This document was generated at 5:28 PM on Monday, March 25, 2024
Copyright 2024 Richard T. Woodward

11 – Numerical Issues #1: The complications of continuity

AGEC 642 - 2024

Introduction and a caveat

This lecture and number 12 are focused on numerical methods to address some of the challenges of applying numerical dynamic programming. The purpose of these notes is to give you a flavor of different approaches and give you intuition that hopefully will help you understand dynamic programming better. The notes do not, therefore, provide you with a recipe for implementation and make no promise that they capture the best and latest methods. With the foundation provided by these notes, I think you will be able to solve simple DP problems or understand how more sophisticated approaches can be used to solve more complicated DP problems.

Part 1: Continuous State Space

I. The basic problem

The DD problems (discrete state and discrete control) that we have considered up until now have had an important limitation: the state variable has been assumed to take on only a finite number of possible values.

- In the simple inventory control problem discussed, the inventory could be only 0, 1,2,….
- In the option pricing model, the price is assumed to go either up or down by a constant factor, going up by u or down by u.
- In the cow replacement problem, the state variable is simply the age of the cow, which is counted in discrete units, and a small finite set of production classes.
- In Burt & Allison's paper, soil moisture is treated as falling in one of five levels.

In reality, of course, most important economic variables are not discrete.

- Inventory of most products is held in such large numbers that it approximates a continuous number.
- A cow's productivity cannot be described by a small set of discrete levels, but instead fall along a continuous distribution.
- Prices vary nearly continuously (up to 1¢ intervals)
- Soil moisture content, Burt and Allison's state variable varies continuously.

Allowing for the continuity of state space in DP problems, however, introduces some very important problems. Consider a finite horizon problem with a known salvage function $V(x, T)$ that maps from the continuous domain of x to $\mathbb{R}$. For a finite-horizon problem, the Bellman's equation for periods $T-1$ would take the form

1. $$V\left(x_{T-1}, T-1\right) = \max_{z_{T-1}} E\left[u\left(z_{T-1}, x_{T-1}, T-1\right) + \beta V\left(x_T, T\right)\right].$$

This equation can be evaluated at any **finite grid** of points, say $X=\{x^1, x^2, x^3,\ldots, x^n\}$, since we know the functional form for $u(\cdot)$, $V(x_T,T)$, and the state equation are known. When we come to the equation for $V(x_{T-2},T-2)$, however, we have

2. $$V\left(x_{T-2}, T-2\right) = \max_{z_{T-2}} E\left[u\left(z_{T-2}, x_{T-2}, T-2\right) + \beta V\left(x_{T-1}, T-1\right)\right].$$

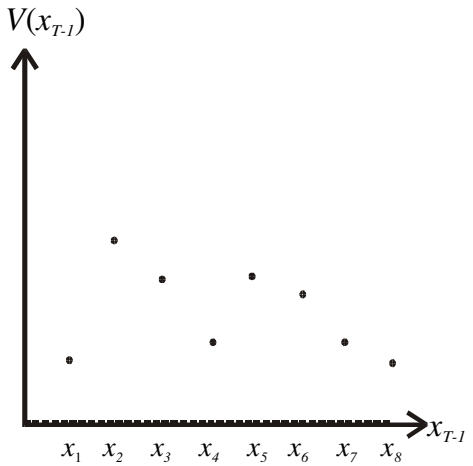
This may cause problems because from 1 we only know the values of $V(x_{T\text{-}1},T\text{-}1)$ at the points at which 1 has been evaluated, namely $X=\{x^1, x^2, x^3,\ldots, x^n\}$. Since we need to find the value of z_{T-2} that maximizes the RHS of 2, it is likely that some candidate values of z will lead to values for $x_{T\text{-}1}$ that are not contained in the grid X. Hence, we are faced with a problem: How do we ensure that we are finding the correct solution to 2 if we only know the values of $V(x_{T\text{-}1,}\,T\text{-}1)$ at a finite set of points?

If the problem is stochastic, then this issue becomes even more relevant. Suppose there is a continuous probability distribution over $x_{T\text{-}1}$ conditional on x_{T-2} and z_{T-2}, then for a given choice, z_{T-2}, the expected future value of next period's stock will be $E\left[V\left(x_{T-1},T-1\right)\right]\Big|_{x_{t-2},z_{T-2}} = \int f\left(x_{T-1}\,|\,x_{t-2},z_{T-2}\right)V\left(x_{T-1},T-1\right)dx_{T-1}$ where $f(x_{T-1}|\,z_{T-2},\,x_{T-2}\,)$ is the probability distribution of x_{T-1} conditional on z_{T-2} and x_{T-2}.

> It might be easier to think of the case of a discrete probability distribution $E\left[V\left(x_{T-1},T-1\right)\right]\Big|_{x_{t-2},z_{T-2}} = \sum_{i=1}^{m} p\left(x_{T-1}^{i};z_{T-2}\right)V\left(x_{T-1}^{i},T-1\right)$ with $p\left(x_{T-1}^{i};z_{T-2}\right)$ being the probability that x_{T-1} takes on a particular value x^i given a particular choice z_{T-2} with m large compared to n. A discrete specification would typically be used to approximate continuous distributions.

However, since we have used numerical methods to solve the first equation at only n points or *nodes*, we don't have "observations" of $V(x_{T-1},T-1)$ at all the possible values of x_{T-1}.

Suppose, for example, you have evaluated $V(x_{T-1})$ at the eight points in the figure below and came up with the values as indicated. To solve for $V(x_{T-2})$ you need to take an expectation of the value function including points that fall between points on this grid. How do we proceed? We now discuss a someays around this problem.



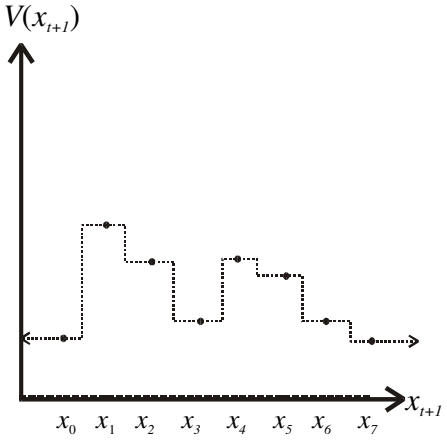
Needless to say, this problem does not occur only at $T-2$, but in all periods except the final one. Hence, it critically affects infinite horizon problems in the same way. We focus in these notes on the infinite-horizon case; extension to a finite-horizon case is straightforward. We should point out, however, that unlike the successive approximation algorithm for DD problems, the convergence of the infinite-horizon algorithm for these problems is not as well-behaved, may not be monotonic, and may not converge uniformly.

II. Solution #1: Rounding

The easiest way to handle a continuous state space is to turn it into a discrete space. That is, we treat the value function as if it can only take on n possible values, those values associated with the points in our set X. If x_{t+1} should happen to fall outside this set of points, either between the points or completely outside the range, we simply round up or down until we get to a value that we have evaluated. Technically, we might write,

$$\hat{V}(x_{t+1}) = V(\Omega(x_{t+1})) \text{ with } \Omega(x_{t+1}) = \arg\min_{x \in X} |x - x_{t+1}|.$$

That is, since we don't have an estimate of $V(x_{t+1})$, we approximate it with the value of the nearest point for which we know.



Using the same points from the first figure, the implicit value function that follows from rounding would take a form like that in the figure above.

It is easy to see that rounding may not be the best way to handle the problem of approximating the value function. For example, if x_1=1.0 and x_2=2.0, then the estimate of the value of x_{t+1}=1.49 would be dramatically different from the approximation of the value of x_{t+1}=1.51.

