Advanced Macroeconomics: HW1

Quang-Thanh Tran

November 2, 2022

1 Deriving the Steady State

The Social Planner problem is

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

$$\begin{aligned} \text{s.t.} \quad c_t + k_{t+1} &= k_t^\alpha + (1-\delta)k_t, \\ c_t &\geq 0, \\ k_{t+1} &\geq 0, \\ k_0 &\geq 0 \text{given}. \end{aligned} \tag{2}$$

From (2), c_t can be expressed as

$$c_t = k_t^\alpha - k_{t+1} + (1-\delta)k_t. \tag{3}$$

The Bellman equation is

$$V(k_t) = \max_{c_t, k_{t+1}} \left[\ln(c_t) + \beta V(k_{t+1})\right]. \tag{4}$$

Plugging (3) into (4), the problem becomes

$$V(k_t) = \max_{k_{t+1}} \ln(k_t^\alpha - k_{t+1} + (1-\delta)k_t) + \beta V(k_{t+1}). \tag{5}$$

The optimal solution must be true for any t. To find the optimal k_{t+1}, taking the FOC with respect to k_{t+1} yields

$$\frac{\partial V(k_t)}{\partial k_{t+1}} = \frac{-1}{k_t^\alpha - k_{t+1} + (1-\delta)k_t} + \beta \frac{\partial V(k_{t+1})}{\partial k_{t+1}} = 0.$$

so that

$$\beta \frac{\partial V(k_{t+1})}{\partial k_{t+1})} = \frac{1}{k_t^\alpha - k_{t+1} + (1-\delta)k_t} \left(\equiv \frac{1}{c_t}\right). \tag{6}$$

To find $\partial V(k_{t+1})/k_{t+1}$, first, differentiate (5) wrt k_t

$$\frac{\partial V(k_t)}{\partial k_t} = \frac{\alpha k_t^{\alpha-1} + (1-\delta)}{k_t^\alpha - k_{t+1} + (1-\delta)k_t}, \tag{7}$$

implying that

$$\frac{V(k_{t+1})}{k_{t+1}} = \frac{\alpha k_{t+1}^{\alpha-1} + (1-\delta)}{k_{t+1}^{\alpha} - k_{t+2} + (1-\delta)k_{t+1}} \left(\equiv \frac{\alpha k_{t+1}^{\alpha-1} + (1-\delta)}{c_{t+1}} \right). \tag{8}$$

Plugging this equation back to (6) yields the Euler equation

$$\frac{c_{t+1}}{c_t} = \beta[\alpha k_{t+1}^{\alpha-1} + (1-\delta)]. \tag{9}$$

The equilibrium is reached when $c_t = c_{t+1} = c^{ss}$, where we set (9) equal 1. Thus, the steady-state capital stock k^{ss} must satisfy

$$\beta[\alpha(k^{ss})^{\alpha-1} + (1-\delta)] = 1.$$

Solving for k^{ss}

$$k^{ss} = \left(\frac{\alpha\beta}{1-\beta+\delta\beta} \right)^{1/(1-\alpha)}. \tag{10}$$

If $\delta = 1$, then we have

$$k^{ss} = (\alpha\beta)^{1/(1-\alpha)}. \tag{11}$$

2 Solving Analytically: Guess and Verify

2.1 Policy Function

A reasonable guess is that the Social Planner consumes a constant fraction of output each period. Suppose that the policy function has the following form

$$c_t = \sigma k_t^{\alpha}. \tag{12}$$

We want to find an analytical expression for σ. Let $\delta = 1$, plugging (12) into (9), we have

$$\frac{k_{t+1}^{\alpha}}{k_t^{\alpha}} = \alpha\beta k_{t+1}^{\alpha-1}. \tag{13}$$

Rearranging and we obtain

$$k_{t+1} = \alpha\beta k_t^{\alpha}. \tag{14}$$

Putting this equation to the budget constraint (2) to obtain

$$c_t = (1-\alpha\beta)k_t^{\alpha}, \tag{15}$$

which implies that $\sigma = 1 - \alpha\beta$.

