# Problem set 1

Aishameriane Schmidt[*], Antonia Kurz[†].

November, 2020

## Question 1

*Consider the deterministic growth model described in class.*

For convenience, we will reproduce here the model from the lecture slides.

Consumers solve

$$\max_{\{c_t,k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left(c_t\right) \tag{1}$$

subject to

$$c_t + k_{t+1} - (1-\delta)k_t = r_t k_t, \qquad \text{for all } t \geq 0 \tag{2}$$

a No-Ponzi constraint, and $k_0$ given. Firms solve

$$\max_{k_t} z_t f\left(k_t\right) - r_t k_t, \qquad \text{for all } t \geq 0 \tag{3}$$

This gives us:

- Parameters: $\beta, \delta, k_0, u(\cdot), f(\cdot)$
- Exogenous variable: $z_t$, for all $t \geq 0$ (for now, also deterministic)
- Endogenous variables: $c_t, k_{t+1}, r_t$, for all $t \geq 0$.

As suggested, we are using the specification of the analytical problem presented in class, where:

- $\delta = 1$;
- $u(x) = \ln(c)$;
- $f(k) = k^{\alpha}$;
- $z = 1$.

[*]Contact: aishameriane.schmidt@tinbergen.nl

[†]Contact: a.kurz@tinbergen.nl

Homework for the Advanced Topics in Macro I Course, Block 2. **Instructors:** dr. M. Z. Pedroni (UvA) and dr. E. Proehl (UvA).

This reduces our problem to

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \ln(c)$$

subject to

$$c_t = z_t\, k_t^{\alpha} - k_{t+1}, \qquad \text{for all } t \geq 0$$

such that we can rewrite it as:

$$\max_{\{k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \ln(z_t k_t^{\alpha} - k_{t+1})$$

## Item a

*Solve the model by value function iteration.*

*Solution.* We opted by using brute-force grid search in this first item[1]. This means that we are computing $V_{i+1}(k)$ by finding $k'$ that solves

$$V_{i+1}(k) = \max_{k'} u(\underbrace{f(k)}_{k^{\alpha}} + \underbrace{(1-\delta)}_{0} k - k') + \beta V_i\left(k'\right). \tag{4}$$

In practice, we need to find the value function $V(k)$ and the policy function $\pi(k)$ which will allow us to find the optimal level of capital stock for the next period, $k'$, conditional on the current $k$.

We start by assembling a grid of capital levels that will be the arguments for our value function and the utility function. In the problem set up, we have the grid

$$\mathcal{K} = \left\{\frac{1}{1000}, K_2, \ldots, K_{999}, 1\right\},\; K_i < K_j,\; i < j = 1, 2, \ldots, 1000\,,$$

and the stopping criterion $\varepsilon = 0.01$. The elements inside the grid will be taken as evenly spaced, that is, $K_{i+1} - K_i = \delta > 0$, $\forall i = 1, \ldots, 999$. The space between elements in our case will be:

$$\delta = \frac{K_n - K_1}{n-1} = \frac{1 - \frac{1}{1000}}{999} = \frac{1}{1000} = \frac{1}{n}$$

The utility function has as inputs both current and future capital levels, so we denote as $U^{\text{MAT}}$ the matrix with the utility computed for all pairs of $(k, k')$ within our grid (in our code, $k'$ varies across columns and $k$ varies across rows). The next step is to compute the value function (4) at each iteration $t$, which is going to be an $n \times 1$ vector. This will then be used in the value function to obtain a new optimal value and so on, until the difference between the current and next values are less than a tolerance level $\varepsilon$.

We opted out for an equally spaced grid with $n = 1000$ points from $K_1 = \frac{1}{1000}$ to $K_n = 1$, and we set $\varepsilon = 0.01$. As parameters of the model we choose $\beta = 0.95$, $\alpha = 0.3$. The steps of our code can be seen on Algorithm 1. What we do in the main loop is computing the value function for each possible combination of $(k_i, k_j')$, then evaluating which combination gives the highest value function. Instead of iterating over all possible levels of $k$ and $k'$ (as, for example, inside a double 'for' loop, we disposed in an augmented matrix our guess for $V(0)$ (we just duplicated it to have the same dimension as the $U^{\text{MAT}}$ matrix), in a way that at each step of the algorithm we compute the value function for all possible values of $k'$ given $k$ at once. We then search for the $k$ that produces the maximum value for next period (which in 1 is represented by $max_{col}$. From there, we start over, but now the vector $V(0)$ will be replaced by the previous value function, until reaching a point where it is not possible to note difference between $V(i)$ and $V(i+1)$.

