Graduate Macro Theory II:
Notes on Value Function Iteration

Eric Sims
University of Notre Dame

Spring 2011

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:

$$V(k) = \max_{c} \left\{ \frac{c^{1-\sigma}-1}{1-\sigma} + \beta V(k') \right\}$$
$$\text{s.t.}$$
$$k' = k^{\alpha} - c + (1-\delta)k$$

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:

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

Then compare how $V^1(k)$ looks compared to your guess, $V^0(k)$. If it is not close, then take your new value function, $V^1(k)$, 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 (ignoring expectations):

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

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:

```
kmin = 0.25*kstar; % minimum
kmax = 1.75*kstar; % maximum
kgrid = 99; % grid points + 1
grid = (kmax-kmin)/kgrid; % grid

kmat = kmin:grid:kmax;
kmat = kmat'; % make column vector as opposed to row
```

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:

```
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. 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:

```
tol = 0.01; % maximum tolerance
maxits = 300; % maximum number of iterations
```

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(@valfun2,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
```

This code can be interpreted as follows. For each spot i in the state space, I get $k0$. 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:

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

In Matlab, my interpolation code is:

```
1  klo=max(sum(k>kmat),1); % identify the gridpoint that falls just below . .
2  % . . the choice for k
3  khi=klo+1;
4  % do the interpolation
5  gg =  v0(klo) + (k-kmat(klo))*(v0(khi) - v0(klo))/(kmat(khi) - kmat(klo));
```

Now, given the choice of k, we can define c in Matlab as:

```
c = k0^alpha - k + (1-Δ)*k0;
```

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

```
if c<=0
    val = -9999999 - 999*abs(c);
else
     val = (1/(1-s))*(c^(1-s)-1) + beta*gg;
end
val = -val; % make it negative since we're maximizing and code is to minimize.
```

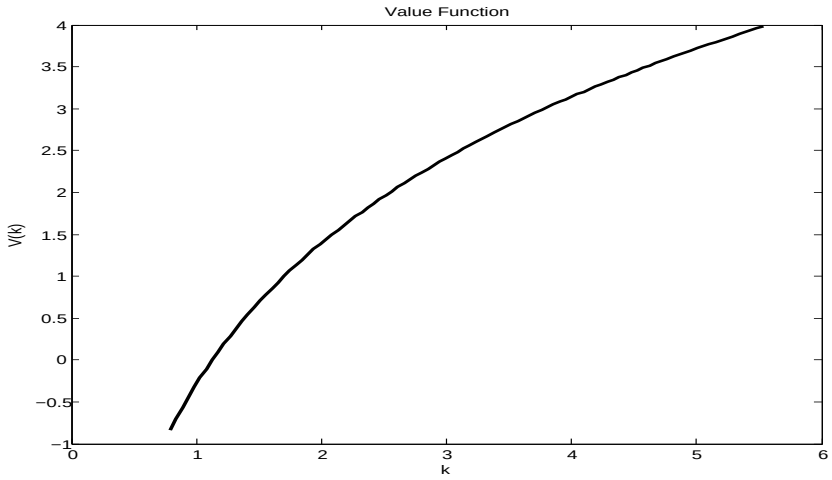
My complete code for the man file and the function file are given below:

```
clear all;
close all;
tic

global v0 beta Δ alpha kmat k0 s kgrid

% set parameters
alpha = 0.33; % capital's share
beta = 0.95;
Δ = 0.1; % depreciation rate (annual)
s = 2;

tol = 0.01;
maxits = 300;
dif = tol+1000;
its = 0;
kgrid = 99; % grid points + 1

% first solve for steady state
kstar = (alpha/(1/beta - (1-Δ)))^(1/(1-alpha)); % steady state k
cstar = kstar^(alpha) - Δ*kstar;
istar = Δ*kstar;
ystar = kstar^(alpha);

kmin = 0.25*kstar; % minimum
kmax = 1.75*kstar; % maximum
kgrid = 99; % grid points + 1
grid = (kmax-kmin)/kgrid; % grid

kmat = kmin:grid:kmax;
```