2.2 Value Function

Since the utility function is a natural log form, we make a guess that the Value function takes the following form

$$V(k_t) = A + B\ln(k_t). \tag{16}$$

Plugging into the Bellman equation (5), we can rewrite the problem as

$$V(k_t) = \max_{c_t, k_{t+1}} \ln(c_t) + \beta[A + B\ln(k_{t+1})]. \tag{17}$$

We need to find the optimal values for (17) subject to the constraint (2). With $\delta = 1$, the Lagrangian is

$$\mathcal{L} = \ln(c_t) + \beta[A + B\ln(k_{t+1})] + \lambda_t(k_t^\alpha - k_{t+1} - c_t). \tag{18}$$

The first-order conditions are

$$(c_t) : \frac{1}{c_t} = \lambda_t, \tag{19}$$

$$(k_{t+1}) : \frac{\beta B}{k_{t+1}} = \lambda_t. \tag{20}$$

From (19) and (20), we then derive the Euler equation

$$k_{t+1} = \beta B c_t. \tag{21}$$

Replacing c_t from (2), we obtain

$$k_{t+1} = \beta B(k_t^\alpha - k_{t+1}). \tag{22}$$

This implies

$$k_{t+1} = \frac{\beta B}{1+\beta B} k_t^\alpha, \tag{23}$$

$$c_t = \frac{1}{1+\beta B} k_t^\alpha \quad \text{(from (19))}. \tag{24}$$

By plugging these 2 expressions into the Value function at (17), we have

$$V(k_t) = \ln\left(\frac{1}{1+\beta B} k_t^\alpha\right) + \beta\left[A + B\ln\left(\frac{\beta B}{1+\beta B} k_t^\alpha\right)\right].$$

We can rearrange it to match our guess at (16)

$$V(k_t) = \underbrace{[\beta(A + B\ln(\beta B)) - (1+\beta B)\ln(1+\beta B)]}_{A} + \underbrace{\alpha(1+\beta B)}_{B}\ln(k_t). \tag{25}$$

With some algebra, we can solve for A and B as

$$A = \frac{\alpha\beta}{(1-\alpha\beta)(1-\beta)}\ln(\alpha\beta) + \frac{1}{1-\beta}\ln(1-\alpha\beta), \tag{26}$$

$$B = \frac{\alpha}{1-\alpha\beta}. \tag{27}$$

Plugging (26), (27) to equations (23) and (24), we have

$$c_t = (1-\alpha\beta)k_t^\alpha, \tag{28}$$

$$k_{t+1} = \alpha\beta k_t^\alpha. \tag{29}$$

Note that (28) is the same with (15), and (29) is the same with (14) so the two methods are consistent.

3 Numerical Methods

3.1 Value Function Iteration

Let us see how the algorithm works by reviewing the Bellman equation at (5)

$$V(k_t) = \max_{k_{t+1}} \ln(k_t^\alpha - k_{t+1} + (1-\delta)k_t) + \beta V(k_{t+1}). \tag{5}$$

`Loop 1:` We first make an initial guess of $V^{(0)}(k_{t+1}) = 0$. Plugging back to (5) yields

$$V^{(1)}(k_t) = \max_{k_{t+1}^{(0)}} \ln(k_t^\alpha + (1-\delta)k_t) + 0.$$

Using this initial guess, $k_{t+1}^{(0)}$ is obviously 0 and we then find the value of k_t that satisfies

$$V^{(1)}(k_t) = \ln(k_t^\alpha),$$

which implies that

$$V^{(1)}(k_t) = \alpha \ln(k_t).$$

Compared to (25), this is far from the true value, but more accurate than the initial guess. We will use this updated value function, plug back again to (5) and do it again.

