Applied Computational Economics
Problem Set 2
Solving Representative Agent General Equilibrium Models

The University of Nottingham

2020

Q1

1. The state variables for the household are k and K. Their problem is then given by

$$\begin{aligned} v(k,K) &= \max_{c,k'} \ \frac{c^{1-\sigma}}{1-\sigma} + \beta v(k',K') \\ c &= R(K)k + W(K) + \Pi(K) + (1-\delta)k - k' \\ K' &= G(K) \end{aligned} \tag{1}$$

   where $G(K)$ is the aggregate capital law of motion. Denote the household's policy function for k' as $g(k,K)$.

2. Just the basic first order conditions

$$\begin{aligned} R(K) &= \alpha K^{\alpha-1} N^{1-\alpha} \\ W(K) &= (1-\alpha) K^{\alpha} N^{-\alpha} \end{aligned} \tag{2}$$

   where $\Pi(K) = 0$ in equilibrium.

3. A recursive competitive equilibrium is defined as a set of functions $G(K)$, $g(k,K)$, $v(k,K)$, $R(K)$ and $W(K)$ such that

   - $G(K)$ is the aggregate capital law of motion,
   - $g(k,K)$ is the household's capital policy function,
   - $v(k,K)$ is the household's value function,
   - $R(K)$ is the rental rate on capital,
   - $W(K)$ is the wage rate on labour,
   - $v(k,K)$, $g(k,K)$ come from the household's optimisation problem (1),
   - The factor returns are determined endogenously through the firm's problem (2),
   - The labour, goods and capital markets clear and
   - Consistency is satisfied $g(k,K) = G(K)$.

4. From the household's problem, we can get the Euler equation for capital investment as

$$1 = \beta \left(\frac{c_{t+1}}{c_t} \right)^{-\sigma} \{ \alpha K_{t+1}^{\alpha-1} + (1-\delta) \} \tag{3}$$

where notice I've used the market clearing condition for labour $1 = N_t \ \forall t$ in addition to substituting-out the factor return to capital. To get the steady state capital stock, impose that $c_{t+1} = c_t \ \forall t$ and that $K_t = K_{t+1} \ \forall t$ and then re-arrange (3) to obtain

$$K = \left(\frac{1}{\alpha} \left[\frac{1}{\beta} - (1-\delta) \right] \right)^{\frac{1}{\alpha-1}}.$$

5. See the code PS2.m. Note that you also need to use the lin_int_v_PS2.m file to do the linear interpolation. The way you solve this problem is effectively

   a. Conjecture $G(K)$, (which I've just set equal to K in the code),

   b. With this conjecture, solve the household's problem to get $g(k, K)$ using value function iteration.

   c. Update $G(K)$ to reflect its difference from $g(k, K)$.

   d. Return to step b and keep repeating until $g(k, K)$ and $G(K)$ have converged.

   The update slowly part pertains to step c in the above. How should we update the aggregate law of motion? Recall that the general procedure for value function iteration (used in step b above and in PS1) had the following procedure

   i. Conjecture an initial value function for the household v_0, (which I just set to zero in the code).

   ii. Get a new update for the value function v_1 using the functional equation

$$v_1(k, K) = \max_{c, k'} \frac{c^{1-\sigma}}{1-\sigma} + \beta v_0(k', K')$$

   where we just stick the initial guess on the right-side of the equation.

   iii. Update the conjecture such that $v_0 = v_1$.

   iv. Return to step ii and continue until convergence.

   In this VFI procedure, we set $v_0 = v_1$ (in step iii). That is — we take the new version of the value function as the updated guess. The natural analogue with the capital law of motion is to set $G(K) = g(k, K)$ at step c in the above. The problem with this though is that, since the consistency loop is not a contraction, there is no guarantee of convergence (or monotonic convergence for that matter). As a result, we need to be a bit more delicate with the updating rule. If you denote the initial guess in step a above as $G_0(K)$, then the rule that I use for updating in step c is $G_0(K) = 0.95 G_0(K) + 0.05 g(k, K)$. So most of the new law of motion is coming from the initial guess.

   Using this gradual updating approach obviously means more iterations are required overall since there are only very small adjustments happening to the law of motion in each iteration.

