Chapter 14

Dynamic Programming: Numerical Methods

Many approaches to solving a Bellman equation have their roots in the simple idea of “value function iteration” and the “guess and verify” methods. These are the very first steps typically one learns about for obtaining analytical solutions, but they are also practical and useful in numerical work. In general, the algorithms for dynamic programming can be broadly classified along two dimensions:

A Algorithms that work with the Bellman equation versus those that work through the Euler equation. Each approach has its strengths, although in more complex problems working with the Bellman equation can be more reliable.

B Algorithms that work with a class of flexible functional forms, such as polynomials, versus finite element methods.

To illustrate how each approach works, let us recall how the two methods work.

14.1 Value Function Iteration

Value function iteration (VFI) is the oldest and most basic approach to solving a dynamic programming problem. It was originally proposed by Richard Bellman in his seminal book (Bellman (1957)) and has been the workhorse of DP analysis since then. Its performance is guaranteed by the contraction mapping theorem (Theorem 13.3) stated in the previous chapter and is applicable in any problem where the conditions of that theorem are satisfied. In particular, we must be able
to establish that the Bellman operator for the specific problem we are dealing with is a contraction mapping over the space of appropriately specified functions, which itself must be a complete metric space. These conditions are rather mild and are satisfied by a broad class of problems you will encounter in economics. VFI is the primary approach to DP problems and arguably the most generally applicable one. It can be used to solve problems with non-convexities and non-differentiabilities (various binding constraints or discrete choice sets), where other methods can easily fail.

The solution algorithm is provided by the following result in the theorem:

\[ TV^* = V^* = \lim_{N \to \infty} T^N V_0 \]

for all \( V_0 \in S \), where \( T^N \) denotes \( N \) repeated applications of \( T \). That is, the unique fixed point is attained by repeated applications of the Bellman operator, starting from any point (i.e., function) that lies in the space of interest, \( S \). This gives us enormous flexibility in choosing an initial guess and we are ensured to converge to the correct solution, eventually.

Thus, conceptually, VFI is a fairly straightforward algorithm. In an abstract form, the *standard VFI algorithm* can be written as follows.

**Algorithm 6: Standard Value Function Iteration**

**Step 0.** Set \( n = 0 \). Choose an initial guess \( V_0 \in S \).

**Step 1.** Obtain \( V_{n+1} \) by applying the mapping: \( V_{n+1} = TV_n \), which entails maximizing the right-hand side of the Bellman equation.

**Step 2.** Stop if convergence criterion is satisfied: \( \| V_{n+1} - V_n \|_\infty < \text{toler} \). Otherwise, increase \( n \) and return to step 1.

Although the algorithm is conceptually simple, implementing it (and other algorithms like it) on a computer raises a host of practical issues. Before delving into the details of implementation, we first discuss some of the most important of these issues.

### 14.2 Four Practical Issues

Four issues in particular require special attention.

How do we represent a general function—which maps infinitely-many points into infinitely-many points—on a computer?
How do we pick a sensible initial guess, $V_0$? How do we determine a reasonable stopping rule for the algorithm (“toler”)?

Initialization: How do we determine the domains of various functions? This requires choosing appropriate bounds for state variables, choice variables as well as a way to represent the state space.

How do we improve the speed of the algorithm for a given desired accuracy?

We begin by discussing the first three of these practical issues in this section. The fourth one—accelerating VFI based algorithms—is a major topic on its own. Therefore, starting in Section 14.3, we devote the rest of this chapter to covering a variety of useful acceleration techniques.

To provide context, it is useful to have a concrete framework to discuss these practical issues. For this purpose consider the stochastic version of the neoclassical growth model, with CRRA preferences, a persistent TFP process, $z_t$, and partial depreciation of capital:

$$V(k, z) = \max_{c, k'} \left\{ \frac{e^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}(V(k', z') | z) \right\}$$

subject to:

$$c + k' = zk^\alpha + (1 - \delta)k$$

$$\ln z' = \rho \ln z + \eta', \quad k' \geq k.$$

We are now looking to obtain a numerical solution, which consists of decision rule for optimal savings, $k' = g(k, z)$, as well as a value function, $V(k, z)$, that “solve” the dynamic programming problem—that is, satisfy the functional equation (P1) with negligible error.\(^1\)

I. How to Represent $V$?

The value function that solves a Bellman equation is the product of all the features of the dynamic program—from the shape of the utility function to the choices (or margins of adjustment) available, and from the nature of uncertainty to the types and tightness of constraints. The simple example (13.13) studied in the previous chapter yielded a value function that was log-linear in $k$, which we could solve by guess-and-verify. And as noted there, this was a very exceptional case: there are only a handful of problems (that are sufficiently rich and interesting) in which the

\(^1\) Once we have the savings rule, the consumption rule, $c(k, z)$, comes for free via the resource constraint.
value function belongs to a simple parametric family. More generally, we should work under the assumption that the value function will not have an analytical formula. Solution methods can be broadly categorized into two groups, according to how they deal with this issue.

**Global Parametric Methods.** The first approach is to appeal to a variety of approximation theorems in mathematics that broadly establish that under rather mild assumptions any “well-behaved” general function can be approximated to any desired accuracy by a finite number of *simpler* (basis) functions, such as polynomials or rational functions. Thus, we can parametrize $V$ over its entire domain using a convenient family of basis functions, which is fully characterized by a finite set of parameters. Therefore, solving the Bellman equations is reduced to searching over a finite set of parameter values, rather than searching in the infinite space of functions. It should be evident that the parametrization should be flexible enough that we can be confident it does not impose any unintended (and artificial) constraints on the problem. This approach is global, in the sense that the same basis functions and coefficients represent the entire value function over its entire domain. A variety of implementations based on this approach exist in the economics literature, and have found widespread usage especially in representative-agent models.

