Homework 3
Quantitative Macroeconomics

Victor Caballero

*With collaboration of Santiago Albarracin and Lorenzo de Carolis

October 29, 2020

Question 1. Value Function Iteration

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

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

Over consumption and leisure $u(c_t, 1 - h_t) = lnc_t - \kappa \frac{h^{1+\frac{1}{v}}}{1+\frac{1}{v}}$, 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 θ=0.679; β=0.988; δ=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 computational variants listed below. In all these variants use the same initial guess for your value function.

In order to solve the model, the first step is to find the equation that explains kss (steady state). For accomplishing this we want to maximize the following Lagrangian:

$$L = E_0 \left[\sum_{t=0}^{\infty} \beta^t \left(ln(c_t) - \gamma \frac{h^{1+\frac{1}{v}}}{1+\frac{1}{v}} \right) \right] + \sum_{t=0}^{\infty} \lambda_t [k_t^{1-\theta} h_t^{\theta} + (1-\delta)k_t - k_{t+1} - c_t] \tag{5}$$

From which we obtain the following First Order Conditions (FOC), deriving over c_t and k_{t+1} respectively:

$$\beta^t \frac{1}{c_t} - \lambda_t = 0 \tag{6}$$

$$-\lambda_t + \lambda_{t+1}[(1-\theta)k_{t+1}^{-\theta} h_{t+1}^{\theta} + (1-\delta)] = 0 \tag{7}$$

By equalizing $\frac{\lambda_t}{\lambda_{t+1}}$ from both previous equations we obtain the following result:

$$\frac{1}{\beta} \frac{c_{t+1}}{c_t} = (1-\theta) h_{t+1}^{\theta} k_{t+1}^{-\theta} (1-\delta) \tag{8}$$

Equation in which to obtain kss, we normalize $k_{t+1} = k_t$ and $c_{t+1} = c_t$, and like the question statement says we set $h_t = h_{t+1} = 1$, with what we obtain:

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

From what we clear k and obtain:

$$kss = \left[\frac{\left(\frac{1}{\beta} + \delta - 1 \right)}{(1-\theta)} \right]^{-\frac{1}{\theta}} \tag{10}$$

As requested, we pose the recursive formulation of the sequential problem without productivity stocks:

$$c = k^{1-\theta} - (1-\delta)k - k' \tag{11}$$

Then, for solving the model we will follow the next steps explained in lectures:

1. We define a grid for the values of k. The grid goes from 1 to 2 times kss, divided in in 200 equal parts.

2. We define return matrix, having nk rows and columns. In each position i, j of the matrix (M) we place $u(c)$ taking into consideration the equation 11, in which if $c > 0$ then:
$$kj \leq k_i^{1-\theta} + (1+\delta)k_i \tag{12}$$
While, if c is not feasible ($c \leq 0$, there we place $M[i,j] = -1000000$ to make the computation faster.

3. We guess the solution for the value function being a null vector at $s = 0$, We set the $\epsilon = 0.01$ being enough in order to report a success in finding the value function. For the maximum value we set $s = 400$.

4. After completing the previous steps, now we can continue with the process of the Value Function Iteration (VFI):
   (a) Compute the matrix X_{ij} being a $nk * nk$ matrix, having $M_{ij} * \beta V_i^s$ in position ij.
   (b) Compute the updated value function V_j^{s+1} being the maximum element in each row of X_{ij}.
   (c) If $\|V^s + 1_j - V_i^s\| > \epsilon$, then update V with the new one and we redo it since the first step of VFI. On the contrary if $\|V^s + 1_j - V_i^s\| < \epsilon$, then we have found the convergence.
   (d) Compute the decision rule once we have found the best V

5. Afterwards we obtain the policy functions g_k and g_c, which are respectively for capital and consumption.

6. Finally, we conclude the time calculation, we present the total quantity of iterations and we plot the value functions.

These steps are going to be repeated for all the following in subsections, changing the specifications of estimation taking into consideration different statements of the value function iteration for the next items.