In most economic problems, however, the value function is not as bumpy as the one in the figures. If $V(\cdot)$ is a monotonic function without huge changes in its slope, then the magnitude of the error using rounding can be quite small. Nonetheless, if you want to use rounding, you need to make sure that your grid is tight enough so that the rounding is not having an overwhelming influence on your results.

A. The rounding algorithm

Implementing rounding numerically is quite simple in principle and could be implemented by an algorithm like the following. Let V be an array of values from the previous stage at each of the points in your grid X, let x be your grid, X which will take on values $x_1, x_2, \ldots, x_{nx}$, and let xtrue be the true value of x_{t+1} for which we want to find an estimate of $V(x_{t+1})$, say Vest. The following algorithm would find the nearest estimate using rounding.

These lines of code would calculate the value of $V(x_{t+1})$ for some value $x_{t+1} \notin X$.

```
diff = ∞
xTrue = g(x_t, z_t, ε_t)        'Calculates the true value x_{t+1} as a function of the current
                                state, ' control and random shock. Call this value xtrue
for all ix
  if abs(xgrid(ix)-xTrue) < diff then        ' closest to xtrue
    diff = abs(xgrid(ix)-xTrue)
    ixGrid = ix                  ' Use V(x(ix)) as an estimate of V(xtrue),
  endif                          '    The x(ix) that is closest to xtrue will be used
next ix
Vest = VRHS(ixGrid)
```

Matlab can carry all of this out much more concisely:

```
[~, ixGrid] = min(abs(x - x_grid)); The min function returns both the minimum, which in
                                    this case is ignored and the index that contains the
                                    closest point
Vest = VRHS(ixGrid)
```

An important modeling decision in virtually every application of numerical DP is how to treat points that are completely outside the grid. In the figure and algorithms above, I assumed that the $V(x_{t+1})$ is the same as $V(x_0)$ if $x_{t+1} < x_0$ and is the same as $V(x_n)$ if $x_{t+1} > x_n$. However, this may not be appropriate. For example, this may give the impression that a decision-maker could drive the state variable to negative infinity without sacrificing any future value. In some problems, therefore, for any x_{t+1} that falls completely outside the grid, it is necessary to set $V(x_{t+1})$ equal to a very large negative number. *It is extremely important to be careful in how you handle the edges of your grid in applied dynamic programming; this seemingly small modeling decision can dramatically affect your results.* As a general rule, for problems in which the true state space is unbounded, the grid you use for your numerical model should be specified such that the edges of the grid do not influence the solution inside the grid and all optimal paths lead to points in the interior of your grid.

While rounding is not always the best way to deal with CD problems, it sometimes works out pretty well. Of course, the more points that you have in your grid, the more accurate your rounding estimation will be.

B. The "Curse of Dimensionality"

The problem with tightening your grid is that this means that in each stage you have to solve more state problems. If your state space is multi-dimensional, as you tighten your grid the number of state problems increases geometrically. This problem is known as the

"THE CURSE of DIMENSIONALITY."

The curse of dimensionality refers to the computational problem that arises when the size of the problem grows geometrically with increases in the dimension of the problem, meaning that small increases in dimensions lead to large increases in the computational burden. When this happens, it becomes impossible to solve high-dimensional problems. The term was first used to refer to DP problems. If a DP problem has m state variables, each of which is allowed to take on n values, then you need to solve the Bellman's equation at n^m points in each stage. For example, a rather coarse grid would be to approximate the state space with only 10 points in each dimension. If you have four state variables then your computer algorithm must solve the 10^4 or 10,000 points. If each evaluation takes only $1/10^{th}$ of a second, then each stage would still take 1,000 seconds or 16.7 minutes. Moving from 4 to 5 state variables under the same assumptions would increase each stage loop to 2.7 hours; add one more variable and the run time will be over one day. It would be 317 years before a problem with just 11 state variables completed just one stage (and of course, the memory of your computer would fill up long before a single stage was completed). If you contrast this with the relative freedom that one has when choosing how many variables to put into a linear econometric model, we see that problems of applied dynamic programming are very different.

As we note in Woodward, Wui, and Griffin (2005), despite the incredible increases in the speed of computers, the curse remains very real.

> Although enormous improvements in computational speed have been achieved in recent years, this computational burden will continue to limit the size of DP problems for many years to come. "Moore's law" is the regular tendency for the density of computer circuitry (and processor speed) to double every eighteen months (Schaller). This "law", which has held up surprisingly well since its conception in 1965, has startling implications for simulation modelers: a simulation model could double in size every 1.5 years without slowing down. The implications for DP, however, are not nearly so promising. For example, in a model in which each state variable takes on just 8 possible values, it would be 4.5 years before one additional state variable could be added without increasing the run time of the program. The solution of DP problems with hundreds of state variables lies only in the far distant future.

In many problems, hundreds of iterations of the stage loop are necessary for convergence. Hence, there is an obvious premium on keeping your grid as sparse as possible and, more critically, on keeping the dimension of your state space as small as possible. Since finding precise answers using rounding usually requires the use of a tight grid, this approach has its limitations and smoother options such as those considered below are attractive. Nonetheless, it should be emphasized that the "curse" affects all the approaches considered below, it is only the extent to which these approaches are affected

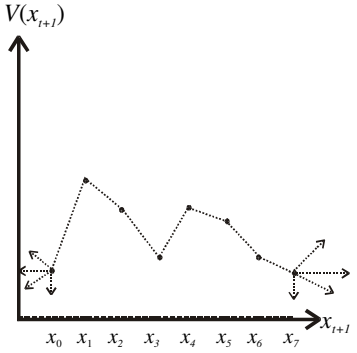
that varies. That is, if you can reduce n, the number of grid points, then the impact of increasing m, the number of dimensions, is not as severe.

There have been some approaches that have attempted to get around the curse. John Rust (1997) proposed an approach that uses rounding in a randomly chosen grid and can be used to solve problems involving very large state spaces. The Approximate Dynamic Programming approach of Powell (2007) represents another approach that also takes advantage of randomization to solve the problem. These approaches are not covered here.

III. Solution #2: Interpolation

A. Linear interpolation (also known as linear splines)

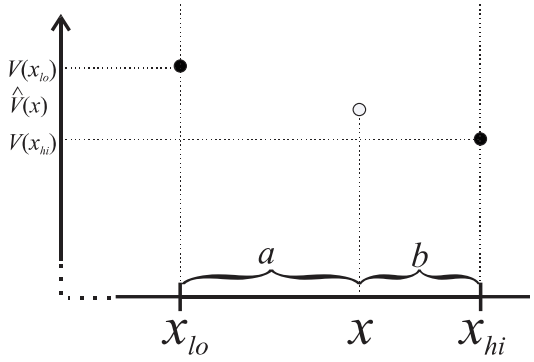
As we have seen above, rounding leads to a step function for the estimated value function. This may not be a big problem, but we can usually do better. One simple way to do better is to use linear interpolation to get estimate the value function at points between those included in our grid, X.



When using linear interpolation, the estimate of the value function is done using a piece-wise linear and continuous function. Again, there is no uniform rule on how to extrapolate beyond the endpoints of the grid. I have indicated this using the multiple arrows in the figure above.

