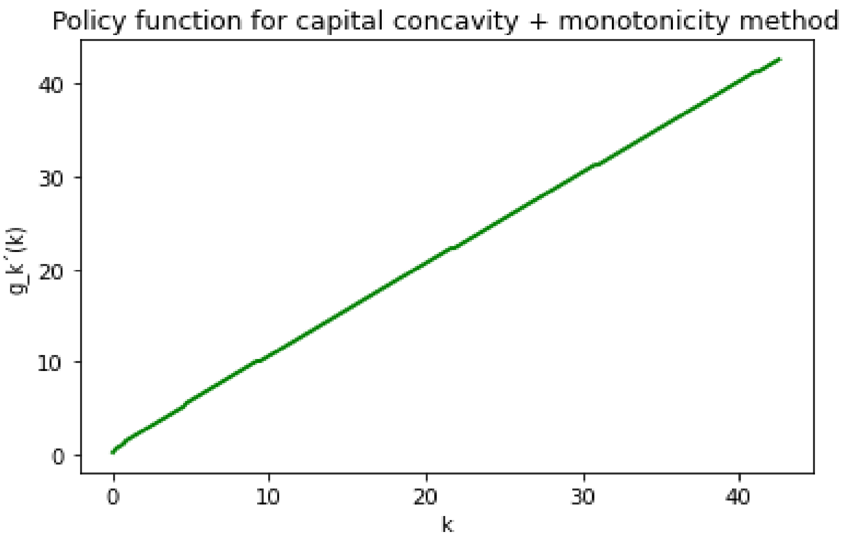


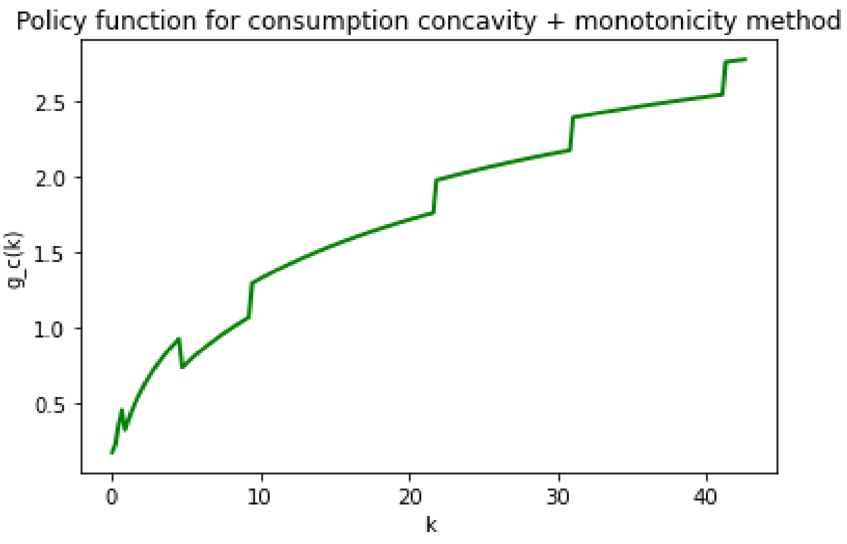


(e)Iterations of the value function taking into account both concavity of the value function and monotonicity of the decision rule.

Iterations taking into account both concavity of value function and monotonicity of decision rule took 373 iterations and 8.05 seconds.Results are presented below.


Value function concavity + monotonicity method
V(k)
k
85
80
75
70
65
60
55
50
0
10
20
30
40


Policy function for capital concavity + monotonicity method
g_k´(k)
k
40
30
20
10
0
0
10
20
30
40


Policy function for consumption concavity + monotonicity method
g_c(k)
k
2.5
2.0
1.5
1.0
0.5
0
10
20
30
40

(f) Use Howard's policy iterations waiting until converged to solve the problem. Start the policy iteration at three different iterations of the value function, and report the differences.
I was not able to do it.

(g)Use policy iterations with 5, 10, 20 and 50 steps in between policy reassessments.
I was not able to do it.

Report the time and number of iterations needed per variant and describe your results.
The time and number of iterations is at the beginning of each method. As expected, the slowest method was the brute force method (a), then the monotonicity method (almost 13 seconds faster than a), then the concavity method (near 19 seconds faster than a), followed by the local search method (near 20 seconds faster than a), and finally the fastest one is the iterations taking into account both concavity of the value function and monotonicity of the decision rule, which was almost 44 seconds faster than a.

Note that more speed does not necessarily mean less iterations, because the speeding up algorithms precisely consist on making the iterations faster, not on making less iterations. For example, the method in (e) is the fastest one and took 1 iteration more than the slowest one.
Regarding to reliability, taking into account the properties that value functions and policy functions are assumed to have, the methods that best approach to it are the brute force method (a), the monotonicity method (b) and the method combining both concavity and monotonicity (e).
Hence, taking into account this results, if I had to choose one method, I would choose iterations taking into account both concavity of the value function and monotonicity of the decision rule, because is the fastest one and in addition does a good approximation of the shape of the value function and decision rules.

2.Redo item 1 adding a labor choice that is continuous. For this, set $\kappa = 5.24$ and $\nu = 2.0$.

Assuming continuous labor choice notice that the recursive problem becomes

$$V(k) = \max_{c\geq 0 \;\; k' \;\; h\in[0,1]} \Big[u(c,h) + \beta V(k')\Big] \tag{16}$$

s.t.

$$c + i = y \tag{17}$$

$$y = k^{1-\theta}h^{\theta} \tag{18}$$

$$i = k' - (1-\delta)k \tag{19}$$

Now if as in item 1, we combine the constraints (17) (18) and (19) isolating consumption, and plug it in (16) we get the unconstrained problem:

$$V(k) = \max_{k'\in[0,k^{1-\theta}+(1-\delta)k] \;\; h\in[0,1]} \Big[u(k^{1-\theta}h^{\theta} + (1-\delta)k - k', h) + \beta V(k')\Big] \tag{20}$$

Which finally, using the concrete utility specification given in the statement is:

$$V(k) = \max_{k'\in[0,k^{1-\theta}+(1-\delta)k] \;\; h\in[0,1]} \left[ln(k^{1-\theta}h^{\theta} + (1-\delta)k - k') - \frac{\tilde{\kappa}h^{1+\frac{1}{\nu}}}{1+\frac{1}{\nu}} + \beta V(k')\right] \tag{21}$$

where again I just use the tilde notation ($\tilde{\kappa}$) to distinguish the kappa parameter from the notation for capital stock.