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

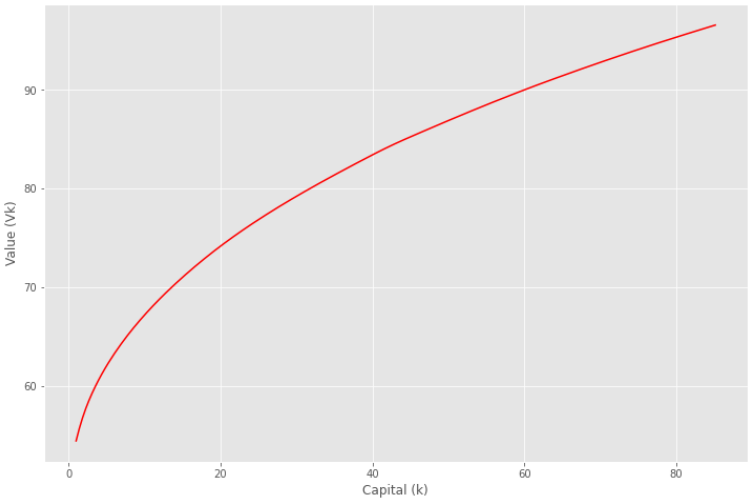

Figure 1: Value Function - Brute force iterations

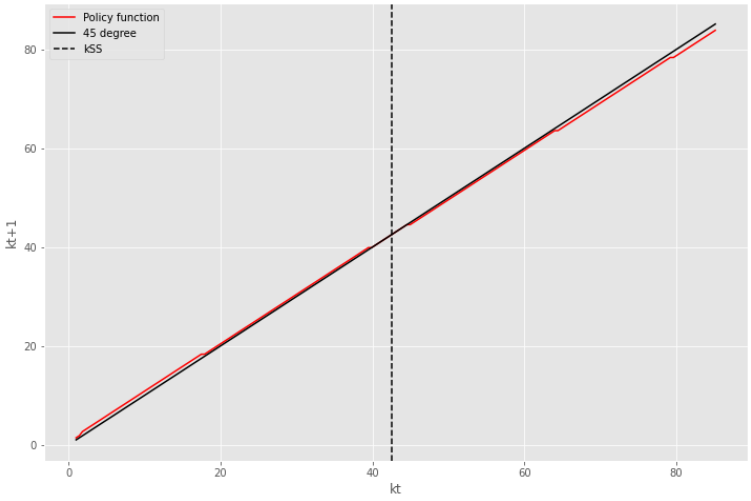

Figure 2: Policy Function - Brute force iterations

Time elapse: 40.8583459854126

Number of iterations: 371

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

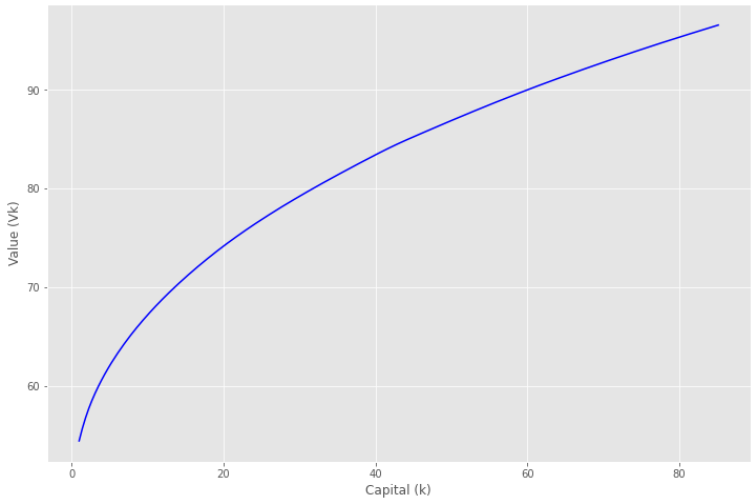

Figure 3: Value Function - Monotonicity of the optimal decision rule

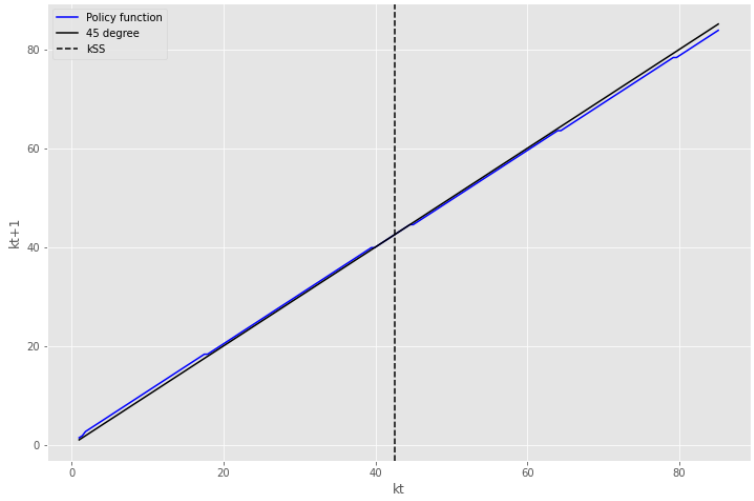

