8. A more formal introduction to Dynamic Programming and Numerical DP

AGEC 642 - 2024

I. Some DP terminology

- The **Bellman's equation** is an equation like: $V_t(x_t) = \max_{z_t} u(x_t, z_t) + \beta V_{t+1}(x_{t+1})$
- We assume that the state variable $x_t \in X \subset \mathbb{R}^m$
- Bellman's equation is a **functional equation** in that it maps from the function to a function, i.e., from $V_{t+1}: X \to \mathbb{R}$ to $V_t: X \to \mathbb{R}$.
- Bellman's equation is a **recursive** expression in that values functions earlier in time are determined by an operation on the function later in time, i.e. $V_T \to V_{T-1} \to V_{T-2} \to \ldots$
- The discount factor, $\beta = 1/(1+r)$ where r is the discount rate. (For our initial discussion we will let β=1, r=0)

A. *Important features of the DP framework*

1. The Bellman's equation at any point in time is a static optimization problem and everything that we know about solving such problems applies here.
2. If you look back at Dorfman's framework (Lecture 5), you will find a lot of similarities between that structure and what we have here.
3. Hence, just as in optimal control, there will be a co-state variable λ_t which is equal to the marginal value of an additional unit x_t, i.e. $\lambda_t = \partial V / \partial x$ or, if x is discrete as in many of our examples, i.e., taken from the set $\{x_1, x_2, \ldots, x_n\}$, then λ at a point x_i could be calculated as $\lambda(x_i) = \dfrac{V(x_i) - V(x_{i-1})}{x_i - x_{i-1}}$.[1]
4. The marginal future consequence of choices today is equal to $\dfrac{\partial V_{t+1}(x_{t+1})}{\partial x_{t+1}} \dfrac{\partial x_{t+1}}{\partial z_t} = \lambda_{t+1}(x_{t+1}) \dfrac{\partial x_{t+1}}{\partial z_t}$ or the analog for a discrete case.
5. If the problem were stochastic, then the Bellman's equation would look like $V_t(x_t) = \max_{z_t} E_t \left\{ u(x_t, z_t, \varepsilon_t) + \beta V_{t+1}(g(x_t, z_t, \varepsilon_t)) \right\}$ where ε_t is a random variable.

   Other than the addition of the expectation, however, we see that the underlying structure is basically unchanged. The fact that the transition from deterministic to stochastic problems is so straightforward is a big advantage of dynamic programming.
6. In AGEC 642 we will typically consider problems with a discrete-time specification. However, the Bellman's equation can be analyzed in a continuous time framework, which is common in the macroeconomic literature. Continuous-time dynamic programming is discussed in detail in Stokey and Lucas (1989).

---

[1] Notice that here I define $\lambda(x_i)$ based on the one-sided difference in the negative direction, i.e. comparing $V(x_i)$ and $V(x_{i-1})$. We could have used a forward difference, i.e. x_i relative to x_{i+1}, or an average of the two one-sided differences. There is no correct approach; but it is important to tell your reader what you did.

II. A general theoretical framework for the consideration of dynamic programming problems

The general problem that we will seek to solve through the use of dynamic programming is to maximize the **expected** benefits over our planning horizon. Benefits in period t will be written $u(z_t, x_t, \varepsilon_t, t)$, where z_t represents a vector of choices in t, x_t is a vector of state variables, and ε_t represents a vector of random variables. The value of your terminal stock, equivalent to the salvage value, will be written $S(x_{T+1})$.
Hence, the decision maker's objective is to maximize

$$E_t\left\{u\left(z_t,x_t,\varepsilon_t,t\right)+u\left(z_{t+1},x_{t+1},\varepsilon_{t+1},t+1\right)+u\left(z_{t+2},x_{t+2},\varepsilon_{t+2},t+2\right)+\ldots+u\left(z_T,x_T,\varepsilon_T,T\right)+S\left(x_{T+1}\right)\right\}$$

$$E_t\left\{\sum_{s=t}^{T}u\left(z_s,x_s,\varepsilon_s,s\right)+S\left(x_{T+1}\right)\right\}$$

where E_t is the expectation operator conditional on the information available in t.

The value function $V(x_t)$ is defined as

$$V\left(x_t,t\right)\equiv\max_{z_s}E_t\left\{\sum_{s=t}^{T}u\left(z_s,x_s,\varepsilon_s,s\right)+S\left(x_{T+1}\right)\right\},$$

which can be rewritten,

1. $$V\left(x_t,t\right)\equiv\max_{z_t,z_s}E_t\left\{u\left(z_t,x_t,\varepsilon_t,t\right)+E_{t+1}\left\{\sum_{s=t+1}^{T}u\left(z_s,x_s,\varepsilon_s,s\right)+S\left(x_{T+1}\right)\right\}\right\}.$$

If this objective is to be achieved, we must assume that the optimal choices will be made not only in t but in t+1, t+2, etc. Hence,

$$V\left(x_{t+1},t+1\right)\equiv\max_{z_s}E_{t+1}\left\{\sum_{s=t+1}^{T}u\left(z_s,x_s,\varepsilon_s,s\right)+S\left(x_{T+1}\right)\right\},$$

which can then be substituted into 1, to get

2. $$V\left(x_t,t\right)=\max_{z_t}E_t\left\{u\left(z_t,x_t,\varepsilon_t,t\right)+V\left(x_{t+1},t+1\right)\right\}$$

with the terminal condition $V\left(X_{T+1},T+1\right)=S\left(X_{T+1}\right)$.