**Grid-Based Methods.** The value functions and decision rules that we will encounter in models with heterogeneity often display challenging features, such as a very high curvature in certain parts of the domain combined with remarkable flatness in other parts, as well as multiple kinks (or points of non-differentiabilities) in some other applications. Thus, a second approach that provides more flexibility—at cost of some computational speed—is to approximate $V$ in a piecewise fashion. That is, as discussed in Chapter 8 (Interpolation), the idea is to divide the domain of $V$ into subdomains—most commonly, a multi-dimensional grid—and approximate $V$ with separate basis functions within each subdomain. Because each approximation is over a small range, this allows us to use lower order polynomials, which also helps with the stability of the numerical algorithm. One idea described in detail in Chapter 8 was Spline interpolation and we will make good use of that knowledge in this and coming chapters.

---

2 Another class of problems with explicit solutions are those with linear constraints and quadratic objective that has been exploited thoroughly by researchers in many fields. While this framework provides a useful departure point, it is rarely very useful for state of the art quantitative and empirical work.

3 Perhaps the most famous of these theorems is the *Weierstrass approximation theorem*, which establishes that any continuous real-valued function over a closed interval can be uniformly approximated with polynomials. This theorem has been substantially extended in the *Stone-Weierstrass theorem* to general compact Hausdorff spaces and more general classes of basis functions.

4 As noted at the outset, I will restrict attention to techniques that are most suitable for heterogeneous-agent models. Parametric methods are covered in great detail in many textbooks. See, e.g., Judd (1998); Heer and Maussner (2009).
For example, in problem (P1), we would construct a discrete grid (for current period state variables) as the product of individual grids in each state variable—\( k_i \in G_k \equiv \{k_1, k_1, \ldots, k_I\} \) and \( z_j \in G_z \equiv \{z_1, z_1, \ldots, z_J\} \). When needed, the values off the grid are obtained by various types of interpolation to provide a continuous and differentiable decision rule over the rectangle \( G_k \times G_z = [k_0, \ldots, k_I] \times [z_0, \ldots, z_J] \). Therefore, \( V \) would be represented by a matrix of values \( \{V_{ij}, i = 1, \ldots, I; j = 1, \ldots, J\} \) on the grid and an appropriate interpolation algorithm everywhere else (off-grid).

In grid-based methods, it will be important to distinguish between the true continuous state variables and the (finite) set of grid points. We use \( (k, z) \) to denote the actual variables and denote particular values on the grid with subscript indices: e.g., \( (k_i, z_j) \). Therefore, we will seek to solve the following problem:

\[
V(k_i, z_j) = \max_{c, k'} \left\{ \frac{c^{1-\gamma}}{1-\gamma} + \beta E[V(k', z') | z_j] \right\}
\]

s.t. 
\[
c + k' = z_j k_i^\alpha + (1 - \delta)k_i
\]
\[
\ln z' = \rho \ln z_j + \eta', \quad k' \geq k.
\]

(P2)

The lack of subscript on \( c \) and \( k' \) is to emphasize that these variables are chosen from a continuous choice set. Here I have also not put a subscript on \( \eta' \), which allows \( z' \) to take on continuous values. We shall allow this approach in some applications whereas in others we will restrict \( z' \) to lie on the grid \( G_z \). As discussed in the previous chapters on numerical tools, the choice of grid, including how to space the grid points and where to place the minimum and maximum, require careful consideration. There is more to be said on these, but I will postpone that discussion until the next chapter. For the purposes of the general discussion of algorithms that follow, it will be sufficient to think that a sensibly constructed grid is available.

II. Initial Guess and Stopping Rule

(Educated) Initial Guess for \( V_0 \)

Many numerical algorithms, especially those solving with large-scale problems, have an iterative nature, which makes the starting point a critical part of the solution algorithm. A good choice can mean quick convergence to the correct solution, a bad one can slow down the algorithm or, even worse, can cause it to crash.\(^6\) Therefore,

\(^5\)And in fact even outside of it through extrapolation, but that is not recommended unless necessary. See Chapter XX.

\(^6\)Unless an algorithm is *globally convergent* (and they are the exception rather than the rule), it is only guaranteed to converge if the starting point is inside its *basin of attraction*. As we have seen, the contraction mapping theorem ensures that the VFI is globally convergent but many of the subsequent algorithms we will cover are not.
the choice of initial points will be a recurring theme in many methods covered in this book. So, how do we find a good starting point?

In many DP problems, a typically harmless choice for $V_0$ is the zero function: $V_0 \equiv 0$ or more generally a constant function $V_0 = c$. This is because a constant function is continuous, bounded, weakly convex and weakly concave, all at once. Most Bellman operators, $T$, will quickly map this function in the next iteration into something that is consistent with the properties of the problem (for example, into a strictly concave and strictly increasing one).\footnote{The zero function also has an intuitive interpretation in a life cycle model as the value function in the last period of life (with no bequests).} However, the fact that this choice is feasible does not mean that it is a good idea. Recall that the convergence rate of VFI is linear in the modulus, which is typically the discount rate, $\beta$, in our problems. This can be very slow when $\beta$ is close to 1, which is often the case (especially, in models with monthly or weekly frequencies). Thus, picking an “initial guess”, $V_0$, that is as close to the true solution as possible is essential, and $V_0 \equiv 0$ is almost never a close enough choice.

A reliable heuristic is to begin by first solving a simplified version of the problem that we eventually intend to solve. Such a simplified problem is obtained, for example, by ignoring certain constraints or eliminating uncertainty. The key is that this problem should be much faster and easier to solve than the original problem. For example, in problem (P1) above, one approach is to solve a deterministic version of this problem by setting $z$ to its mean. This problem should be much faster to solve because it has one fewer state variable: for example, if we use a discrete grid for $z$ with 20 points, the deterministic problem will require 20 times fewer maximizations. Furthermore, because there is no uncertainty, there is no need to compute a conditional expectation.\footnote{Of course, this solution will generate a} Finally, there is no need—and really not much benefit—to solve this preliminary problem to full convergence, since it is only an initial guess that will be quickly updated in the real problem. Even iterating on this problem 30 or 50 times will provide a much better initial guess than simply starting with an uneducated guess.