Figure 4: Policy Function - Monotonicity of the optimal decision rule

Time elapse: 29.69220209121704

Number of iterations: 371

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

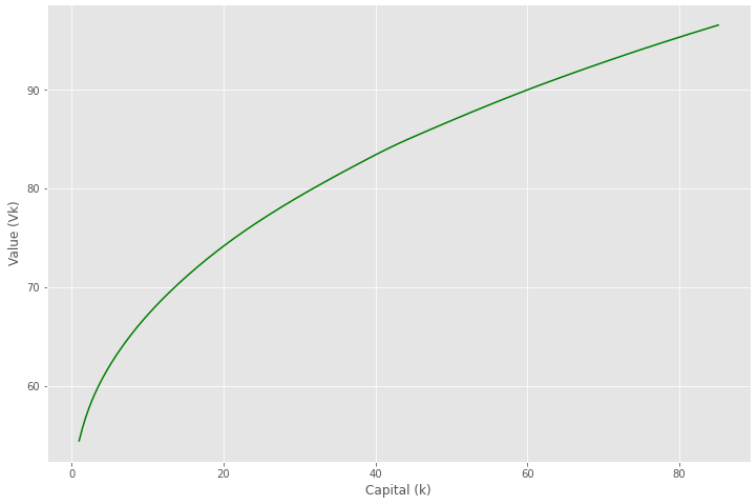

Figure 5: Value Function - Concavity of the value function

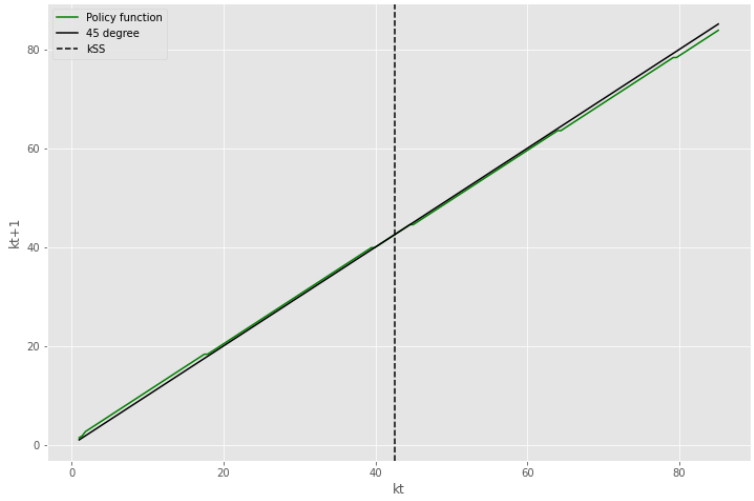

Figure 6: Policy Function - Concavity of the value function

Time elapse: 23.921939849853516

Number of iterations: 371

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

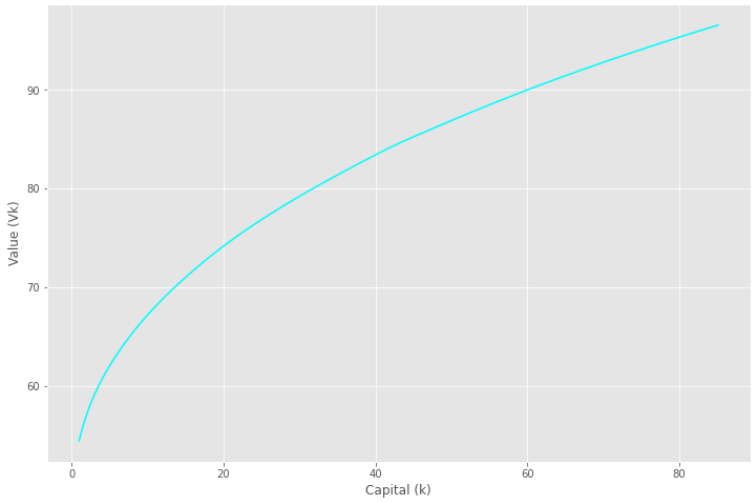

Figure 7: Value Function - Local search on the decision rule