Note: As in Judd (1997), technically this should be a *sup* operator instead of *max*. We can interpret this as a technical distinction that doesn't really affect us. For all the problems we'll be considering, *sup* and *max* are equivalent since $u(\cdot)$ and the state equation will be defined over X,Z.

While $V(x_t,t)$ is the max, we are probably also interested in the choice variables that actually solve this problem. The *policy function*, $z^*(x_t,t)$ maps from state and period to a set of choices, i.e.,

$$z^*\left(x_t,t\right)=\arg\max_{z_t}E_t\left\{u\left(z_t,x_t,\varepsilon_t,t\right)+V\left(x_{t+1},t+1\right)\right\}.$$

Notation: "arg max" means the argument (i.e., the value of z_t,) that maximizes the function.

The policy function is a decision rule: if you wake up on the t^{th} morning and discover that you have an endowment of x_t, then you can refer to the rule, $z^*(x_t,t)$, carry out the action, and go back to bed.

III. The types of DP problems

Numerical DP problems can be broken into four basic types of problems, each requiring slightly different approaches.

A. Discrete state and discrete control (DD)

Discrete-Discrete problems (DD-DP problems) are problems for which computers are best suited. In such cases both the state variable and the control options take on a finite number of values. The inventory control problem presented in Lecture 7 is a DD problem. There are a finite number of states (i.e., inventory takes on only discrete values) and the decision-maker's problem is to choose from a discrete set of possible production choices.

Because both the state space and the control space can be exactly modeled by the computer, there is no approximation error in the solution and it is possible to obtain an exact solution.

B. Continuous state and discrete control (CD)

In Continuous-Discrete problems (CD-DP problems) the state space is continuous, but there are only a finite number of choice possibilities. One example of this would be what are generally referred to as optimal-stopping problems. For example, the seminal paper by Rust (1987) considered the problem of the optimal time to replace the engine of a bus. The main state variable in Rust's model was the age of the current engine, a continuous variable. The choice was whether to replace the engine or continue to give it maintenance. Hence, the state variable is continuous, the age of the engine, while the choice variable is discrete, replace or not.

CD problems are common in economics and are particularly well-suited for dynamic programming. Unlike in DD problems, in CD problems it is not possible to find the value of V at every point in the state space because there are an infinite number of such points. Further, we do not typically have a closed-form expression of V that would allow us to exactly calculate V at any point. We have to approximate. One way to solve CD problems is to calculate $V(x_t)$ at a finite number of points and then estimate the value elsewhere in the state space based on the points where the solution has been found. Hence, the best we can do is to find an approximate solution to the problem. We'll discuss this problem in much more detail in Lecture 11.

C. Continuous state and continuous control

The third class of problems is where both the control and the state variables are continuous, CC-DP problems. In these problems both the state and control variables are continuous. The PV utility maximization problem discussed above is a good example of such a problem. In this case, there are two sources of error – the approximation error in finding the correct choice variable from the continuous set of possibilities and the approximation error because V can only be estimated. Computer programs like GAMS or Matlab have embedded algorithms that are quite good at finding the solution to static optimization problems. However, even in the best of worlds the solution will frequently only be approximate (though the error might be economically insignificant) and, if the

problem is not concave, the computer program may find a solution that is entirely incorrect.

D. Discrete state and continuous control (DC)

The 4th class of problems are discrete state and continuous control (DC). This can arise, for example when a piece of machinery is either working or not and the question is how much to spend on maintenance each period. The solution algorithms for such problems are often an amalgam of DD and CD problems, so we will not spend much time considering this type of problem independently, though you should be able to solve such problems. There are problems in which there is a closed-form expression for the optimal choice in t as a function of the values of V_{t+1}. In cases like that, the exact solution can be found without numerical approximation.

IV. A general algorithm for the solution of finite-horizon DD problems using a grid search method.

In DD problems, the state space is made up of a finite number of points, say X. We will assume that future benefits are discounted using the factor β. If discounting is not required, then β=1.

There are three basic steps required

1) Initialization: set $t=T+1$ and define $V(x_{T+1},T+1)$ for every state $x\in X$. If you don't have a salvage value, set $V(x_{T+1},T+1)=0$ for all x_{T+1}.

2) Recursion step: Starting with $t=T$ find the value of z that solves the problem

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

for every value of x_t, then store $V(x_t, t)$ and

$$z^*(x_t,t)=\arg\max_{z_t} E\{u(z_t,x_t,\varepsilon_t,t)+\beta V(x_{t+1},t+1)\}$$

3) Termination step: if t=0, stop. Otherwise set $t=t-1$ and return to step 2.