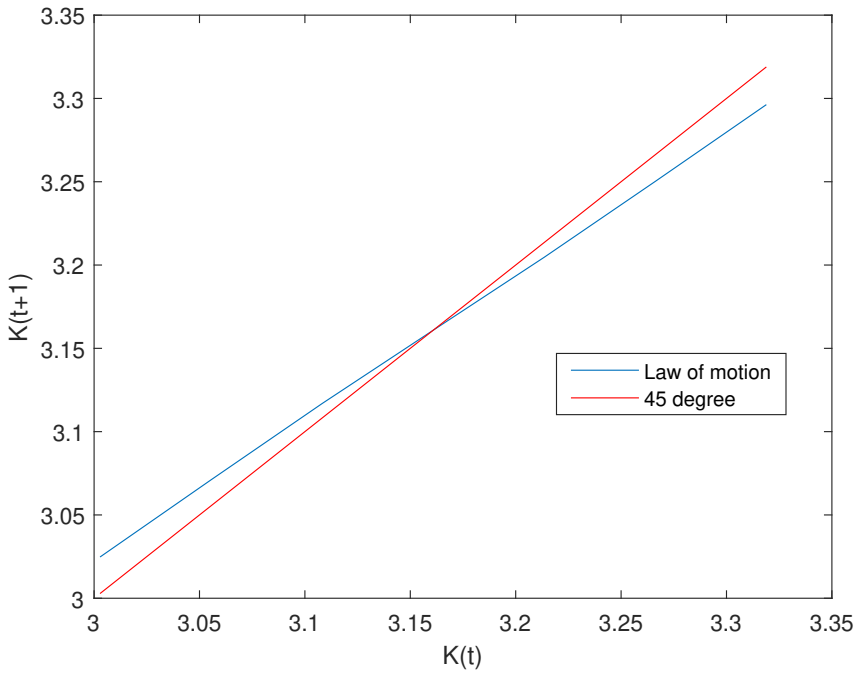

Figure 1: Interpolation law of motion for capital

It's particularly necessary though given that the problem is discretised; we take the interval for capital and chop it up into grid points. Since in effect we're iterating on an index over that discretised grid, we can end up with a situation where the difference bounces between two gridpoints, switching back and forth at each iteration. This can prevent proper convergence. See the next part with gridsearch to get a better idea of what I mean by this. Figure 1 shows the resulting law of motion from interpolation.

6. Figure 2 shows the numerical law of motion found for the 21, 51 and 71 gridpoint gridsearch solutions. Look at how rough these solutions are! Again this is related to the fact that we're iterating to get consistency on a coarse grid for capital. We're going from $K \in \{\underline{K}, ..., \bar{K}\}$ to a law of motion that's from the same set of gridpoints. In my code, when I would run it for the 21, 51 and 71 gridpoint solutions, often the difference between $G(K)$ and $g(k, K)$ would stay constant for several iterations at a time, (often the while loop wouldn't actually converge properly). This is because the maximal difference between the two functions would just bounce between two gridpoints. E.g. say on iteration n, the maximal difference is for gridpoint K_i. Often on iteration $n+1$, the maximal difference would be for gridpoint K_j for $j \neq i$. Then on iteration $n+2$, the difference would be largest for K_i again and so on. This "bouncing back and forth" behaviour makes convergence difficult and makes the law of motion very jagged and discontinuous.

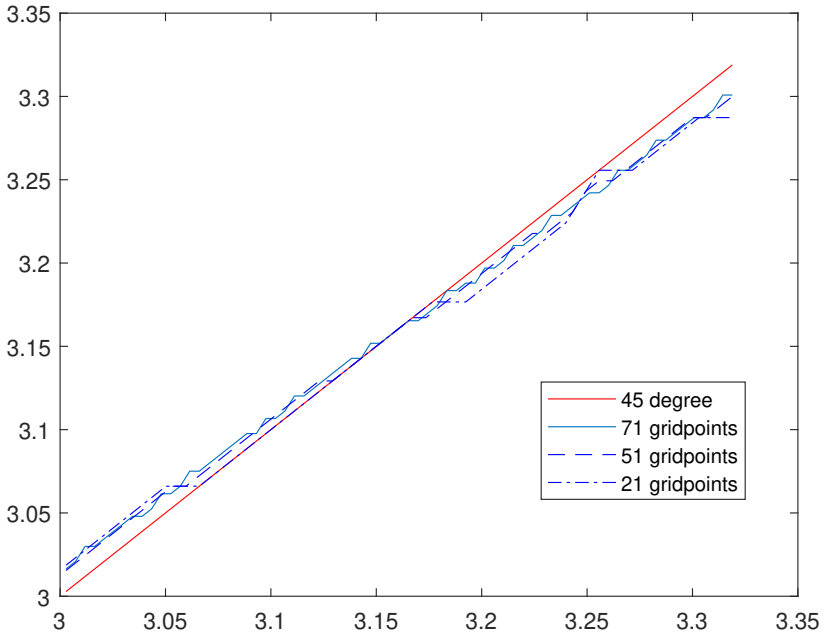

Figure 2: Gridsearch law of motion for capital

The takeaway from this exercise and comparing figures 1 and 2 is the following. There are just some types of algorithms that we use, which are better suited to interpolation over the value function rather than gridsearch. It's the fact that we're trying to get consistency for a policy function in the aggregate that makes it the better technique in this particular example.