Programming a linear interpolation algorithm is quite easy. The basic elements are presented in the figure below. If we know the value of V at x_{lo} and x_{hi}, then the estimated value at x, which lies between these two points is simply

$$\hat{V}(x) = \frac{b}{a+b} V(x_{lo}) + \frac{a}{a+b} V(x_{hi})$$



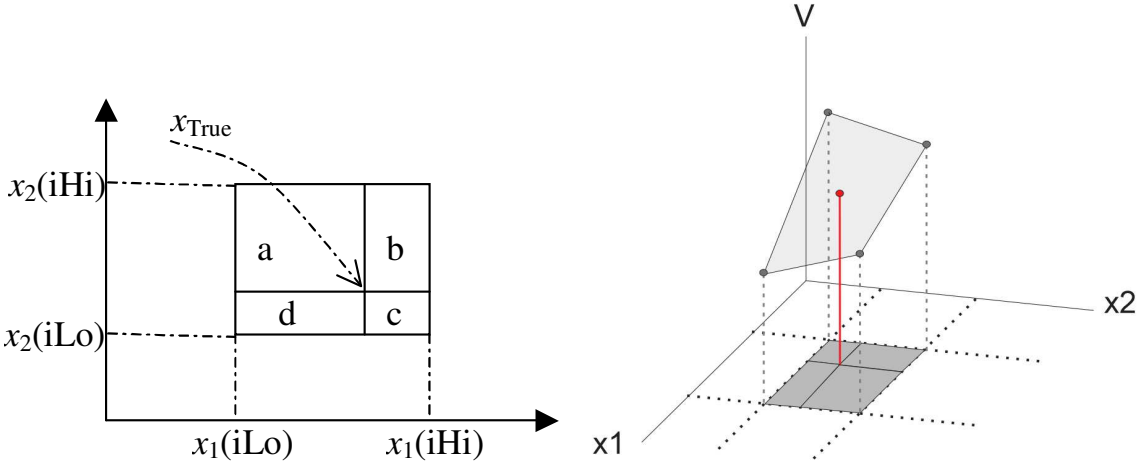
Here's pseudo code to implement linear interpolation for a point xTrue that is in the interior of points in an ordered grid:

```
Find the index of value, in the grid that is closest to xTrue as in the Rounding
algorithm
If xTrue>xGrid(ixGrid) then
    iLo = ixGrid
else
    iLo = ixGrid–1
end if
a = xTrue–xGrid(iLo)
b = xGrid(iLo+1)–xTrue
Vest = VRHS(iLo)*(b/(a+b)) + VRHS(iLo+1)*(a/(a+b))
```

As noted above, boundary violations are very important to address, but the approach can vary greatly from one problem to the next.

Linear interpolation has advantages over rounding, but it still has some limitations. In particular, although the estimated value function is smoother than the rounding approach, the derivatives of the estimated value function are discontinuous. As we will see in the next lecture, this can be problematic if your control variable is also continuous. Also, if the value function is highly nonlinear, then a tight grid will still be needed to obtain a good estimate.

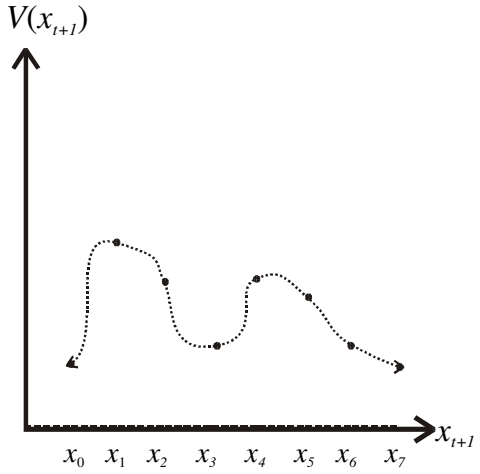
Interpolating in 2 or more dimensions is a straightforward analog to the 1-dimensional case. As seen in the figures below, the 2-dimensional example simply involves calculating the weights a, b, c, and d, which are normalized to sum to 1. The weight placed on each corner is proportional to the area diagonally across from the corner, e.g.,
$V(X_{True}) \cong a \cdot V(x_1(\text{iHi}),x_2(\text{iLo})) + b \cdot V(x_1(\text{iLo}),x_2(\text{iLo})) + c \cdot V(x_1(\text{iLo}),x_2(\text{iHi})) + d \cdot V(x_1(\text{iHi}),x_2(\text{iHi}))$



B. Cubic interpolation or cubic splines

An improvement over linear interpolation is to use cubic interpolation. Cubic splines yield a smooth approximating function something like the one below. In this case, you

are basically interpolating using both the levels *and* the partial derivatives of the function at the points on the grid. I will not present the algorithm for cubic interpolation here. A detailed discussion of the use of cubic splines is available in *Numerical Recipes* a book by Press et al. (1989) (available online at http://www.nr.com/) that contains careful discussion of many numerical techniques and Fortran code for implementing these techniques that could easily be adapted to other languages. Judd (pp. 225-227) also discusses the use of cubic splines.



In each of the approaches discussed so far, the estimate of the value function, say $\widehat{V}\left(x_{t+1}\right)$, is obtained using a finite number of "observations" of V at the points in the state grid, $x \in X$, which were found in the previous stage loop. If x_{t+1} is a number that is not contained in X, then there will be some error, and the expected magnitude of that error declines as we move from rounding, to linear splines, to cubic splines. Moreover, if x_{t+1} is a value contained in the grid X then there will be no estimation error.

Matlab has an intrinsic function, `spline`, that carries out cubic interpolation. For example, if you have an array of grid points, `xgrid`, and a collection of values `VRHS`, with values for each point in `xgrid`, then to estimate the value at a point, `xTrue`, you can simply write `Vest = spline(xgrid, VRHS, xTrue)`.

C. Shape-preserving splines

Cai and Judd (2012, 2015) propose an approach that they show is efficient and is likely to give quite reliable results. Their approach is only currently derived for problems with a single state variable, so that case is presented here. The authors present an interpolating algorithm that retains the curvature of the true value function throughout the successive approximation algorithm. This is done by adding a piece of information to the interpolation algorithm – the slope of the value function at the point at which it is evaluated.

I present the case of a one-dimensional state variable here from Cai and Judd (2012). The approach is generalized to multiple state variables in Cai and Judd (2015). Their approach starts with solving a modified Bellman's equation,

$$V^{LHS}(x_i) = \max_{z,y} u(y,z) + \beta E V^{RHS}(x^+)$$

$$s.t. x^+ = g(y,z,\varepsilon) \text{ and } y = x_i$$

for each of the points in the state space, x_i, for $i = 1,\ldots n$, wherein the first iteration, $V^{RHS}(x)$ is set at your best first guess (e.g., 0) for the true value function.

The difference between this problem and the standard Bellman's equation is that instead of x_i on the RHS of the Bellman's equation, we have y, which is treated as a choice variable. But y is constrained to equal x_i, so why do we bother? The answer is that by solving the problem in this way we also can find the Lagrange multiplier, say μ_i, the shadow value of the constraint that $x_i = y$. This shadow value tells us the slope of the value function at x_i. Hence, by the end of the stage loop, we have obtained a set of values, say v_i and μ_i, for $i = 1,\ldots,n$, where $v_i = V^{LHS}(x_i)$. The set of values $\{(x_i, v_i, \mu_i): i = 1\ldots m\}$ are called Hermite data, which can then be used to solve the next iteration of the algorithm as follows:

$$V^{RHS}(x;c) = c_{i1} + c_{i2}(x - x_i) + \frac{c_{i3}c_{i4}(x - x_i)(x - x_{i+1})}{c_{i3}(x - x_i) + c_{i4}(x - x_{i+1})} \text{ for } x \in [x_i, x_{i+1}]$$

where $c_{i1} = v_i$, $c_{i2} = \dfrac{v_{i+1} - v_i}{x_{i+1} - x_i}$, $c_{i3} = \mu_i - c_{i2}$, and $c_{i4} = \mu_{i+1} - c_{i2}$.

