Graduate Macro Theory II: Notes on Value Function Iteration

Eric Sims University of Notre Dame

Spring 2024

1 Introduction

These notes discuss how to solve dynamic economic models using value function iteration.

2 A Deterministic Growth Model

The solution to the deterministic growth model can be written as a Bellman equation as follows:

$$V(K) = \max_{C} \left\{ \frac{C^{1-\sigma} - 1}{1 - \sigma} + \beta V(K') \right\}$$
s.t.

$$K' = K^{\alpha} - C + (1 - \delta)K$$

Variables with a \prime denote future values. K is a predetermined state variable. C is a jump variable. You can impose that the constraint holds and re-write the problem as choosing the future state:

$$V(K) = \max_{K'} \left\{ \frac{(K^{\alpha} + (1 - \delta)K - K')^{1 - \sigma} - 1}{1 - \sigma} + \beta V(K') \right\}$$

The basic idea of value function iteration is as follows. Create a grid of possible values of the state, K, with N elements. Then make a guess of the value function, $V^0(k)$. This guess will be a $N \times 1$ vector – one value for each possible state. For that guess of the value function, compute $V^1(k)$ as follows:

$$V^{1}(K) = \max_{K'} \left\{ \frac{(K^{\alpha} + (1 - \delta)K - K')^{1 - \sigma} - 1}{1 - \sigma} + \beta V^{0}(K') \right\}$$

In doing this process, it is important to remember to take care of the "max" operator inside the Bellman equation: conditional on $V^0(K)$, you have to choose K' to maximize the right hand side. For each grid value of K, this gives you a new value for the value function, call this $V^1(K)$. Then compare $V^1(K)$ to your initial guess, $V^0(K)$. If it is not close, then take your new value function, $V^1(K)$, as the "guess" and repeat the process. On the n^{th} iteration, you will have $V^n(K)$ and $V^{n-1}(K)$. For n large, if your problem is well-defined, these should be very close together and you can stop.

2.1 A Numerical Example

Suppose that the parameter values are as follows: $\sigma = 2$, $\beta = 0.95$, $\delta = 0.1$, $\alpha = 0.33$. Basically, this can be taken to be a reasonable parameterization at an annual frequency. The Euler equation for this problem is:

$$C^{-\sigma} = \beta C'^{\sigma} (\alpha K'^{\alpha - 1} + (1 - \delta))$$

In steady state, $C' = C = C^*$. This occurs where $K' = K = K^*$ where:

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

For the parameters I have chose, I get a value of $K^* = 3.169$. We then need to construct a grid of possible values of K. I will construct a grid with N = 100 values, ranging between 25 percent of the steady state value of K and 175 percent of its steady state value. The MATLAB code is:

```
1 kmin = 0.25*kstar;
2 kmax = 1.75*kstar;
3 grid = (kmax-kmin)/kgrid;
4
5 kmat = kmin:grid:kmax;
6 kmat = kmat';
7 [N,n] = size(kmat);
```

The resulting vector is called "kmat" and is 100×1 . Next I guess a value function. For simplicity I will just guess a vector of zeros:

```
1 \text{ v0} = zeros(N, 1);
```

I need to specify a tolerance for when to stop searching over value functions. In particular, my objective is going to be the norm of the of the absolute value of the difference between $V^n(K)$ and $V^{n-1}(K)$. The norm I'm going to use is the square root of the sum of squared deviations. One could do other measures of closeness of the value functions. I need to set a tolerance for this for when MATLAB to stop. I set this tolerance at 0.01. I also can specify the maximum number of

iterations that MATLAB can take. I set this at 300. The MATLAB code for these preliminaries is:

```
1 tol = 0.01;
2 maxits = 1000;
```

I'm going to write a loop over all possible values of the state. For each value in K, find the K' that maximizes the Bellman equation given my guess of the value function. I have a function file to do the maximizing that we will return to in a minute. Outside of the loop I have a "while" statement, which tells MATLAB to keep repeating the text as long as the difference between value functions is greater than the tolerance. The MATLAB code for this part is:

```
1 while dif>tol & its < maxits
2     for i = 1:N
3         k0 = kmat(i,1);
4         k1 = fminbnd(@valfun,kmin,kmax);
5         v1(i,1) = -valfun(k1);
6         k11(i,1) = k1;
7     end
8     dif = norm(v1-v0);
9     v0 = v1;
10     its = its+1
11    end</pre>
```

This code can be interpreted as follows. For each value i in the state space, I get the initial value of K. Then the command on line 4 finds the argmax of the Bellman equation, which is found in the function file "valfun2.m". Line 5 collects the optimized value into the new value function (called v1), and line 6 finds the policy function associated with this choice (ie. the optimal choice of K', which is called k1 in this code). After the loop over the possible values of the state I calculate the difference and write out the iteration number. I do not include a semi-colon after these statements so that I can see the converge taking place in the command window.

