University of Minnesota
ECON 8581 Prof. Fatih Guvenen
Homework 3 November 23, 2022
Angelo Mendes and Johanna Torres

Numerical Derivatives

We started by evaluating the real derivative $g'(p)$ at $p = 1.5$. We then evaluate the one-sided difference for $\epsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$. The minimum difference found was: $\hat{g}'_\epsilon(1.5) - g'(1.5) \approx 8.74 \times 10^{-9}$ for $\epsilon = 10^{-8}$. We then turn to the double-sided finite differences method and approximate the derivative of g as follows:

$$\hat{g}'_\epsilon(p) = \frac{g(p+\epsilon) - g(p-\epsilon))}{2\epsilon}$$

Using the same grid for the increment ϵ we get that the best approximation was $\hat{g}'_\epsilon(1.5) - g'(1.5) \approx 6.17 \times 10^{-12}$ for $\epsilon = 10^{-5}$. The conclusion for this exercise is powerful: double-sided differences seem to be much more precise to approximate derivatives, that is, with a bigger increment ϵ we can reach a smaller error.

Root-finding

In order to evaluate the three different root-finding algorithms we set the tolerance level to 10^{-6}.

	Iterations	Error	Root
Bisection	26	8.73E-07	2.327571958
Secant	21	6.61E-07	2.327586335
Newton	4	6.12E-07	2.327574404

All the algorithms tested seemed to be very precise. However, Newton-Raphson was much more efficient since it needed only 4 iterations to reach a tolerated value.

The Stochastic Growth Model

This question asks you to solve the same dynamic programming problem we studied in HW1 but this time making tomorrow's capital a continuous choice not restricted to a grid and assuming that A is not fixed but follows a persistent stochastic process.

The problem

The problem can be written as:

$$V(k,A) = \max_{c} \left[\frac{c^{1-\gamma}}{1-\gamma} + \beta V(k',A') \right]$$

s.t.

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

Imposing standard assumptions (i.e. the constraint holds). We can rewrite the problem as choosing the future state:

$$V(k,A) = \max_{k'} \left[\frac{(Ak^{\alpha} + (1-\delta)k - k')^{1-\gamma}}{1-\gamma} + \beta V(k',A') \right]$$

The productivity parameter A follows a 15-state Markov process with a persistence of 0.98, which generically means that the current state is a sufficient statistic for forecasting future values of the state. We'll additionally assumed that the standar deviation of the shocks to the lof of the productivity level as $\sigma = 0.007$.
In order to solve this problem and obtain the policy functions and the value function will follow two procedures:

1. Value Function Iteration discretizing only the state space (not control space)

2. Value Function Iteration with discrete choice

The parameters assumed for both algorithms are as follows: $\gamma = 2$, $\beta = 0.99$, $\delta = 0.03$, $\alpha = 0.36$. This can be taken to be a reasonable parameterization at an annual frequency.

Steady State

Before proceeding with the solution, lets solve by-hand the model in steady state.

The Euler equation for this problem (ignoring expectations) is:

$$c^{-\sigma} = \beta c'^{-\sigma}(A\alpha k'^{\alpha-1} + (1-\delta))$$

In steady state: $c = c' = c^*$. This occurs when $k = k' = k^*$. Then, solving for the capital in steady state, we get:

$$k^* = \left(\frac{A^*\alpha}{1/\beta - (1-\delta)}\right)^{\frac{1}{1-\alpha}}$$

1. Value Function Iteration discretizing only the state space

Code

To solve the model we proceed with the following algorithm (for details check the comments in the code *S_ValueFunctionIteration.m*).

- Construct a grid of possible values of k. We started with a small number (31) of grid points and then increased this number in increments of 100, up to 1000. We choose our grid ranging between 25 percent of the steady state value of k and 125 percent of its steady state value.

- We also need a grid for A. Recall A follows a Markov process. To create the grid for Z and the transition matrix we use an implementation of the Tauchen method.

- We guess a value function. We have to take account of the fact that there are expectations over future states now. We started with an initial guess of the value function, $V^0(k, A)$. Then the Bellman equation:

$$V^1(k, A) = \max_{k'}\left[\frac{(Ak^\alpha + (1-\delta)k - k')^{1-\gamma}}{1-\gamma} + \beta E(V^0(k', A'))\right]$$

  Here the expectation is taken over the future value function. The uncertainty comes in over future values of A'. Given a initial k, there are 15 possible values of A' given a current value of A, which just corresponds to the row of the probability matrix for the current state of A. For simplicity we just guessed a vector of zeros.

- We specified a tolerance for when to stop searching over value functions. In particular, the convergence criterion if defined as follows:

$$\max_t \left(\frac{k_t^{n+1} - k_t^n}{1 + k_t^n} \right) < 10^{-6} \tag{1}$$

- We wrote a loop over all possible values of the state. For each value in k, find the k' that maximizes the Bellman equation given the guess of the value function. Since k' that can take continuous values above a certain positive minimum $k \geq k \geq 0$ (i.e. off the grid), we have a function file to do the maximizing that we will return the maximum. This file is call *valfun.m*. Outside of the loop there is a "while" statement, which tells MATLAB to keep repeating until we reached the convergence criterion.