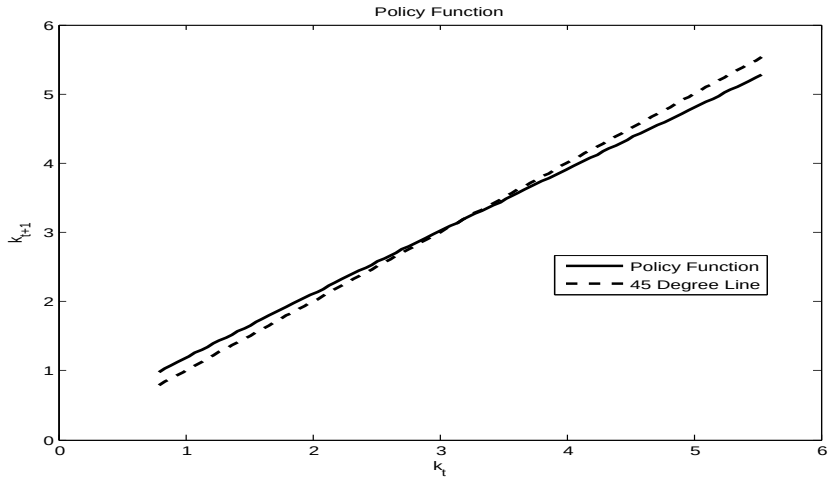
```
kmat = kmat';

[N,n] = size(kmat);

v0 = zeros(N,1);

while dif>tol & its < maxits
    for i = 1:N
        k0 = kmat(i,1);
        k1 = fminbnd(@valfun2,kmin,kmax);
        v1(i,1) = -valfun(k1);
        k11(i,1) = k1;
    end
dif = norm(v1-v0)
v0 = v1;
its = its+1
end


toc
```

```
function val=valfun2(k)

% This program gets the value function for a neoclassical growth model with
% no uncertainty and CRRA utility

global v0 beta Δ alpha kmat k0 s kgrid


klo=max(sum(k>kmat),1); % identify the gridpoint that falls just below . .
% . . the choice for k
khi=klo+1;

% do the interpolation
gg =  v0(klo) + (k-kmat(klo))*(v0(khi) - v0(klo))/(kmat(khi) - kmat(klo));

c = k0^alpha - k + (1-Δ)*k0; % consumption
if c≤0
    val = -9999999 - 999*abs(c);
else
     val = (1/(1-s))*(c^(1-s)-1) + beta*gg;
end
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_{t+1}, what is the optimal choice of k_t? I plot this with a 45 degree line showing all points where $k_{t+1} = k_t$, so that we can visually verify that a steady state indeed exists and is stable:



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:

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

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:

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

The stochastic part comes in in that 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:

$$\tilde{A} = \begin{pmatrix} 0.9 \\ 1.0 \\ 1.1 \end{pmatrix}$$

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

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

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

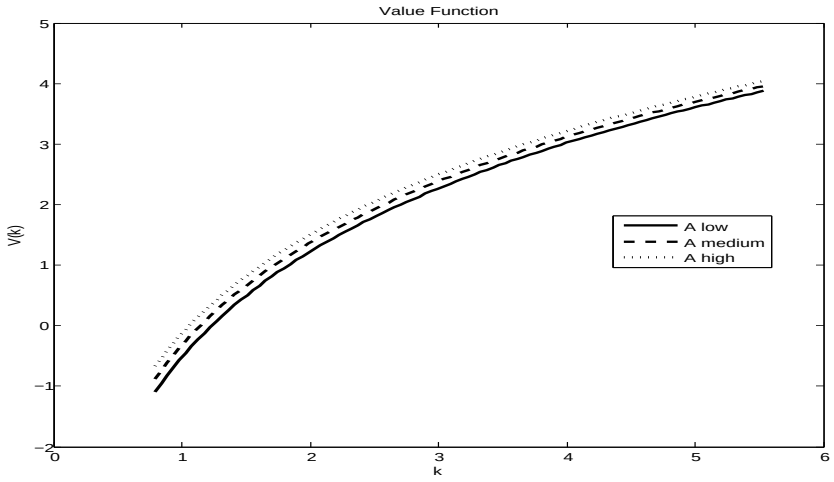
there is now a double loop (i.e. looping over both states):

```
while dif>tol & its < maxits
    for j = 1:3
    for i = 1:N
        k0 = kmat(i,1);
        a0 = amat(j,1);
        k1 = fminbnd(@valfun_stoch,kmin,kmax);
        v1(i,j) = -valfun_stoch(k1);
        k11(i,j) = k1;
    end
    end
    ggg = v1 - v0;
    gg = vec(ggg);
dif = norm(v1 - v0)
v0 = v1;
its = its+1
end
```

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

```

klo=max(sum(k>kmat),1); % identify the gridpoint that falls just below . .
% . . the choice for k
khi=klo+1;

% do the interpolation
g =  v0(klo,:) + (k-kmat(klo))*(v0(khi,:) - v0(klo,:))/(kmat(khi) - kmat(klo));
gg = interp1(kmat,v0,k,'linear');

c = amat(j,1)*k0^alpha - k + (1-Δ)*k0; % consumption
if c≤0
    val = -8888888888888888-800*abs(c);
else
     val = (1/(1-s))*(c^(1-s)-1) + beta*(gg*prob(j,:)');
end
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 small – 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:

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

s.t.

$$\frac{\theta}{1-n} = c^{-\sigma}(1-\alpha)k^\alpha n^{-\alpha}$$