A complication arises in function file because "fminbnd" 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. I will use linear interpolation. Suppose that Matlab chooses a point k1 that is off the grid. I can approximate the value function at that point assuming that the value function is linear between the two points on the grid immediately around that point. In particular, I want to find K(i) < K < K(i+1). Then I can approximate the value function at this point as:

$$V(k) \approx V(K(i)) + \frac{v(K(i+1)) - v(K(i))}{K(i+1) - K(i)} (K - K(i))$$

One can do other forms of interpolation. If the problem is not too non-linear, and your grid points are close enough to one another, linear interpolation ought to be pretty good. Matlab has a built-in interpolation code called "interpl" that will do this for you.

```
1 g = interp1(kmat,v0,k,'linear'); % smooths out previous value function
```

Now, given the choice of K', we can define C in Matlab as:

```
1 c = k0^alpha - k + (1-\Delta)*k0; % consumption
```

We need to include a penalty to prevent consumption from going negative and then calculate the Bellman equation as follows:

The commented out part is for use when s = 1, when the utility function collapses to the natural log. My complete code for the man file and the function file are given below:

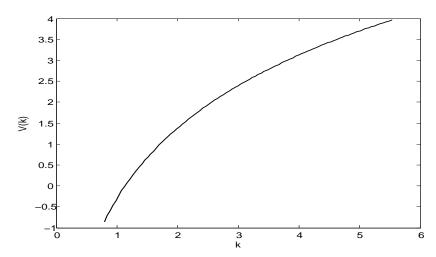
```
1 clear all;
2 close all;
4 tic
  % Eric Sims
  % University of Notre Dame
  % Graduate Macro II
10
11 % This program uses Value Function Iteration to solve a neoclassical growth
  % model with no uncertainty
14 global v0 beta \Delta alpha kmat k0 s
15
16 plott=0; % set to 1 to see plots
  %s = 2;
18
19
20 % set parameters
21 alpha = 0.33; % capital's share
22 beta = 0.95;
23 \Delta = 0.1; % depreciation rate (annual)
24 s = 2;
26 tol = 0.01;
```

```
27 \text{ maxits} = 1000;
28 \text{ dif} = \text{tol} + 1000;
29 its = 0;
31 kgrid = 99; % grid points + 1
32 kstar = (alpha/(1/beta - (1-\Delta)))^(1/(1-alpha)); % steady state k
33 cstar = kstar^(alpha) - \Delta *kstar;
34 istar = \Delta * kstar;
35 ystar = kstar^(alpha);
37 kmin = 0.25 * kstar;
38 \text{ kmax} = 1.75 * \text{kstar};
39 grid = (kmax-kmin)/kgrid;
41 kmat = kmin:grid:kmax;
42 kmat = kmat';
43 [N,n] = size(kmat);
45 polfun = zeros(kgrid+1,3);
46
47 \text{ v0} = zeros(N, 1);
48 dif = 10;
49 its = 0;
51 while dif>tol & its < maxits
      for i = 1:N
            k0 = kmat(i,1);
53
            k1 = fminbnd(@valfun,kmin,kmax);
            v1(i,1) = -valfun(k1);
            k11(i,1) = k1;
       end
58 \text{ dif} = \text{norm}(v1-v0);
59 \text{ v0} = \text{v1};
60 its = its+1
61 end
63 for i = 1:N
      con(i,1) = kmat(i,1)^(alpha) - k11(i,1) + (1-\Delta)*kmat(i,1);
        polfun(i,1) = kmat(i,1)^(alpha) - k11(i,1) + (1-\Delta)*kmat(i,1);
66 end
67
69 figure
70 plot(kmat, v1, '-k', 'Linewidth', 1)
71 xlabel('k')
72 ylabel('V(k)')
74 figure
75 plot(kmat,polfun,'-k','Linewidth',1)
```