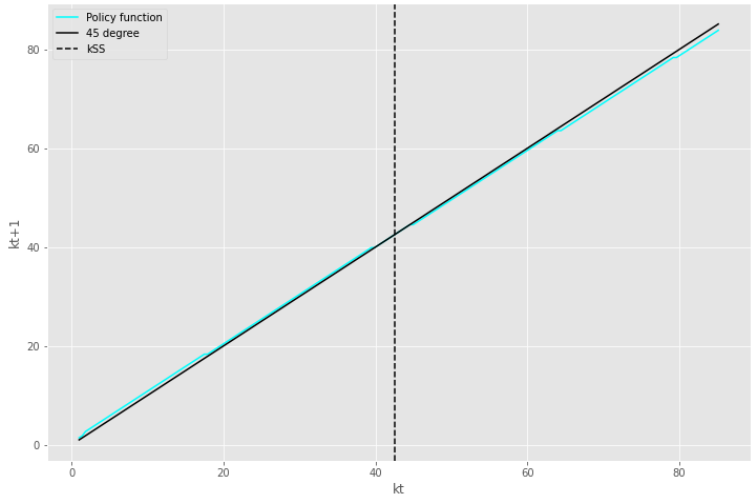

Figure 8: Policy Function - Local search on the decision rule

Time elapse: 36.85394477844238

Number of iterations: 372

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

* We cannot explicate why there appears a change at the ending right tale of the following figures, maybe it's a computational mistake at the ending point

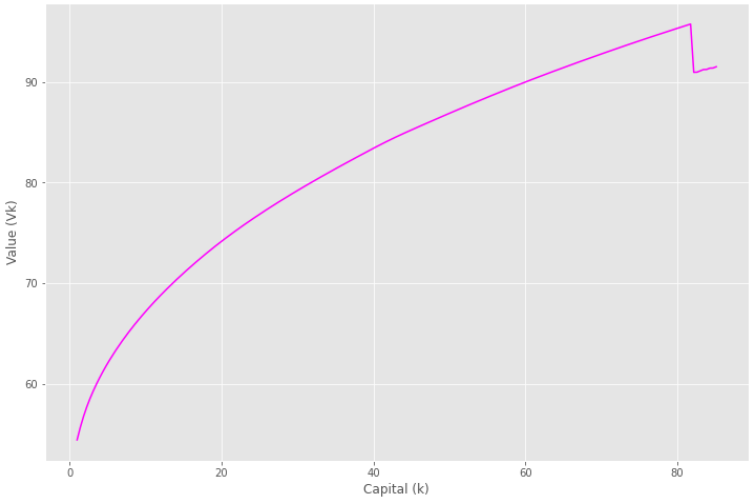

Figure 9: Value Function - Concavity of the value function and monotonicity of the decision rule

