Homework 3: Quantitative Macro

Rosanna Gomar Lloret

October 29, 2020

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

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

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:

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

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

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

Set $\theta = 0.679$, $\beta = 0.988$, and $\delta = 0.013$. Also, to start with, set $h_t = 1$, hat 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. Report the time and number of iterations needed per variant and describe your results.

The interpretation of v(k) is the discounted lifetime utility of the representative agent from the current period onwards if the social planner is given capital stock k at the beginning of the current period and allocates consumption across time optimally for the household. This function v, the so-called value function) solves the following recursion:

$$V(k) = \max_{0\leq c,k'} U(c) + \beta V(k') \tag{5}$$

Subject to:

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

and:

$$k' = i + (1-\delta)k \tag{7}$$

From the previous equation, we isolate the investment:

$$k' - (1-\delta)k = i$$

And then, we can plug this expression in equation (6):

$$c + k' - (1-\delta)k = y$$

Rearranging $y = k^{1-\theta}$ since the labor is inelastically supply:

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

So we obtain that the consumption is equal to:

$$c = k^{1-\theta} + (1-\delta)k - k' \tag{8}$$

We can plug equation (8) in equation (5) and we get:

$$V(k) = \max_{k' \in [0, f(k)+(1-\delta)k]} U(k^{1-\theta} + (1-\delta)k - k') + \beta V(k') \tag{9}$$

Once we obtain this, we observe that the functional equation postulate that the discounted lifetime utility of the representative agent (there is perfect aggregation in the economy, so we can work with a single individual) is given by the utility that this agent receives today plus the discounted utility from tomorrow onwards.

The capital stock that the planner brings into the current period, result of past decisions, completely determines what allocations are feasible from today onwards. This is called a state variable, as this name suggest, summarizes the state of the economy. In addition to this, the control variable is k' since it is decided in the current period.

By solving the functional equation we mean finding a value function V and two optimal policy k'=g(k) and c=c(k) that describes the optimal k' as a function of k, for every possible k.

There are some steps that are common for all the settings, we have set the parameters (β, θ, δ, h=1), then the first step consists of defining a grid for the values of capital (the grid size is equal to 200 and I have keep this for all the parts of this exercise.), so it is important to obtain the steady state for capital for a better approximation. Let's calculate the capital in the steady state, to do so, we need two equations: the Euler Equation (this was obtained in the previous Homework, so I pick directly the results, for not repeating the process) and the marginal productivity of capital (this has changed since in this case we don't have a productivity shock and the labor is inelastically supplied):

$$f'(k) = (1-\theta)k^{-\theta} \tag{10}$$

$$f'(k) = \frac{1}{\beta} - 1 + \delta \tag{11}$$

Putting together equations (10) and (11):

$$\frac{1}{\beta} - 1 + \delta = (1-\theta)k^{-\theta} \Rightarrow 1 - \beta(1-\delta) = \beta(1-\theta)k^{-\theta} \Rightarrow k^{-\theta} = \frac{1-\beta(1-\delta)}{\beta(1-\theta)} \Rightarrow$$

We obtain the capital in steady state:

$$k^* = \left(\frac{1-\beta(1-\delta)}{\beta(1-\theta)}\right)^{\left(\frac{-1}{\theta}\right)} \tag{12}$$

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

We have solved with brute force iterations of the value function, following several steps, in first place we need to discretize by defining a grid for the values of k (uniformly distributed over the state space). The second step consist of guessing a solution, say the null vector, after that, we define the return matrix that takes into account the feasibility constraint :

$$0 \leq k' \leq f(k) + (1-\delta)k$$

So, the return matrix evaluates the utility for the agent at every possible combination of k_i and k_j if the previous condition holds.

- What happens if the feasibility constraint does not hold?

We know that there are some values for c and k' that are not feasible, so we need to replace that combinations. Computationally, if we focus on the return matrix, these are cells with NaN, so we need to replace this by a very negative number (for example, in the code is -1000).
The step 5 consist in computing the matrix χ, where:

$$\boxed{\chi_{i,j} = M_{i,j} + \beta V_j^s}$$

Then, we can carry out with the value function matrix iteration, since the updated value function consist of choosing the maximum element of each row of χ.

```
Vj[i]=np.max(Chi[i,:]) #updated value function, maximum element in each row of Chi
g[i] = np.argmax(Chi[i,:])
```

To conclude, the last step consist of looking at the difference between the updated value function and the previous one, if this difference is smaller than the tolerance error (for example, in the code is equal to 0.005), we have finished. If not, we have to go back to the previous step. In addition to this, we can compute the policy functions for capital and consumption, as I have explained before these are the controls variables of the economy, once we have the policy function for capital is easy to compute the consumption policy:

```
gk=np.empty(nk)
gc=np.empty(nk)
for i in range(nk):
    gk[i]= k_grid[int(g[i])]
    gc[i]=(k_grid[i]**(1-Theta))+(1-Delta)*k_grid[i]-gk[i]
```

As we can see, we pick the argmax element of the χ matrix for the capital, once we know the policy function for capital, we can compute the policy function for consumption.

The graphs for this part, concretely for the value function and the policy functions are figure 1[1], 2 and 3 in the Appendix. We need 430 iterations of the value function to reach converge and the time needed is 53.062 seconds. The results are consistent with the theory since the value function is increasing in capital, in addition to this, the curvature of the value function is higher for lower values of capital due to the diminishing returns of capital. There is a part when we compute the matrix χ that varies across the different parts of this exercise, I'm going to write the modification in the pdf, for the part a:

```
for i in range(nk):
    for j in range(nk):
        if k_grid[j]>=k_grid[int(g[i])]:
            Chi[i,j]=M[i,j]+Beta*Vi[j]
        else:
            continue
```

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

To do so, we use the known property that the optimal decision rule increases in k. Therefore, if $k_j > k_i$, then $g(k_j) \geq g(k_i)$. We go back to step 5 and we establish that we only want to comput the matrix χ for the pairs (k_i, k_j), that satisfy the lower bound, $k_j \geq g^s(k_i)$. Therefore, the following modification has to be done in Phyton:

[1] All the figures of the document have cross references

```
for i in range(nk):
     for j in range(nk):
         if k_grid[j]>=k_grid[int(g[i])]:  #MONOTONICITY
             Chi[i,j]=M[i,j]+Beta*Vi[j]
         else:
             continue
```

In this case, we need 430 iterations of the value function with monotonicity to reach convergence and the time needed is 41.086 seconds. So, we observe that we need less time in this method than in the previous one. The results are in figure 4, 5 and 6 in the Appendix, the graphs are identically to the previous ones.