You can see that c_{i1} is the value of $V^{RHS}(\cdot)$ at x_i, c_{i2} is the slope of V^{RHS} between x_i and x_{i+1}, and c_{i3} is a measure of the function's curvature over the range from x_i to x_{i+1}.

The algorithm works for all points between the minimum and maximum values, x_1 and x_n. They require that the bounds on the state variable are chosen so that points outside of the grid rarely need to be evaluated. For points that fall outside the grid, an arbitrary rule must be established or the values could be rounded to the nearest endpoint.

IV. Solution #3: Functional approximation

The next set of solution methods assume that there is an underlying continuous function that describes the value function. In this case, the analyst assumes the functional form and the DP algorithm is used to identify its parameters. This differs in an important way from the methods we have considered so far. Up to now the Bellman's equation in the k^{th} iteration (i.e. k^{th} stage) was calculated using the values of V that you found in iteration $k-1$. In the functional approximation approach, the value function on the RHS of the Bellman's equation is defined not by a set of values at fixed points in the state grid, but by a set of parameters – the $k-1^{th}$ set of coefficients of the assumed functional form. The updating step between each stage loop, therefore, involves finding a new set of coefficients for the value function. The test of convergence in a successive approximation algorithm might be based on the extent to which the coefficients change from one stage to the next, though it is important to be aware that the scale of these coefficients might be very important.

A. Functional approximation using ordinary polynomials

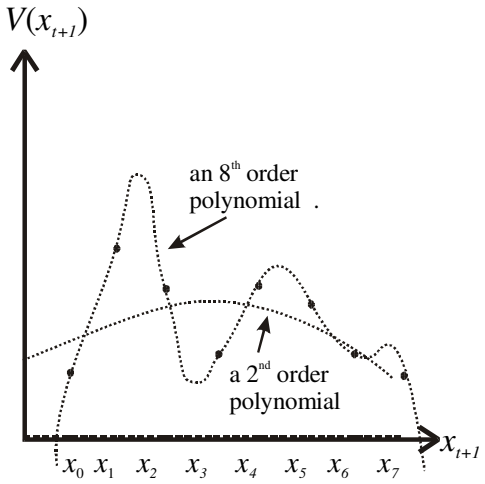
The first functional approximation method that we consider is the use of ordinary polynomials. For example, you may assume that the value function can be closely approximated by a second-order Taylor series approximation, i.e.,

$$\hat{V}\left(x_{t+1}\right) = c_0 + c_1 x_{t+1} + \frac{c_2}{2}\left(x_{t+1}\right)^2 .$$

In this case, your problem becomes one of choosing the parameters c_0, c_1, and c_2 at each iteration. Let c^k be the vector of coefficients of the value function in the k^{th} iteration of a successive approximation algorithm. $\hat{V}\left(x_{t+1}; c^k\right)$ is then the estimated value function conditional on the parameters c^k. Then the k+1$^{\text{th}}$ set of parameters would be found in two steps. First, solve the problem $V\left(x_t\right) = \max_z E\left[u\left(z_t, x_t\right) + \beta \hat{V}\left(x_{t+1}; c^k\right)\right]$ at every point in your grid. Then, use the values $V\left(x_t\right)$ like data to find the new set of coefficients, c^{k+1}, that give you the best possible approximating function. How might this be done? Well OLS is an option. Hence, we can get a new set of parameters, c^{k+1}. Each stage, therefore represents a mapping from c^k to c^{k+1}.

One significant advantage of using this approach is that we then have a closed-form expression for our estimated value function $\hat{V}\left(x_{t+1}; c^k\right)$. Evaluating a point on this line is as simple as plugging it into the equation with the most recent set of parameters. Moreover, it is likely that the first derivative, $\left(\partial \hat{V}\left(x_{t+1}; c^k\right) / x_{t+1}\right)\left(\partial x_{t+1} / \partial z_t\right)$ also can be expressed analytically meaning that it may be possible to find analytical closed-formed solutions to the Bellman's equation at each point in the state space, especially if the choice variable is also continuous (CC). Hence, rather than solving the Bellman's equation numerically using a grid-search or hill-climbing algorithm (see below), it would be possible to simply find the optimal choice, $z^*(x)$, as a function of the parameters of the model (state equation and benefit function) and the parameter vector, c.

Using ordinary polynomials as we have in the case above, however, has significant limitations and is ***not*** recommended. As seen in the figure below, for low-order polynomials, the assumed functional form places very strong restrictions on the form that $\hat{V}\left(x_{t+1}; c^k\right)$ might take. Hence, if the true value function is highly nonlinear, the estimate using a 2nd-order polynomial would be quite inaccurate. As the order of the polynomial rises, $\hat{V}\left(x_{t+1}; c^k\right)$ will get closer and closer to the points $V(x_t)$ on the grid. However, the errors at points between the values on the grid can actually rise and extrapolation beyond the grid is extremely dangerous.

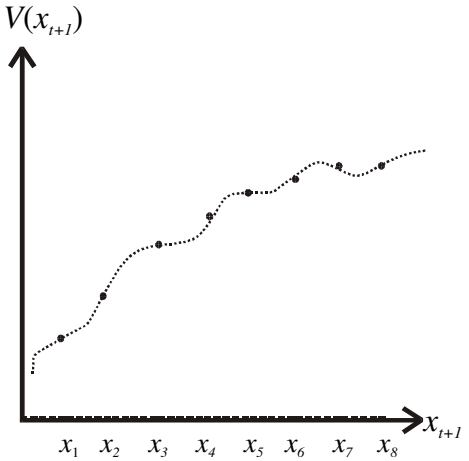


As will be discussed in section C below, there are alternative polynomial forms that are far superior to the ordinary polynomials used here.

B. Functional approximation using prior knowledge about the functional form of V

If a modeler uses the polynomial approach to approximating the value function, we can say that he or she has assumed that the value function will be taking a particular functional form. In this case, however, the functional form is arbitrarily chosen to make analysis easy and/or the error between $V(x_t)$ and $\hat{V}\left(x_{t+1};c^k\right)$ small. In some problems, the modeler can use prior knowledge regarding the function form of $V(x_t)$.

If it is known that $V(x_t)$ is of the form $\tilde{V}\left(x_t;c\right)$, with parameters c, then the successive approximation algorithm can be implemented in the same way as was done for ordinary polynomials, stepping from c^k to c^{k+1}.



Judd (p. 437-438) points out that it may be very important to use information about the value function if you have it. For example, he considers the case of points that are strictly increasing as in the figure. A cubic spline might lead to an approximating function that is highly nonlinear like the one indicated by the line in the figure. This can lead to quite erroneous outcomes since it indicates, for example, that although $V(x_8)>V(x_7)$, the

estimated value of almost all the points between x_6 and x_7 exceeds the value of the points between x_7 and x_8.

Hence, if you know that the true value function is monotonic or concave, choosing an approximation method that preserves those characteristics can avoid errors.

C. Functional approximation using Chebyshev polynomials

If the modeler is interested in using a functional approximation method but does not have prior knowledge of the form of $V(\cdot)$, the use of polynomials is still a possibility. While ordinary polynomials can give very large errors, the *Chebyshev polynomial* is a polynomial with an unintuitive functional form but very attractive numerical properties. As noted by Press et al. (1989),

> The Chebyshev approximation is very nearly the same polynomial as the holy grail of approximating polynomials the *minimax polynomial*, which (among all polynomials of the same degree) has the smallest maximum deviation from the true function $f(x)$. (p. 149)

The computation of Chebyshev polynomials is complicated and unintuitive, but it is relatively easy to implement. For details, I refer you to *Numerical Recipes* or many other sources online.