In the preliminary problem, it is generally advisable not to change the grid or the bound from the original problem and would definitely not change the preference parameters.\footnote{There is a set of acceleration algorithms that work by aggregating the number of states. That is, they proceed by solving the problem on a coarser grid initially, and increase the fineness of the grid as one gets closer to the true solution. Give citation.} After all, our goal is to get a good approximation to the general shape of the value function, and preferences (especially the curvature of utility) matters greatly for this shape, often much more than uncertainty or constraints.

The heuristic described here is applicable much more generally. For example, in a heterogeneous-agent model a good choice of $V_0$ is often obtained by first solving the representative-agent counterpart of the same problem. Similarly, in a general
equilibrium model with aggregate shocks, $V_0$ can be taken to be the value function that solves the counterpart problem with no aggregate shocks. And so on. In fact, it is not unusual to solve a sequence of increasingly richer preliminary models and using the solution of each as the initial guess for the next problem, all the way up to the final model that we intend to solve. However, in more complicated problems, sometimes

We discuss more examples in Chapter 21.

**Stopping Rule: When to Terminate the Algorithm?**

We terminate a numerical algorithm when the current solution $V_n$ is “sufficiently close” to the fixed point, $V^*$. In practice, of course, the latter is not known, so the stopping rule needs to rely on numbers that can be computed during the course of the numerical algorithm. Since Richard Bellman’s original contribution, a number of authors have derived increasingly tighter “error bounds” that link the computed solutions to some measure of the true error: $d(V_n, V^*)$ where $d$ is a distance measure. For example, the basic bound given in (13.1.2) says that the maximum gap between the current solution and the true one cannot be greater than the maximum gap between the current and last solutions, multiplied by $1/(1 - \beta)$. When $\beta$ is close to 1, this scaling factor can be very large, so it bears to pay attention to it.

One challenge with this formula is that it is expressed in the level of the value function. But it is not unusual for the value function to cover a very wide range of values across its domain. This is especially true when we consider a wide capital grid, when the utility curvature is high, and very low income levels are possible. One idea is to stop when the change in the value function is small in percentage terms. But this requires a lot of care because the value function can change signs (and hence cross zero!). Parameterizing the value function as a CES form as suggested in Chapter 8.5 comes in handy for this purpose too, as it ensures that the value function is always positive and its range of values does not vary too much. We will discuss other ideas later, such as basing the stopping rule on the change in decision rules (which almost always converge faster than does the value function), checking accuracy further with Euler equation errors (Marcet and den Haan, etc.), and using alternative tighter bounds that can also help accelerate the algorithm as we shall see in the next section.

**III. Initialization**

The third issue is chronologically the first step in implementing any algorithm: the initialization of various mathematical objects that are part of the algorithm. It includes the choice of bounds for each state variable, as well as the spacing of grid points: i.e., the choice of $G_k$ and $G_z$. It also covers decisions regarding how to
handle cases when one or more of the bounds are exceeded during the course of the algorithm.

**Choice of Bounds.**

The choice of bounds should be done by a careful study of the economics of the dynamic programming problem. To see a concrete example, consider again problem (P2). A natural approach would be to use the budget constraint to eliminate $c$ as a choice variable, so the Bellman objective reads:

$$
\max_{k' \in \Gamma(k_i, z_j)} \left\{ \frac{(z_j k_i^\alpha + (1 - \delta) k_i - k')^{1 - \gamma}}{1 - \gamma} + \beta \mathbb{E} (V(k', z') | z_j) \right\}. \quad (14.2.1)
$$

There are three issues to address here: (i) how to determine the bounds for $k_i$ grid? (ii) how to determine the bounds for $k'$, that is the choice set, $\Gamma(k_i, z_j)$, when performing the maximization? These two issues are closely related and will be addressed together. A third issue is how to determine bounds for $z_j$ grid. This choice is largely dictated by the stochastic properties of $z$ and has been discussed extensively in Chapter 10. We now discuss the first two issues.

**Grid for $k_i$** Because of the constraint $k' \geq k_i$, the capital value can never fall below $k_i$, which gives us the lower bound: $k_1 \equiv k_i$. The choice of upper bound, $k_I$, is less straightforward and depends on many considerations. I discuss some of them here, and others that are model-specific will be mentioned later when those models are presented.

An important consideration in deciding on the value of $k_I$ is the maximization problem that the agent faces at that state. Let $\tilde{k}'^*(k_I, z_j)$ be the optimal savings choice when the agent’s assets today is at the upper bound. It is entirely possible that this savings choice leads to a stationary distribution of capital in the long run, our job becomes a bit easier. Because then we can choose the upper bound such that the upper end
of the stationary distribution is fully within the grid that we constructed. Then
we can be sure that at \( k_I \) the agent’s optimal decision will be not to increase his
capital holdings—so the constraint will not bind. The tricky issue of course is that,
we cannot compute the stationary distribution, without first solving the dynamic
program! One solution is to first take a sufficiently large value of \( k_I \) that we can
be confident the constraint will not bind. This will require us to use lots of grid
points, possibly more than what is efficient. But once we solve this problem we can
compute the stationary distribution and then adjust \( k_I \) downward to a level that
still leaves breathing room but is not excessively wide. The problem with a grid
that is too wide is that it is not efficient as we solve the problem in a range of capital
values that will never be realized in any simulation.\(^{10}\) As I will discuss in Chapter
19, \( \text{den Haan} \) (\( \text{2010} \)) evalua tes sev e rend \( \text{f} \) erent methods that solve the Krusell-Smith
model and concludes that the most precise and fastest methods are those that do
not waste grid space in regions of the capital space that are irrelevant.