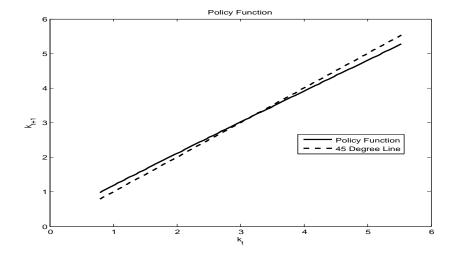
```
76 xlabel('k')
77 ylabel('c')
78
79 toc
```

```
function val=valfun(k)
2
  % This program gets the value function for a neoclassical growth model with
  % no uncertainty and CRRA utility
  global v0 beta \Delta alpha kmat k0 s
  g = interp1(kmat, v0, k, 'linear'); % smooths out previous value function
  c = k0^alpha - k + (1-\Delta)*k0; % consumption
  if c \le 0
11
      12
13
      val=(1/(1-s))*(c^{(1-s)}-1) + beta*g;
14
  end
15
16 val = -val; % make it negative since we're maximizing and code is to minimize.
```

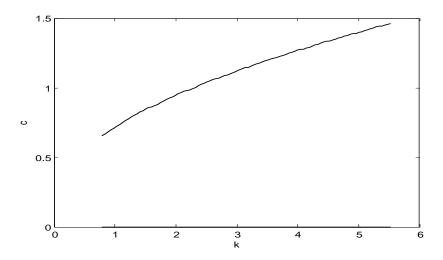
Here is a plot of the resulting value function, plotted against the value of the current capital stock:



Next I show a plot of the policy function – given K, what is the optimal choice of K'? I plot this with a 45 degree line showing all points where K' = K, so that we can visually verify that a steady state indeed exists and is stable:



Once I know K', I can get C. Here is a plot of C against K:



3 The Stochastic Growth Model

Making the problem stochastic presents no special challenges, other than an expansion of the state space and having to deal with expectations and probabilities. The problem can be written:

$$V(K,A) = \max_{C} \left\{ \frac{C^{1-\sigma} - 1}{1-\sigma} + \beta EV(K',A') \right\}$$
 s.t.
$$K' = AK^{\alpha} - C + (1-\delta)K$$

You can impose that the constraint holds and re-write the problem as choosing the future state:

$$V(K, A) = \max_{K'} \left\{ \frac{(AK^{\alpha} + (1 - \delta)K - K')^{1 - \sigma} - 1}{1 - \sigma} + \beta EV(K', A') \right\}$$

The stochastic part comes in because A is stochastic. I assume that A follows a Markov process, which generically means that the current state is a sufficient statistic for forecasting future values of the state. Let \tilde{A} denote a $q \times 1$ vector denoting possible realizations of A. Then let P be a $q \times q$ matrix whose (i,j) element is taken to be the probability of transitioning to state j from state i; hence the rows must sum to one. I assume that A can take on the following three values: low, medium, and high:

$$\tilde{A} = \left(\begin{array}{c} 0.9\\1.0\\1.1 \end{array}\right)$$

I define the probability matrix so that all of its elements are $\frac{1}{3}$:

$$P = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

I can specify the A process as follows:

```
1 amat = [0.9 1 1.1]';
2 prob = (1/3)*ones(3,3);
```

This just means that the expected value of A next period is always 1, regardless of what current productivity is. It also means that the non-stochastic steady state value A is also 1.

Another complication now is that my value function is going to have $N \times q$ elements. In other words, there is a different value to being in every different capital/productivity combination. I construct this is a $N \times q$ matrix, though I could conceivably do Kronecker products and make the state space a one dimensional object.

The value function iteration proceeds similarly to above, but I have to take account of the fact that there are expectations over future states now. I start with an initial guess of the value function, $V^0(K, A)$. Then I construct the Bellman equation:

$$V^{1}(K,A) = \max_{k'} \left\{ \frac{(AK^{\alpha} + (1-\delta)K - K')^{1-\sigma} - 1}{1-\sigma} + \beta E\left(V^{0}(K',A')\right) \right\}$$

Here the expectation is taken over the future value function. The uncertainty comes in over future values of A' – you are choosing K' based on an expectation of A'. Given a choice of K',

there are three 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.