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

We use the known property that the maximand in the Bellman equation is strictly concave in k', we have to return to step 5.1 in the VFI to improve the efficiency of the algorithm, if $\chi_{i,j-1} > \chi_{i,j}$ stop and report that the optimal value is $\chi_{i,j-1}$. Doing this step, we avoid to compute all the elements of the matrix.

```
for i in range(nk):
     for j in range(nk):
         Chi[i,j]=M[i,j]+Beta*Vi[j]
         if Chi[i,j]<Chi[i,j-1]:  #CONCAVITY
             break
```

We need 430 iterations of the value function with concavity to reach convergence and the time needed is 33.235 seconds. In this method, we need less time than in the previous parts. The results are in figure 7, 8 and 9 in the Appendix, same figures as before.

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

Here, we exploit the property of continuity of the optimal decision rule, we know that $k_j = g(k_i)$, then $g(k_{i+1})$ should be in a small neighborhood of k_j. I implement this change computationally:

```
 for i in range(nk):
     for j in range(nk):
          if (j >= g[i]) and (j <= g[i] + 5) :  #LOCAL SEARCH
             Chi[i,j]=M[i,j]+Beta*Vi[j]
```

We need 430 iterations of the value function to reach convergence and the time needed is 46.838 a bit faster than the part a but slower than the other methods that are used previously. The results are in figure 10, 11 and 12 in the Appendix.

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

Here we introduce this two modifications together, the one that I have calculated in part a and b, let's see this computationally:

```
  for i in range(nk):
     for j in range(nk):
         if k_grid[j]>=k_grid[int(g[i])]: #MONOTONICITY
             Chi[i,j] = M[i,j] + Beta*Vi[j]
             if Chi[i,j]<Chi[i,j-1]: #CONCAVITY
                 break
```

With this method we only need 8.585 seconds, so we can conclude that in this method we gain a lot of efficiency with respect to the others methods. The results are in figure 13, 14 and 15 in the Appendix.

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.

This method consists on iterating on the policy functions instead of on the value function. We need to do an initial guess on the policy function, and then use this to update the value function several times. The following step consist of updating the policy function, and if it has not converged you repeat the process. I have tried to get this but it does not work, you can see the attempt in the file of code.

g) Use policy iterations with 5, 10, 20 and 50 steps in between policy reassessments.

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

In this case, we add another choice variable to the previous model, the labor supply, if we focus on the maximization problem of the agent, the agent maximises his/her utility by choosing how much to work today and how much to save and not only how much to save as before. When we set the parameters, all are exactly the same as before but we need to add κ and ν, parameters that represents the disutility of labor for the agent.

I have defined a grid for capital and a grid for labor, but for simplifying the steps of the code, I have considered that they have the same size, in this case n evenly space points or in other words, the set density of the grid is equal for both. When the grid for capital is defined, as we have explained before, it is useful to know the level of steady state of the economy, whereas the labor takes values between 0 and 1. Following, we can compute all the parts A to E, as we have done in exercise 1.

The important modification here is that now when the feasible return matrix is computed, the feasibility constraint changes. This happens since now the production function not only depends on capital, the labor is a choice variable in this economy. In addition to this it also has an effect on the utility function (due to the feasibility constraint and to the disutility of labor). Another change with respect to exercise 1 is that now we have another policy function, for the new choice variable, labor. We have computed the different settings as in exercise 1, so I am not going to repeat and let's analyze directly the results:

- Brute force value function. The results for these setting can be seen in figure 16, 17, 18 and 19. In this case we need 514 iterations of the value function to reach convergence and the time needed is 59.634, we need more iterations and more time than in exercise 1 part a. The facts that are remarkable are: brute force value function has negative values now (the dimension has changed), the policy function for capital is not a straight line and the policy function for consumption is convex.

- Monotonicity. The results for these setting can be seen in figure 20, 21, 22 and 23. The brute force value function is the same but now the policy function change with respect to the previous setting, this is an interesting fact, in exercise 1, we obtain the same result for all settings and it only changes the execution time. Now, we need different iterations, different time and we obtain different results in the policy functions. In this case we need 804 iterations of the value function with monotonicity to reach convergence and 75.15 seconds, more iterations and more time that in setting a of this exercise.

- Concavity. The results for these setting can be seen in figure 24, 25, 26 and 27. We need 456 iterations of the value function with concavity to reach convergence and the time needed is 11.564 seconds. We need less time and less iterations than in the two previous methods to

get the convergence. The slope of the value function has change a little bit for values of capital between 0 and 10, the policies functions have changed as well.

- Local search.The results for these setting can be seen in figure 28, 29, 30 and 31. We need 947 iterations of the value function with local search to reach convergence and the time needed is 116.209 seconds, this is not what we should expect since we need more time and more iterations than in part a, and in thhis part of exercise 1 we obtain the same number of iterations but less time. The value function looks better than for concavity, and the policy function are different from the previous settings.

- Monotonicity and concavity.The results for these setting can be seen in figure 32, 33, 34 and 35. We need 1794 iterations of the value function with monotonicity and concavity to reach convergence and the time needed is 28.038 seconds. The result is very close to the first setting of this exercise, the policy functions are so close and also the value function. This is the first time that we observe a trade off between the number of iterations and the time needed, if we compare with a, more iterations but less time. In the case where we have only concavity, it is fast but the form of the value function is a bit different as it has been commented, so in this case, we need more time but it is a better approximation to what we should expect.

To conclude, in each setting of this exercise we need more iterations and more time than in exercise 1, this is due to the fact of introducing the labor. Another interesting fact is, that now, we observe how policies functions vary across the different settings, and also the slope of a segment of the value function when we do concavity.

We have analyzed the number of iterations and time of each method, the method that we need less time in this exercise is when we study concavity. In contrast to exercise 1, that it was the case of monotonicity and concavity, and in all methods we have the same number of iterations.

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

The value function doing Chebyshev can be seen in figure 36, its very fast since we need only 3 seconds and 100 iterations to reach convergence. It has been very important the Notes of Makoto Nakajima. We know the properties of Chebyshev Polynomials: the range is between -1 and 1, symmetry, the expression for the roots, and so on. In general, a Chebyshev polynomial of order i can be evaluated following a algorithm: we specify the order of the polynomial, then we compute the order since the order can be computed using the values of Chebyshev polynomials of one previous order and two previous order. Then we need to approximate the function, we have done some transformation because we are interested in the grid of capital and not only in the domain (-1,1). Then, in the notes of Nakoto we can see the Algorithm 2 that helps us to solve the exercise. The remarkable facts when we do Chebyshev regression algorithm are:

- The coefficients associated with Chebyshev polynomials are strictly decreasing as the order increases. Looking at how the coefficients decrease helps determining the order of Chebyshev polynomials n used for approximation.