The described approach is not always feasible however. For example, sometimes
we will be calibrating or estimating such a model and as we search over the param-
eter space, the grid that is wide enough for one parameterization can be small for
another one. So the artificial constraint could be binding sometimes and not bind-
ing other times, which in turn makes the whole exercise suspect. As a precaution
against such scenarios, one could also do the following: create a virtual grid point
beyond the upper bound, \( \tilde{k}_{I+1} = k_I + (k_I - k_{I-1}) \). It should be placed probably no
more than one full grid spacing beyond the upper bound. Then define the choice set
to be \( k' \in \Gamma(k_I, z_j) = [k_1, \tilde{k}_{I+1}] \).\(^{11}\) Of course, the value function is only computed
for values of \( k \in [k_1, k_I] \), so we need to (carefully!) extrapolate between \( k_I \) and
\( \tilde{k}_{I+1} \). Because extrapolation is a very risky endeavor, one needs to be very cautious
in how this is implemented. One relatively safe idea is to take a convex combination
of a spline extrapolation and a linear extrapolation, whereby we increase the weight
of the linear component from zero at \( k_I \) to one at \( \tilde{k}_{I+1} \).\(^{12}\) Like I said, most of the
time our grid will be wide enough that this is not a major issue and \( k'(k_I, z_j) < k_I \naturally.

One of the algorithms that will be discussed later in this chapter—the endoge-
nous grid method—has the appealing feature that maximization at the upper bound
ceases to be an issue. We will see how that works momentarily.

One point of this rather long discussion is to hopefully convince the reader
that choosing grids for state variables, especially endogenous ones, requires careful
footnotes:
\(^{10}\)Expanding grids as described in Chapter 8.4 help in this respect, because doubling the upper
bound of the grid typically results in a small increase the number of points. This is because the
value function and decision rules are typically quite linear at the top end, so even few grid points
at the top end usually deliver a satisfactory results when interpolating.

\(^{11}\)Of course, we need to make sure that this choice set does not violate any other constraints,
such as, e.g., making consumption non-positive.

\(^{12}\)Specifically, \( \hat{V}(k) = \theta V_{\text{spline}}(k) + (1 - \theta) V_{\text{linear}}(k) \) and \( \theta = (\tilde{k}_{I+1} - k)/(\tilde{k}_{I+1} - k_I) \), where \( \hat{V} \)
is the final extrapolated value.
consideration. It also makes sense to automate the choice of upper bound by linking it to the parameters of the model, such as $\beta$ and $\gamma$ as well as the variance of the shock, so that when we change the calibration the grid scales automatically.

Finally, there are some other issues that arise in the construction of multidimensional grids, such as paying attention that the constraints in the problem do not prevent the construction of a rectangular grid.\footnote{Non-rectangular grids can be handled but often at high cost. They prevent the application of many convenient techniques or make them significantly more complicated.} These will be discussed in more detail in Chapter xyz.

Spacing of Grid Points  In Chapter 8, we discussed grid spacing in detail. One issue that has not been covered there is what to do when the value function or the decision rule (or both) has a kink (or multiple kinks). In such cases the advise of putting more grid points where there is curvature still broadly applies, but the location of the kinks hence of curvature is not as easy to pin down. One idea that has been pursued in some papers is to construct a double exponential grid—that is a grid that expands in two directions—starting from somewhere in the middle of the range.

14.2.1 VFI Algorithm: Details

Now that we have discussed the various essential preliminaries, we are ready to discuss how to implement the VFI algorithm on a computer.

\begin{algorithm}
\caption{VFI Algorithm: Details}
\begin{flushleft}
\textbf{Step 0.} Construct an appropriate grid for state variables: $(k_i, z_j) \in G_k \times G_z$. Choose an educated initial guess $V_0 \in S$. Set $n = 0$.

\textbf{Step 1.} For each element in the $(k_i, z_j)$ grid, execute the mapping $V_{n+1}(k_i, z_j) = TV_n(k, z)$. For $(k, z) \notin G_k \times G_z$, obtain $V_n(k, z)$ by an appropriate interpolation method.

\textbf{Step 2.} Stop if convergence criterion satisfied: $\|V_{n+1} - V_n\|_\infty < \text{toler}$. Otherwise, increase $n$ and return to step 1.
\end{flushleft}
\end{algorithm}

While this algorithm is reliable, it is often too slow. In the rest of this chapter we cover a variety of acceleration techniques that can speed it up, often by \textit{several orders of magnitude}. So, this basic algorithm serves as a foundation for what is to come, and is almost never a good one to implement on its own.
CHAPTER 14. DYNAMIC PROGRAMMING: NUMERICAL METHODS 197

14.3 Accelerating VFI: Two Useful Tricks

14.3.1 Policy Iteration Algorithm

To understand why the standard VFI algorithm is often slow, notice that it works by repeatedly applying the Bellman mapping, which in fact does two things at once. First, it maximizes the right hand side of the Bellman equation for every grid point today’s state space. That is, define \( s \equiv k' \) (for clarity of notation in the following discussion) and for all \( i, j \) solve:

\[
\hat{s}_n(k_i, z_j) := \arg \max_{s \in \mathcal{X}} \left\{ \frac{(z_j k_i^\alpha + (1 - \delta) k_i - s)^{1-\gamma}}{1 - \gamma} + \beta \mathbb{E}(V_n(s, z')|z) \right\}. \quad (14.3.1)
\]

This step can be extremely time consuming for obvious reasons. When the maximization problem is multi-dimensional, this is even more the case. The second step is then to take the maximand in the first step and evaluating the right hand side with this policy once to update the value function from \( V_n \) to \( V_{n+1} \):

\[
V_{n+1} = T \hat{s}_n V_n.
\]

Recall that this second step is simply the Howard mapping defined above and it is very fast since it only requires a single evaluation of the Bellman objective. A simple but key insight is that the Howard mapping itself is also a contraction with modulus \( \beta \). Therefore if we were to repeatedly apply \( T \hat{s}_n \), it would also converge to a fixed point itself at rate \( \beta \). Of course, this fixed point would not be the solution of the original Bellman equation we would like to solve (unless the current policy \( \hat{s}_n \) happens to be the optimal one).\textsuperscript{14} But because each application of the Howard mapping costs little, we would be converging much faster than applying \( T \). So one strategy called the (Howard’s) policy iteration algorithm is to apply Howard operator until convergence in between every application of the Bellman operator. Therefore, the \textit{VFI with policy iteration algorithm} can be written as:

Two important properties of policy iteration algorithm can be understood via an important result established by Puterman and Brumelle (1979). These authors show that policy iteration is equivalent to Newton’s method (for solving nonlinear equations) applied to dynamic programming. Thus, just like Newton’s method it has two properties: (i) it is guaranteed to converge to the true solution when the initial point, \( V_0 \), is in the domain of attraction of \( V^* \), and (ii) when (i) is satisfied, it converges at a \textit{quadratic rate} in iteration index \( n \).