This Bellman equation (21) will be the one that I will solve in Python. When we say solve in this environment, we mean to find (or rather approximate) a value function $V(.)$ that solves the Bellman equation and it's associated (approximations) for optimal policy functions or decision rules $g_c(k)$, $g_h(k)$ and $g_{k'}(k)$. Note that the only thing that changes computationally with respect to item 1 is that now we don't keep $h = 1$ all the time. Instead, it is a choice variable moving in the range [0,1].What I do in Python is to in addition of constructing a grid for capital I also construct one for h.However, don't get confused, this DOES NOT mean that h is an state variable. The problem continues having a single state variable k, as the notation in (16),(20) and (21) reflects. That is, the fact that you define a grid for a variable does not imply that that is an state variable. For example notice that in item 1 (and will do also in this item 2), when we construct a grid for capital, actually the grid is useful for k but also for k', and notice that k' is not an state variable, is a choice one.

Notice that now, before proceeding to Python, since with endogenous labor choice, the problem has changed, we have to actualize the capital at steady state k^*, which provides an upper bound for the grid of capital. Notice that now, the Euler equation corresponding to the sequential formulation of the problem is:

$$u'(c_t, h_t) = \beta u'(c_{t+1}, h_{t+1})[1 - \delta + (1-\theta)k_{t+1}^{-\theta}h_{t+1}^{\theta}] \tag{22}$$

At the setady state (22) is:

$$1 = \beta[1 - \delta + (1-\theta)k^{*-\theta}h^{*\theta}] \tag{23}$$

Normalizing output at steady state to one: $y^* = 1$, (3) implies:

$$h^{*\theta} = \frac{1}{k^{*1-\theta}} \tag{24}$$

So that using (24), equation (23) can be rewritten as:

$$1 = \beta[1 - \delta + (1-\theta)k^{*-1}] \tag{25}$$

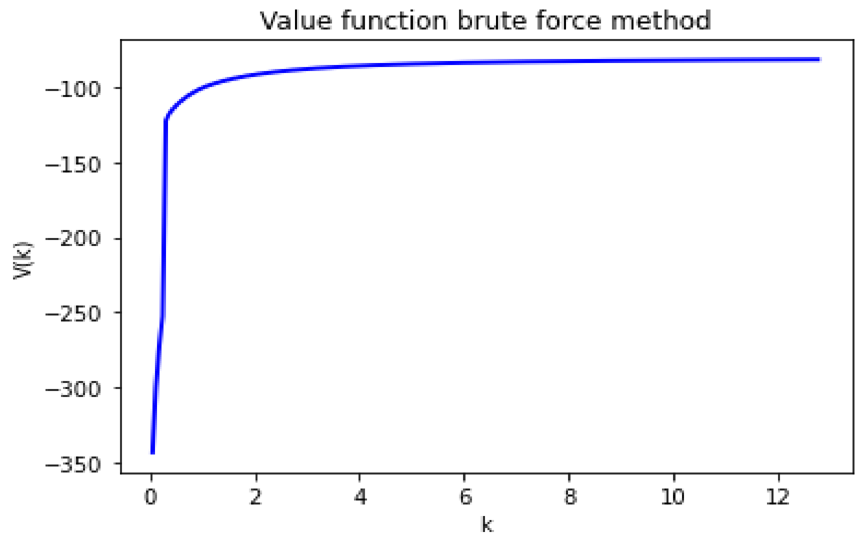
And finally from here we can obtain an explicit expression for k^*:

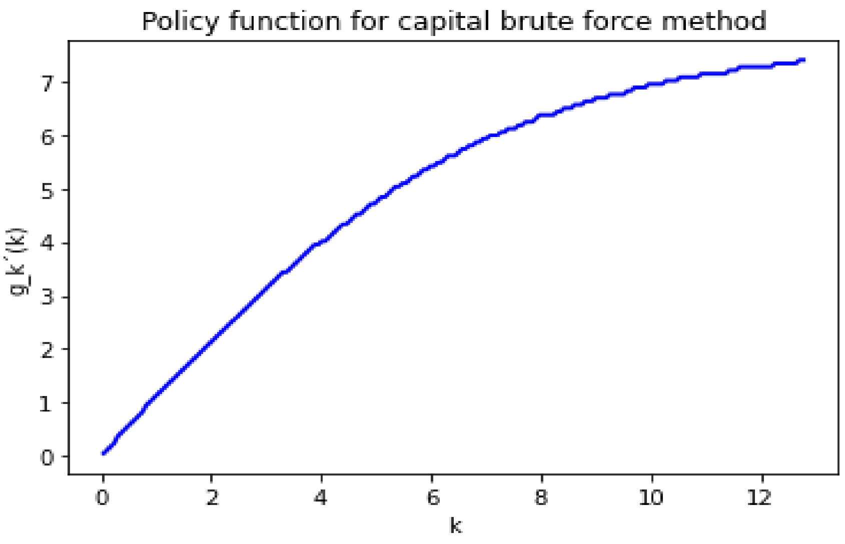
$$k^* = \left[\frac{1-\theta}{\frac{1}{\beta} + \delta - 1}\right] \tag{26}$$

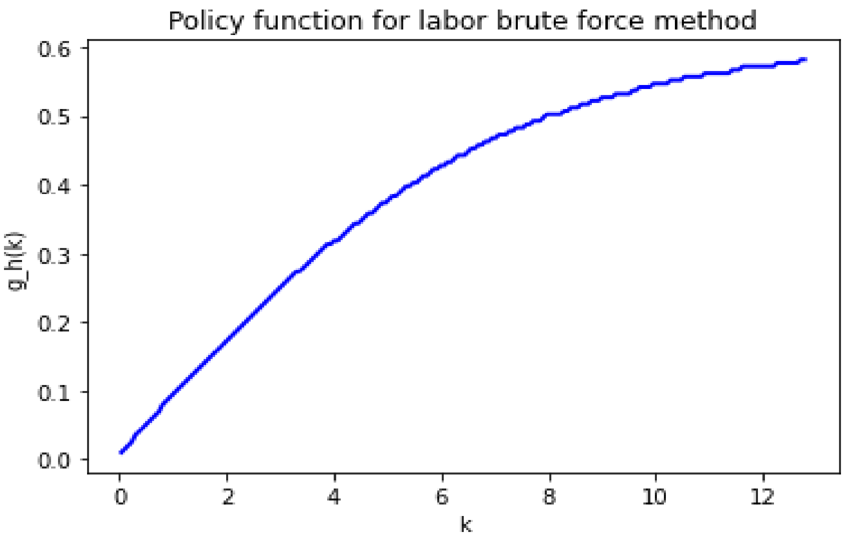
As the range for h is very well defined,$h \in [0, 1]$ and is not an state variable, I did not considered necessary to compute h^* to use it as upper bound for labor grid. I use the natural bound 1 instead.
For all the methods in item 2 I use a grid for capital of 200 elements,a grid for labor of 200 elements, an initial guess of 0 for the value function and a tolerance error of 0.01.

