LECTURE NOTES

COMPUTATIONAL METHODS FOR MACROECONOMICS

FALL 2006

KAREN A. KOPECKY

Value Function Iteration

1 Value Function Iteration

Consider the standard neoclassical growth model in recursive form,

$$V(K) = \max_{C,K'} \{U(C) + \beta V(K')\}$$

subject to

$$C + K' = zF(K, 1) + (1 - \delta)K,$$

$$C \geq 0,$$

$$K' \geq 0,$$

K_0 given

where $U(\cdot)$ and $F(\cdot, \cdot)$ satisfy the standard assumptions, $0 < \beta < 1$ and $0 \leq \delta \leq 1$, and z is assumed to be a constant.

We wish to solve for the value function, $V(K)$, and optimal capital accumulation policy, $K' = g(K)$. The model can be solved analytically in the case of full depreciation ($\delta = 1$) and log utility. However, when we deviate from this often unrealistic case we can no longer solve analytically. So instead we use numerical methods to compute approximations to the value function and policy for capital. A large number of such numerical methods exist. The most straightforward as well as popular is value function iteration. By the name you can tell that this is an iterative method. The iteration rule is as follows. At iteration n, we have some estimate of the value function, $V^{(n)}$. We then use the Bellman equation to compute an updated estimate of the value function, $V^{(n+1)}$, as follows:

$$V^{(n+1)}(K) = \max_{K'} \left\{U(zF(K, 1) + (1 - \delta)K - K') + \beta V^{(n)}(K)\right\} \tag{1}$$

We know from Stokey and Lucas that (1) a value function V^* that is a solution to the fixed point problem above exists and is unique and (2) if we repeat this procedure many times (i.e., start with an initial guess $V^{(0)}$ and use equation (1) to obtain $V^{(1)}$ and again to obtain $V^{(2)}$, and so on) our value function $V^{(n)}$ will converge monotonically to the V^* for any continuous function $V^{(0)}$. This means that after many iterations, each subsequent iteration no longer changes our guess, i.e. $\lim_{n\to\infty} ||V^{(n+1)} - V^{(n)}|| = 0$. The fact that value function iteration will eventually converge for virtually any initial guess, make value function iteration a very stable algorithm.

The next task is to decide how to represent V, a function defined over a continuous state space, by a finite number of numbers. In other words, we need to decide how we want to approximate V. It is the approximation that we will solve for since it is not possible to solve for V itself numerically. We will approximate the function over the continuous domain by a discrete set of function values at a discrete set of points in the domain. Suppose we are interested in V on the region $[\underline{K}, \overline{K}]$ this could be an interval around the non-trivial steady state level of capital $K^* > 0$ or an interval starting below the steady state but above the trivial steady state of $K^* = 0$ and ending above the non-trivial steady state for example. To approximate V on this domain we will choose a discrete state space or grid of n_k K's, $\{K_1, K_2, K_3, ..., K_{n_k}\}$, where $K_i \in [\underline{K}, \bar{K}]$, for all i. Instead of solving for V itself, we will solve for the approximation to V which consists of the values of V at each point on the capital grid.

The next step is to determine the grid. We would ideally like to select a very large n_k to obtain as good an approximation to the value function as possible. However, the usual tradeoff between efficiency and accuracy arises since as n_k increases so do the time required to compute the approximation at each iteration. With only one state variable the rate of increase of the time cost with increasing n_k is not outrageous but in more complicated models with multiple state variables, processing time increases exponentially with increases in grid size, the so-called *curse of dimensionality*.

The simplest way to discretize the state space is to evenly space the n_k grid points along the real line between the chosen upper and lower bounds but alternative methods that place more grid points in areas where the value function is more non-linear may yield better approximations for the same cost of computation time. The value function is in

general more non-linear for lower values of capital. Thus is is often better to make the grid points evenly spaced in logs.

1.0.1 Value function iteration algorithm

Step 1. Choose a relative error tolerance level, ε.

Step 2. Discretize the state space by constructing a grid for capital:
$K = \{K_1, K_2, K_3, ..., K_{n_k}\}$.

Step 3. Start with an initial guess of the value function, $V^{(0)}(K)$. This is a vector of length n_k, i.e. $V^{(0)}(K) = \{V_i^{(0)}\}_{i=1}^{n_k}$. where $V_i^{(0)} = V^{(0)}(K_i)$ The initial guess can be as simple as $V^{(0)}(K) = 0$, or it can be a more sophisticated guess.