In addition to 1, we also implemented the analytical solution from the slides, to have a comparison. The plots of the value function, the capital path and consumption path using both methods are displayed in Figure 1.

□

[1] For this first part, we are basing our algorithm in what we saw in the course of Macroeconomics 1 last year.

**Algorithm 1:** Value Function Iteration Algorithm with brute-force grid search

```
Input: An initial value, $V_0(k)$, a grid $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$, $k_i < k_j$, $i < j = 1, 2, \ldots, n$, and a
       stopping criteria, $\varepsilon > 0$.
Output: $\{V^t(k), \pi^t(k)\}_{t=1}^T$
begin
    For each $k$ on the grid, assign a value for $V_0(k)$
    Let $i = 0$ while $\|V_{i+1}(k) - V_i(k)\| \geq \varepsilon$ do
        1. Compute a new function $V^{i+1}(k)$ and the associated policy function $\pi^{i+1}(k)$ from
               $[V^{i+1}(k), \pi^{i+1}(k)] = max_{col}(U^{MAT} + \beta V^{MAT,i})$.
        2. Increase $i$
    end
    Compute $k^t$ by locating the positions $\pi^t(k)$ in the grid; compute $c^t$ from $k^t$.
end
```

Item b

*Exploit the monotonicity of the policy function and evaluate its performance.*

*Solution.* As discussed in the lectures, when the policy function is monotone[2], we can solve the value function iteration problem in a more efficient way: instead of at each step compute the value function for all combinations of $(k, k')$, we can simply check the values of $(k_j, k')_{j \in \{i, \ldots, n\}}$, where $i$ denotes the index for which we obtained the maximum of the value function in the previous step. This calls for an update of the procedure from item a, which is described in Algorithm 2.

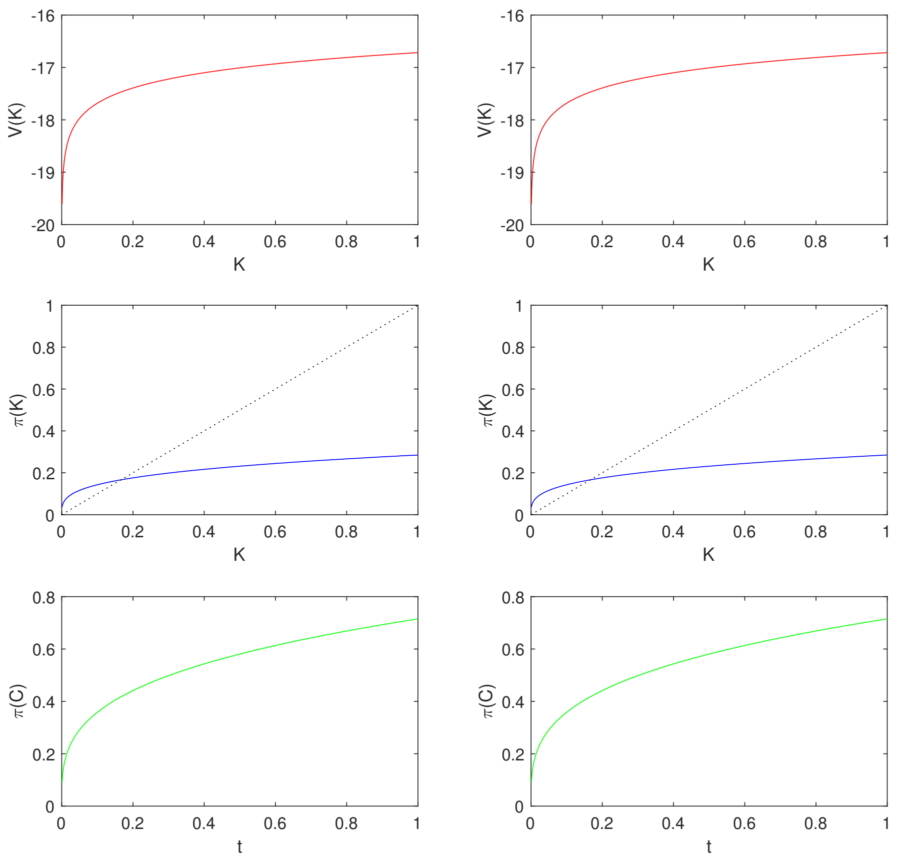
**Algorithm 2:** Value Function Iteration Algorithm exploring monotonicity of policy function

```
Input: An initial value, $V_0(k)$, a grid $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$, $k_i < k_j$, $i < j = 1, 2, \ldots, n$, and a
       iteration index, $dT$.
Output: $\{V^t(k), k'^t\}_{t=1}^T$
begin
    1. For each $k$ on the grid, assign a value for $V_0(k)$
    for $i \in \{1, \ldots, dT\}$ do
        1. Find the location of the $i$-th capital point on the grid;
        2. Compute the value function only for capital values located in indexes equal or higher than $i$,
           store the values for the $k$ that maximizes the value function over $k'$ and store its location in
           the grid;
        3. Use the vector from the last step as new starting point;
        4. Shift the index by 1.
    end
end
```