`Loop 2:` The next iteration yields

$$V^{(2)}(k_t) = \max_{k_{t+1}^{(1)}} \ln(k_t^\alpha - k_{t+1} + (1-\delta)k_t) + \beta\alpha \ln(k_{t+1}).$$

The FOC is:

$$-\frac{1}{k_t^\alpha - k_{t+1} + (1-\delta)k_t} + \alpha\beta\frac{1}{k_{t+1}} = 0.$$

Solving for k_{t+1} yields (in MATLAB, we use `fmindbnd` to find the maximizer).

$$k_{t+1} = \frac{\alpha\beta}{1+\alpha\beta}\left(k_t^\alpha + (1-\delta)k_t\right).$$

From now on, let us assume that $\delta = 1$. We use this value and plug back to (5)

$$\begin{aligned} V^{(2)}(k_t) &= \ln\left(k_t^\alpha - \frac{\alpha\beta}{1+\alpha\beta}k_t^\alpha\right) + \beta\alpha \ln\left(\frac{\alpha\beta}{1+\alpha\beta}k_t^\alpha\right) \\ &= \underbrace{\ln\left(1 - \frac{\alpha\beta}{1+\alpha\beta}\right) + \alpha\beta \ln\left(\frac{1}{1+\alpha\beta}\right)}_{\text{a constant}} + \alpha(1+\alpha\beta)\ln(k_t). \end{aligned}$$

`Loop n:` If we repeat this process long enough, we will reach

$$V^{(n)}(k_t) = \text{ some constant } + \underbrace{\alpha(1 + \alpha\beta + (\alpha\beta)^2 + \dots)}_{\to \frac{\alpha}{1-\alpha\beta} \text{ as } n\to\infty} \ln(k_t).$$

which share a similar form to the true value function at (25). Assuming that this is true, we can work out the policy function by inserting the true estimated value function to (5) and solve for k_{t+1} to derive the policy function

$$k_{t+1} = \frac{\alpha\beta + (\alpha\beta)^2 + \dots}{1 + \alpha\beta + (\alpha\beta)^2 + \dots} k_t^\alpha \to \alpha\beta k_t^\alpha.$$

3.2 The Code

A popular numerical method is iterating the value function. First, since we already know the steady state of k from (10), we can construct a grid of possible values of k. In this case, the number of grid points is 100, and the k candidates range from $0.5k^{ss}$ to $2k^{ss}$.

```
% initiate grid
kmin = 0.5*kss;
kmax = 2*kss;
kgrid = 100; %no of grid points
kmat = linspace(kmin, kmax, kgrid); %make an array
kmat = kmat'; %convert to column vector
[N,n] = size(kmat);
```

At some iteration n, we have some estimate of the value function $V^{(n)}$. We then use the Bellman equation to compute the updated estimate of the value function $V^{(n+1)}$. Such iteration rule is expressed as follows

$$V^{(n+1)}(k_t) = \max_{k_{t+1}} \left[u(k_t^{\alpha} - k_{t+1} + (1-\delta)k_t) + \beta V^{(n)}(k_{t+1}) \right]. \tag{30}$$

The algorithm should eventually converge as demonstrated in the previous section. For simplicity, let us set the initial value for $V^{(0)} = 0$.

```
tol = 0.0001;
maxiter = 300;
dif = tol+1000;
iter = 0;
% guess
v0 = zeros(N,1);
```

If we calculate the difference (or the distance in the vector space) between $V^{(1)}$ and $V^{(0)}$, then it would be

$$d = \|V_i^{(1)} - V_i^{(0)}\|. \tag{31}$$

Convergence means that $d \to 0 \approx \text{tol}$. This is when we stop the loop.

The algorithm follows the iteration explained in the previous section. For each value of k, we find the k_{t+1} that maximizes the Bellman equation given the initial guess of the value function.

```
while dif > tol & iter < maxiter
  for i = 1:N
    k0 = kmat(i,1);
    k1 = fminbnd(@valfun, kmin, kmax);
    v1(i,1) = -valfun(k1); %because k1 is returned as negative
    kt(i,1) = k1;
  end
dif = norm(v1-v0);
v0 = v1;
iter = iter + 1;
end
```

For each spot i in the state space, we get a `k0` from the grid. Then line 4 searches and returns the `k1` that maximizes the Bellman equation, which is stored in the file `valfun.m`. Line 5 then collects the optimized value and calculates the new value function `v1` while line 6 stores this `k1` as the policy function. After the code loops over the possible values of the state, we calculate the difference and update the value function for the next loop.