- As long as the number of collocation points m is fixed, reducing n does not affect the value of coefficients. This property is called Chebyshev economization.

- Chebyshev polynomial is not always the best one, especially when we are approximating a non-smooth function. For example, it is known that the performance of the Chebyshev approximation is not great if the original function has a kink.

List of Figures

1 Appendix

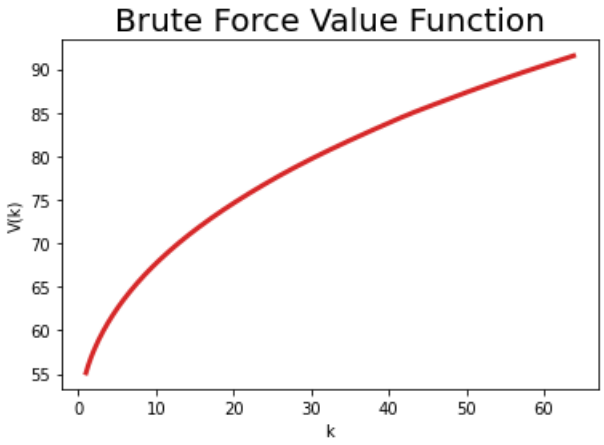

Figure 1: Brute force value function part a

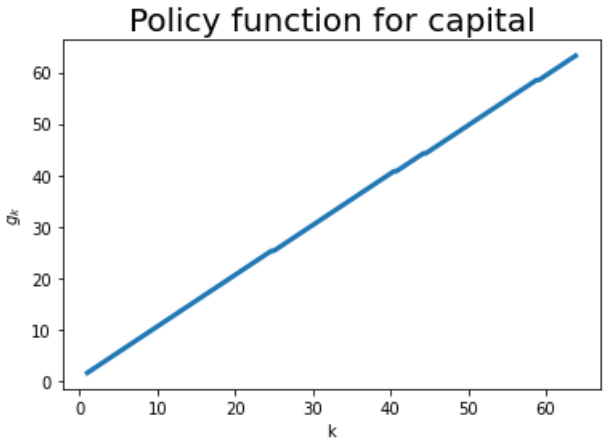

Figure 2: Policy function for capital part a

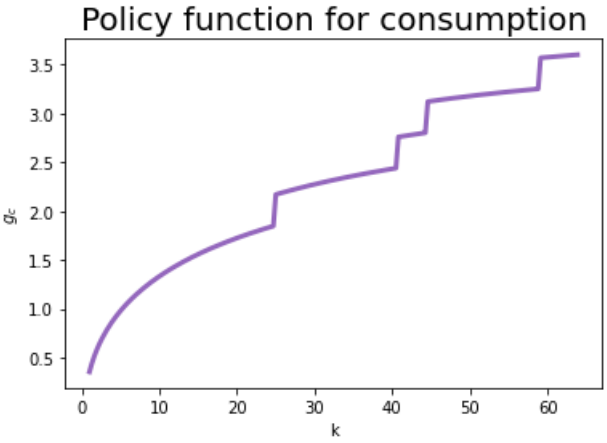