Our previous algorithm (from item a) in some sense was already efficient since we computed the objects outside the optimization routine and didn't use any loops to find the optimal path, so in terms of time for computation there is no big gains, but this is due to the way we wrote Algorithm 2: since we use a

[2] Which implies that higher values of capital will be associated with higher values of the policy function.

Figure 1 – Value function $V(K)$, policy function $\pi(K)$ and policy function $\pi(C)$ for Question 1, item (a).



**Notes:** The value function (top) corresponds to the one that solves the social planner problem, while the policy function $\pi(K)$ (middle plot) represents the optimal next-period capital stock as a function of current capital $K$. In this plot, the steady state is point where $\pi(K) = K$. The bottom graph represents the policy function for consumption. The left column contains the results using Value Function Iteration with a discrete grid, solving numerically as described in Algorithm 1, while the right column has the results when we use the analytical closed-formula results.

for loop with a fixed end point, we will continue making the computations even after achieving convergence. This can be seen in Figure 2: $k'$ (middle graph) achieves stability at very early iterations, meaning that we possibly could cut the iterations by a large number.The monotonicity algorithm needs 1.96s to run, in comparison to 0.027s from the search grid.

$\square$

## Item c

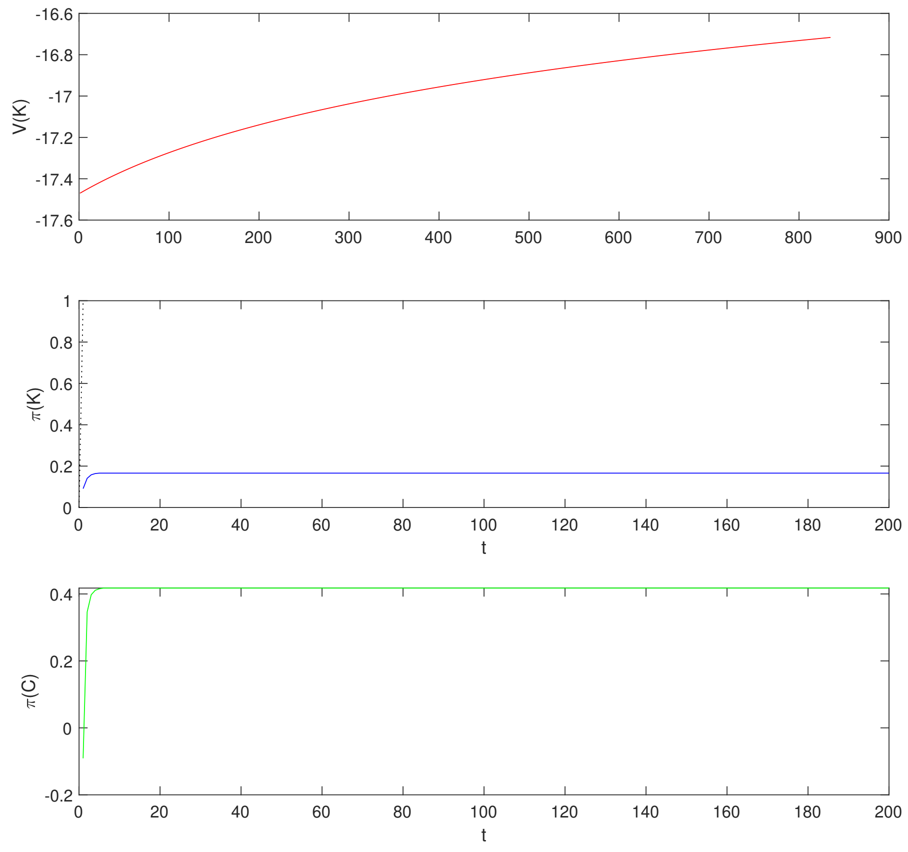
*Exploit the concavity of the policy function and evaluate its performance.*