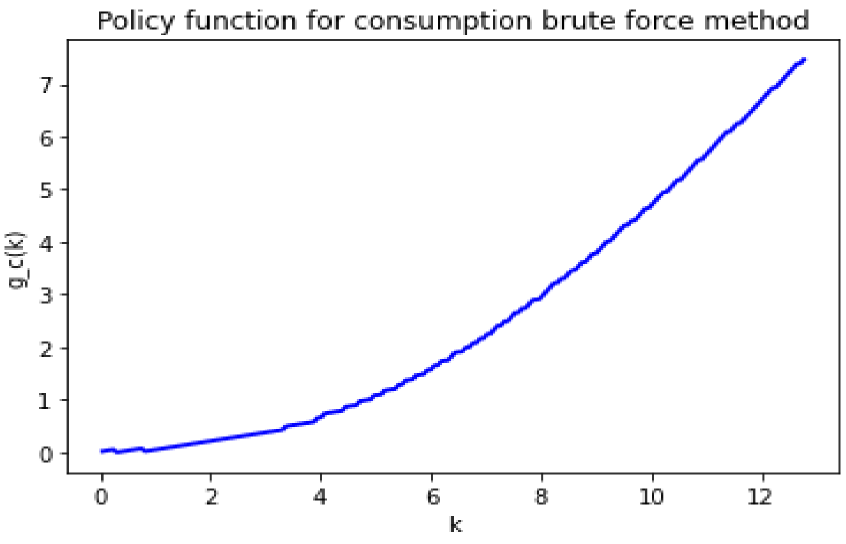
(a)Solve with brute force iterations of the value function. Plot your value function.

Brute force iterations took 500 iterations and 68.64 seconds.



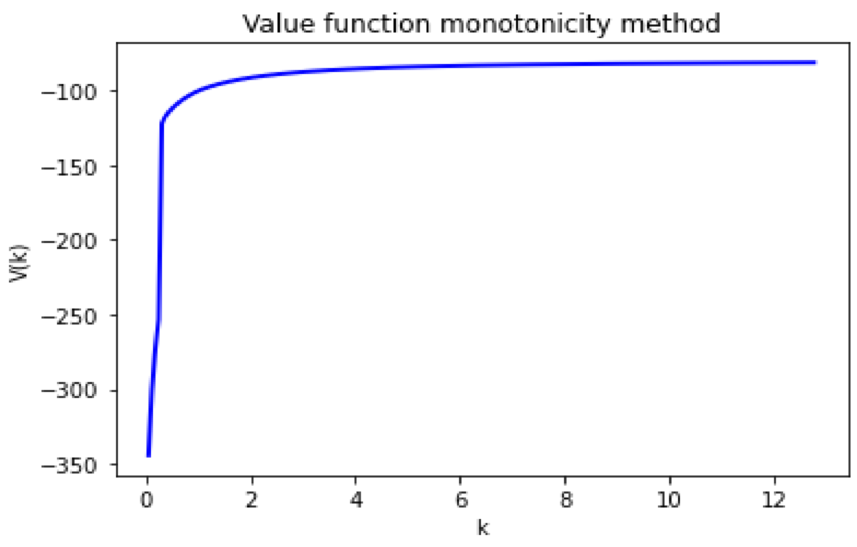

Policy function for capital brute force method
g_k´(k)
k
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12

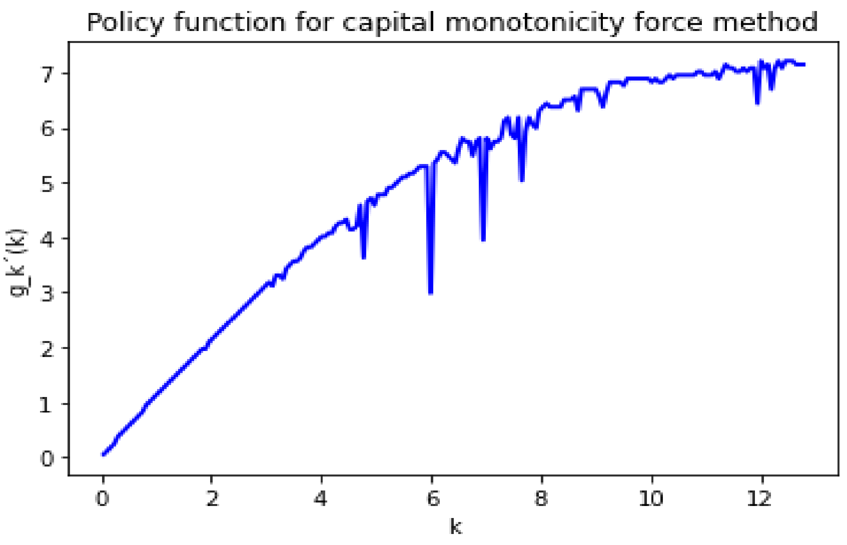

Policy function for labor brute force method
g_h(k)
k
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12

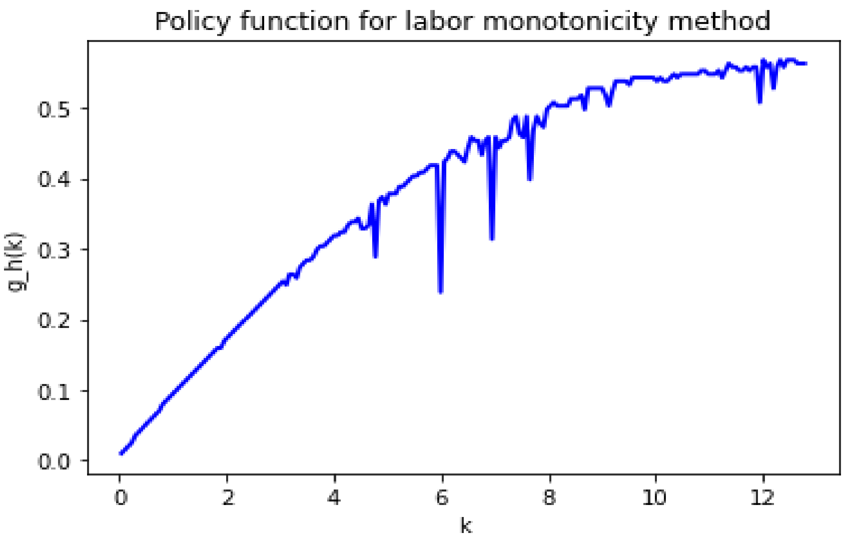

Policy function for consumption brute force method
g_c(k)
k
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12

