Homework4 - Macroeconomics

Lillian Haas

October 2019

1 Exercise 1

1.1 Bruce force iteration

1. Set the grid $k_{min} > 0$ and $k_{max} >= steadystate$
2. Guess a Indirect Utility Function V.
3. Check if consumption is positive. Along the definition condition:

   $y(k_t) + (1 - \delta) * k_t) > k_{t+1}$. For all k's this condition does not hold, I put a very negative value -10000.
4. Computing the X[k_i,k_j] values for all k's the positive consumption holds.
5. Value Function V^1 is provided by selecting the maximising value at position k_j for each row k_i.
6. Checking if the reached Value Function presents the optimal path by comparing the new Value Function with the previous one.

For the bruce force value function iteration, I created the m-matrix, indicating all capital investment options for the future, outside of the loop. It doesn't change throughout the iteration process.

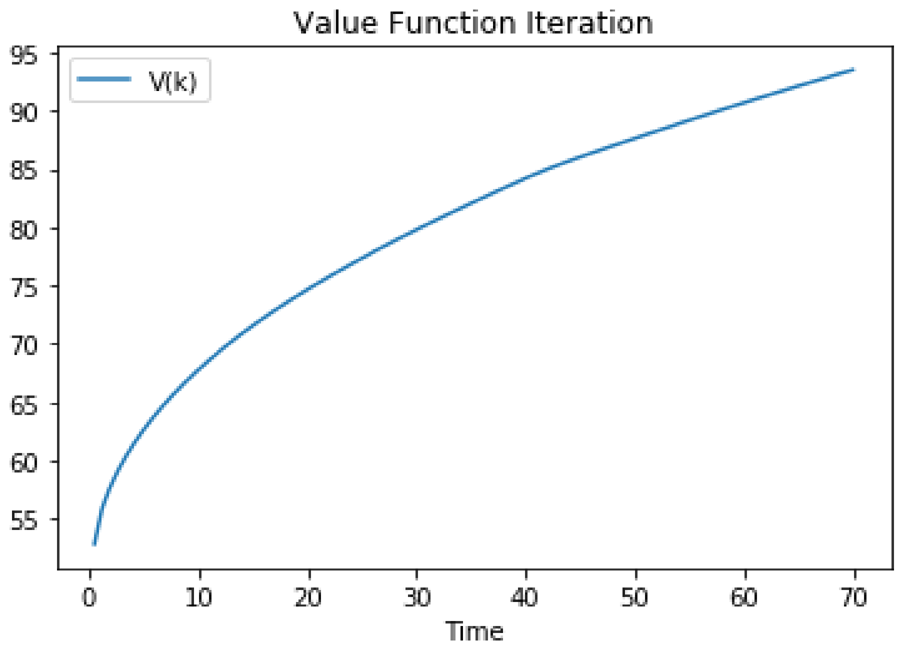
In the while loop, I compute the Chi-matrix, V-next and respective policy functions.

You will find two codes in python to solve for this exercise. The first one is the code along the slides.

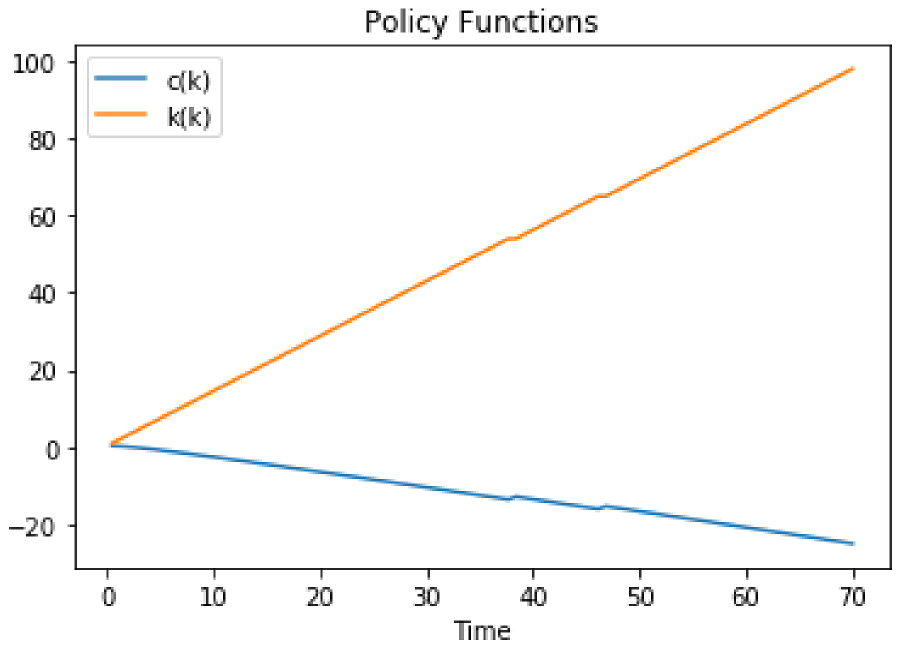
Total time taken this loop: 14 seconds. Total number of iterations: 616