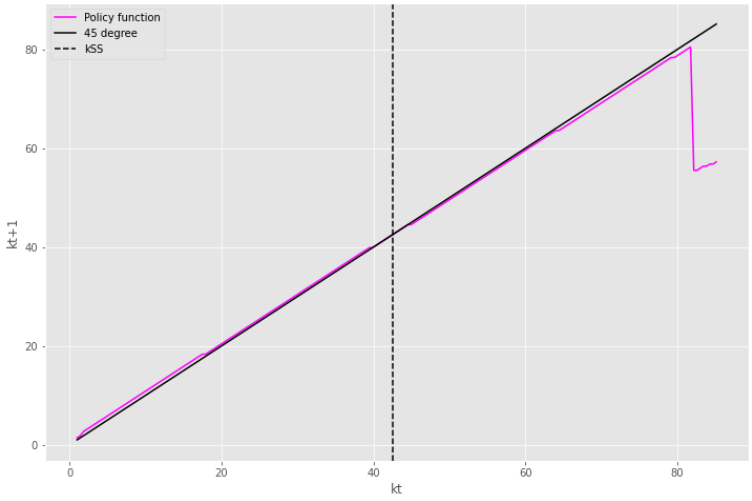

Figure 10: Policy Function - Concavity of the value function and monotonicity of the decision rule

Time elapse: 17.671236991882324

Number of iterations: 371

2. Redo item 1 adding a labor choice that is continuous. For this, set κ= 5:24 and ν= 20

Then, for solving the model we will base our solution in the one from the exercise 1 with the corresponding computation changes following the next steps:

1. We define a new function for solving the maximization of the model taking into consideration c and u_i
2. We define a grid for the values of k. This time we reduced the values taken for the grid so that we can reduce the time of iteration and make it more efficient once we know hos it works. The the grid goes from 0.01 to 2, divided in in 100 equal parts.
3. We define all our zero vectors using the *np.zeros* python function.
4. We define ours return matrices, having nk rows and columns. In each position i, j of the matrix (Chi) we place the values of M and the Value function, value function that we updated as the maximum element in each row.
5. We set the $\epsilon = 0.01$ being the threshhold to stop the itertions once the values are lower, then we can report a success in finding the value function.
6. Finally, we conclude the time calculation, we present the total quantity of iterations and we plot the value functions.

These steps are going to be repeated for all the following in subsections, changing the specifications of estimation taking into consideration different statements of the value function iteration with continuous labor choice . In this question we will only provide the VF figures due that the policy functions are not that relevant at the moment of analysis.