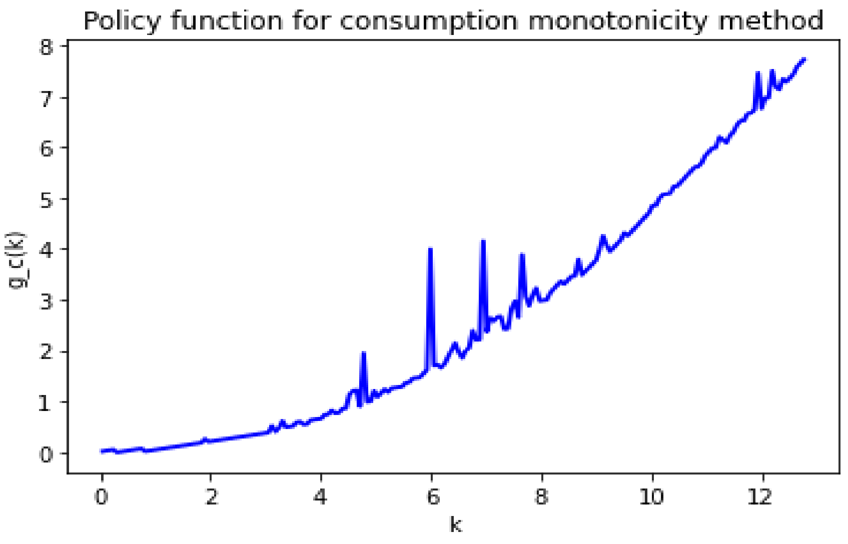
(b)Iterations of the value function taking into account monotonicity of the optimal decision rule.

Iterations taking into account monotonicity of decision rule took 1205 iterations and 161.52 seconds.



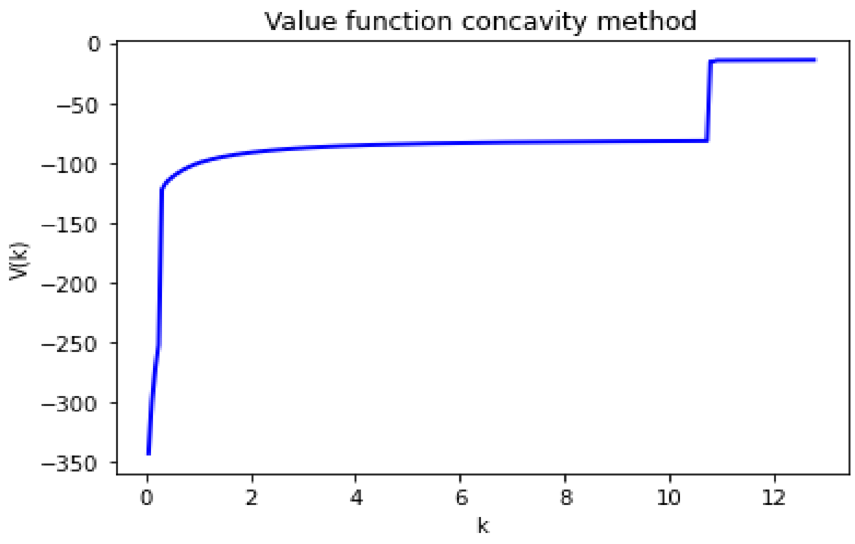


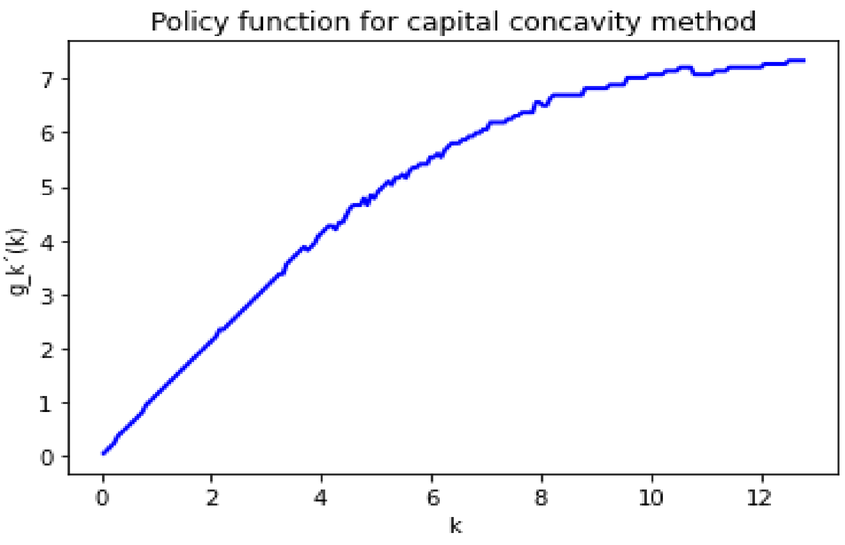


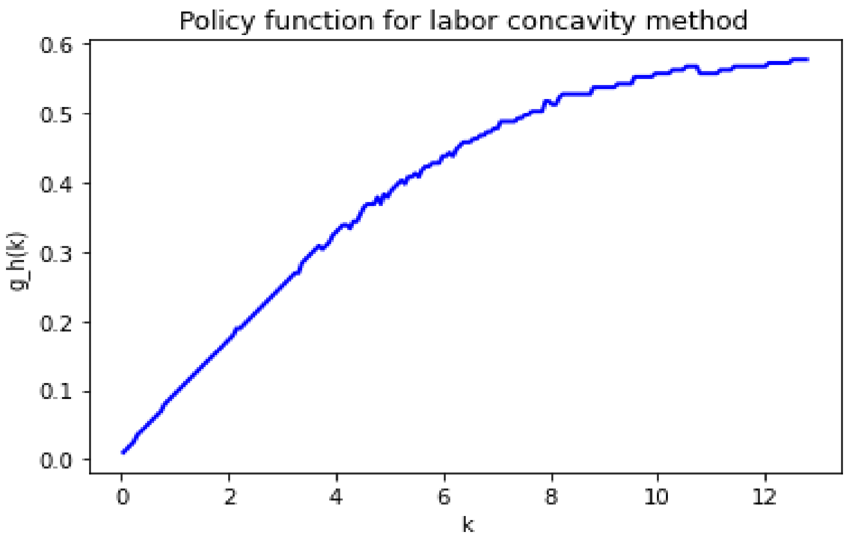


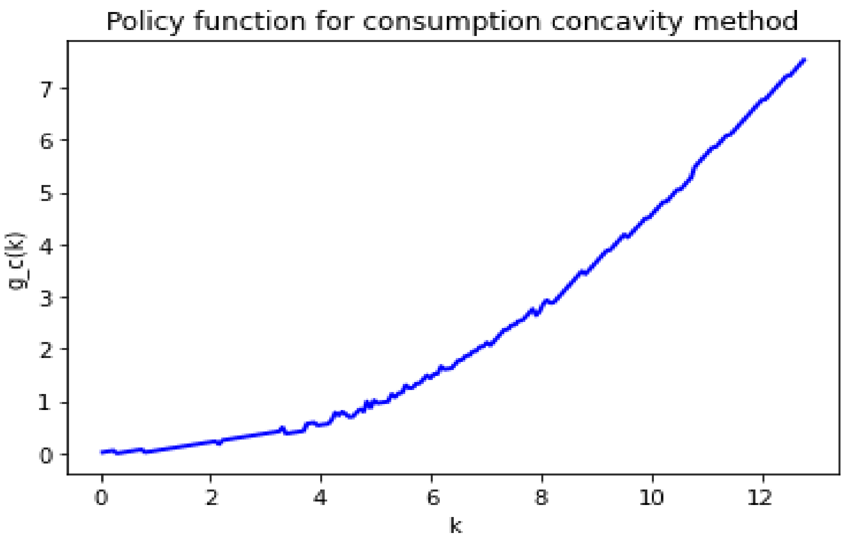
(c)Iterations of the value function taking into account concavity of the value function.