The second code uses the QuantEcon Template. Total time taken this loop: 1.19 minute. Total number of iterations: 698.

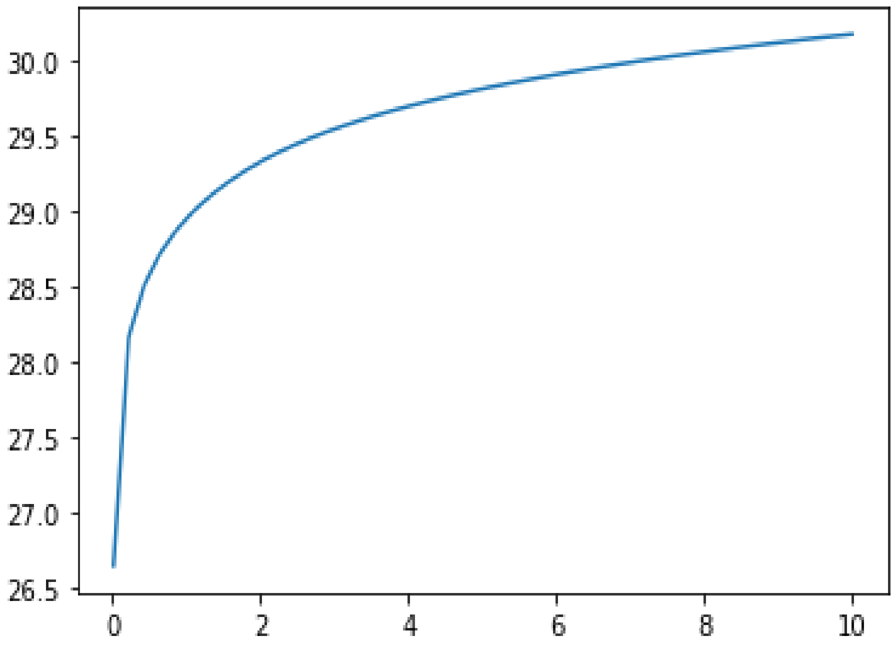
Title: Value Function $V(x)$ force.png force.png



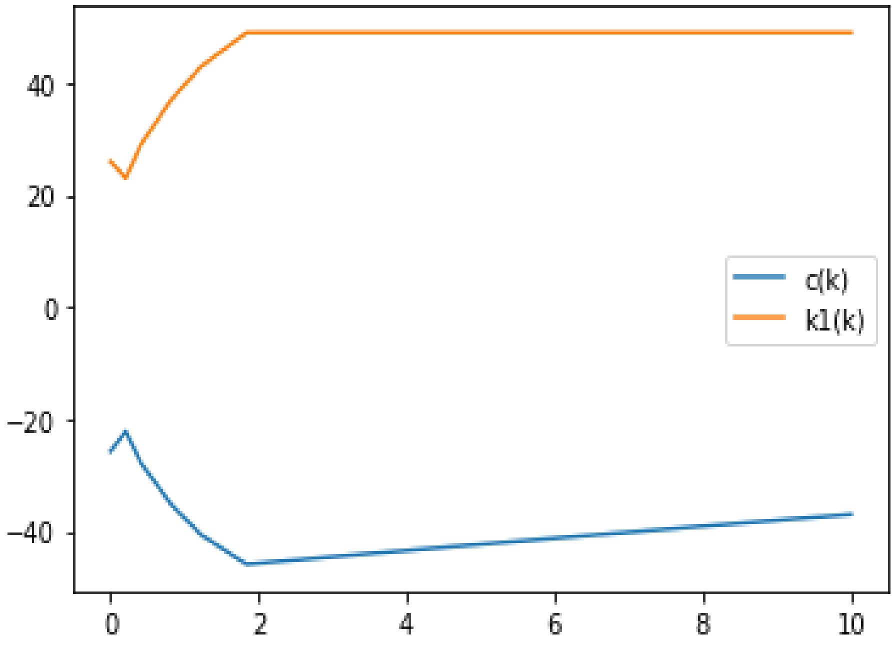
Title: Policy functions
Functions.png Functions.png