The *pseudocode* presented below gives a general representation of the solution algorithm, which could then be programmed in almost any language that you choose. An implementation of this using Visual Basic is provided toward the end of these notes.

Pseudocode for a finite-horizon DD DP program

$$\max_{z_t} E_0 \sum_{t=0}^{T} u\left(z_t, x_t, \varepsilon_t, t\right) + V\left(X_{T+1}, T+1\right)$$

.

Calculate $V(x_{T+1},T+1)$ for every state $x \in X$ based on prior information.

Stage loop: For every period $t=T, T-1, T-2, \ldots, 0$

State loop: for every point in your state space, $x_t \in X$

set $V(x_t,t) = -\infty$ (in practice, a big negative number such as -999999)

Control loop: for each possible decision choice, z_t

calculate $E_t u(\cdot) = \sum_{\varepsilon_t} p(\varepsilon_t) u\left(z_t, x_t, \varepsilon_t, t\right)$, where $p(\varepsilon_t)$ is the probability that a particular value of ε_t will be realized

calculate $E_t V\left(x_{t+1}, t+1\right) = \sum_{\varepsilon_t} p(\varepsilon_t) V\left(x_{t+1}\left(z_t, x_t, \varepsilon_t, t\right), t+1\right)$ where $x_{t+1}\left(z_t, x_t, \varepsilon_t, t\right)$ is the value for the state variable in $t+1$ given z_t and a particular realization of ε_t

calculate $V' = E_t\{u(\cdot) + \beta V(\cdot)\}$ (V' is a *candidate* value)

if $V' > V(x_t,t)$ then

$V(x_t,t) = V'$

$z^*(x_t,t) = z_t$

EndIf

continue {End of Control loop}

continue {End of State loop}

continue {End of Stage loop}

As you write your programs, each of these steps and the three main loops should be clearly identified. The control loop is where Bellman's equation is solved at each state-stage combination.

V. Infinite-horizon dynamic programming problems

The finite-horizon problem that we consider above starts with some salvage value, $V\left(x_{T+1}, T+1\right)$, sometimes equal to zero, and then works backwards. Hence, you might wonder how one might solve an infinite horizon problem, where there is no end point. It turns out, however, that because of discounting you still use the backward recursion solution approach. In effect, what we will do is start far out in the future and then work our way back to the present until, because of discounting, that starting value function far in the future, has no effect on what is done in the present.

Consider yourself in a situation in which you have a resource that you can use up to generate immediate utility or, with effort, you can increase the resource stock. If you had 100 units of the resource and only 4 days to use it, then your choices today would be very

much affected by the time horizon. Your actions on day 3 would be very much affected by the fact that the end of your problem is imminent. And if, on the 3rd day somehow you were given an extra day to use the resource, your choices would probably change a lot. Now imagine the same problem but assume that you have 100 years to use the resource. It is unlikely that your choices today would differ very much if someone told you that you had 100 years + 1 day. If that's true, if there is no change in your choices today as the time horizon changes, you are behaving in the current period, t=0, *as if* the time horizon doesn't matter, i.e., as if you face an infinite time horizon. This intuition is essentially what we implement in the successive approximation algorithm.

A. Successive approximation of infinite-horizon problems

We now want to consider how we would solve DP problems that don't have a terminal time, i.e., infinite-horizon problems. As indicated above, the basic intuition behind the infinite-horizon Bellman's equation is that, because of discounting, if T is much larger than t, then the function $V(x,t)$ will look very much like $V(x,t+1)$, and likewise for $z^*(x,t)$ and $z^*(x,t+1)$. The solution algorithm converges if as $T\to\infty$, $V(x,t+1)=V(x,t)$. What this means is that the value *function* will not change over time. Hence, we can drop our reference to t and write Bellman's equation:

$$3.\quad V(x_t)=\max_{z_t} E_t\left\{u(z_t,x_t,\varepsilon_t)+\beta V(x_{t+1})\right\}.$$

Note that in this equation we have also dropped the reference to t in the benefit function $u(\cdot)$. This is a limitation – if you want to consider infinite-horizon DP problems you have to specify your benefit function and state equation(s) as autonomous.[2] Many problems that might appear to be non-autonomous (e.g., if there is technological change) can be made autonomous by including additional state variables.

Equation 3 also includes $\beta=1/(1+r)$ in the general specification of the Bellman's equation. This is important; $\beta<1$ ensures that the objective function is finite, and will play a central role in the numerical solution of the problem. The proof of convergence of the successive-approximation algorithm requires that β be less than 1. Some undiscounted problems converge, but typically they will not.

Solving this problem, however, presents a dilemma. In the finite-horizon case, the function $V(x_{T+1},T+1)$, being the salvage value, was known. In the infinite-horizon case, however, we are trying to solve for an unknown function $V(\cdot)$ on the LHS, but it also appears on the RHS. The starting point is not obvious.