*Solution.* When the value function is convex, we can solve the value function problem at each step until finding the value of capital in which the value function will be lower than the previous value (i.e., the value function "derivative"[3] changes sign from one step to the next one). The concavity ensures us that this happens on the point of maximum value of $V$. Once again we will update the procedure from item a, which is described in Algorithm 3.

Exploiting the concavity results, one can see that capital converges very fast within the first periods, see figure 3. The computing time is 3.457s - but this was obtained using a different machine and different

[3] Derivative is not the best terminology since we are dealing with a discretized problem, but the idea is that the function changes behavior from being increasing to decreasing.

Figure 2 – Value function $V(K)$, capital path and consumption path for Question 1, item (b).



**Notes:** The value function (top) corresponds to the one that solves the social planner problem, while the middle point shows the path that the capital takes towards time. In comparison to item (a), we see that convergence seems to be achieved faster. The bottom graph represents the policy function for consumption. The algorithm used is described in 2.

specification from the other item, which might compromise the comparison. (if one uses this different machine also for monotonicity, computing time is 5.179s, so much slower!)

□

## Item d

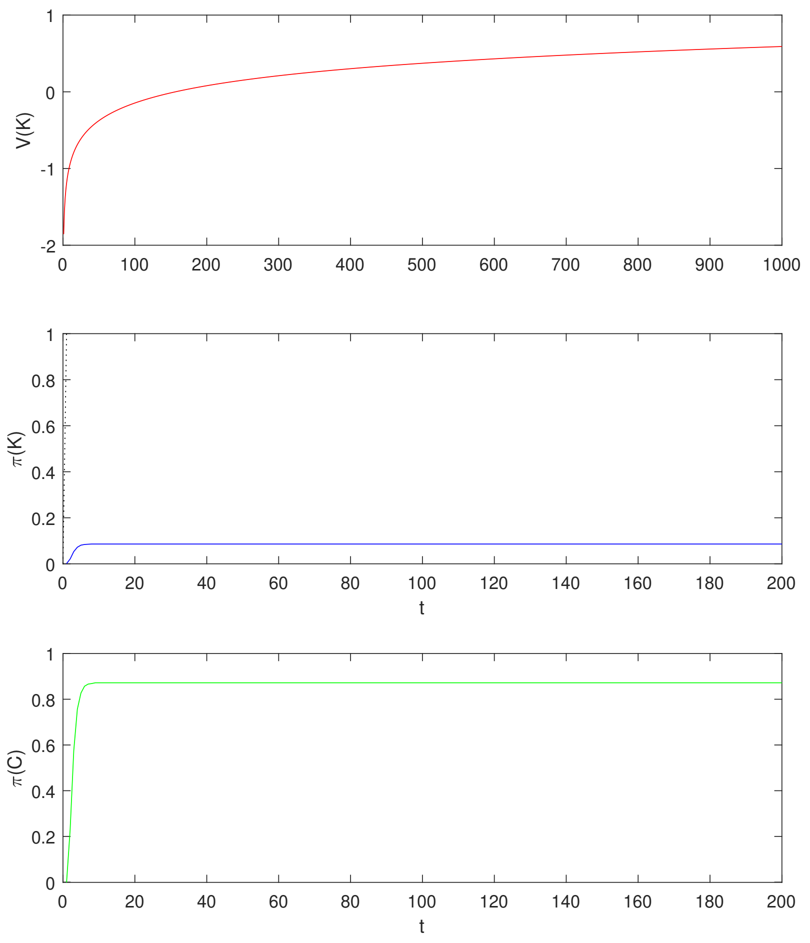
*Code Howard's policy iteration and evaluate its performance.*