As noted by Cai and Judd (2013), "methods of fitting a curve to the data, like Chebyshev polynomial approximation, may produce a non-concave value function approximation, which in turn may lead to nonconcavity of the objective function in the maximization step and then instability in [the DP] Algorithm" (p. 409-10). They propose a modification of the standard Chebyshev polynomial in which curvature restrictions such as monotonicity and concavity can be imposed.

V. Setting up your grid

An important modeling decision that you must make if you are solving problems with a continuous state variable using any of the above techniques is how your grid will be established. Regardless of the method chosen to approximate the value function, a tighter grid will lead to a more precise estimate of your final solution.

There is no general rule to guide how you should set up your grid and how tight you should make it. In many problems, a uniform grid (e.g. x_1=0.1, x_2=0.2, x_3=0.3,…) is as good as any. In other cases, if you know that the probability of hitting a particular range in the grid is high, then you'll want to have more grid points in that range than in a range where there is a very low probability of actually ending up. However, if the relative values of these low probability ranges are very high and, therefore, important to get the correct answer, then the grid may need to be tight in that area as well. If you use Chebyshev polynomials, then the grid must be set up in a very precise way.

How tight your grid is, i.e., the value of *nx* is typically a decision that you make based on practical concerns. You do not want your results to be sensitive to the size of the grid, so you should tighten your grid until further tightening does not affect your results anymore,

an obviously subjective decision. However, if your problem is large (i.e. you have lots of state variables), then tightening your grid may add hours or days to the time it takes your program to run. Clearly, practical considerations regarding the tradeoff between computing time and precision also enter into the choice of the grid.

There has also been work that uses non-standard grids. A series of papers by Grune (e.g., Grune and Semmler, 2004) use adaptive grid schemes that focus on points where there is the greatest need for precision. Woodward Wui and Griffin (2005) use a uniform but non-rectangular grid in their approach, focusing only on the portion of the state space where the decision process tends to reside.

Part 2: Continuous Choice Variables

VI. The additional difficulties associated with continuous choice variables

As we saw above, some important problems arise when the state space of a dynamic programming problem is continuous. Additional complications arise when the control variable(s) that you are trying to model are continuous.

Before we start talking about continuous controls, it's probably worth pointing out that many control variables that are relevant in economics are **not** continuous. The cow replacement problem and the option price problem considered in Lecture 10 are two good examples. These problems are typically referred to as "optimal stopping problems." In those cases, the decision was binary (replace or not, exercise or not). However, many if not most economic decisions are continuous, not discrete -- how much to consume, how much to produce, how much of an input should be used, etc. In such problems the question is not simply whether or not a particular action should be taken, but *the level* at which that action should be taken.

Recall the backward-recursion algorithm for solving finite and infinite horizon DP problems is as follows: For each stage (t=T, T−1, T−2, …,0 for finite horizon problems; k=1, 2, … for infinite horizon problems) we want to find the value of each point in the state space. InTodentify the value at each point in the state space, we need to solve a maximization problem -- identify the choice variable z_t that maximizes

3. $E\left[u\left(z_t, x_t, \varepsilon_t\right) + \beta V\left(x_{t+1}\right)\right]$, where $x_{t+1}=g(z_t,x_t,\varepsilon_t)$.

When the choice variable is discrete, this is easy - we just try all the values and see which one is the best. But when the choice variable is continuous, it is impossible to check every possible value using a computer. We will now explore how you might address this difficulty in practice.

VII. Methods for solving CC problems

A. Discretize the control space

When your choice variable is continuous, the simplest approach is to treat it as if it were a discrete variable. Suppose, for example, that you were interested in a variable z that can take on any value between zero and one. Instead of using the infinite number of values between zero and one, perhaps you can get a sufficiently precise answer by only looking

at the n+1 values, say $Z = \left\{0, \frac{1}{n}, \frac{2}{n}, \cdots \frac{n-1}{n}, 1\right\}$. By treating your variable as if it were discrete, you have greatly simplified your problem and you can now solve each state problem by simply evaluating which of these n+1 options is the best.

How tight should your control grid be? Again, this is a subjective decision that depends on your needs. If you are interested in the qualitative results of your model, i.e., the general trends, then you should tighten your grid until any further tightening does not alter the qualitative features of your results. If you need precise results, then your grid will probably need to be much tighter.

The tighter your grid, the more precise will be your results. However, a tighter grid means a slower program. Let's take a simple example. If you have one choice variable, then doubling the number of points in your control grid will approximately double the time it takes your program to run. If you have two control variables, then doubling the number of points in each dimension will quadruple your run time. An n-fold increase in the grid points of a problem with m choice variables will increase your run time by a factor of n^m. If you have two control variables and your program takes 30 seconds to run with 10 points in each choice grid, then increasing your grid to 100 in each dimension would increase your run time to 3000 seconds, 50 minutes!

> Tip: If you use the discretization approach, debug your program with a quite sparse grid (relatively few points) and then increase your precision once your program is running and you think it is giving you the correct answers.

B. Using a hill-climbing algorithm

Most programming languages in which economists work (e.g., Matlab, C, Gams, Python) have built-in or downloadable plug-ins that solve optimization problems for continuous variables. They do this by carrying out an organized search over the continuous range of your choice variables between user-defined upper and lower bounds. Although modern algorithms are more sophisticated, Newton's method is the first such approach and still works remarkably well. Starting with an initial guess, hill-climbing algorithms make successively better guesses of the optimum using the function and its derivatives.

A hill-climbing algorithm can be plugged into your solution algorithm at the point where the control loop normally fits. This approach has some advantages and disadvantages when compared to a grid-search method. First, good hill-climbing algorithms approximate a continuous variable, so that it is possible to come very close to the exact solution to the state problem in every loop. Secondly, particularly given their accuracy, these algorithms can be quite fast. Having to loop over a very fine grid of control options would be comparatively quite slow.

Recall that we would use the hill-climbing algorithm to solve the problem,
$V(x_t) = \max_{z_t} E\left[u(z_t, x_t, \varepsilon_t) + \beta V(x_{t+1})\right]$
subject to the state equation $x_{t+1}=g(z_t,x_t,\varepsilon_t)$.

Hill climbing algorithms use information about the slope and curvature of a function to iteratively find the solution to an optimization problem. In the context of solving Bellman's equation, this means that the algorithm needs to know (or calculate)

$$\hat{V}(x_t, z_t) = E\left[u(z_t, x_t, \varepsilon_t) + \beta V(x_{t+1})\right],$$

$$\frac{\partial \hat{V}(x_t, z_t)}{\partial z_t} = E\left[\frac{\partial u(z_t, x_t, \varepsilon_t)}{\partial z_t} + \beta \frac{\partial V(x_{t+1})}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial z_t}\right],$$

$$\frac{\partial^2 \hat{V}(x_t, z_t)}{\partial z_t^{\,2}} = E\left[\frac{\partial u^2(z_t, x_t, \varepsilon_t)}{\partial z_t^{\,2}} + \beta \partial\left(\frac{\partial V(x_{t+1})}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial z_t}\right) \Big/ \partial z_t\right],$$

$$\frac{\partial^2 \hat{V}(x_t, z_t)}{\partial z_t^{\,2}} = E\left[\frac{\partial u^2(z_t, x_t, \varepsilon_t)}{\partial z_t^{\,2}} + \beta\left(\frac{\partial^2 V(x_{t+1})}{\partial x_{t+1}^{\,2}}\left(\frac{\partial x_{t+1}}{\partial z_t}\right)^2 + \frac{\partial V(x_{t+1})}{\partial x_{t+1}} \frac{\partial^2 x_{t+1}}{\partial z_t^{\,2}}\right)\right].$$