It turns out that there's an easy solution to this problem. We can take <u>any</u> arbitrary guess at the value function, say $V^0(x)$, then using this function, solve for $V^1(x)$ as follows

$$V^1(x_t)=\max_{z_t} E_t\left\{u(z_t,x_t,\varepsilon_t)+\beta V^0(x_{t+1})\right\}.$$

[2] (at least for all t greater than some finite number)

Then we repeat this process, finding $V^2(x_t)$, $V^3(x_t)$, …, until we reach a point when $V^{k+1}(x)$ is "close" to $V^k(x)$. In each iteration we are only going to be using the k^{th} approximation of the value function to obtain the $k+1^{th}$ approximation, we do not need to store all of our approximations of the value functions. Instead, at any time we only need to retain two value functions, which I will call $V^{RHS}(x)$, which is the previous estimate, and $V^{LHS}(x)$ which is the new estimate.

Note that the superscript on V refers to the iteration of the approximation algorithm, not to time. Hence, $V^0(x_t)$ is the is the initial guess of the value function, $V^1(x_t)$ is estimate of the value function after the first iteration, and $V^k(x_t)$ is the estimate after the k^{th} iteration.

B. The successive approximation algorithm for the solution of infinite-horizon DD problems

The successive approximation algorithm used to solve infinite-horizon problems is as follows:

As in the finite-horizon case, there are three basic steps required.

1) Initialization: For every state $x \in X$ take a first guess at $V(x)$, say $V^0(x)$. You can use $V^0(x) = 0$ for all x or some other value if you have prior information about the solution. We will put $V^0(x)$ into the array $V^{RHS}(x)$.

2) Update step: Given our estimate V^{RHS}, obtain a new vector of values for $V^{LHS}(x)$ at every point $x \in X$ by solving the problems

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

Store V^{LHS} and the candidate policy function,

$$\hat{z}(x_t) = \arg\max_{z_t} E\left\{u(z_t, x_t, \varepsilon_t) + \beta V^{RHS}(x_{t+1})\right\}.$$

3) Convergence check: Check to see if $\max_x \left\| V^{LHS}(x) - V^{RHS}(x) \right\| < \tau$, where τ is a small number.

· If the convergence criterion is satisfied, then we're done: set $V(x)=V^{LHS}(x)$ and stop.

· Otherwise set $V^{RHS}(x) = V^{LHS}(x)$, and return to step 2 to find a new function, $V^{LHS}(x)$.

Here is the pseudocode for an infinite-horizon deterministic DD problem using a grid-search algorithm.

Set $V^{RHS}(x)=0$ (or some other starting value) for every state $x \in X$.

Stage Loop: For iteration 1,2,…, max iter

  State Loop: for every point in your state space, $x_t \in X$

    set $V^{LHS}(x_t) = -\infty$ (in practice, a big negative number)

    Control Loop: for each possible decision choice, z_t

      $\tilde{u} = u\left(z_t, x_t\right)$

      $\tilde{V} = V^{RHS}\left(x_{t+1}\left(z_t, x_t\right)\right)$

      [*We now have "candidate" values* $u(\cdot)$ and $V(x_{t+1})$ *for a given choice*]

      if $\tilde{u} + \beta\tilde{V} > V^{LHS}\left(x_t\right)$ then [*The current "candidate"* z_t *is the best so far*]

        $V^{LHS}\left(x_t\right) = \tilde{u} + \beta\tilde{V}$

        $z^*(x_t) = z_t$

      endif

    continue [end of control loop]

  continue [end of state loop]

  set $diff = \max_x \left|V^{LHS}\left(x\right) - V^{RHS}\left(x\right)\right|$

  set $V^{RHS}\left(x\right) = V^{LHS}\left(x\right)$

  if *diff*<convergence criterion, then

    exit stage loop

  endif

  continue [end of stage loop, start next iteration with a new V^{RHS}]

set $V\left(x\right) = V^{LHS}\left(x\right)$ for all x.

And here is the pseudocode for an infinite-horizon **stochastic** DD problem.