Figure 3: Policy function for consumption part a

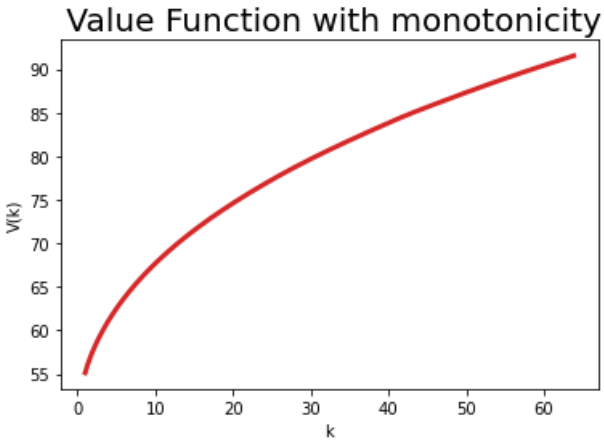

Figure 4: Value function with monotonicity part a

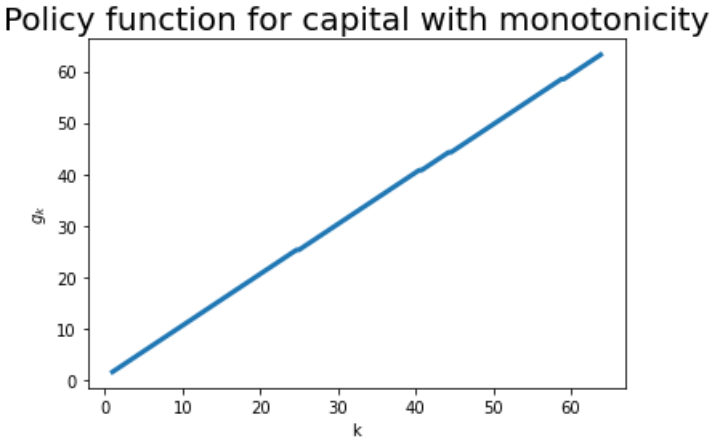

Figure 5: Policy function for capital with monotonicity part a

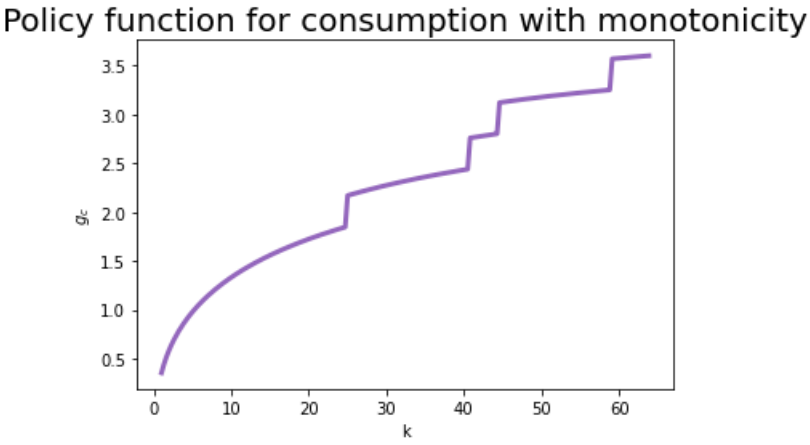

Figure 6: Policy function for consumption with monotonicity part a

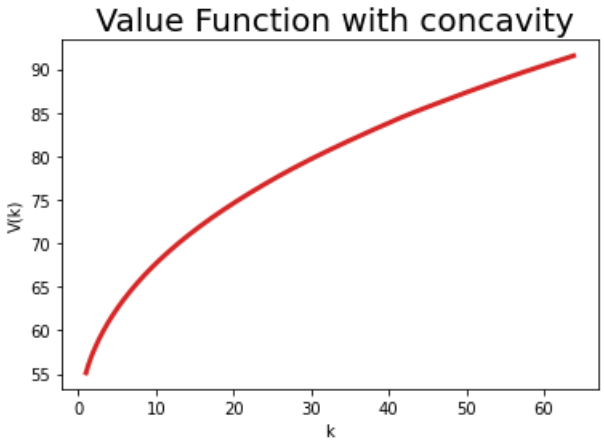

Figure 7: Value Function with concavity part a

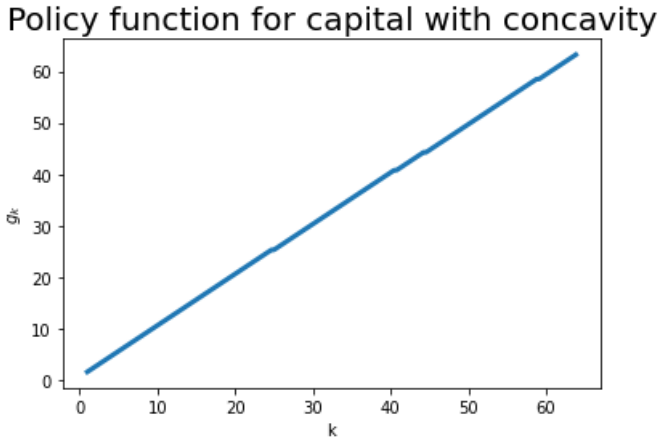

Figure 8: Policy function for capital with concavity part a

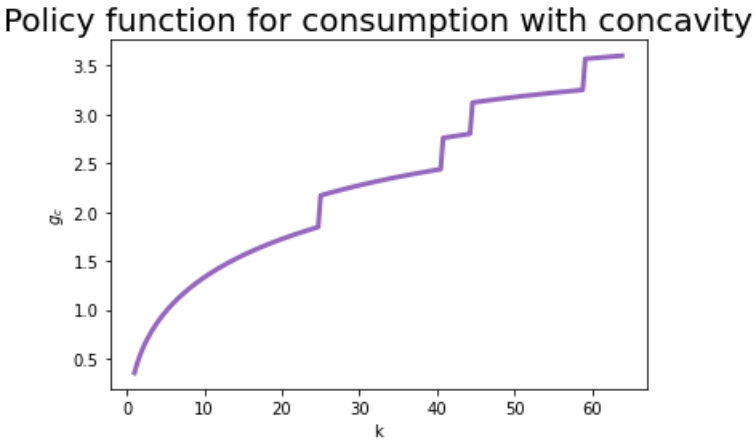

Figure 9: Policy function for consumption with concavity part a