Step 4. Update the value function using equation (1). Specifically,

i) Fix the current capital stock at one of the grid points, K_i, beginning with $i = 1$

ii) For each possible choice of capital next period, K_j, $j = 1, ..., n_k$, calculate

$$T_{i,j} = U(zF(K_i, 1) + (1 - \delta)K_i - K_j) + \beta V_j^{(0)} \tag{2}$$

If consumption is negative for a particular K_j, assign a large negative number to T_i, j. Note that T_i is a vector of length n_k, with each element, $T_{i,j}$, representing the value of the right hand side of equation (5) conditional upon the respective choice of capital next period, K_j, i.e. $T_i = \{T_{i,j}\}_{j=1}^{n_k}$

iii) Find the location of the maximum and maximum of T_i. Store the maximum as the ith element of the updated value function, $V^{(1)}(K)$. Store the location of the maximizer, as the ith element in the policy vector g.

iv) Choose a new grid point for the current capital stock in step (a) and repeat steps (b) and (c). Once we have completed steps (a) through (c) for each value of K_i, i.e. for $i = 1, .., n_k$, we will have updated value and policy functions and can continue to the next step.

Step 5. Compute the distance, d, between $V^{(0)}(K)$ and $V^{(1)}(K)$. A common definition of d is the sup norm, i.e.

$$d = \max_{i \in \{1,...,n_k\}} |V_i^{(1)} - V_i^{(0)}|$$

Step 6. If the distance is within the error tolerance level, $d \leq \varepsilon ||V_i^{(1)}||$, the value function has converged, so we have obtained the numerical estimates of the value

and policy functions. If $d > \varepsilon$, return to Step 4, setting the initial guess to the updated value function, i.e. $V^{(0)}(K) = V^{(1)}(K)$. Keep iterating until the value function has converged.

1.0.2 Additional steps

1. Check that the policy function isn't constrained by the discrete state space. If K_{g_i} is equal to the highest or lowest value of capital in the grid for some i, relax the bounds of K and redo the value function iteration.

2. Check that the error tolerance is small enough. If a small reduction in ε results in large changes in the value or policy functions, the tolerance is too high. Reduce ε until the solution to the value and policy functions are insensitive to further reductions.

3. Check that n_k is large enough. If an increase in n_k results in a substantially different solution, the grid might be too sparse. Keep increasing n_k until the value and policy functions are insensitive to further increases in n_k.

4. A good initial guess of the value function can reduce computation time substantially. Solving the model using a small number of gridpoints for capital will give you an optimal decision rule from which to develop a good guess at the value function using a higher value for n_k.

2 Speed Improvements

Value function iteration is stable, in that it converges to the true solution, however it is also very slow. Fortunately, there are some methods that we can use to reduce computation time without sacrificing stability.

The simplest way to speed up the algorithm is to ensure that you are not doing redoing costly computations over and over again in the iteration loop. For example, notice that $U(zF(K_i, 1) + (1 - \delta)K_i - K_j)$ does not depend on the updated value of V or g and hence can be computed for all i and j once at the beginning stored in an array and the values retrieved when needed.

2.1 Howard's Improvement

Selecting the maximizer is the most time-consuming step in the value function iteration. Howard's improvement reduces the number of times we update the policy function relative to the number of times we update the value function. Essentially, on some iterations, we simply use the current approximation to the policy function to update the value function without updating the policy function. Updating the value function using a the current estimate of the policy function will bring the value function closer to the true value since the policy function tends to converge faster than the value function.

To implement Howard's Improvement, we need to select an integer, n_h, which is the number of iterations using the existing policy function we will perform after each policy function update. Some experimentation is usually required to determine the optimal n_h. To many may result in a value function moving further from the true one since the policy function is not the optimal policy. We also add an additional step to the value function iteration algorithm between the existing Step 4 and Step 5, that we'll name Step 4.5, as follows.

Step 4.5 Complete the following steps n_h times.

a) Set $V^{(0)}(K) = V^{(1)}(K)$

b) For $i = 1, ..., n_k$, update the value function using the following equation:

$$V_i^{(1)} = U(zF(K_i, 1) + (1 - \delta)K_i - K_{g_i}) + \beta V_{g_i}^{(0)}$$

c) Repeat parts (a) and (b). Note that the value of K_j has been replaced by the optimal decision rule K_{g_i}. We now need just one computation for each value of K_i, rather than n_k computations, one for each possible choice of capital stock next period. Also note that there is no optimization, since we do not update the policy function.

Note that as $n_h \to \infty$, the difference between $V^{(0)}(K)$ and $V^{(1)}(K)$ goes to zero and they converge to the fixed point of

$$V_i^{\infty} = U(zF(K_i, 1) + (1 - \delta)K_i - K_{g_i}) + \beta V_{g_i}^{\infty}$$

Instead of iterating on the bellman equation you could solve for V_i^{∞} directly using matrix operations. It requires the solution to a system of n_k linear equations however.