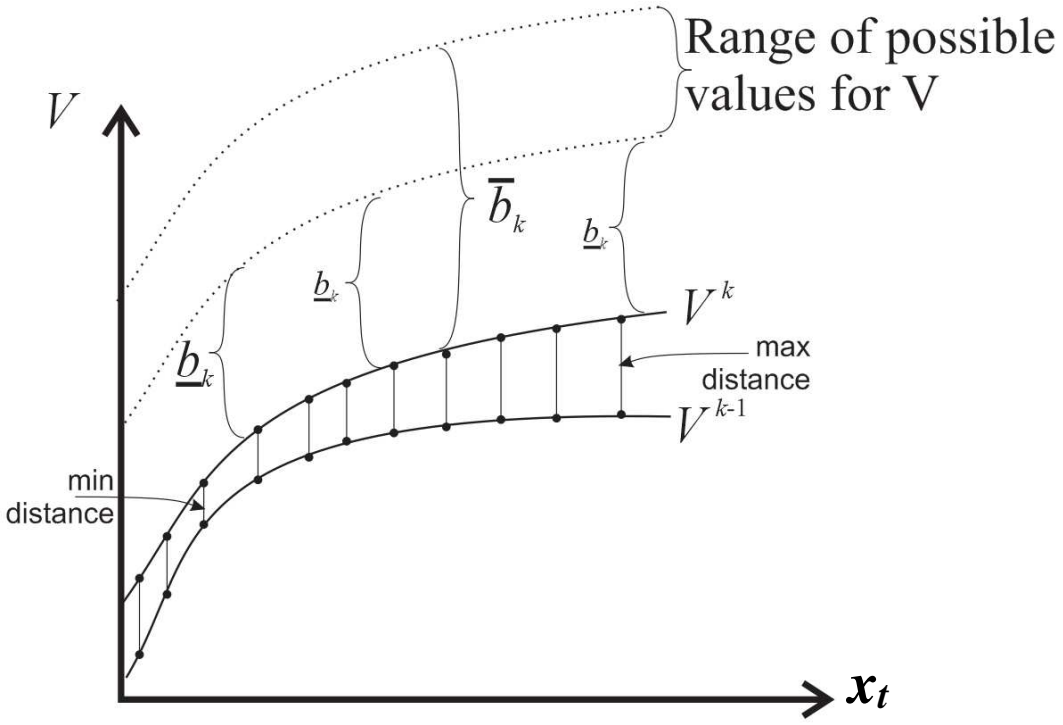
Set $V^{RHS}(x)=0$ (or some other starting value) for every state $x \in X$.
For iteration 1,2,…, max iter
  for every point in your state space, $x_t \in X$
    set $V^{LHS}(x_t) = -\infty$ (in practice, a big negative number)
    for each possible decision choice, z_t
      set $E_t u(\cdot) = E_t V(\cdot) = 0$
      for each state of nature, ε_t with probability $p(\varepsilon_t)$
        evaluate $u(z_t, x_t, \varepsilon_t)$
        evaluate $V\left(x_{t+1}(z_t, x_t, \varepsilon_t)\right)$ where $x_{t+1}(z_t, x_t, \varepsilon_t)$ is the state value that occurs in t+1 given a particular action z_t and a particular realization of ε_t
        Add $p(\varepsilon_t)u(z_t, x_t, \varepsilon_t)$ to $E_t u(\cdot)$
        Add $p(\varepsilon_t)V\left(x_{t+1}(z_t, x_t, \varepsilon_t)\right)$ to $E_t V(\cdot)$
      continue [end of uncertainty loop]
      [*We now have "candidate" values* $E_t u(\cdot)$ and $E_t V(\cdot)$ and $\tilde{V} = E_t u(\cdot) + \beta E_t V(\cdot)$ *for a given choice*]
      if $E_t u(\cdot)+\beta E_t V(\cdot)\} > V^{LHS}(x_t)$ then
        $V^{LHS}(x_t)=E\{u(\cdot)+\beta V^{RHS}(\cdot)\}$
        $z^*(x_t) = z_t$
      endif
    continue [end of control loop]
  continue [end of state loop]
  set $diff = \max_x \left| V^{LHS}(x) - V^{RHS}(x) \right|$
  set $V^{RHS}(x) = V^{LHS}(x)$
  if *diff*<convergence criterion, then exit loop and set $V(x) = V^{LHS}(x)$,
  else
continue [end of stage loop]

Stage Loop: State Loop: Control Loop: Uncertainty Loop

The reason that this algorithm converges to the correct $V(x)$ is because the Bellman's equation is a *contraction mapping*. What this means is that the distance between V^{LHS} and V^{RHS} decreases monotonically over iterations. Hence, if you have a maximum difference between the two value functions of 0.01 after 20 iterations, then even if you iterate 1000 more times, you will never see a maximum difference of more than 0.01.

Technical details and a proof of that this algorithm is a contraction mapping can be obtained from a variety of sources including the books by Bertsekas, Judd, and Miranda and Fackler.

The proof of the contraction mapping property gives rise to an interesting result. If V^k and V^{k-1} are two successive iterations of the value function, then we can place bounds on the true infinite-horizon value function using these two functions as follows. Define $\underline{b}_k = \frac{\beta}{1-\beta}\min\left[V^k - V^{k-1}\right]$ and $\overline{b}_k = \frac{\beta}{1-\beta}\max\left[V^k - V^{k-1}\right]$. Then if V is the true infinite-horizon value function, then $V^k + \underline{b}_k \le V \le V^k + \overline{b}_k$ (Rust, 1996 p. 653).[3]



That is, there are bounds between which we know the true value function will actually lie. Furthermore, these bounds will get tighter as the number of iterations increases. Because that can be shown, if follows that the successive approximation algorithm will converge to the true value function.[4]

Here are a couple of things worth highlighting about the infinite-horizon problem:

- The Bellman's equation of the infinite-horizon problem is what is known as a *functional fixed point*. In the case of DD problems the function $V(\cdot)$ is simply a set of values for each value $x_t \in X$. At the fixed point, when we evaluate the Bellman's equation to find $V^{LHS}(x_t)$, we will find the same vector of values as those in

[3] I have found instances when this formula seems to break down, but I have not explored it in detail and I cannot rule out programming errors on my end. Regardless, as $k \to \infty$, the formula certainly holds.