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

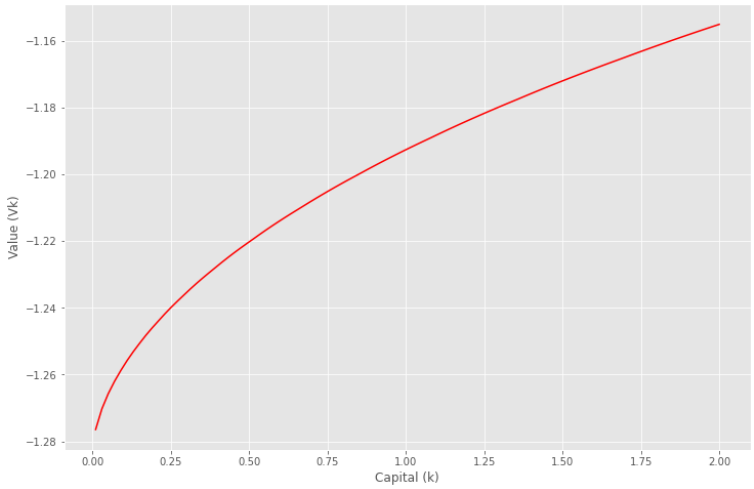

Figure 11: Value Function - Brute force iterations

Time elapse: 5.837031126022339

Number of iterations: 99

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

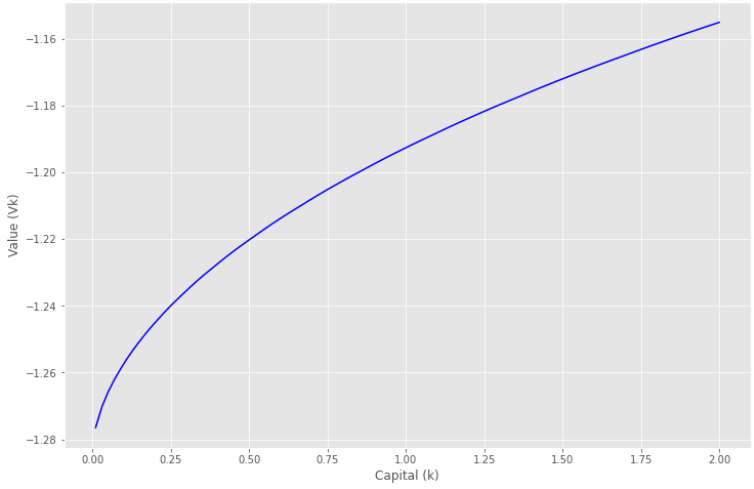

Figure 12: Value Function - Monotonicity of the optimal decision rule

Time elapse: 7.264854907989502

Number of iterations: 99

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

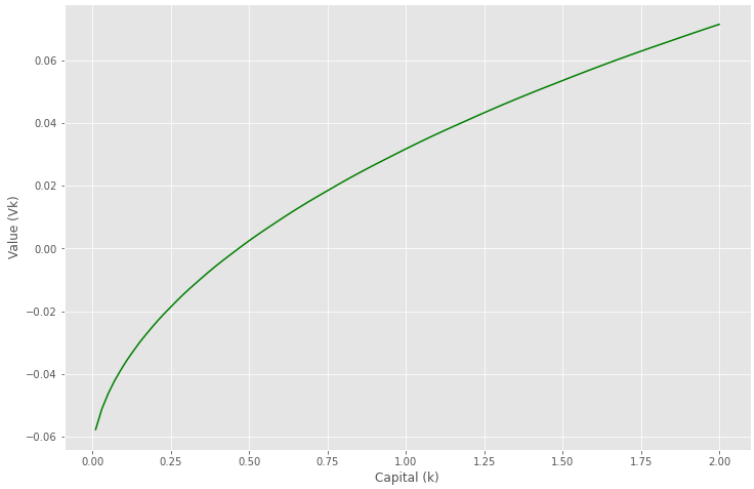

Figure 13: Value Function - Concavity of the value function

Time elapse: 4.915533065795898

Number of iterations: 101

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

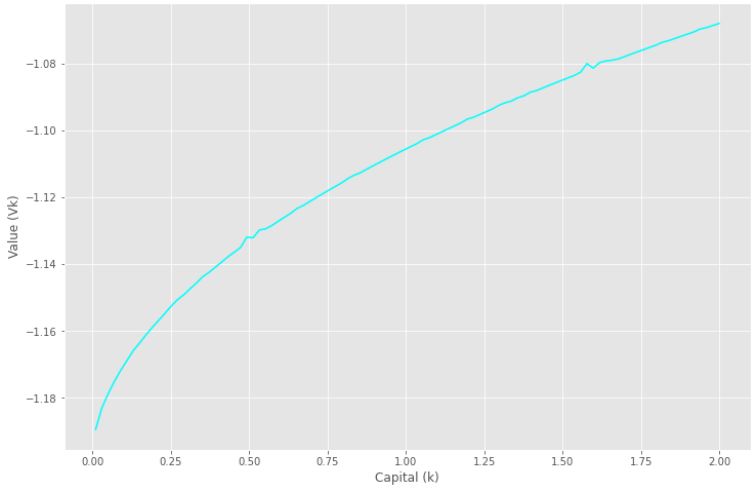

Figure 14: Value Function - Local search on the decision rule

Time elapse: 5.081228971481323

Number of iterations: 101

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

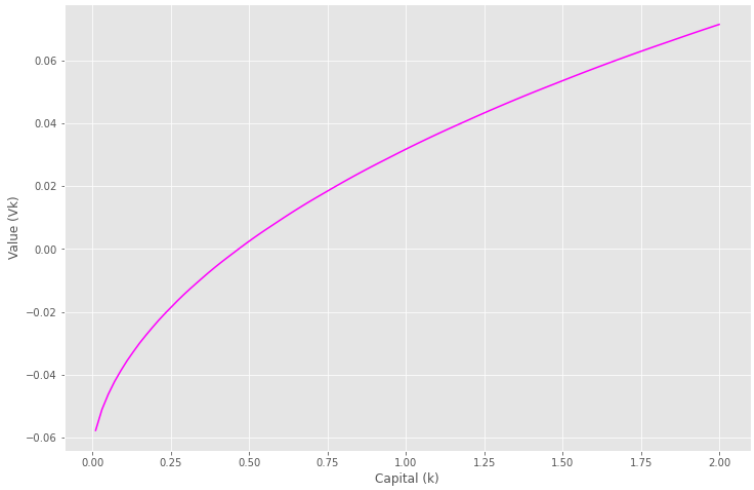

Figure 15: Value Function - Concavity of the value function and monotonicity of the decision rule

Time elapse: 4.672188997268677

Number of iterations: 101

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

For the solution of the VFI with the Chebyshev regression algorithm we started with the same programming procedure that from the previous exercises, after which we tried to use the lecture notes and references from "Quantitative Economics with Python" of Sargent, but we couldn't program anything after the nodes definition because of the quantity of error that we encountered. Our programming skill we weren't enough to make all the computational analysis and we couldn't finish the procedure for the established due date.