2.2 Concavity of the Value Function

The value function that solves the neoclassical growth model here is strictly concave in the choice of K'. Therefore the value of the Bellman equation that we compute in Step 4 will have a unique maximum. For a given i, if we start with $j = 1$ and sequentially calculate the value of $T_{i,j}$ for each $j = 1, \ldots, n_k$, we know that K_{t-1} is the maximizer if $T_{i,t} < T_{i,t-1}$. To exploit this and reduce computation time, as soon as we find the maximizer, we can stop looking, i.e. we don't need to compute the value of $T_{i,j}$ for $j = t + 1, ..., n_k$.

We can modify Step 4 as follows:

Step 4 Update the value function using equation (1). Specifically,

a) Fix the current capital stock at K_i, beginning with $i = 1$.

b) Fix next period's capital stock at K_j, beginning with $j = 1$. Compute $T_{i,j}$ using:

$$T_{i,j} = U(zF(K_i, 1) + (1 - \delta)K_i - K_j) + \beta V_j^{(0)}$$

c) If $j = 1$, return to part (b) but set $j = 2$. If $j > 1$, compare $T_{i,j}$ to $T_{i,j-1}$. If $T_{i,j} < T_{i,j-1}$, set $g_i = j - 1$, $V_i^{(1)} = T_{i,j-1}$, and ignore part (d).

d) If $j = n_k$, set $g_i = n_k$ and $V_i^{(1)} = V_{n_k}^{(1)}$. Otherwise, set $j = j + 1$ and repeat parts (b) and (c).

e) Choose the next grid point for K_i in step (a) and repeat steps (b) through (d). Once steps (a) through (d) have been complete for $i = 1, ..., n_k$, we will have updated value and policy functions and can continue to the next step.

2.3 Monotonicity of the Policy Function

The policy function that solves the neoclassical growth model here is monotonic non-decreasing in K_i, i.e. if $K_i < K_{i+1}$, then $g(K_i) \leq g(K_{i+1})$. Once we have found the optimal decision rule for K_i, we know that the optimal decision rule for K_{i+1} cannot be any of the grid points from K_1 to K_{i-1}. Therefore, we can begin our search for g_{i+1} at g_i, reducing the number of computations we need to perform.

We can modify Step 4 as follows:

Step 4 Update the value function using equation (1). Specifically,

a) Fix the current capital stock at K_i, beginning with $i = 1$.

b) Compute $T_{i,j}$ for j from g_{i-1} or 1 if $i = 1$ to n_k using:

$$T(j) = U(zF(K_i, 1) + (1 - \delta)K_i - K_j) + \beta V_j^{(0)}$$

c) Find the maximizer and maximum of T. Store the maximum as the ith element of the updated value function, $V^{(1)}(K)$. Store the maximum location as the ith element in the policy function, g.

d) Choose the next grid point for K_i in step (a) and repeat steps (b) and (c). Once steps (a) through (c) have been complete for $i = 1, ..., n_k$, we will have updated value and policy functions and can continue to the next step.

3 Linear Interpolation

Consider beginning the value function iteration procedure as normal. Start with an initial capital level, K_i, and choose the level of capital next period that maximizes the value function, K_J. We know that value function is higher when we choose K_J compared to either K_{J-1} or K_{J+1}, however, no other level of capital in the range (K_{J-1}, K_{J+1}) was available. It is almost certain that the true maximizer in a continuous state space would be a point other than K_J in this range, so our approximation to the true optimal policy and value functions almost certainly contains some error. How can we reduce this error?

One solution is to increase the number of grid points. As $n_k \to \infty$, the discrete state space approaches a continuous state space over the range of the grid. Computing power, however, is not infinite, and increasing the number of grid points increases computation time exponentially. In the single state variable case, we need to compute the modified Bellman equation n_k^2 times, so increasing the grid size from 9 to 10 increases the number of computations from 81 to 100. If there were 2 state variables, the number of computations would increase from 9^4 to 10^4, i.e. by 3,439. The memory requirements to store value and policy functions also increase exponentially.

An alternative to increasing n_k that improves accuracy without increasing computation time or memory requirements substantially is linear interpolation. This involves approximating the initial guess of the value function for points off the main capital grid by straight lines. Once we have chosen the optimal capital next period, K_J, we construct a new grid

for capital of length, m_k, over the range, $[K_{J-1}, K_J]$. We want to check whether any of the points on this sub-grid is preferred to K_J.