- The loop can be interpret as follow: For each spot i, m (i for K and m for A) in the state space, we finds the argmax of the Bellman equation, which is found in the outside function. Then the code collected the optimized value into the new value function (called tval), and find the policy function associated with this choice (ie. the optimal choice of k, which is called kdeci in this code). After the loop over the possible values of the state we calculate the difference and write out the iteration number.

- Note that the optimization procedure we follow includes the function *fminbnd*, which assumes a continuous choice set. Basically it will search over all values of k1 between "kmin" and "kmax", which will include points not in the original capital grid. For this reason, we need to interpolate values of the value function off of the grid. We used cubic spline interpolation.

- Additionally notice that within the function *valfun.m*, when we calculate the utility, we included a penalty to prevent consumption from going negative.

- Finally, notice that since the we made the value obtain negative since we are maximizing and code is to minimize.

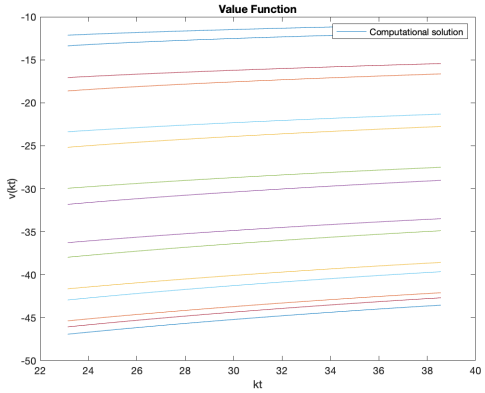
Results

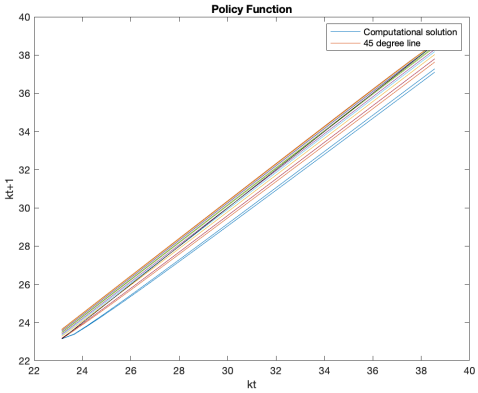
We used the described algorithm following the basic VFI as well as by applying the modified policy iteration using $m = 5$, 20, 100, and 500 Howard steps.

Compare both the solution (value function and decision rules) and the time it takes to obtain a solution

First, in Figure 1, we plot the value function and policy functions with 31 initial grid points. As we can see, the results look standard. The value function is increasing in the initial capital and the policy function has an standard shape. There are 15 lines on the figure corresponding to the different levels of current TFP. The lowest line corresponds to the lowest TFP and so on.

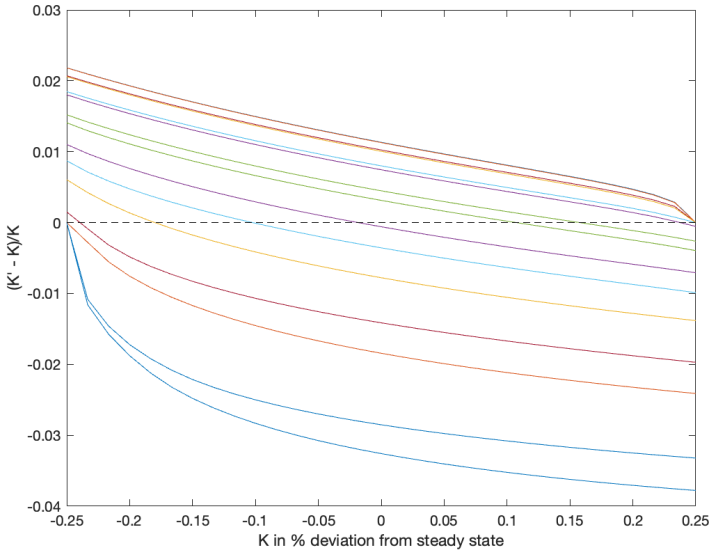
Figure 1: Value function and Policy functions with 31 initial grid points





We can also plot the differences the current capital stock as a percentage difference from the steady state and the vertical axis is the next period's capital stock as a percentage difference from this period's (Figure 2). As we can see, if current capital is low enough then the optimal policy is to increase the capital stock for all TFP levels and if the current capital is high enough then the optimal policy is to eat down the capital stock for all TFP levels. In between, the optimal policy is to build capital if TFP is high and reduce capital if TFP is low.

Figure 2: K in % deviations of the steady state



Finally, the times are shown in Table 1. The most time-efficient approach is reach with 10 Howards Steps.