Iterations taking into account concavity of the value function took 500 iterations and 25.85 seconds.


Value function concavity method
V(k)
k
0
−50
−100
−150
−200
−250
−300
−350
0
2
4
6
8
10
12

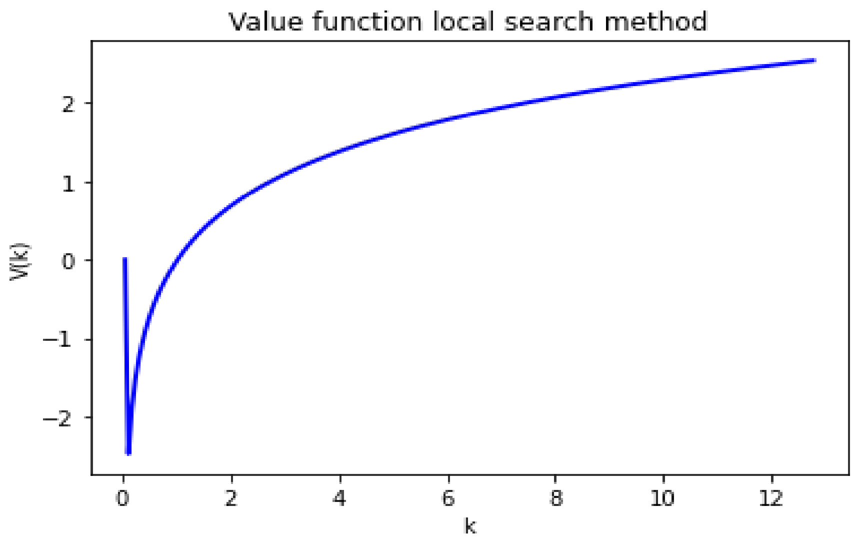

Policy function for capital concavity method
g_k´(k)
k
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12

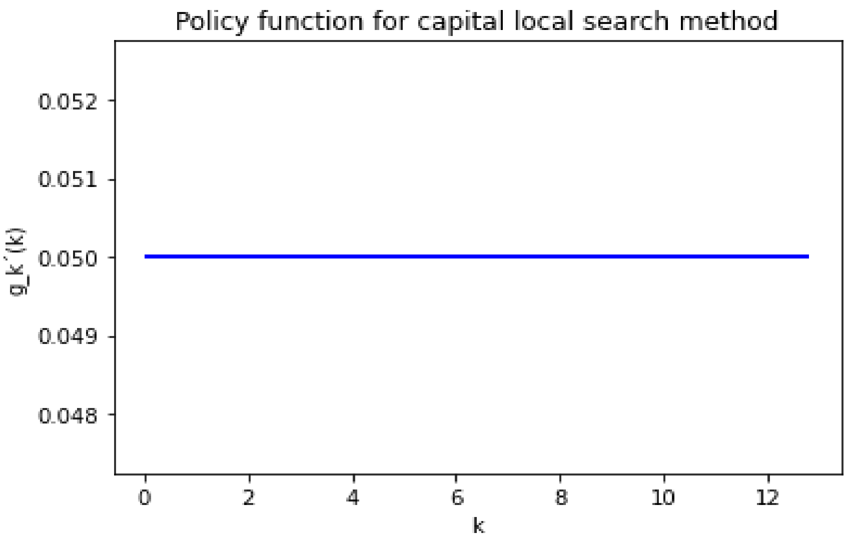

Policy function for labor concavity method
g_h(k)
k
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12

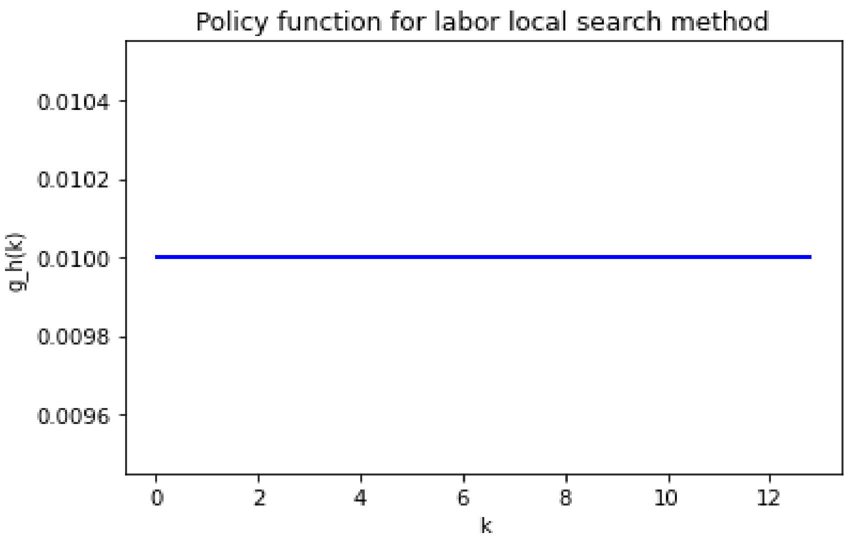


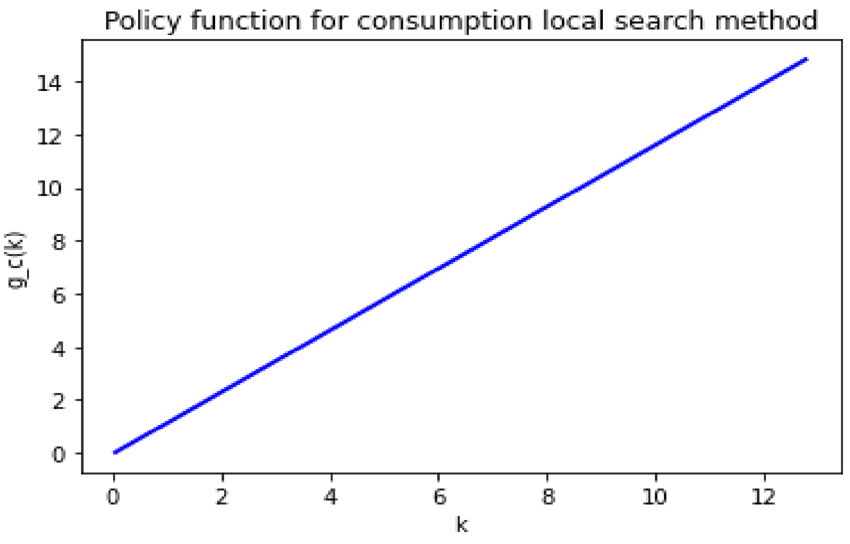
(d)Iterations of the value function taking into account local search on the decision rule.