If we use the function `fminbnd`, it will search over all values of k_1 between `kmin` and `kmax`, including the points not in the original capital grid. Although doing so will help us find the true value function more accurately, the computation will take forever to complete. An alternative approach is a linear interpolation. We approximate the initial guess of the value function for points off the capital grid by producing a straight line.

$$V(k) \approx V(k(i)) + \frac{V(k(i+1)) - V(k(i))}{k(i+1) - k(i)}(k - k(i)). \tag{32}$$

By feeding the newly founded maximizer `k1` into the linear interpolation of the value function, we can approximate the value function. Since the value function is an increasing function of k, we can narrow down the search after each loop.

```
function val=valfun(k)
    global v0 beta delta alpha kmat k0 kgrid
    ki0 = max(sum(k>kmat), 1); %identify the grid that falls just
    below the choice for k
    ki1 = ki0 + 1;
    %interpolation algorithm
    g = v0(ki0) + (k-kmat(ki0))*(v0(ki1)-v0(ki0))/(kmat(ki1)-kmat(
    ki0));
    % another interpolation algorithm
    %g = interp1(kmat, v0, k, 'linear');
    c = k0^alpha - k + (1-delta)*k0; %consumption
    if c <= 0
        val = -888-800*abs(c);
    else
        val = log(c) + beta*g;
    end
    %change value to negative since "fminbnd" finds minimum
val = -val;
```

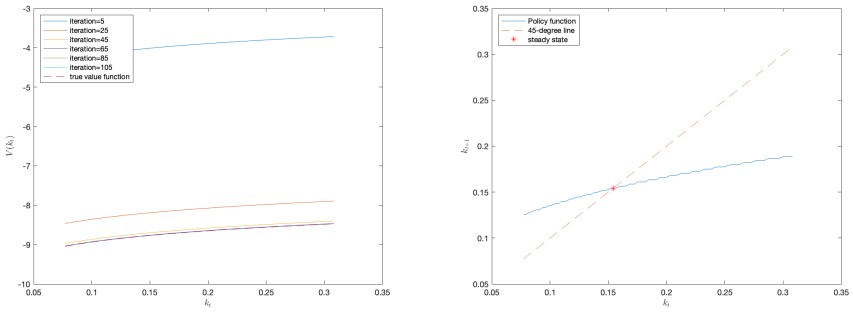
There are 2 ways to perform the interpolation. The first is optimizing the selection of the next k candidate given our own algorithm. The second is to perform interpolation on the whole grid by using MATLAB `interp1` function. The time to run the first is 4.336054 seconds, while the latter is 5.954239 seconds. Thus, we conclude that optimizing grid selection can improve efficiency.

3.3 Read more

- The code: https://github.com/thanhqtran/advanced_macroeconomics
- Eric Sims' Note: https://www3.nd.edu/~esims1/val_fun_iter.pdf
- Ramsey model: Solving with value function iteration https://www.youtube.com/watch?v=oYVfEXAbWEU
- McCandless, G. (2008). The ABCs of RBCs: An introduction to dynamic macroeconomic models. Harvard University Press.

3.4 Figures

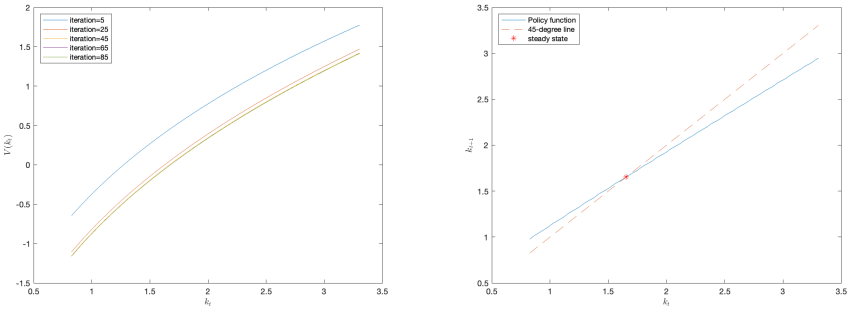
With $\delta = 1$:



(a) Value function (b) Policy function

Figure 1: The algorithms when $\delta = 1$.

With $\delta = 0.1$:



(a) Value function (b) Policy function

Figure 2: The algorithms when $\delta = 0.1$.