\textsuperscript{14}Another way to think of what the standard VFI entails is that we go to great lengths to maximize the Bellman objective to obtain a new optimal policy but then use it only for the current period when updating from \( V_n \) to \( V_{n+1} \). Instead, Howard’s algorithm discards \( V_n \) once step \( n \) is completed and takes the new policy, \( \hat{s}_n \), to apply at all future dates to obtain \( V_{n+1} \).
Algorithm 8: VFI with Policy Iteration Algorithm

**Step 0:** Set $n = 0$. Choose an initial guess $V_0 \in S$.

**Step 1:** Obtain $\tilde{s}_n$ as in (14.3.1) and take the updated value function to be:
$$V_{n+1} = \lim_{m \to \infty} T^{m}_{\tilde{s}_n} V_n,$$
which is the (fixed point) value function resulting from using policy $\tilde{s}_n$, forever.\(^{15}\)

**Step 2:** Stop if convergence criteria satisfied: $\|V_{n+1} - V_n\|_\infty < \text{toler}$. Otherwise, increase $n$ and return to step 1.

So there is both good news and bad news. The good news is that we are able to move from the linear rate under VFI to a quadratic rate. However, this is not an apples to apples comparison, because Howard’s algorithm needs to perform many more computations per iteration compared with standard VFI, because the former needs to solve for the value function assuming policy $\tilde{s}_n$ is used forever, whereas VFI requires a single evaluation of the Bellman equation for the policy. This difference becomes especially important when the state space is large, which can slow down Howard’s algorithm considerably. We will look into a modification to alleviate this problem in a moment.

Second, Puterman and Brunelle’s bad news is that Howard’s algorithm can simply fail if we start from afar. To understand why, notice that $\lim_{m \to \infty} T^{m}_{\tilde{s}_n} V_n$ is the wrong limit for the Bellman equation, especially early on, when the current iterate of the policy rule is far away from the true solution. This means that, at a given stage $n$, the more we iterate on the Howard mapping, the more we might (and generally would) converge rapidly to the wrong point. Of course, then applying the Bellman mapping should bring us back towards the correct solution, but in general there are no guarantees. In other words, the policy improvement algorithm can generate oscillations (sometimes large ones) that can make the whole algorithm unstable, causing it to crash.\(^{16}\) Therefore, a sensible approach is to start with standard VFI (or other methods we discuss in a moment) that are guaranteed to converge and then applying policy iteration as we get closer to the solution to benefit from quadratic rates of convergence.

**Modified Policy Iteration Algorithm**

A simple modification to the basic algorithm solves both problems. The modified policy iteration algorithm essentially takes the number of applications of the Howard

---

\(^{15}\)One can prove convergence from any $v_0$ when the state space is discrete. However, in most of the analysis in this book we will consider a continuous state space, especially in key endogenous variables.

\(^{16}\)
mapping to be a finite number \( m \).\(^{17}\) Therefore we modify Step 1 above to the following:

**Algorithm 9: VFI with Modified Policy Iteration Algorithm**

*Step 1’:* Modify Step 1 of Howard’s algorithm: Obtain \( \tilde{s}_n \) as in (14.3.1) and take the updated value function to be: \( V_{n+1} = T^m \tilde{s}_n V_n \), which entails \( m \) applications of Howard’s mapping to update to \( V_{n+1} \).

In small and well-behaved problems, especially those that do not involve an outer loop (iterating on prices or other general equilibrium objects), we can usually take \( m \) to be fairly large, 50 or even 100. In my experience, for high \( \beta \), sometimes values as high as 500 continue to improve the computation speed. However, this comes with a caution that the higher \( m \) is the closer we get to the original Howard algorithm with its potential pitfalls of oscillations and non-convergence. Therefore, in high dimensional problems and with general equilibrium, it seems prudent to use a smaller value of \( m \), say 20 or less. And as noted above, it is usually advisable to start with a lower \( m \) (e.g., \( m = 1 \) is a good choice) and gradually increase it as we approach the solution.

### 14.3.2 Acceleration by Error Bounds

Another simple trick makes use of some sharp “error bounds” that can be derived for VFI algorithm. To describe the method, it is useful to adopt the discrete state representation for dynamic programs given in Section 13.6. The following theorem defines these bounds.

**Theorem 14.1. [MacQueen-Porteus Error Bounds]** In the discrete state dynamic programming problem,

\[
V(x_i) = \max_{y \in I(x_i)} \left[ U(x_i, y) + \beta \sum_{j=1}^{J} \pi_{ij}(y)V(x_j) \right], \quad (14.3.2)
\]

define

\[
\varepsilon_n = \frac{\beta}{1 - \beta} \times \min \left[ T^n V_0 - T^{n-1} V_0 \right] \quad (14.3.3)
\]

\[
\bar{\varepsilon}_n = \frac{\beta}{1 - \beta} \times \max \left[ T^n V_0 - T^{n-1} V_0 \right] \quad (14.3.4)
\]

\(^{17}\) Denardo (1967) appears to be the first one to propose this algorithm. It has been studied more extensively by van Nunen (1976) and especially Puterman and Shin (1978) (which also explores several variations of it) and Puterman and Brumelle (1979) who establish its equivalence to the Newton-Kantorovich iteration for solving nonlinear equations.
Then, for all \( x \in X \), we have:

\[
T^n V_0(\bar{x}) + \zeta_n \leq V^*(\bar{x}) \leq T^n V_0(\bar{x}) + \tau_n.
\] (14.3.5)

Furthermore, with each iteration, the two bounds approach the true solution monotonically.