Iterations taking into account local search on the decision rule took 2 iterations and 0.59 seconds. Obviously, something must be not good in this one. Hence I will omit it in the comments. It seems some mistake at the really beginning of the kgrid, because except in that neighborhood of $k = 0$ notice that the value function is behaved more or less as expected. Hence I think that if that small detail was solved the method would work well. But I was not able to correct it.



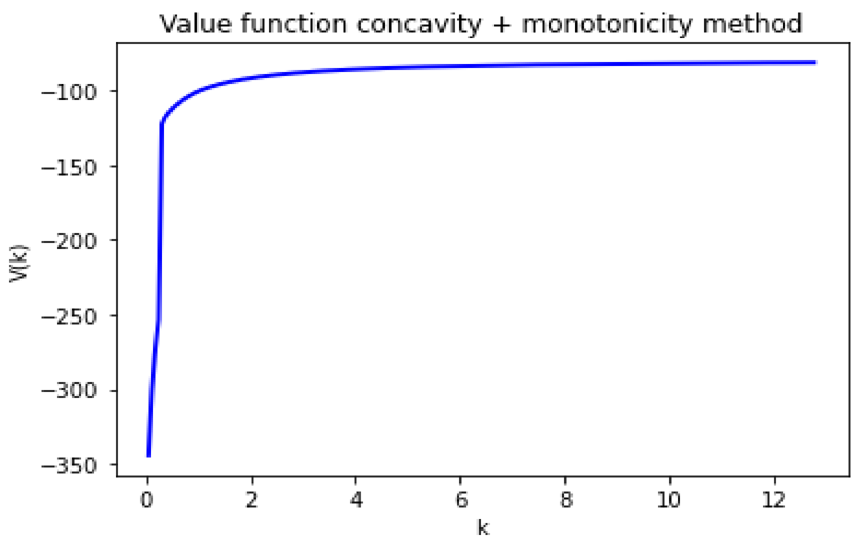

Policy function for capital local search method
g_k´(k)
0.052
0.051
0.050
0.049
0.048
0
2
4
6
8
10
12
k

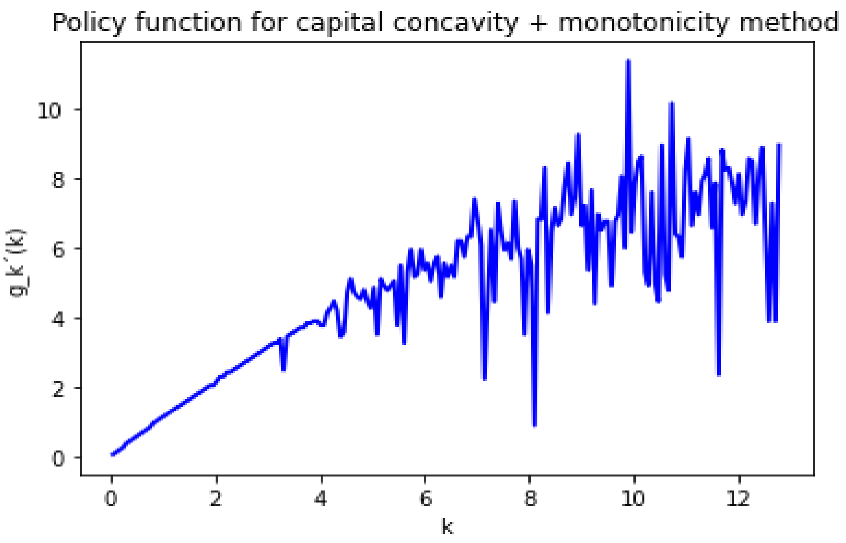

Policy function for labor local search method
g_h(k)
0.0104
0.0102
0.0100
0.0098
0.0096
0
2
4
6
8
10
12
k

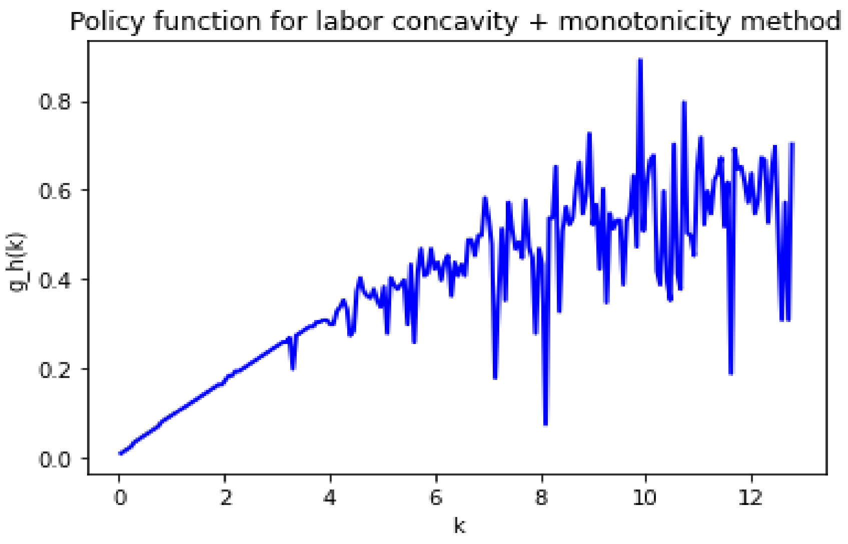

Policy function for consumption local search method
g_c(k)
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
k

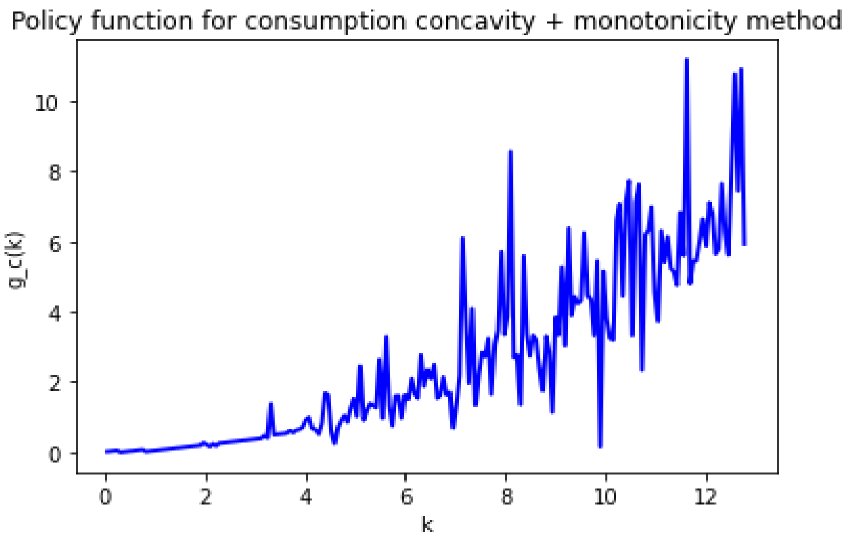
(e)Iterations of the value function taking into account both concavity of the value function and monotonicity of the decision rule.