[4] As will be discussed in Lecture 12, these bounds can be used to accelerate the solution algorithm.

$V^{RHS}(x_t)$. *Note that this does <u>not</u> mean that* $V^{LHS}(x_t) = V^{RHS}\left(x_{t+1}(x_t, z_t)\right)$ because unless x_t is an equilibrium value, $x_t \neq x_{t+1}\left(x_t, z^*(x_t)\right)$.

- The successive approximation algorithm is monotonic in the sense that if $V^{k+1}(x) > V^k(x)$ for all $x \in X$, then in every subsequent iteration for $n \geq k+1$, $V^{n+1}(x) > V^n(x_t)$. This property is critical to the convergence of the algorithm. As a practical manner, you can sometimes use this property in your programming – if it is violated you have an error in your program. (Note that this inequality holds point by point!)
- In the successive approximation algorithm there is no need to store the value function at each stage since in the end all you are looking for is the true value function $V(x)$. For debugging purposes, however, it is sometimes helpful to save them.

VI. DP with VB

On the next several pages we present two Visual Basic programs that solve the inventory control problem presented in the previous lecture. In the first case, note the close correlation between the pseudocode and the actual program. The second program is more sophisticated and general. Either program can be modified to solve virtually any finite-horizon DD-DP problem. Spreadsheets with these programs are available on the programs page of the class web site.

BASIC VB Code for solving the Inventory control problem

```
Option Explicit
Option Base 1

Sub VerySimpleDP()
'  Dimension the minimal set of variables needed
Dim it As Integer, ix As Integer, ixnext As Integer, iz As Integer
Dim xt As Double, xnext As Double, zt As Double
' Dimension V(ix, it) and zStar(ix,it)
Dim V(0 To 3, 1 To 4), zStar(0 To 3, 1 To 4)
Dim u, VLHS, VRHS

' Parameters of the model
Dim p, d, alpha, gamma, N
N = 3
p = 10
d = 2
alpha = 1
gamma = 3
' ------------------------------------------------------------

' Terminal Values. Sell product and pay fine if required
 it = 4
 For ix = 0 To 3
    xt = ix*1.0
    V(ix, it) = (p * xt - d * (N – xt) )
 Next ix

' ------------------------------------------------------------
' Recursion step starting in T-1
' Stage Loop, stepping backward
 For it = 3 To 1 Step -1
'   ------------------------------------------------------------
'   Start State Loop
    For ix = 0 To 3
    xt = ix*1.0                  ' Inventory, a real number
'   ------------------------------------------------------------
'   Start by setting V(ix,it) = -99999
    V(ix, it) = -99999
'       ------------------------------------------------------
'       Start Control Loop
        For iz = 0 To 3
           zt = iz*1
'          ------------------------------------------------
'          Evaluate the candidate policy. VLHS = u + VRHS
'          ------------------------------------------------
           u = -(alpha * zt ^ 2 + gamma * xt)
           xnext = xt + zt     ' State Equation
           If xnext > N Then xnext = N  ' boundary issues
           ixnext = xnext    ' Get index for xnext
           VRHS = V(ixnext, it + 1)
           VLHS = u + VRHS

'          Check to see if this z is an improvement.
'          If so, save it and save zstar
           If VLHS > V(ix, it) Then
              V(ix, it) = VLHS
              zStar(ix, it) = zt
           End If
'       ------------------------------------------------
'       End of control loop
        Next iz
'   ------------------------------------------------------
'   End of state loop
    Next ix
' ------------------------------------------------------------
' End of stage loop
 Next it
' ------------------------------------------------------------
' Store output in Sheet 1.
' use 5 - ix & 15-ix so that results are from highest x to lowest as in
notes
 Sheets("Sheet1").Select
 For ix = 0 To 3
 For it = 1 To 4
    Cells(5 - ix, it) = V(ix, it)
    Cells(15 - ix, it) = zStar(ix, it)
 Next it
 Next ix
End Sub
```