*Solution.* When using Howard's policy iteration and only maximising the value function every 5th step, the convergence of k needs more periods than the concavity or monotonicity approach, see 4.

Exploiting the results, one can see that capital converges stepwise. Due to less maximisations needed, the computing time is 2.772s (again, the different machine) - faster than the concavity method (3.457s - but this was obtained using a different machine and different specification from the other item, which might compromise the comparison. (if one uses this different machine also for monotonicity, computing time is 5.179s although the latter does not need any maximisation, using the second specification (see the code for more details). And it is faster than monotonicity such that it makes it the winner of all tried methods, as we expected.

□

Figure 3 – Value function $V(K)$, capital path and consumption path for Question 1, item (c).



**Notes:** The value function (top) corresponds to the one that solves the social planner problem, while the middle point shows the path that the capital takes towards time. In comparison to item (a), we see that convergence seems to be achieved faster, as it was the case in item (b). The bottom graph represents the policy function for consumption. The algorithm used is described in 3. **Remark:** The specification in this algorithm is using $z = 2$ (instead of 1) and $\beta = 0.3$ (instead of 0.95), due to time constraints it was not possible to us to make it in a single speficication to unite with the previous items. The lower $\beta$ is expected to increase computational time, but we can see in the graphs that the capital achieves convergence in $t < 5$.

## Question 2

*Calculate the steady state of the model discussed in class*

From question 1, our problem can be represented the following way:

$$V(k) = \max_{k'} u(\underbrace{f(k)}_{zk^\alpha} + \underbrace{(1-\delta)}_{0} k - k') + \beta V(k'). \tag{5}$$

With guessing[4] that $V_i(k) = a + b\log(k)$, we find the policy function $k'(k) = z\ \alpha\beta k^\alpha$, in which the variable z represents Total Factor Productivity (TFP).

The steady state capital therefore must be

$$k^{SS}_{old} = (z\ \alpha\beta)^{\frac{1}{1-\alpha}}$$

If the new TFP is twice the old one (assuming that $z$ is old TFP), the new steady state must be

$$k^{SS}_{new} = (2z\ \alpha\beta)^{\frac{1}{1-\alpha}} = k^{SS}_{old}\ (2)^{\frac{1}{1-\alpha}}$$

[4] We find that $a = \frac{1}{1-\beta}\left[\log(1-\alpha\beta) + \frac{\alpha\beta}{1-\alpha\beta}\log(\alpha\beta) + \frac{1}{1-\alpha\beta}\log z\right]$ and $b = \frac{\alpha}{1-\alpha\beta}$

**Algorithm 3:** Value Function Iteration Algorithm exploring concavity of value function

```
Input: An initial value, $V_0(k)$, a grid $\mathcal{K} = \{k_1, k_2, \dots, k_n\}$, $k_i < k_j$, $i < j = 1, 2, \dots, n$, and a and a
       iteration index, $dT$.
Output: $\{V^t(k), k'^t\}_{t=1}^T$
begin
    1. For each $k$ on the grid, assign a value for $V_0(k)$
    for $i \in \{1, \dots, dT\}$ do
        1. Start a counter $j = 1$;
        while $V(j) < V(j+1)$ do
            1. $j = j + 1$
        end
    end
    2. Locate in the grid the position of $k_j$ and store its value and position;
    3. Compute the value function using $k_j$;
    4. Rebuild the grid for $(k', k)$.
end
```