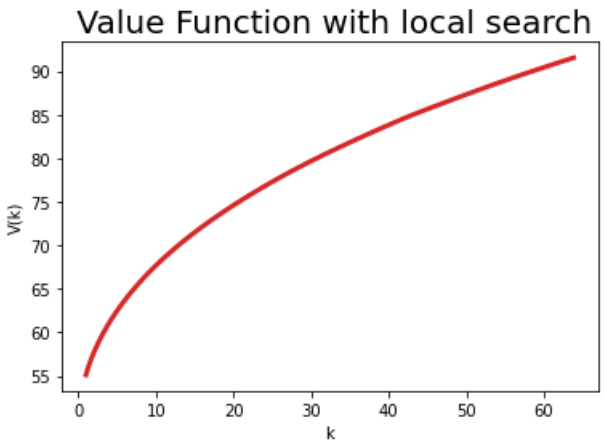

Figure 10: Value Function with local search part a

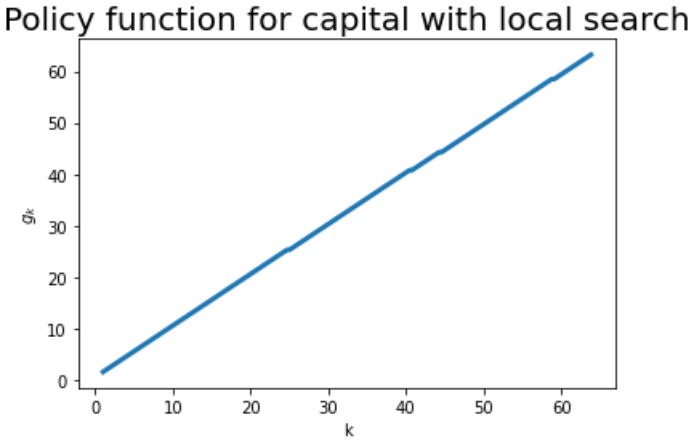

Figure 11: Policy function for capital with local search part a

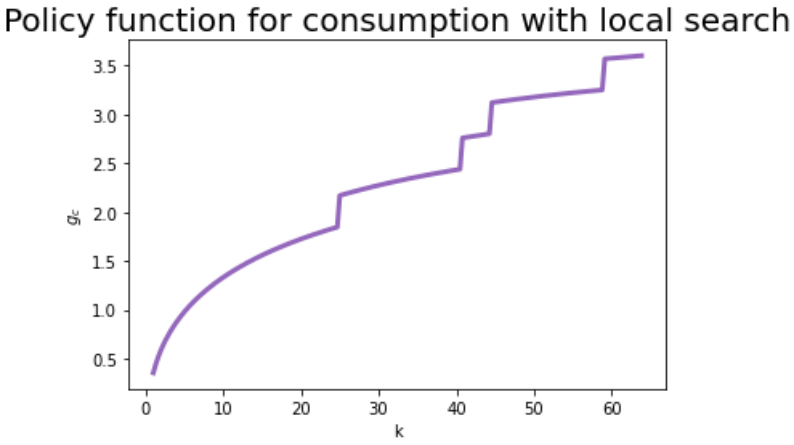

Figure 12: Policy function for consumption with local search part a

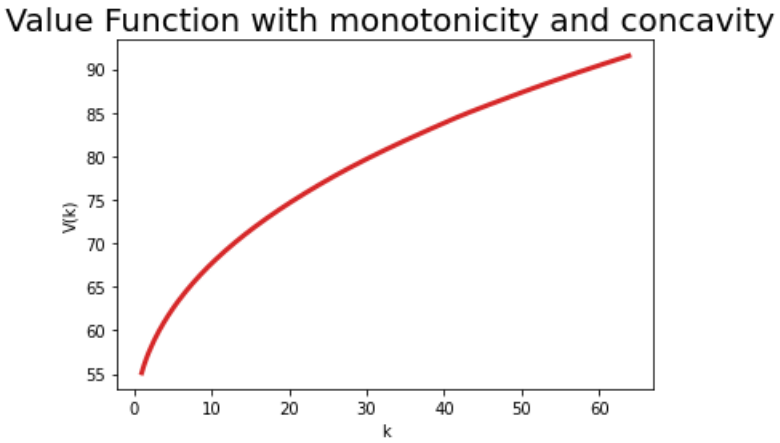

Figure 13: Value Function with monotonicity and concavity part a

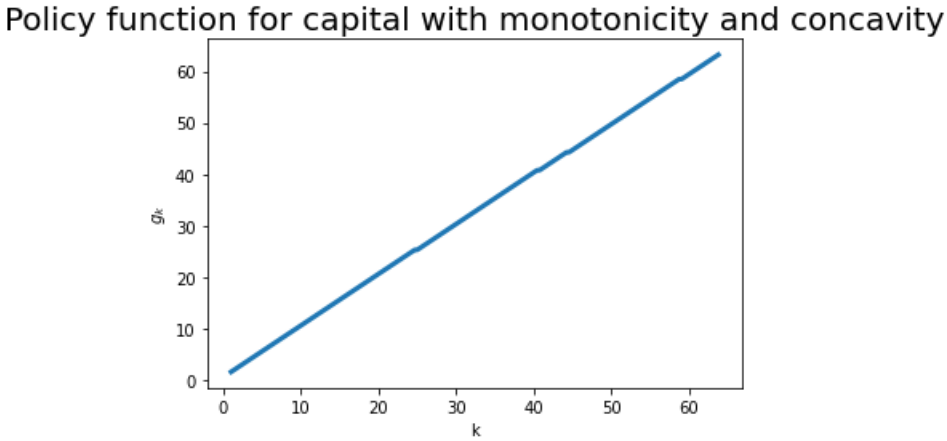

Figure 14: Policy function for capital with monotonicity and concavity part a

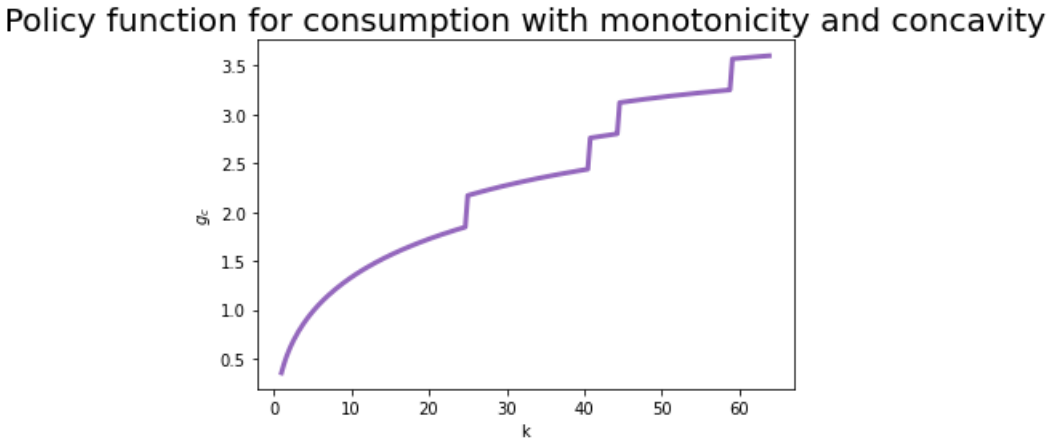

Figure 15: Policy function for consumption with monotonicity and concavity part a