Stage loop

State loop

Control loop

```
' ------------------------------------------------------------------------------------------
' This program solves the inventory control problem that is solved
' By hand in AGEC 637
' Author: Rich Woodward
‘ http://agecon2.tamu.edu/people/faculty/woodward-richard/642/Programs/
' ------------------------------------------------------------------------------------------
' When basing problem-set programs on this code, you need to rewrite
' and add comments to convince me that you fully understand
' the program.
'##########################################################################

' VARIABLE DECLARATIONS & Global options
' ---------------------------------------------------------------------
   Option Explicit     ' All variables used must be explicitly dimensioned
'----------------------------------------------------------------------
   Option Base 1       ' Unless specified otherwise, arrays will be indexed
                       '   starting at 1.  You override this by writing,
                       '   for example,  " Dim V(0 To nx) "
'------------------------------------------------------------------------------------------
' Notice that all variables are dimensioned outside the subs so that their
' values are accessible from all the subs.
'------------------------------------------------------------------------------------------
' Arrays are dimensioned with the () after them so that they can
'   be redimensioned later after dimension parameters have been read in
'------------------------------------------------------------------------------------------
   Dim ValFn() As Double     ' Final Value Function
   Dim VLHS() As Double      ' Value Function on LHS of Bellman's eq
   Dim VRHS() As Double      ' Value Function on RHS of Bellman's eq

   Dim zStar() As Integer     '  Optimal policy function of xt and it
'------------------------------------------------------------------------------------------
' Parameters of the program
'------------------------------------------------------------------------------------------
   Dim nT As Integer, nx As Integer, nz As Integer
'------------------------------------------------------------------------------------------
' Parameters of the model
'------------------------------------------------------------------------------------------
   Dim Beta As Double, p As Double, d As Double, n As Double
   Dim alpha As Double, gamma As Double
'------------------------------------------------------------------------------------------
' Intermediate variables
'------------------------------------------------------------------------------------------
   Dim utility As Double, V As Double, Vnext As Double
'------------------------------------------------------------------------------------------
' Counters.  Some variables require both a real # and integer representation.
'------------------------------------------------------------------------------------------
   Dim it As Integer, iz As Integer, ix As Integer, ixnext As Integer
'------------------------------------------------------------------------------------------
'   x and z are potentially real numbers that will be taken from the
'   grids xGrid and zGrid
'------------------------------------------------------------------------------------------
   Dim zt, xt, xnext
   Dim zMin, zMax, xMin, xMax
   Dim xGrid, zGrid

'##########################################################################
Sub Main()

   Call initializeValues

'------------------------------------------------------------------------------------------
'   CreateGrids of for x and z
'     The CreateGrid Function is available in the MatrixSubs that
'     can be downloaded from the web site
'------------------------------------------------------------------------------------------
   xGrid = CreateGrid(0, nx, xMin, xMax)
   zGrid = CreateGrid(0, nz, zMin, zMax)

'------------------------------------------------------------------------------------------
' Set Terminal Value  [or the initial estimate of V() in infinite-horizon problems]
   Call TerminalValue
' Store terminal values in ValFn and on the spreadsheet
   For ix = 0 To nx
      ValFn(ix, nT) = VRHS(ix, 1)
      Range("ValueFunction").Offset(ix + 1, nT).Value = ValFn(ix, nT)
   Next ix

'------------------------------------------------------------------------------------------
' stage loop (note we start at it=nT and move backwards until it=1
   For it = (nT - 1) To 1 Step -1
   ' ---------------------------------------------------------------------------------------
   ' state loop
      For ix = 0 To nx
      xt = xGrid(ix)
      VLHS(ix, 1) = -99999#
      ' ----------------------------------
      ' Solve the maximization problem at current state & stage
      ' In DD problems we use a grid search
      ' control loop
      ' ----------------------------------
         For iz = 0 To nz
            zt = zGrid(iz)
            Call UtilityFunction
            Call StateEquation
            Call VnextCalc
         ' ----------------------------------
         ' Bellman's equation
            V = utility + Beta * Vnext

' ----------------------------------
         ' Compare stored V with V at current z and update if necessary
         '    In infinite-horizon problems you would use VLHS(ix)
            If V > VLHS(ix, 1) Then
               VLHS(ix, 1) = V
               zStar(ix, it) = zt
            End If
      ' ----------------------------------
      '   End of control loop
         Next iz
```

```
        ' ----------------------------------
        ' Store optimal value in the full array of the value function
            ValFn(ix, it) = VLHS(ix, 1)
        ' ----------------------------------
        ' store optimal values in spreadsheet
            Range("ValueFunction").Offset(ix + 1, it).Value = ValFn(ix, it)
            Range("Zstar").Offset(ix + 1, it).Value = zStar(ix, it)
' ------------------------------------------------
        Next ix

' ------------------------------------------------
' Move VLHS over to the RHS
'       In infinite-horizon problems at this point
'       check convergence, and carry out the policy-iteration or
'       modified-policy-iteration step
    For ix = 0 To nx
        VRHS(ix, 1) = VLHS(ix, 1)
    Next ix

' ------------------------------------------------
'   End of Stage loop
    Next it

' ------------------------------------------------
' The optimization problem is solved!