The theorem has several substantive implications. First, these bounds can be quite tight. For example, suppose that in the \( k \)th iteration, the value function changes by a constant amount across all values of \( \bar{x} \), that is, \( T^n V_0(\bar{x}) - T^{n-1} V_0(\bar{x}) = \alpha \) for all \( \bar{x} \). Then equations (14.3.3) and (14.3.4) imply that \( \zeta_n = \tau_n \), which then implies

\[
\frac{\alpha \beta}{1 - \beta} = V^*(\bar{x}) - T^n V_0(\bar{x}) = \frac{\alpha \beta}{1 - \beta}.
\]

Therefore, \( V^*(\bar{x}) = T^n V_0(\bar{x}) + \frac{\alpha \beta}{1 - \beta} \), and we are done! Although this is a special example that is not likely to arise in real applications, it illustrates the power of these bounds more generally.

Second, it is also useful to contrast these bounds to the simpler bounds derived from the simple contraction argument, given in the previous chapter. This was

\[
\|V^n - T^n V_0\|_\infty \leq \frac{1}{1 - \beta} \|T^n V_0 - T^{n-1} V_0\|_\infty = \frac{\alpha \beta}{1 - \beta}.
\] (14.3.6)

First, unless \( \beta = 1 \), this bound is looser than the MacQueen-Porteus bound. But second and more importantly, it provides an upper bound in the supnorm, whereas (14.3.5) provides both an upper and a lower bound. For example, in the usual VFI, we would iterate until the upper bound in (14.3.6) is smaller than a small tolerance value. However, with Theorem 14.1 we would stop when \( \tau_n - \zeta_n \) is smaller than a small tolerance. In the example just given, \( \tau_n = \frac{\alpha \beta}{1 - \beta} \) and this value might be a large number, say 100, but we would still stop because \( \tau_n - \zeta_n = 100 - 100 = 0 \), whereas we would keep iterating under VFI with regular bounds since the upper bound is still high.

Therefore, an improved algorithm replaces the last step of the VFI algorithm as follows:

This algorithm deserves attention for several reasons. First, it is really simple to implement. It only requires the computation of the maximum and minimum deviation, which can be performed quite easily and quickly. Second, its convergence is guaranteed under the same assumptions about the convergence property of the contraction mapping theorem. Third, its convergence rate can be quite fast in certain situation. In particular, using the discrete state Markov decision problem framework presented in Section 13.6, Bertsekas (2001) proves that the convergence
Algorithm 10: VFI with MacQueen-Porteus Error Bounds

Step 2': Stop when $c_n - c_{n-1} < \text{toler}$. Then take the final estimate of $V^*$ to be either the median
\[
\hat{V} = T^n V_0 + \left( \frac{c_n + c_{n-1}}{2} \right)
\]
or the mean (i.e., average error bound across states):
\[
\hat{V} = T^n V_0 + \frac{\beta}{n(1-\beta)} \sum_{i=1}^{n} (T^n V_0(x_i) - T^{n-1} V_0(x_i)).
\]

| Table I – Mc-Queen Porteus Bounds and Policy Iteration: Speed Improvements |
|-------------------------|---------|---------|---------|---------|---------|---------|---------|
| $\beta$ →              | 0.95    | 0.99    | 0.999   |
| $\frac{MQP/N_{max}}{=} $| 0      | 50      | 500     | 0      | 50      | 500     | 0      | 50      | 500     |
| γ = 1                  |         |         |         |         |         |         |         |         |         |
| no                     | 14.99   | 1.07    | 1.00    | 26.48  | 1.28    | 1.00    | 33.29  | 1.41    | 1.00    |
| yes                    | 0.32    | 0.60    | 0.79    | 0.10   | 0.23    | 0.27    | 0.01   | 0.03    | 0.04    |
| γ = 5                  |         |         |         |         |         |         |         |         |         |
| no                     | 13.03   | 0.96    | 1.00    | 26.77  | 1.28    | 1.00    | 33.37  | 1.45    | 1.00    |
| yes                    | 0.67    | 0.67    | 0.69    | 0.14   | 0.24    | 0.30    | 0.02   | 0.04    | 0.06    |

Note: $N_{max}$ indicates the maximum number iterations in the modified policy algorithm. Results remain unaffected if we stop at $N_{max} = 100$ or for intermediate values of utility curvature. The timing for $N_{max} = 500$ and no MacQueen-Porteus bounds is normalized to 1.0 for each $(\beta, u)$ combination to show relative speed gains. The second row in each panel (marked with MPQ=yes) report the fraction of time it takes the computation to finish when MacQueen-Porteus bounds are applied relative to the cell immediately above it (with the same number of iterations in policy stage).

Rate of this algorithm (for both $\bar{V}$ and $\hat{V}$) is governed by the modulus of the subdominant eigenvalue of the transition matrix $\pi_{ij}(y^*)$, that is, the eigenvalue with the second largest modulus.\(^\text{18}\) Recall that if the transition matrix has a unique ergodic distribution, then its largest eigenvalue is 1 and when multiplied with the discount factor, this yields a convergence at rate $\beta$. When $\beta$ is close to 1, the VFI algorithm becomes extremely slow. Bertsekas’ result shows that under the right conditions, we can improve from the largest eigenvalue to the second largest. As long as this is less than 1, the convergence rate can be faster and often much faster. In the important special case where a Markov chain is constructed by discretizing an AR(1) process, the subdominant eigenvalue is given by the persistence parameter $\rho$. Therefore, the less persistent the stochastic process is the more MacQueen-Porteus bounds will help accelerate convergence.