Iterations taking into account both concavity of the value function and monotonicity of the decsion rule took 5487 iterations and 97.69 seconds.









(f) Use Howard's policy iterations waiting until converged to solve the problem. Start the policy iteration at three different iterations of the value function, and report the differences.
I was not able to do it.

(g)Use policy iterations with 5, 10, 20 and 50 steps in between policy reassessments.
I was not able to do it.
Report the time and number of iterations needed per variant and describe your results.

The time and number of iterations is at the beginning of each method. Notice that now things change a lot from item 1, in terms of speed.Now, the slowest method was the iterations taking into account the monotonicity of the decision rule (method b), which took

161.52 seconds, followed by the iterations taking into account both concavity and monotonicity (almost 64 seconds faster than b). Then we have the brute force iterations (almost 93 seconds faster than b) and finally, the concavity method (almost 137 seconds faster than b). I ignore local search method in this list for the reasons explained above.

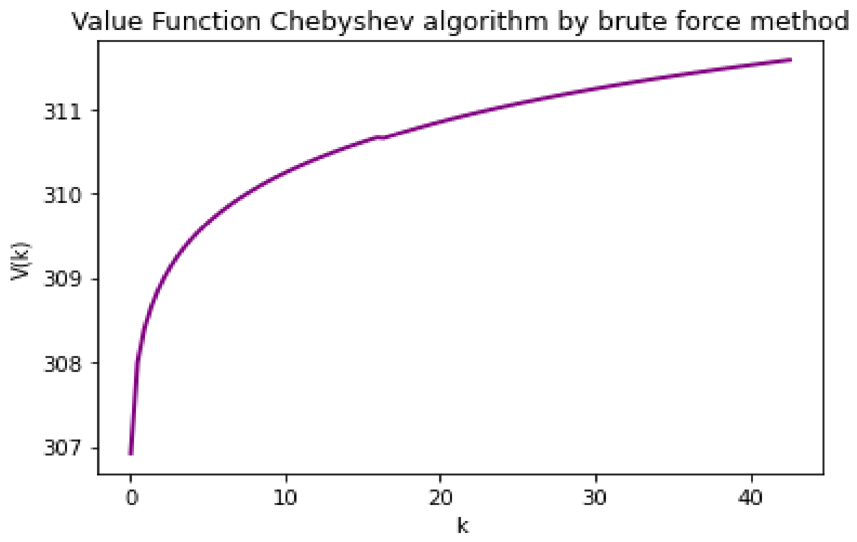
Regarding to reliability, taking into account the properties that value functions and policy functions are assumed to have, the method that best approaches to it is the brute force one (a). However, iterations taking into account concavity (c) also provides a more or less good approximation to the expected shape and has the advantage that is notably faster. Also methods in (b) and (e) approach a bit to the expected shape.
Hence, taking into account this results, if I had to choose one method, in this case I would choose the brute force method, because is the one that seems more reliable to approach the expected shape of the different functions, and in addition is the second one in therms of speed. If someone was more interested in speed and not so much in reliablility, then a good option for him/her could be the fastest one, the concavity method (c) because is sensibly faster (43 seconds faster) than brute force, and also provides a more or less good approximation to the expected shape as I said before.

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

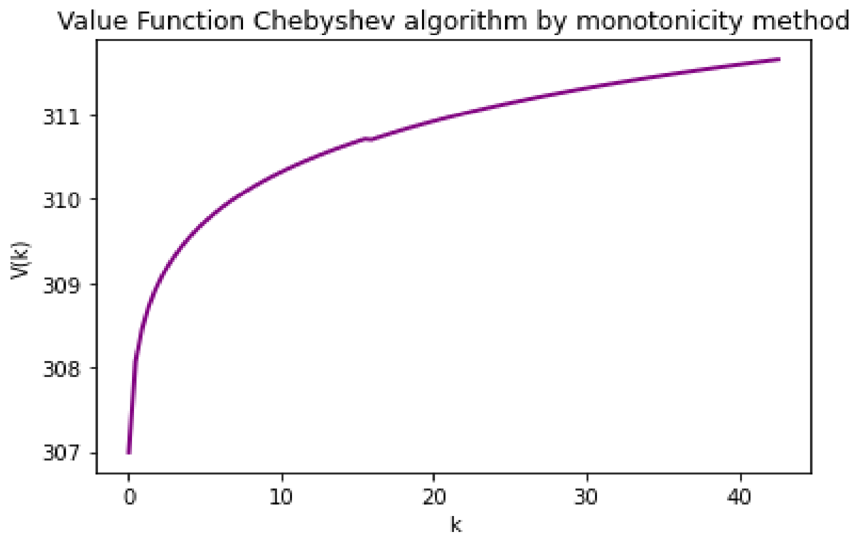
Here we return to the scenario of item 1, in which the labor supply was assumed to be inelastic. Hence, we will be again solving the exact same problem that appears in (12), but now using a continuous method, which is Chebychev regression algorithm.
Notice that in order to make it more comparable with methods in 1, here, as we again assume inelastic labor supply $h = 1$, I again use the monotonic transformation of omitting the constant therm: $-\frac{\kappa}{1+\frac{1}{\nu}}$ as I already explained and justified in 1.

Finally let me say that in this exercise I will be only providing value functions graphs because I was not sure about the way of subtracting the associated optimal policy functions, and instead of giving something that is not correct I preferred to just provide the value functions. Results are shown below. At the end, some comments are made.