Title: Value Function $V(x)$ by QuantEcon Package



Title: Policy functions by QuantEcon Package



1.2 Monotonicity

This section follows the mentioned steps above up until step 4

5 For each X[k_i,k_j] where j¡i, we know by monotonicity that respective policy function will reveal smaller values: $g(k_i) <= g(k_j)$ if i¡j.

6 Value Function V^1 is provided by selecting the maximising value at position k_j for each row k_i. This step is supposed to run faster as it considers a matrix with reduced amount of entries. I replace every value in the m-matrix with 0 for i¡j. This results in a m-matrix, where the left-hand side is empty/zeros.

7 The remaining iteration process does not change. This step reduces entries of m-matrix, consequently the Chi-matrix.

Total time taken this loop: 44 seconds. Total number of iterations: 616

1.3 Concavity

This section follows the mentioned steps above up until step 5

6 Value Function V^1 is provided by selecting the maximising value at position k_j for each row k_i. Because the capital function is strictly concave in k_j, we know that any entry in the X-matrix for $m(i,j) + \beta * V_j^s$¿$m(i,j+1) + \beta * V_{j+1}^s$ than $m(i,j+1) + \beta * V_{j+1}^s$¿$m(i,j+2) + \beta * V_{j+2}^s$ Hence, we don't have to consider all values for the row k_i in the X-matrix after $m(i,j+1) + \beta * V_{j+1}^s$.

7 Value Function V^1 is provided by selecting the maximising value at position k_j for each row k_i. This step is supposed to run faster as it considers a matrix with reduced amount of entries.

8 Checking if the reached Value Function presents the optimal path by comparing the new Value Function with the previous one.

Total time taken this loop: 32 seconds. Total number of iterations: 595

1.4 Local search

7 By continuity of these policy functions, we know that if $g(k_i) = k_j$ that $g(k_{i+1})$ is in the neighbourhood of k_j. In local search, I make use of the continuity of the policy functions. From the second iteration on, only values in a epsilon region around k_t are computed. The epsilon was set for 102 time periods. This is quite large but ensures to find the optimal policy.

8 Hence, the iteration process only processes k's in a range of (k_l,k_h)=($k_t -$ 102,$k_t + 102$)

Total time taken this loop: 9 seconds. Total number of iterations: 616

1.5 Concavity Monotonicity

Combines the two steps explained above. Total time taken this loop: 15 seconds Total number of iterations: 1099

1.6 Howard's policy iteration

1 Howard's suggests to compute optimal policy functions after n-iterations of the the Value Function itself. As in the beginning of the iteration process we are not close to the optimal value. I choose to iterate $n = 100$ times, computing bruce force the Value Function and its Policy Functions.

2 By continuity of these policy functions, we know that if $g(k_i) = k_j$ that $g(k_{i+1})$ is in the neighbourhood of k_j. Hence, I compute along each optimal k given in the policy function of capital a new Value Function. Using the new Value Function in a repeated process of iterating along the capital values of the policy function, provides new solutions.

3 If a Value function equals a new Value function, the iteration is stopped. After 50 iterations, the program creates new policy functions to which to make use of for the repeated Value function iteration.

Total time taken this loop: 13 seconds
Exercise 2 was presented by Albert. Exercise 3 will be handed in, if I find time to do it.
After hours of sitting in front of a non-running code, I am really happy to found the mistakes and make it run all by myself.