Recall equation (5).

$$T(j) = U(zF(K_i, 1) + (1 - \delta)K_i - K_j) + \beta V_j^{(0)}(K_j)$$

For points on the sub-grid, we can calculate $U(zF(K_i, 1) + (1 - \delta)K_i - K_j)$, however we cannot calculate $V_j^{(0)}(K_j)$, since it only allows points on the main grid as inputs. We need to interpolate for points between K_{J-1} and K_J and will do so by constructing a straight line between $V_j^{(0)}(K_{J-1})$ and $V_j^{(0)}(K_J)$.

Recall that the line that joins two points, (x_1, y_1) and (x_2, y_2), is given by the following.

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \tag{3}$$

We can therefore approximate the value function for intermediate guesses, K_t, by the following.

$$V_j^{(0)}(K_t) - V_j^{(0)}(K_{J-1}) = \frac{V_j^{(0)}(K_J) - V_j^{(0)}(K_{J-1})}{K_J - K_{J-1}}(K_t - K_{J-1}) \tag{4}$$

We should also check whether any point in the range $[K_J, K_{J+1}]$ would be chosen over K_J. We therefore construct a second sub-grid over this range and use an analagous approximation to that in equation (4) for the value function.

Implementing linear interpolation requires systematically calculating the value of $T(j)$ for each point in the two sub-grids and selecting the maximum. Note that we use the linear function to approximate values in the initial guess of the value function, $V_j^{(0)}(K_j)$, not the updated value function, $V_j^{(1)}(K_j)$. Note also that if K_J is either K_1 or K_{n_k}, we can only construct one sub-grid for the interpolation.

To add linear interpolation to the value function iteration, replace the original Step 4 with the following.

4. Update the value function using equation (1). Specifically,

    a) Fix the current capital stock at one of the grid points, K_i, beginning with $i = 1$

    b) For each possible choice of capital next period, K_j, $j = 1, ..., n_k$, calculate

$$T(j) = U(zF(K_i, 1) + (1 - \delta)K_i - K_j) + \beta V_j^{(0)}(K_j) \tag{5}$$

If consumption is negative for a particular K_j, assign a large negative number to $T(j)$. Note that T is a vector of length n_k, with each element, $T(j)$, representing the value of the right hand side of equation (5) conditional upon the respective choice of capital next period, K_j, i.e. $T = \{T(j)\}_{j=1}^{n_k}$

c) Find the maximizer and maximum of T. Denote the maximizer K_J.

d) If $K_J > K_1$:

   i Construct a sub-grid for capital of length m_k on the interval $[K_{J-1}, K_J]$.

   ii For each point in the sub-grid, K_t, calculate

$$T^{(l)}(t) = U(zF(K_i, 1)+(1-\delta)K_i-K_t)+\beta \left(V_j^{(0)}(K_{J-1}) + \frac{V_j^{(0)}(K_J) - V_j^{(0)}(K_{J-1})}{K_J - K_{J-1}}(K_t - K_{J-1}) \right)$$

e) If $K_J < K_{n_k}$:

   i Construct a sub-grid for capital of length m_k on the interval $[K_J, K_{J+1}]$.

   ii For each point in the sub-grid, K_t, calculate

$$T^{(r)}(t) = U(zF(K_i, 1)+(1-\delta)K_i-K_t)+\beta \left(V_j^{(0)}(K_J) + \frac{V_j^{(0)}(K_{J+1}) - V_j^{(0)}(K_J)}{K_{J+1} - K_J}(K_t - K_J) \right)$$

f) Find the maximum of all points in the two sub-grids, $T^{(l)}(t)$ and $T^{(r)}(t)$, combined. Store the maximum as the ith element of the updated value function, $V^{(1)}(K)$. Store the maximizer, $K_j \in \{K_{J-1}, ..., K_J, ..., K_{J+1}\}$, as the ith element in the policy function, $g(K)$.

g) Choose a new grid point for the current capital stock in step (a) and repeat steps (b) through (f). Once we have completed steps (a) through (f) for each value of K_i, i.e. for $i = 1, .., n_k$, we will have updated value and policy functions and can continue to the next step.

4 Adding Leisure Choice

Consider the standard neoclassical growth model with leisure choice.

$$V(K) = \max_{C,K',L} \{U(C,L) + \beta V(K')\}$$

subject to

$$C + K' = zF(K,N) + (1-\delta)K$$

$$C \geq 0$$

$$K' \geq 0$$

$$L + N = 1$$

Continuous choice static variable - don't solve inside the model.

Solving the problem numerically is that the value function is defined on a continuous state space (i.e. K is a continuous variable) however computers can only handle discrete numbers. We need to approximate the value function using a finite set of values for K, which we'll call a grid of length n_k, i.e. $K = \{K_1, K_2, K_3, ..., K_{n_k}\}$.

To solve for model numerically, we need to approximate the value function by using discrete numbers. We need to choose a grid of size, n_k, for capital