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

```
import numpy as np
import matplotlib.pyplot as plt
import quantecon as qe
import math

# ============================================================
# EXERCISE 3: Chebyshev
# ============================================================

# parameters
theta = 0.679
beta = 0.988
delta = 0.013
h = 1
kappa = 5.24
nu = 2.0

kSS = ((1/beta-1+delta)/(1-theta))**(-1/theta)

print("The steady state is "+str(kSS))

qe.tic()
nk = 200
k = np.linspace(1,2*kSS,nk)

V0 = np.zeros(nk)

def f(k,h):
    return k**(1-theta)*h**theta

def u(c,h):
    return math.log(c) - kappa*h**(1+1/nu)/(1+1/nu)

def re_M(k1,k2):
    c = f(k1,h) + (1-delta)*k1 - k2
    if c>0:
        return u(c,h)
```

```
    else:
        return -1000000

M = np.empty([nk,nk])
i=0
while i<=nk-1:
    j=0
    while j<=nk-1:
        M[i,j] = re_M(k[i],k[j])
        j = j+1
    i = i+1

# Chebyshev

def cheb_nodes(x,a,b):
    k = []
    z = []
    for j in range(1,x+1):
        z_k=-np.cos(np.pi*(2*j-1)/(2*x))
        k_ch=(z_k+1)*((b-a)/2)+a
        z.append(z_k)
        k.append(k_ch)
    return np.array(z), np.array(k)

#
#
# We tried different thing but nothing from this point on worked.
# We couldn't complete Question 3
#
#

T = qe.toc()

plt.plot(k, V, label='V(k)', color="red")
plt.xlabel('Capital (k)')
plt.ylabel('Value (Vk)')
plt.show()

print("Time elapse in seconds: " + str(T))
print("Number of iterations: " + str(j))
```

Computer specifications

- MacBook Pro (15-inch)
- Processor: 2.6 GHz Quad-Core Intel Core i7
- Memory: 16 GB 2133 MHz LPDDR3