My new MATLAB code looks like this is in the main file. It is almost identical to above, except there is now a double loop (i.e. looping over both states):

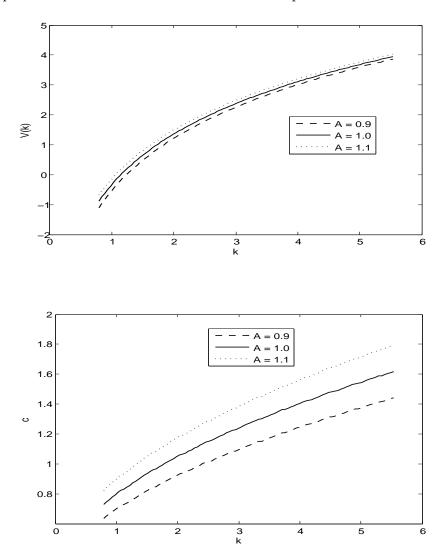
```
while dif>tol & its < maxits
2
       for j = 1:3
3
           for i = 1:N
4
           k0 = kmat(i,1);
           a0 = amat(j, 1);
6
           k1 = fminbnd(@valfun_stoch,kmin,kmax);
           v1(i,j) = -valfun_stoch(k1);
           k11(i,j) = k1;
           end
10
11
       end
12
       %q = abs(v1-v0);
  dif = norm(v1-v0)
  v0 = v1;
  its = its+1
  end
```

The function file takes account of the expectations in the following way:

```
function val=valfun_stoch(k)
  % This program gets the value function for a stochastic growth model with
  % CRRA utility
  global v0 beta \Delta alpha kmat k0 prob a0 s j
  g = interp1(kmat,v0,k,'linear'); % smooths out previous value function
  c = a0*k0^alpha - k + (1-\Delta)*k0; % consumption
10
11
      12
  else
13
14
      val = (1/(1-s))*(c^{(1-s)} - 1) + beta*(g*prob(j,:)');
15
16 end
17 val = -val; % make it negative since we're maximizing and code is to minimize.
```

Basically, I post-multiply the interpolated value function by the (transpose of) the proper row of the probability matrix. The resulting value functions -V(K) plotted against K for different values of A – are plotted below. As one would expect, the value of being in a "high" A state exceeds the value of being in the low A states. These differences are not very large – this primarily results

because my specification of P is such that there is no real persistence in A.



4 Dealing with "Static" Variables

As written above, all of the variables are dynamic – choices today directly affect the states tomorrow. In many models we will also want to work with "static" variables, variables whose choice today does not directly influence the value of future states. An example here is variable employment. Suppose that we take the neoclassical growth model with variable employment.

Let the time endowment of the agents be normalized to 1, and let their work effort be given by n. They get utility from leisure, which is 1 - n. The dynamic program can be written as:

$$V(K) = \max_{C,n} \left\{ \frac{C^{1-\sigma} - 1}{1 - \sigma} + \theta \ln(1 - n) + \beta V(K') \right\}$$
s.t.

$$K' = K^{\alpha} n^{1-\alpha} - C + (1-\delta)K$$

The first order condition for the choice of employment is:

$$\frac{\theta}{1-n} = C^{-\sigma}(1-\alpha)K^{\alpha}n^{-\alpha} \tag{2}$$

This first order condition must hold for any choice of C (equivalently, K') along an optimal path. Hence, you can effectively treat this first order condition as a "constraint" that must hold. It essentially determines n implicitly as a function of C and K. Then you can write the modified Bellman equation as:

$$V(K) = \max_{K'} \left\{ \frac{\left(K^{\alpha} n^{1-\alpha} + (1-\delta)K - K'\right)^{1-\sigma} - 1}{1-\sigma} + \theta \ln(1-n) + \beta V(K') \right\}$$
s.t.
$$\frac{\theta}{1-n} = \left(K^{\alpha} n^{1-\alpha} + (1-\delta)K - K'\right)^{-\sigma} (1-\alpha)K^{\alpha} n^{-\alpha}$$

The "constraint" (the FOC for labor) implicitly defines n as a function of K and K'. So you can effectively plug that in everywhere you see n showing up, leaving the problem as one of just choosing K' given K.