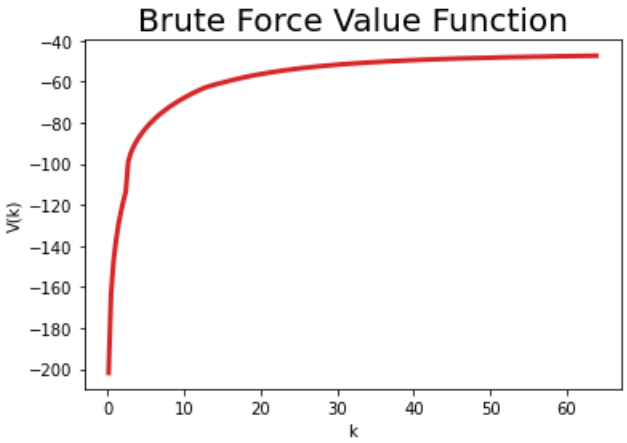

Figure 16: Brute force value function part b

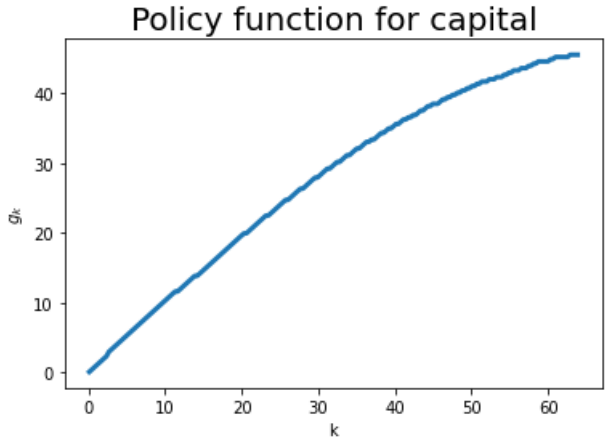

Figure 17: Policy function for capital part b

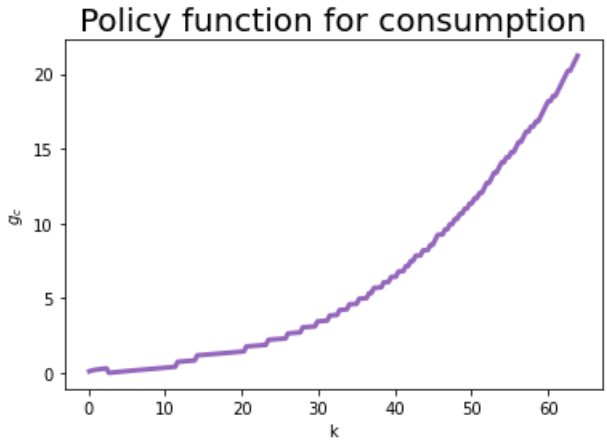

Figure 18: Policy function for consumption part b

Figure 19: Policy function for labor part b

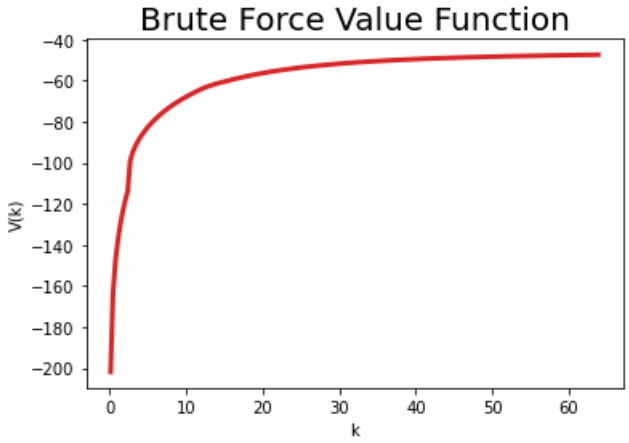

Figure 20: Value function with monotonicity part b

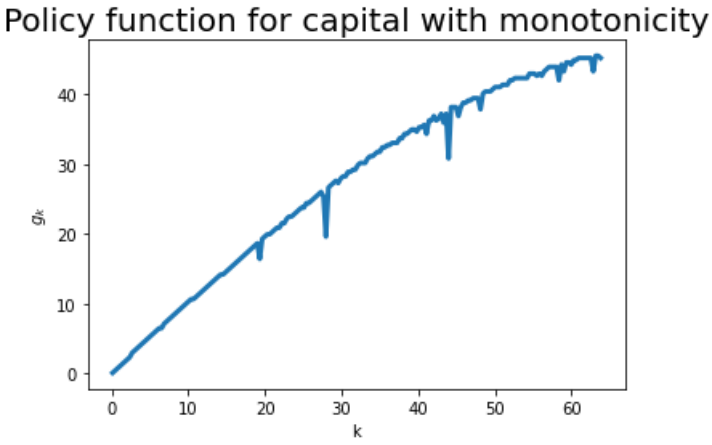

Figure 21: Policy function for capital with monotonicity part b

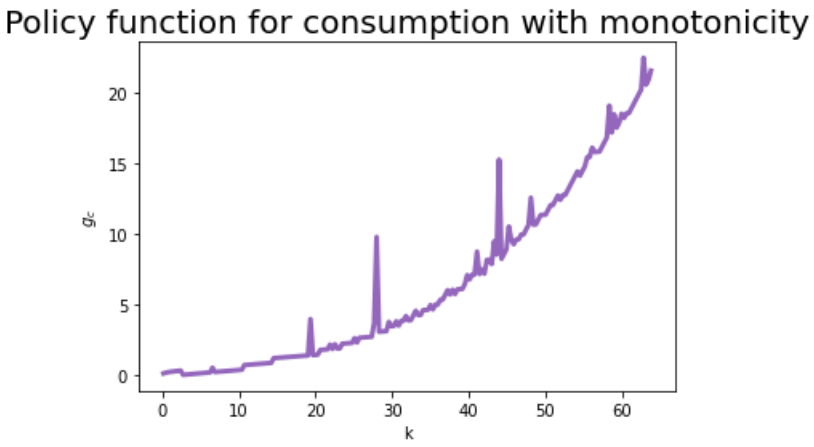

Figure 22: Policy function for consumption with monotonicity part b

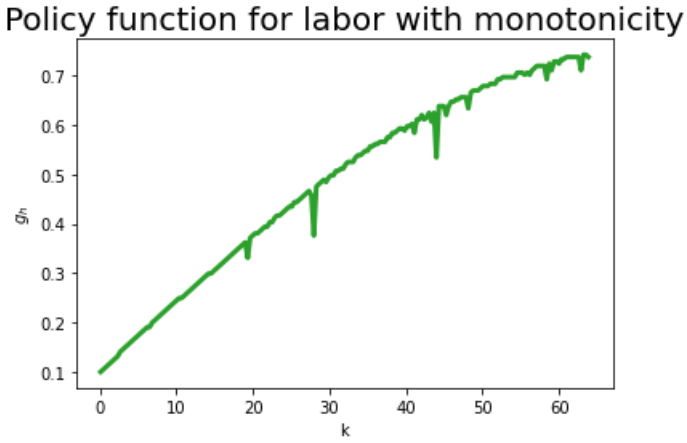

Figure 23: Policy function for labor with monotonicity part b