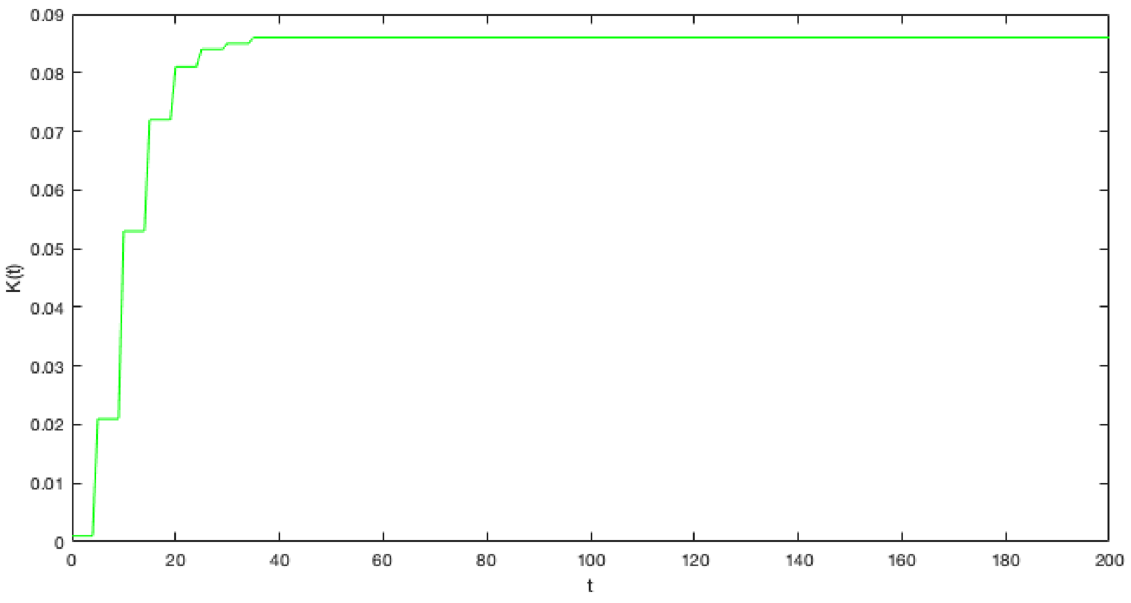


Figure 4 – Capital over time following Howard's policy iteration

The new consumption level is given via the policy function

$$c_{new}(k) = 2z\ k^\alpha - k'(k) = 2z\ k^\alpha - 2z\ \alpha\beta k^\alpha = (1-\alpha\beta)2z\ k^\alpha$$

Such that the steady state for consumption is:

$$c_{new}^{SS} = (1-\alpha\beta)2z\ \ (2z\ \alpha\beta)^{\frac{\alpha}{1-\alpha}}$$

Given our parameter choice

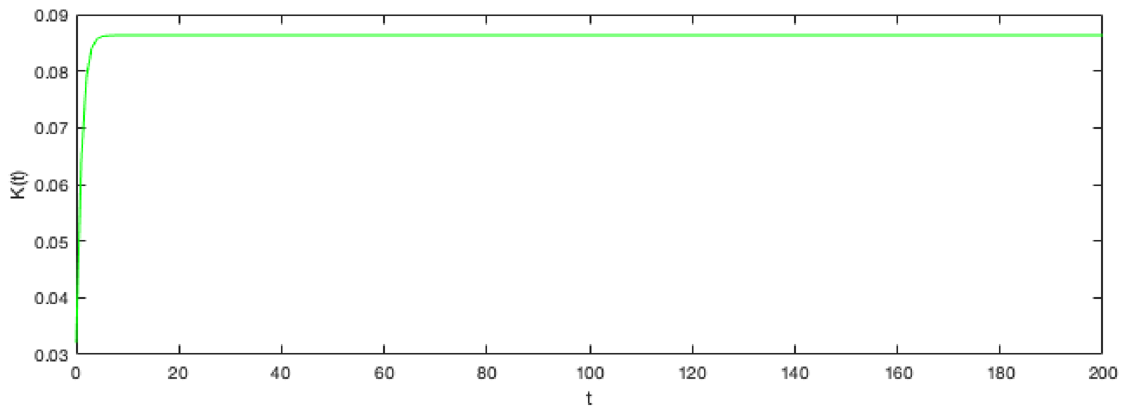


Figure 5 – Capital over time following the policy function

- $\alpha = 0.3$
- $\beta = 0.3$
- $\delta = 1$
- $T = 200$
- $z_0 = 1$

we get

- $k_{old}^{SS} = 0.0321$
- $k_{new}^{SS} = 0.0863$

Using the policy function $k'(k) = 2 * z\ \alpha\beta k^{\alpha}$, we let k converge over time, starting in $k_{old}^{SS}$, as one can see in figure 5. Capital reaches its new steady state in period 30, calculation time is 0.005 s.