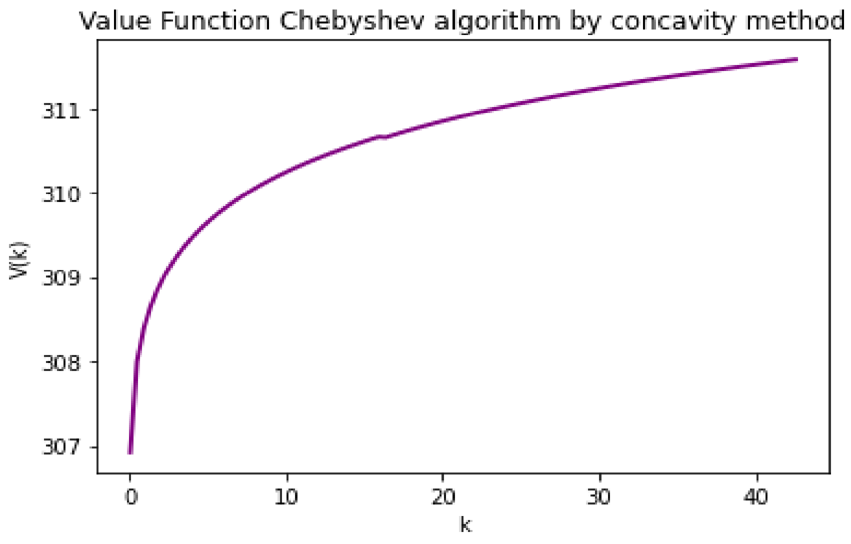
(a)Solve with brute force iterations of the value function. **Plot your value function.**
Brute force iterations with Chebychev regression algorithm took 484 iterations and 17.10 seconds.



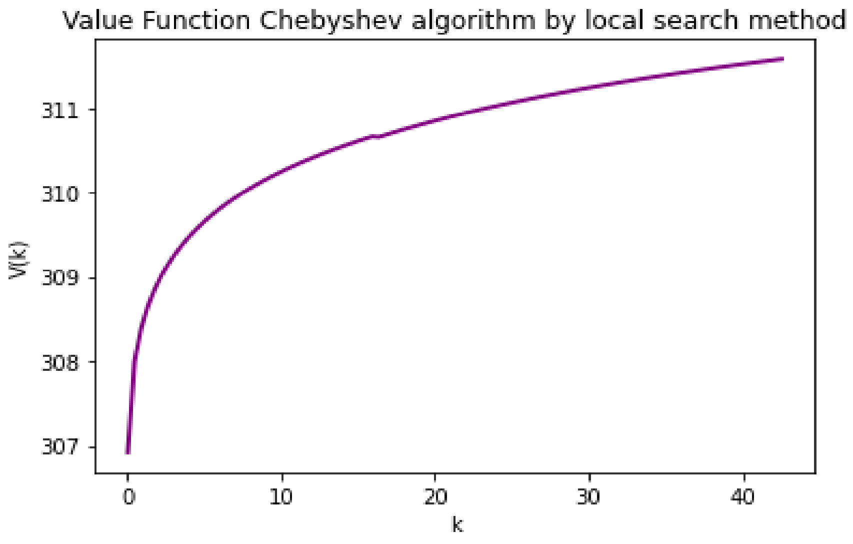
(b)Iterations of the value function taking into account monotonicity of the optimal decision rule.
Iterations with Chebychev regression algorithm taking into account monotonicity of the optimal decision rule took 491 iterations and 21.81 seconds.



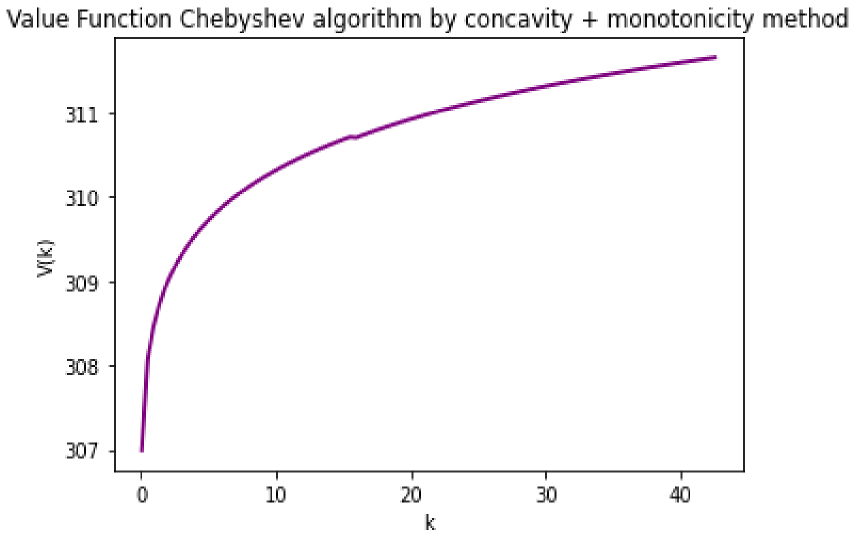
(c)Iterations of the value function taking into account concavity of the value function.
Iterations with Chebychev regression algorithm taking into account concavity of the value function took 484 iterations and 1.38 seconds.



(d)Iterations of the value function taking into account local search on the decision rule.
Iterations with Chebychev regression algorithm taking into account local search on the decision rule took 484 iterations and 16.67 seconds.



(e)Iterations of the value function taking into account both concavity of the value function and monotonicity of the decision rule Iterations with Chebychev regression algorithm taking into account both concavity of the value function and monotonicity of the decision rule took 491 iterations and 1.67 seconds.



(f) Use Howard's policy iterations waiting until converged to solve the problem. Start the policy iteration at three different iterations of the value function, and report the differences.
Not able to do it.

(g)Use policy iterations with 5, 10, 20 and 50 steps in between policy reassessments.
Not able to do it.

Report the time and number of iterations needed per variant and describe your results.

Notice that, in general, the method works fast and provides a very good approximation to the expected shape of the value function.
In terms of comparison, as we assume again inelastic labor supply, we must compare it with item 1. Notice that all the variants/methods using Chebychev, are faster than the same variants when using discrete methods, and notice that faster does not mean that they use less iterations. Indeed, they use more iterations, but they do them in a faster way. Why? One cause (among many others more technical) is that now we can use a grid that is much smaller than the one we used when going discrete. This saves a lot of work to the CPU at each iteration.
Regarding to the comparison of speed between the different methods when using Chebychev, we can see that the slowest one is the monotonicity method (b) which takes 21.81 seconds, followed by the brute force one (a) (4.71 seconds faster than (b)). Then we have

the local search method (d), which is 5.14 seconds faster than (b), followed by the concavity+monotonicity method(e) (20.14 seconds faster than (b)). Finally, the fastest one is the concavity method, which is 20.43 seconds faster than (b).
Note again that in general this methods are super fast, the slowest one takes only 21.81 seconds and the fastest one 1.38 seconds!

In terms of quality all the methods approximate very well the expected shape of the value function, which was not happening in items 1 and 2. Hence, if I had to choose a method to choose a method to use with Chebychev regression algorithm, I would choose the concavity method (c) because is the fastest one (takes just 1,38 seconds), and provides a very good approximation to the expected shape of the value function.