\(^\text{18}\)In Bertsekas’ formulation the state space is discrete but can be infinite. Notice that $\pi_{ij}(y^*)$ is the transition matrix induced by the optimal policy.
14.4 VFI with Endogenous Grid Method

As noted earlier, the different versions of the VFI algorithm described so far all face the same bottleneck: the maximization step of the algorithm is slow and accounts for a substantial fraction of the computational time of each algorithm. The endogenous grid method (EGM) introduces two key modifications that can significantly speed up the maximization step and, consequently, the VFI algorithm itself.

First, we construct the capital grid based on tomorrow’s values and let today’s capital grid be determined endogenously. Second, we adopt a slightly different timing for selecting the state variables. In particular, we write the value function as a function of the end-of-period total resources, rather than the usual choice of beginning-of-period capital.

To understand how these modifications help, first recall that the VFI method presented in the previous section proceeds by backward iteration and maximizes the right-hand-side of the Bellman equation. Assuming an interior solution, the FOC is:

\[ c^{-\gamma} = \beta \mathbb{E}(V_k(k', z') | z_j), \]

where \( V_k \) denotes the derivative of \( V \) with respect to capital. This equation can be rewritten (by substituting out consumption using the budget constraint) as

\[
(z_j k_i^\alpha + (1 - \delta) k_i - k')^{-\gamma} = \beta \mathbb{E}(V_k(k', z') | z_j),
\]

(14.4.1)

In the standard VFI, we solve this equation for \( k' \) for each value of \( k_i \) and \( z_i \). This is a costly computation for three reasons. First, this is a non-linear equation in \( k' \). Second, when searching for a solution over values of \( k' \), we need to repeatedly evaluate the conditional expectation on the right hand side for every trial value of \( k' \) (since it appears inside the expectation). Third, in standard VFI, the value function is stored at grid points defined over \( k \), so for every trial value of \( k' \), we need to interpolate (or, even worse, extrapolate!) the tabulated values \( \{V(k_i, z_j), i = 1, ..., I; j = 1, ..., J\} \) to obtain off-grid values \( V(k', z'_j) \) for each \( z'_j \).

Each one of these steps add significantly to the cost of solving equation (14.4.1).

In the endogenous grid method (hereafter, EGM), we take the same equation but use it differently. The idea is to construct a grid for tomorrow’s capital, \( k' \), and today stochastic state \( z \). We then solve for \( c \) and \( k \) as a function of \( k'_i \) and \( z_j \) only

---

19 Of course, if we proceed this way, at the very least we should construct the interpolating function, such as a spline, and compute its coefficients once and store them. Then for every choice of \( k_{t+1} \) we can simply use the stored coefficients to evaluate the interpolant at new points, which is much faster than re-computing coefficients (which is redundant).
on these grid points. That is, we view the problem as:

\[
V(k, z_j) = \max_{c, k'} \left\{ \frac{c^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}(V(k', z') | z_j) \right\}
\]

s.t. \[c + k' = z_j k^\alpha + (1 - \delta)k\]
\[\ln z' = \rho \ln z_j + \eta',\]
\[k' \geq k.\]  

Comparing this equation to problem P2 above, notice the subtle change in the placement of the subscript \(i\) from today’s capital stock previously, to tomorrow’s capital here. (I also switched to a discrete shock space, which will come in handy in a moment). To see how this simplifies things, notice that the problem is going to be solved for capital values, \(k_i\), on the grid tomorrow. This step involves no interpolation since \(k_0\) is on the grid and, as well shall see, we will compute this once. Now rearrange the right hand side of (14.4.1) to obtain:

\[
z_j k^\alpha + (1 - \delta)k = [\beta \mathbb{E}(V_k(k', z') | z_j)]^{-1/\gamma} + k'.
\]

Solving this equation for \(k\) is easier, because—even though we still need to solve a nonlinear equation—the conditional expectation needs to be computed once, unlike what we had to do before (as we searched over values of \(k'\)).

The second clever idea in the EGM method is that we can avoid even solving this nonlinear equation by carefully changing the state variable from the beginning-of-period capital stock, \(k_i\), to end-of-period total cash-on-hand, or equivalently, output plus undepreciated capital. Define

\[
Y \equiv zk^\alpha + (1 - \delta)k
\]

and rewrite the Bellman equation as:

\[
V(Y, z) = \max_{k'} \left\{ \frac{(Y - k')^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}(V(Y', z') | z_j) \right\}
\]

s.t. \[\ln z' = \rho \ln z_j + \eta'.\]
\[Y' = z' k'^\alpha + (1 - \delta)k'.\]

The key observation is that \(Y'\) is only a function of \(k'_i\) and \(z'\), so we can write the conditional expectation on the right hand side as:

\[
\mathbb{E}(V(k'_i, z_j) \equiv \beta \mathbb{E}(V(Y'(k'_i, z'), z') | z_j).
\]
The problem can then be written as:

$$V(Y, z) = \max_{k'} \left\{ \frac{(Y - k')^{1-\gamma}}{1 - \gamma} + V(k'_i, z_j) \right\}$$

Now the FOC of this new problem becomes:

$$c^*(k'_i, z_j) = \nabla V(k'_i, z_j).$$  \hspace{1cm} (14.4.3)

Having obtained $c^*(k'_i, z_j)$ from this expression, we use the resource constraint to compute today’s end-of-period resources, $Y^*(k'_i, z_j) = c^*(k'_i, z_j) + k'_i$, as well as

$$V(Y^*(k'_i, z_j), z_j) = \frac{(c^*(k'_i, z_j))^{1-\gamma}}{1 - \gamma} + V(k'_i, z_j)$$
on the left-hand-side. Therefore, given tomorrow’s grid, we are able to find today’s state variable, $Y^*$, and choice variable without requiring a non-linear solver. However, the values of $V$ we obtained are on a discrete grid of values given by $(Y^*(k'_i, z_j), z_j)$ which is not on the $(k'_i, z_j)$ grid, since $Y^*$ is not on this grid (necessarily). But this is easy to fix. We can obtain the values of $V$ on the latter exogenous grid by interpolation. Then we can plug the updated and interpolated value function into the RHS and iterate until convergence.