Some algorithms that you can use will allow you to supply code that calculates $\hat{V}(x_t, z_t)$ and then will automatically calculate the first and second derivatives numerically by using small shocks such as

$$\frac{\partial \hat{V}(x_t, z_t)}{\partial z_t} \approx \frac{\hat{V}(x_t, z_t + \varepsilon) - \hat{V}(x_t, z_t - \varepsilon)}{2\varepsilon}.$$

If you can provide an analytical expression for $\frac{\partial \hat{V}(x_t, z_t)}{\partial z_t}$ and $\frac{\partial^2 \hat{V}(x_t, z_t)}{\partial z_t^{\,2}}$, the algorithm will be much faster. This is one advantage of using a functional approximation of the Bellman's equation. Hill-climbing algorithms can also run into problems if $\frac{\partial \hat{V}(x_t, z_t)}{\partial z_t}$ or $\frac{\partial^2 \hat{V}(x_t, z_t)}{\partial z_t^{\,2}}$ change discontinuously. Hence, the algorithm will usually behave poorly if rounding or linear interpolation methods are used.

A disadvantage of a hill-climbing approach lies in the certainty that one can have in your solution to each state problem. The recursive algorithm for solving finite- or infinite-horizon problems requires that this problem be ***solved*** at each point in the state space; that is, ***the correct*** answer must be obtained. It is well known that for nonlinear problems, numerical hill-climbing algorithms may not yield the correct solution. If a "black box" is used, a computer program into which most users cannot look, one does not have complete confidence that a global maximum has been achieved. Hence, our confidence in our final results is diminished if this method is applied and you are advised to include in your code checks to ensure that the algorithm is consistently finding the global optimum.

C. Closed-form solution for particular functional forms[1]

If it is assumed that the value function takes on a particular functional form, e.g., a polynomial, then it is sometimes possible to find a closed-form solution to the Bellman's equation. Let's look at this in more detail.

Suppose, for example, we assume or know based on some prior information that the value function of a two-dimensional DP problem takes the form

4. $V(x_1, x_2) = a_{00} + a_{10}x_1 + a_{20}x_1^{\ 2} + a_{01}x_2 + a_{02}x_2^{\ 2} + a_{11}x_1x_2$

where the a_{ij} are parameters that we need to identify. The k+1$^{\text{th}}$ approximation of the value function is found by solving the problem

5. $$\begin{cases} V^{k+1}(x_1, x_2) = \max_z E\left[u(z_t, x_t, \varepsilon_t) + \beta \cdot \left(a_{00}^k + a_{10}^k g^1(\cdot) + a_{20}^k g^1(\cdot)^2 + a_{01}^k g^2(\cdot) + a_{02}^k g^2(\cdot)^2 + a_{11}^k g^1(\cdot) g^2(\cdot) \right) \right] \\ \text{with } x_{1t+1} = g^1(z_t, x_t, \varepsilon_t) \text{ and } x_{2t+1} = g^2(z_t, x_t, \varepsilon_t). \end{cases}$$

where a_{ij}^k is the k^{th} approximation of the ij^{th} coefficient of the value function.

What is particularly attractive about this specification is that for relatively simple probability distributions and state equations, closed-form solutions for the optimal policy function of the i^{th} choice variable, say $z_i^*(x;a)$. This policy function indicates the best choice as a continuous function of all possible values of the state variables, contingent on a particular set of coefficients of the value function, a=$\{a_{ij}\}$.

If an analytical representation of the policy function, $z_i^*(x;a)$, can obtained, then two approaches might be taken. First, a numerical approach could be taken in which a set of grid points, X, so that the values $V(x)$ can be calculated explicitly for all $x \in X$ by plugging $z_i^*(x;a)$ into 5. With this set of values, the k+1$^{\text{th}}$ set of coefficients could be determined using, for example, OLS approximation.

Alternatively, an analytical representation of the value function $V^{k+1}(x)$ can be found. Since this almost certainly would not take the same form as in 4, Androkovich and Stollery (1994) suggest taking a second-order Taylor series approximation of the value function, to obtain a new set of coefficients, a_{ij}^{k+1}. In either case, the solution to the infinite horizon problem could be found by iterating until $\left| a_{ij}^{k+1} - a_{ij}^k \right| < \Delta$ for some critical value Δ.

---

[1] I've only seen this method applied once, in a paper by Androkovich, Robert A. and Kenneth R. Stollery. 1994, "A Stochastic Dynamic Programming Model of Bycatch Control in Fisheries" *Marine Resource Economics* 9:19-30. Nonetheless, it's an interesting approach and helps highlight how we solve DP problems in practice.

On the other hand, if the true underlying value function cannot accurately be depicted using a second-order polynomial like 4, then this approach will lead to erroneous results. Moreover, the approach is intrinsically inconsistent in that they never obtain a value function $V(x)$ such that $V(x_t) = \max_{z_t} E\left[u(z_t, x_t, \varepsilon_t) + \beta V(x_{t+1})\right]$.

D. Other approaches

Two other approaches are frequently used to solve CC problems, these are what I'll call Euler equation iteration and linear quadratic approximation methods. Both of these draw on the fact that CC problems are differentiable. Before introducing these, it will be useful to go over a little theory.

VIII. A slight detour -- Numerical integration over continuous probability density functions

We have not yet covered the basic principles of numerically taking expectations with continuous probability distributions. Hence, I provide here a very quick overview of some methods. Further development is available in Chapter 7 of Judd (1998), Chapter 5 of Miranda and Fackler's text, and Chapter 4 of Press et al.[2]

Suppose you want to take an expected value from a continuous distribution using a computer. That is, you hypothesize that the underlying distribution of your random variable, e, is continuously distributed, say normally with mean $\bar{e}$. The PDF of the variable, $f(e)$, therefore, would look like the figure below.

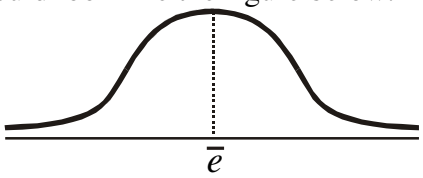


Figure 1

A. Numerical integration using a uniform grid

The expected value of some function, $u(e)$ with a probability density function $f(e)$ is simply $\int_{-\infty}^{+\infty} u(e) f(e) de$. The computational problem is that we do not have a closed-form expression for this integral. Hence, numerical approximation methods must be used. The most simplistic way to deal with this problem is simply to divide the range of e into a grid and then calculate the probability of falling into each portion of the grid. This process is demonstrated in the figure below.

[2] Miranda and Fackler's notes are probably the easiest to read option of the three sources noted. Press et al.'s *Numerical Recipes for Fortran 77* are also quite readable and have the advantage of including well commented Fortran 77 code, which you should be able to translate into VB or any other language. These can also be accessed through the internet at (http://www.nr.com/).

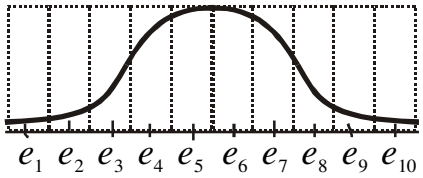


Figure 2

In this case, the expected value of u(e) would be approximated using the function $\sum_{i=1}^{10} u(e_i)w(e_i)$, where $w(e_i)$ is the probability weight associated with the grid cell centered at e_i. The value of $w(e_i)$ is equal to the area under $f(e)$ in the grid box centered at e_i with an adjustment to account for the fact that we have truncated off the ends of the distribution, i.e., $w(e_i) = \frac{\int_{\underline{e}_i}^{\bar{e}_i} f(e)\,de}{\int_{\underline{e}_1}^{\bar{e}_{10}} f(e)\,de}$, where $\bar{e}_i$ and $\underline{e}_i$ are the upper and lower bounds on the grid cell centered at e_i.