' ------------------------------------------------
' ------------------------------------------------
'   Now simulate the optimal policy path
    Call Simulation

' ------------------------------------------------
' End of main program
End Sub
'#####################################################################
Sub UtilityFunction()
' ------------------------------------------------------------------
' The benefit function subroutine as a function of z and x

  utility = -alpha * (zt / 100) ^ 2 - gamma * (xt / 100)
End Sub
'#####################################################################
Sub StateEquation()
' ------------------------------------------------------------------
'   The state equation subroutine
' ------------------------------------------------------------------
    xnext = xt + zt

' ------------------------------------------------------------------
'   You must decide what happens if you go outside your state grid.
'       This can often be a critical modelling decision

    If xnext > xGrid(nx) Then xnext = xGrid(nx)
' ------------------------------------------------------------------
'   The potentially real number xnext, must be converted to an
'       integer in DD problems
    ixnext = InvertGrid(xnext, xGrid)

End Sub

'#####################################################################
Sub TerminalValue()

        For ix = 0 To nx
            xt = xGrid(ix)
            VRHS(ix, 1) = p * xt / 100 - d * (n - xt) / 100
            If xt > n Then VRHS(ix, 1) = p * n / 100
        Next ix

End Sub
'#####################################################################
Sub VnextCalc()
' ------------------------------------------------------------------
' Subroutine to calculate V(xt+1, t+1)
' ------------------------------------------------------------------
    Vnext = VRHS(ixnext, 1)
End Sub
'#####################################################################
Sub initializeValues()
' ------------------------------------------------
' This subroutine sets up the space for writing output
'   and reads in the data

' ------------------------------------------------
'   Clean up the area for writing output
    Call ClearAreas

' ------------------------------------------------
'   Read in data from spreadsheet
    nT = Range("nT")
    nx = Range("nX")
    nz = Range("nz")
    p = Range("p")
    d = Range("d")
    n = Range("N")
    alpha = Range("alpha")
    gamma = Range("gamma")
    Beta = Range("beta")

' ------------------------------------------------------
' Set up maxes & mins for the state and control variables
' ------------------------------------------------------
    zMin = 0
    zMax = nz * 100
    xMin = 0
    xMax = nx * 100
```

```
' ------------------------------------------------
'   Redimension arrays based on read in data
'      When you redimension an array you
'      set its value to zero.  Be careful!
'      Note that in the inventory problem we want the arrays to start at zero
    ReDim ValFn(0 To nx, nT)
    ReDim VLHS(0 To nx, 1)
    ReDim VRHS(0 To nx, 1)

    ReDim zStar(0 To nx, nT)


' ------------------------------------------------
' Set up areas for writing output
    Range("output").Offset(2, 0).Name = "ValueFunction"
    Range("ValueFunction").Offset(nx + 4, 0).Name = "Zstar"
    Range("Zstar").Offset(nx + 4, 0).Name = "Simt"
    Range("Simt").Offset(1, 0).Name = "SimXt"
    Range("SimXt").Offset(1, 0).Name = "SimZt"
    Range("SimZt").Offset(1, 0).Name = "SimXtPlus1"
    Range("SimXtPlus1").Offset(1, 0).Name = "SimUtility"

' ------------------------------------------------
' Add titles to the output arrays
    Range("ValueFunction").Value = "V"
    Range("Zstar").Value = "Zstar"
    For it = 1 To nT
       Range("ValueFunction").Offset(0, it).Value = "t = " & it
       Range("Zstar").Offset(0, it).Value = "t = " & it
    Next it

    For ix = 0 To nx
       Range("ValueFunction").Offset(ix + 1, 0).Value = "x = " & ix
       Range("Zstar").Offset(ix + 1, 0).Value = "x = " & ix
    Next ix

    Range("Simt").Offset(-1, 0).Value = "Simulation"
    Range("Simt").Value = "t"
    Range("SimXt").Value = "x"
    Range("SimZt").Value = "z"
    Range("SimXtPlus1").Value = "xt+1"
    Range("SimUtility").Value = "Utility"

End Sub
```

```
'###################################################################
Sub Simulation()
   ' Now simulate the path over nT periods
    xt = 0
    For it = 1 To nT
'-------------------------------------------------------------
'     Find the index associated with the real # xt
'          The InvertGrid Function is available in the MatrixSubs
'          that can be downloaded from the web site
'-------------------------------------------------------------
       ix = InvertGrid(xt, xGrid)
'-------------------------------------------------------------
'     Choose the optimal choice z* from the zstar array
'-------------------------------------------------------------
       zt = zStar(ix, it)

'-------------------------------------------------------------
'     figure out what x(t+1) will be given z*
'-------------------------------------------------------------
       Call StateEquation  ' finds xnext as fn of xt zt
       Call UtilityFunction

'-------------------------------------------------------------
'     write the output to the spreadsheet
'-------------------------------------------------------------
       Range("Simt").Offset(0, it).Value = it
       Range("SimXt").Offset(0, it).Value = xt
       Range("SimZt").Offset(0, it).Value = zt
       Range("SimXtPlus1").Offset(0, it).Value = xnext
       Range("SimUtility").Offset(0, it).Value = utility
'-------------------------------------------------------------
'     Update to the next period
'-------------------------------------------------------------
       xt = xnext
'-------------------------------------------------------------
'   end of simulation loop
'-------------------------------------------------------------
   Next it

End Sub
'###################################################################

Sub ClearAreas()
'-------------------------------------------------------------
'   This program may be used directly without personalized comments
'-------------------------------------------------------------
'   Clear space for printing value function
    With Range("OUTPUT")
       Range(.Offset(1, 0), .Offset(100, 50)).ClearContents
    End With

End Sub
'###################################################################
```