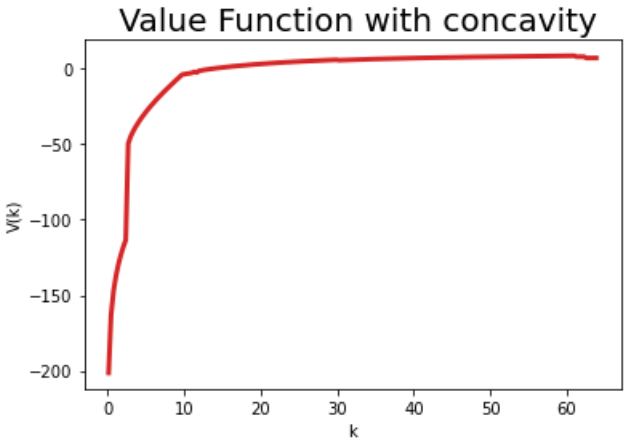

Figure 24: Value Function with concavity part b

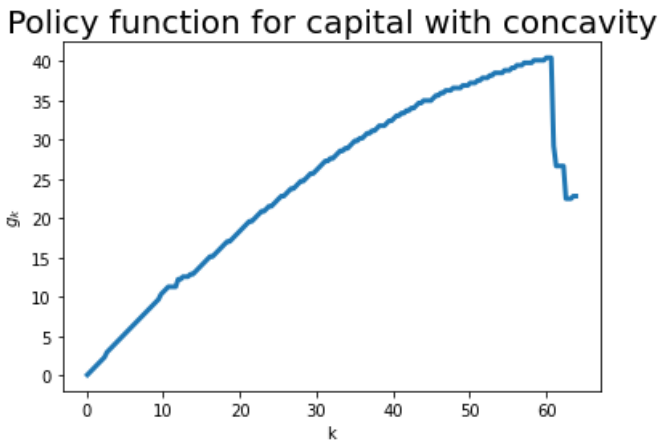

Figure 25: Policy function for capital with concavity part b

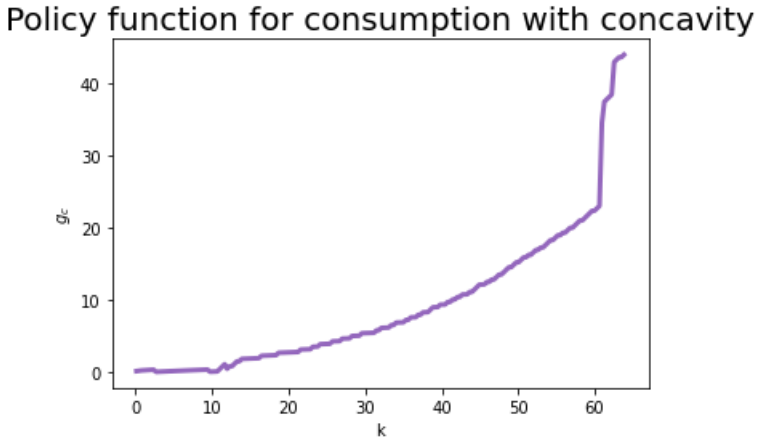

Figure 26: Policy function for consumption with concavity part b

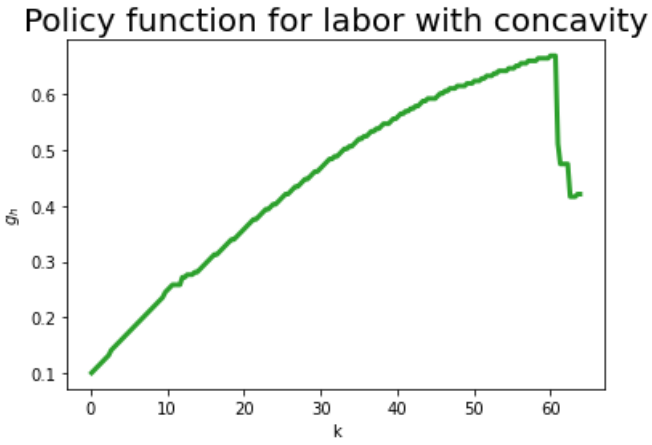

Figure 27: Policy function for labor with concavity part b

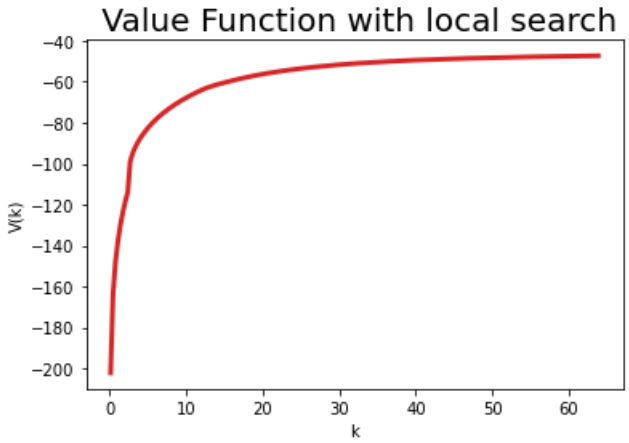

Figure 28: Value Function with local search part b

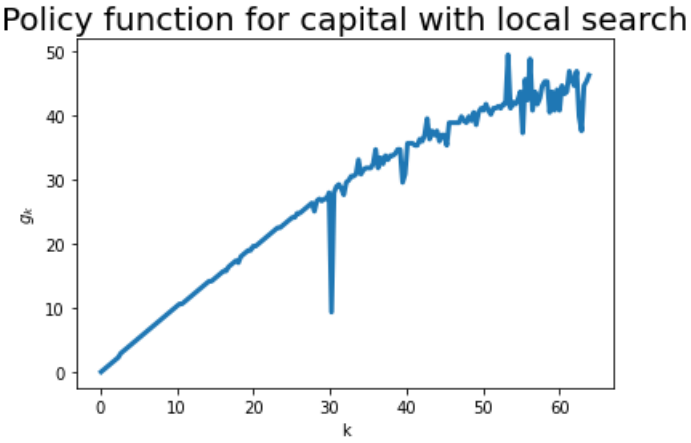

Figure 29: Policy function for capital with local search part b

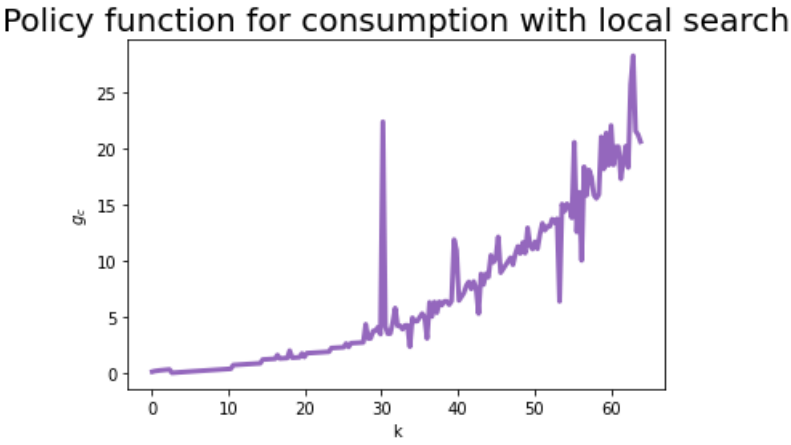