For the second method, guessing a starting value of $k = 0.5$, we see a slower, but still rapid convergence, see figure 6. It doesn't perfectly converge to the new steady state, but very close (0.0860 vs. 0.0863). Calculation time is 3.129s.

The third method can't be implemented: Assuming the steady state to be a fixed point and all capital states in the range of ]0;1] are possible, one cannot infer the period zero capital without additional information about the starting point.

Furthermore, we tried to see this question as solving a sequence problem with the Euler equation as discussed in class, with a second order difference equation, solving for the third capital level in a row. This approach was not successful as the capital level converged to 2.6918.

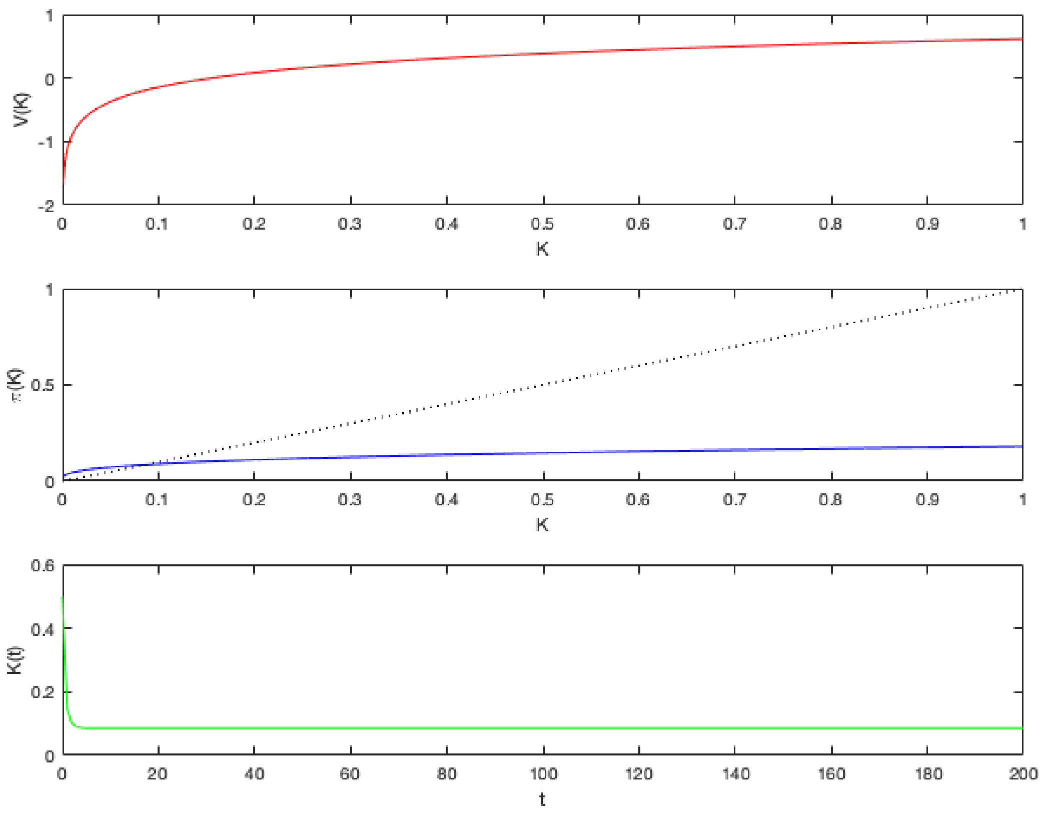


Figure 6 – Capital over time, converging to the new steady state