Table 1: Running time of the algorithm with a discrete choice (in seconds)

		HOWARD STEPS					
		Non-Howard	**5**	**10**	**50**	**100**	**500**
Grid Points (N)	**31**	25.51	7.55	5.83	6.31	14.82	69.33
	50			9.36			
	100			18.93			
	300			78.33			

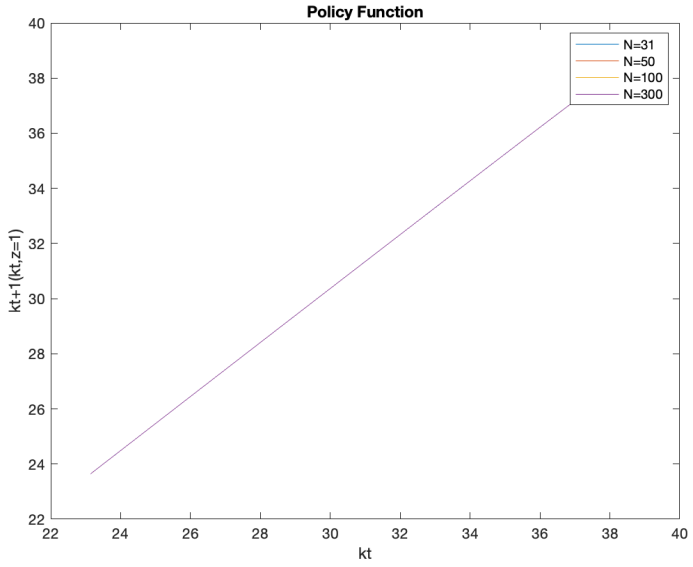
Was 31 grid points for k sufficient to get an accurate solution?

With 31 points the algorithm converges, but the accuracy differs depending of the number of initials plots as its shown in the next question.

Plot your value functions and decision rules and apply spline interpolation

Figure 3 show the solution for the algorithm with 10 Howards Steps (since this one was the most time-efficient approach) but for different initial grid points. We plot the median productivity shock ($Z = 1$) and to get a "fair" measure, we plotted the interpolate value functions and policy functions on a bigger grid of 1000 points.

Figure 3: Policy functions with different grid points



The results look similar, even for the solution with only 31 grid points. The absolute maximum difference between the policy function with 31 and 300 grid points is 0.0240.

2. Value Function Iteration with discrete choice

Code

To solve the model we proceed with the following algorithm (for details check the comments in the code *ValueFunctionIteration.m*).

For this case, we used the basic VFI algorithm but assuming that the choice of k lies on a discrete grid tomorrow. The procedure is more simple and can be resumed in the following steps:

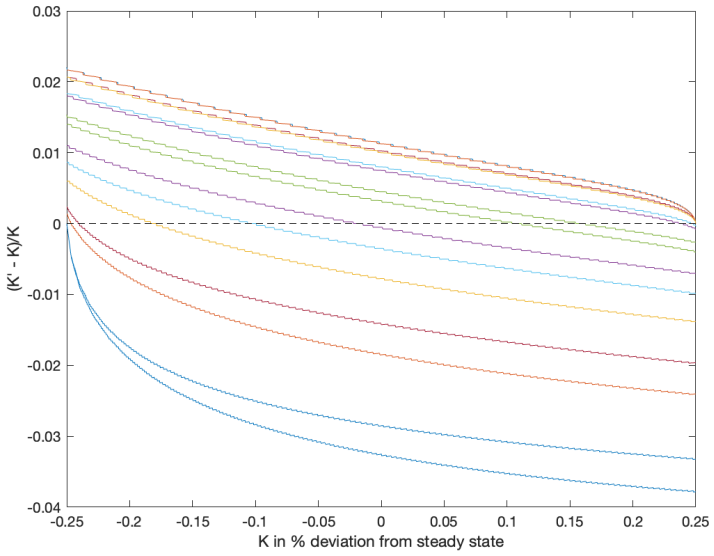
- Set parameter values
- Define a grid for state variable k and z. We also used Tauchen to discretize the stochastic process and get the transitions probabilities matrix.
- Initialize value function V
- Start iteration, repeatedly computing a new version of V.
- Stop when reach the convergence criterion.

Results

How many grid points for k did you need to pick?

Figure 4. shows the results with 5000 grid points. As we can see, there is a non-smoothness in the result even compare with our results with the continuous choice of K'.

Figure 4: K in % deviations of the steady state obtain from the algorithm with a discrete choice and 2000 grid points



Compare the timing to that of the basic VFI in the first procedure. How do they compare?

We run the algorithm with 1000 and 2000 grid points. The times are shown below:

Table 2: Running time of the algorithm with a discrete choice (in seconds)

Grid Points (N)	**1000**	58.14
	2000	553.17

As we can observe, comparing the results against the times in Table 1., non-discretizing the choice set of *K'*, allow us to get more accurate results in half of the time when non accounting for any improvement. Accounting for an improvement, like including 10 Howard Steps, the algorithm without a discrete choice is almost 20 times faster with only 1,5% of grid points (31 vs 2000).