This is a fairly straightforward process and you could even use a spreadsheet to generate values for $w(e_i)$ for any grid size.

B. Numerical integration using non-uniform grids

While the uniform grid approach is quite intuitive, it is not very efficient. For example, it provides just as much information about the points e_1 and e_{10} as it does about e_5 and e_6, but e_5 and e_6 are in the center of the distribution and thus are more important in our expectation. An efficient algorithm would spread out the cells to get as precise an estimate of the true expectation as possible for any fixed number of grid points.

Numerous methods are used to accurately approximate a continuous integral. Gaussian Quadrature methods are efficient methods for integrating smooth functions. For a detailed discussion of these methods, I refer to the above-mentioned sources.

The basic idea in Gaussian Quadrature methods is that the points are chosen wisely so that a more accurate approximation of the expectation can be achieved. The basic principle is seen in Figure 3. Grid points towards the tails are spaced further apart than the grid points near the mean, as in the figure below (though the differences are exaggerated here).

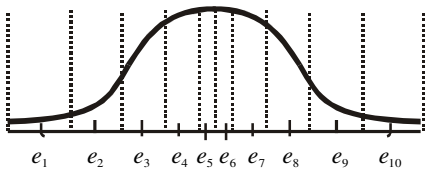


Figure 3

The formulas that are used to calculate the values for e_i and $w(e_i)$ are quite complicated and involve some pretty tricky programming but well-tested subroutines are available for your use.

C. A programming note

Solving a stochastic DP problem involves finding, in each stage and for each state, the choice z that maximizes $E[u(z,x,\varepsilon)+\beta V(x_{t+1})]$. There are two ways you might address this problem in your program. Suppose ε takes on only two values, say ε_1, ε_2, with probabilities p_1 and p_2. With only a small number of probabilities you might take your expectation directly using commands such as the following:

u1 = $u(z,x,\varepsilon_1)$
xnext1 = $g(z,x,\varepsilon_1)$
Vnext1 = V(xnext1)
u2 = $u(z,x,\varepsilon_2)$
xnext2 = $g(z,x,\varepsilon_2)$
Vnext2 = V(xnext2)
EV = p_1*(u1+βVnext1)+p_2*(u2+βVnext2)

You would then compare EV with $V(x)$ and, if it is better, store it; if not, move on to the next value of z.

When you have a vector of values for your state variable, say `xgrid`, and you have the value at each point in that grid, `VRHS(xgrid)`, a corresponding vector of probabilities, say `pgrid`, the expected value can be obtained through simple expectation: `VEST = pgrid'VRHS`.

IX. A little theory about infinite-horizon problems

A. The Euler Equilibrium conditions

The key theoretical feature that distinguishes CC problems from problems with discrete choices is the ability to apply the standard principles of differential calculus to the problem. The Bellman's equation of an infinite horizon problem takes the following form:

$$V(x_t) = \max_{z_t} E\left[u(z_t, x_t, \varepsilon_t) + \beta V(x_{t+1})\right]$$

where z, x, and ε can be scalars or vectors and $x_{t+1}^i = g^i(z_t, x_t, \varepsilon_t)$, for each state variable, $i=1,\ldots,m$. If the functions u and all the g^i are differentiable in z and x, and $V(\cdot)$ is differentiable in x, then we know that, for an unconstrained DP problem, the first-order conditions would be satisfied at the optimum for each choice variable z_j, i.e.,

$$E\left[\frac{\partial u}{\partial z_j} + \beta \sum_{i=1}^{m} \frac{\partial V}{\partial x_i}\frac{\partial g}{\partial z_j}\right] = 0.$$

Letting $\frac{\partial V}{\partial x_i} \equiv \lambda_i$ and applying the envelope theorem to the problem,

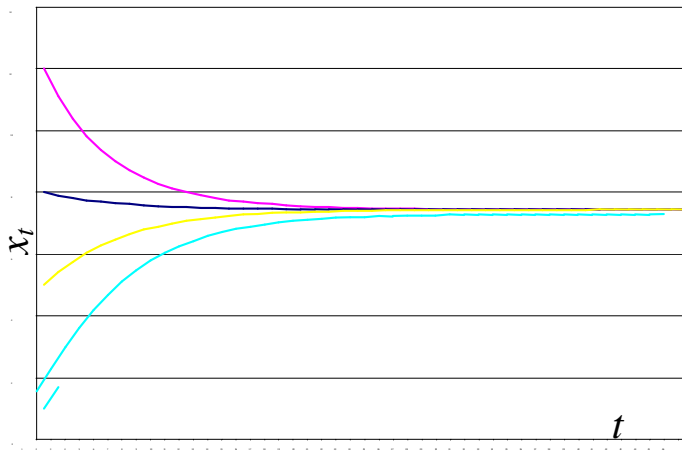
6. $\lambda_i = E\left[\frac{\partial u}{\partial x_i} + \beta\sum_{j=1}^{m}\frac{\partial V}{\partial x_j}\frac{\partial g^j}{\partial x_i}\right] = E\left[\frac{\partial u}{\partial x_i} + \beta\sum_{j=1}^{m}\lambda_j\frac{\partial g^j}{\partial x_i}\right]$.

The equations in 6 are typically referred to as the ***Euler conditions***. You should see in them a close similarity to the maximum conditions of optimal control. In particular, if you look at Dorfman's derivation, you'll find the deterministic version of the conditions we have here.

If the problem is subject to intratemporal constraints, the Euler conditions would be altered to reflect the Kuhn-Tucker conditions, but the intuition is fundamentally the same.

B. The steady state and certainty-equivalent steady state of CC problems

For many deterministic DP problems, the optimal strategy will lead to a steady state. That is, following the policy rule set out by your policy function, $z^*(x)$, will lead to an evolution in the state space that will lead to a steady state. Consider, for example, a simple problem in which the optimal policy function takes a linear form, $z^*(x)=\alpha x$ and $x_{t+1}=g(x_t,z_t)=x_t+(x_t)^\gamma-z_t$. In this case, following the optimal policy would lead the state variable to a unique steady state value as in the figure below from any initial starting value.



An appreciation of the steady state can be quite useful in understanding a problem. This is particularly true when the steady state is reached quickly so that it can be safely assumed that the agents you are studying will probably be at the steady state at any time.

The steady state of infinite horizon problems with m state variables and n control variables can be found by solving three sets of equations,

$$E\left[\frac{\partial u}{\partial z_j} + \beta\sum_{i=1}^{m}\lambda_i\frac{\partial g}{\partial z_j}\right] = 0 \text{ for } j=1,\ldots,n$$

$$\lambda_j = E\left[\frac{\partial u}{\partial x_j} + \beta\sum_{i=1}^{m}\lambda_i\frac{\partial g^i}{\partial x_j}\right] \text{ for } j=1,\ldots,m, \text{ and}$$

$$x_j = g(z,x) \text{ for } j=1,\ldots,m.$$

That is, at the optimum steady state, the FOCs of the problem must be satisfied and the state variables must not be changing over time. Note that solving for the steady state does not require knowledge of V, instead, it is based on information about the slope of the value function at the steady state, λ. While it still may be impossible to analytically solve this system of equations for closed-form expressions for the variables z_i, x_j and λ_j, this system of equations is well specified and it should be possible to solve the system numerically (see Judd, 1998 chapter 5).