Figure 30: Policy function for consumption with local search part b

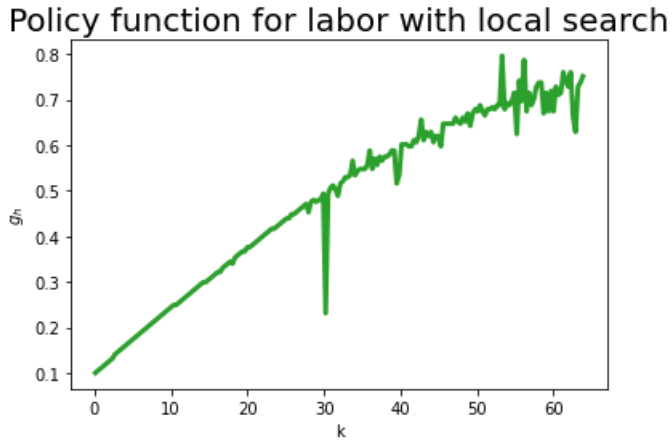

Figure 31: Policy function for labor with local search part b

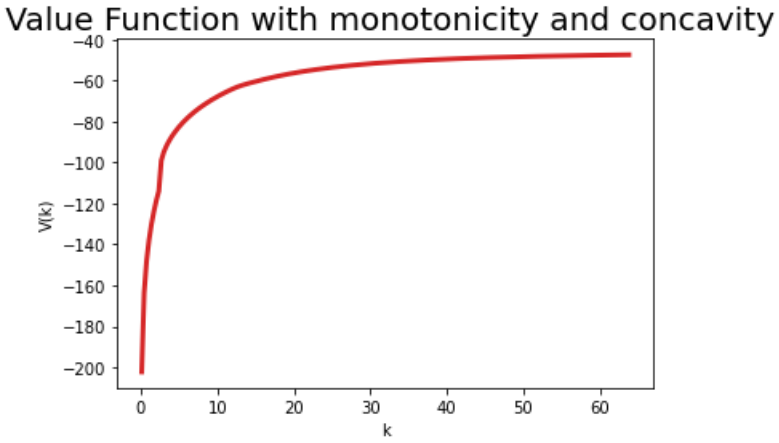

Figure 32: Value Function with concavity and monotonicity part b

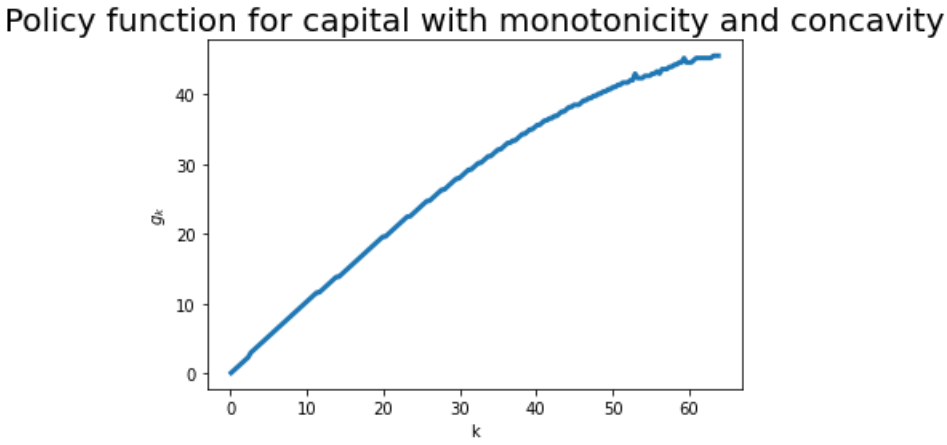

Figure 33: Policy function for capital with concavity and monotonicity part b

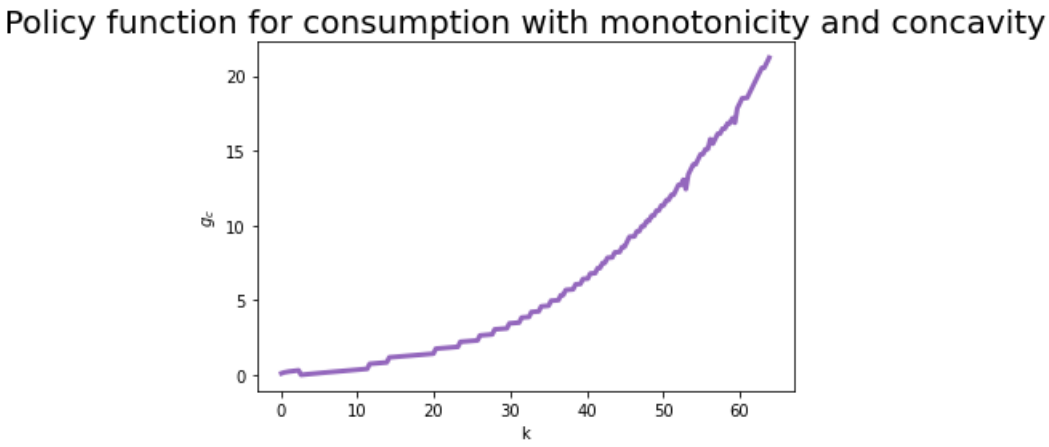

Figure 34: Policy function for consumption with concavity and monotonicity part b

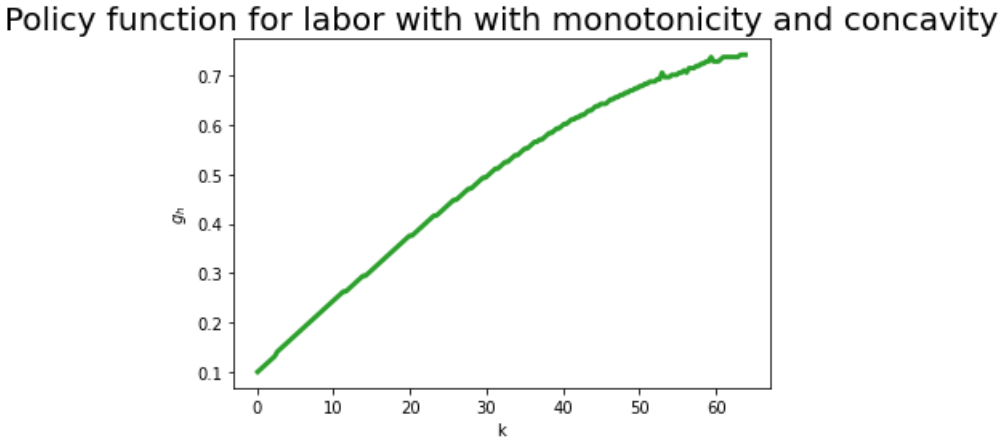

Figure 35: Policy function for labor with concavity and monotonicity part b

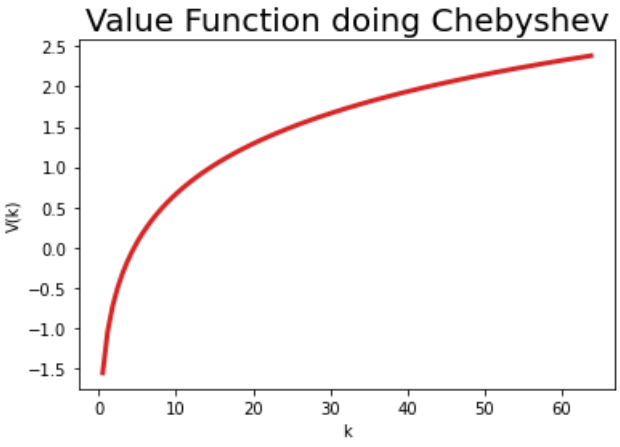

Figure 36: Value function doing Chebyshev