X. Solution methods for CC problems that utilize the optimality conditions

A. Linear quadratic (LQ) approximation[3]

Another method that has been used to solve CC DP problems is to assume that the problem that you're interested in solving falls into a class of problems for which a nice clean solution exists. If the state equations, $g^i(\cdot)$, are linear in z and x and the benefit function, $u(\cdot)$, is quadratic, then it is possible to find an analytical solution to stochastic DP problems. This has led to a great deal of analysis of these types of problems. In many presentations of the material covered in this class, LQ problems are presented separately and analyzed in depth. In these notes, such specifications are given substantially less emphasis as I see it as one more means of finding an approximate solution to a true DP problem. If the true problem that you want to solve fits the LQ requirements, then it should obviously be solved using the LQ methods. However, if your problem does not meet these quite restrictive conditions, then using this approach is just one more way to find an approximate solution to your true underlying problem. In some instances, particularly in the neighborhood of the certainty-equivalent steady state, this approach might be quite useful.

1. In an LQ problem, the benefit function takes the form of a 2nd order polynomial:

$$u(x,z) = A_0 + A_1 x + A_2 z + \tfrac{1}{2} x' A_3 x + x' A_4 z + \tfrac{1}{2} z' A_5 z ,$$

where z and x are $n\times 1$ and $m\times 1$ vectors, A_0 is a scalar, A_1 and A_2 are $1\times m$ and $1\times n$ vectors, and A_3, A_4 and A_5 are conformable matrices.

2. The state equations in the LQ setup are linear functions in the state variable and control variable

$$x_{t+1} = G_0 + G_1 \cdot x_t + G_2 \cdot z_t + \varepsilon_t$$

where G_0, G_1 and G_2 are $m\times 1$, $1\times m$ and $1\times n$ vectors and ε_t is an $m\times 1$ vector of random shocks with zero mean.

What makes these types of problems particularly important is that they can be solved explicitly. The optimal policy function and shadow price function (λ) are linear functions of the state variables:

$$z(x) = Z_0 + Z_x x$$

$$\lambda(x) = \Lambda_0 + \Lambda_x x.$$

---

[3] Details of this section are taken from Miranda and Fackler (1999).

in which Z_0 is an $n\times1$ vector, Z_x is an $n\times m$ matrix, Λ_0 is an $n\times1$ vector and Λ_x is an $n\times m$ matrix.

The parameter matrices Λ_0 and Λ_x are characterized by the nonlinear Riccati equations. Riccati equations are fixed-point equations that define the coefficients of the $z(x)$ and $\lambda(x)$ above. The elements of the matrices Λ_0 and Λ_x appear on both the right- and left-hand sides of the Riccati equations. The solution of these equations is discussed in Judd (1998, p. 432) and in Miranda and Fackler.

One thing that is particularly interesting about the solution to these problems is that the solution is independent of the type of stochastic shock. Regardless of the distribution of the shock, the problem will have the same solution.

B. Using LQ approximation around the certainty-equivalent steady state

One way that the LQ method can be particularly useful is to describe the behavior of a system around the certainty-equivalent steady state (CESS). In this case, the first step is to find the variables at the CESS, say x^*, λ^*, and z^*.

The second step is to take first- and second-order Taylor series approximations of the state equations and benefit function respectively at the CESS.

The third step is to then solve the approximate LQ problem. The resulting solution should yield quite reasonable estimates of the optimal policies in the neighborhood of the CESS. This could then be used to analyze the behavior of the system in the long run. For example, this approach might give a quite accurate approximation of the long-term reaction to a one-period policy change.

LQ methods have advantages and disadvantages when compared to methods that rely on approximating the value function. The numerical methods give an approximate solution to a problem very close to the one you're interested in. LQ methods give an exact solution to a problem that is a rough approximation of the one you're interested in.

You choose your poison.

C. Euler equation iteration

The standard value function iteration approach uses successive approximations of $V(x)$, using one guess at the value function, say $V^k(x)$, to obtain the next guess, $V^{k+1}(x)$. This is repeated until convergence is achieved at a value function $V(x)$ that can then appear on both the right and left-hand sides of Bellman's equation

$$V(x_t) = \max_{z_t} E\left[u(z_t, x_t, \varepsilon_t) + \beta V(x_{t+1})\right].$$

An alternative approach is to use a similar successive approximation algorithm on the Euler equations. In this case, the unknown that we need to successively approximate is the co-state variable $\lambda(x)$.

The algorithm follows a pattern much as we do with the successive approximation of the value function

1. **Initialization step**: make an initial guess of the values of λ at each point in your grid, say $\lambda_i^0(x)$.

Then, for *k*=1, 2,…

2. **Update the policy function**: For each point in your state grid, solve the system of equations for a set of candidate optimal policies, say $\hat{z}$

$$E\left[\frac{\partial u(\hat{z},x,\varepsilon)}{\partial z_i}+\sum_{j=1}^{m}\lambda_j^{k-1}(x)\frac{\partial g(\hat{z},x,\varepsilon)}{\partial z_i}\right]=0$$

This system of equations could be solved numerically.

3. **Update the costate variables:** these candidate policies for each point in your grid are then plugged into the Euler equations $\lambda_i^k = \partial V^k / \partial x_i$ to obtain an updated value for λ at each point in the grid, i.e.,

$$\lambda_i^k(x)=E\left[\frac{\partial u(\hat{z},x,\varepsilon)}{\partial x_i}+\sum_{j=1}^{m}\lambda_j^{k-1}(x)\frac{\partial g(\hat{z},x,\varepsilon)}{\partial x_i}\right].$$

4. **Convergence check:** Calculate $\Delta\lambda(x)=\left|\lambda_j^k(x)-\lambda_j^{k-1}(x)\right|$ for all *x*. If $\Delta\lambda(x)$ is "small" for all *x*, then stop. If not, return to step 2 and continue.

As with value function approximation, a variety of methods can be used to approximate the function $\lambda(x)$, including rounding, interpolation, or functional approximation.

XI. References

Androkovich, Robert A. and Kenneth R. Stollery. 1994. A Stochastic Dynamic Programming Model of Bycatch Control in Fisheries. *Marine Resource Economics* 9(1):19-30.

Cai, Yongyang, and Kenneth L. Judd. 2012b. Dynamic programming with shape-preserving rational spline Hermite interpolation. *Economics Letters* 117(1):161-164.

Cai, Yongyang, and Kenneth L. Judd. 2013. Shape-preserving dynamic programming. *Mathematical Methods of Operations Research.* 77:407-421.

Cai, Yongyang, and Kenneth L. Judd. 2015. Dynamic programming with Hermite approximation. *Mathematical Methods of Operations Research* 81(3):245-267.

Grune, Lars, and Willi Semmler. 2004. Using dynamic programming with adaptive grid scheme for optimal control problems in economics. *Journal of Economic Dynamics and Control* 28(12):2427-2456.

Judd, Kenneth L. 1998. *Numerical Methods in Economics*. Cambridge, Mass.: The MIT Press.

Powell, Warren B. 2007. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Hoboken, New Jersey: John Wiley & Sons, Inc.

Press, William H., Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling. 1989. *Numerical Recipes: The Art of Scientific Computing (FORTRAN Version)*, Cambridge University Press, Cambridge.

Rust, John. "Using Randomization to Break the Curse of Dimensionality." *Econometrica* 65(May 1997):487-516.

Woodward, Richard T., Yong-Suhk Wui, and Wade L. Griffin. "Living with the Curse of Dimensionality: Closed-loop optimization in large-scale fisheries simulation model." *American Journal of Agricultural Economics* 87(Feb. 2005):48-60.