Notice that we have not solved for the beginning-of-period capital stock, $k$. This can be obtained by solving (14.4.2) for $k$, given grid points of $Y(i, j)$. This is a non-linear equation, but needs to be solved only once, after the Bellman equation has been solved. Here is the full algorithm.

14.4.1 EGM With CES Curvature Reduction Formulation

WORK in PROGRESS....

14.4.2 Other Advantages/Disadvantages of EGM

◆

14.4.3 EGM with Labor Supply

Algorithm 11: Endogenous Grid Method

Step 0: Set $n = 0$. Construct a grid for tomorrow’s capital and today’s shock: $(k^t_i, z_j)$. Choose an initial guess $V^0(k^t_i, z_j)$.

Step 1: For all $i, j$, obtain

$$c^*(k^t_i, z_j) = (V^0_k(k^t_i, z_j))^{-1/\gamma}.$$  

Step 2: Obtain today’s end-of-period resources as a function of tomorrow’s capital and today’s shock:

$$Y^*(k^t_i, z_j) = c^*(k^t_i, z_j) + k^t_i,$$

and today’s updated value function,

$$V^{n+1}(Y^*(k^t_i, z_j), z_j) = \frac{(c^*(k^t_i, z_j))^{1-\gamma}}{1 - \gamma} + V^n(k^t_i, z_j)$$

by plugging in consumption decision into the RHS.

Step 3: Interpolate $V^{n+1}$ to obtain its values on a grid of tomorrow’s end-of-period resources: $Y' = z'(k^t_i)^{\alpha} + (1 - \delta)k^t_i$.

Step 4: Obtain

$$V^{n+1}(k^t_i, z_j) = \beta E \left( V^{n+1}(Y'(k^t_i, z'), z') | z_j \right).$$

Step 5: Stop if convergence criterion is satisfied and obtain beginning-of-period capital, $k$, by solving the nonlinear equation $Y^{n*}(i, j) = z_j k^\alpha + (1 - \delta)k$, for all $i, j$. Otherwise, go to step 1.

14.4.4 EGM with Non-Differentiable Value Functions

$\blacklozenge$ Fella (2014)

14.5 Euler Equation Method

$$1 = E \left[ \beta U''(c^\alpha(z^{t+1})) \right].$$

Let $F(k) = zk^\alpha + (1 - \delta)k$

$$U'(c_{n+1}(k, z)) = E [\beta U'(c_n(F(k, z) - c_{n+1}(k, z), z')) \times F'(F(k, z) - c_{n+1}(k, z), z')] .$$
Algorithm 12: Endogenous Grid Method with CES Trick

Step 0: Set $n = 0$. Construct a grid for tomorrow’s capital and today’s shock: $(k'_{i}, z_{j})$. Choose an initial guess $V^{0}(Y', z_{j})$ and compute $V^{0}$.

Step 1: For all $i, j$, obtain

$$c^{*}(k'_{i}, z_{j}) = E \left( (1 - \gamma) \frac{\partial Y'(Y', z_{j})}{\partial Y'} \frac{\partial Y'(k_{i}, z')}{\partial k'_{i}} (V^{n}(Y', z_{j}))^{1-\gamma} \right)^{-1/\gamma}.$$

Step 2: Obtain today’s end-of-period resources as a function of tomorrow’s capital and today’s shock:

$$Y^{*}(k'_{i}, z_{j}) = c^{*}(k'_{i}, z_{j}) + k'_{i},$$

and today’s updated value function,

$$V^{n+1}(Y^{*}(k'_{i}, z_{j}), z_{j}) = \left[ \frac{(c^{*}(k'_{i}, z_{j}))^{1-\gamma}}{1 - \gamma} + V^{n}(k'_{i}, z_{j}) \right]^{1/(1-\gamma)}$$

by plugging in consumption decision into the RHS.

Step 3: Interpolate $V^{n+1}$ to obtain its values on a grid of tomorrow’s end-of-period resources: $Y' = z'(k'_{i})^{\alpha} + (1 - \delta)k'_{i}.$

Step 4: Obtain

$$V^{n+1}(k'_{i}, z_{j}) = \beta E \left( V^{n+1}(Y'(k'_{i}, z'), z') | z_{j} \right).$$

Step 5: Stop if convergence criterion is satisfied and obtain beginning-of-period capital, $k$, by solving the nonlinear equation $Y^{**}(i, j) \equiv z_{j}k^{\alpha} + (1 - \delta)k$, for all $i, j$. Otherwise, go to step 1.

$$U'(c_{n+1}(k_{i}, z_{j})) = E [\beta U'(c_{n}(F(k_{i}, z_{j}) - c_{n+1}(k_{i}, z_{j}), z')) \times (1 + R(z'))].$$

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

At $t = T$, we know that $c(k'_{i}, z'_{j}) = k'_{i}$

Guess and Verify (Policy Functions)

Let the policy rule for savings be: $k' = g(k)$. The Euler equation is:

$$\frac{1}{Ak^{\alpha} - g(k)} - \frac{\beta \alpha A (g(k)^{\alpha-1})}{A (g(k)^{\alpha} - g(k))} = 0 \quad \text{for all } k.$$
which is a functional equation in \( g(k) \). Guess \( g(k) = sAk^{\alpha} \), and substitute above:

\[
\frac{1}{(1 - s)Ak^{\alpha}} = \alpha A \left( (sAk^{\alpha})^{\alpha - 1} - sA(aAk^{\alpha})^{\alpha - 1} \right)
\]

As can be seen, \( k \) cancels out, and we get \( s = \alpha \beta \). By using a very flexible choice of \( g \), this method too can be used for solving very general models.

- We need Euler equation to be sufficient, so concave utility and convex choice set. Not always possible.

14.6 Taking Stock

Today, the state-of-the-art for solving dynamic programming problems is the endogenous grid method. It dominates the other methods discussed in this chapter (as well as others not covered here) in terms of both computational speed and programming time. Furthermore, it has been extended to models with non-concave objectives, problems with endogenous labor supply and is able to deal easily with certain types of constraints commonly encountered in economic applications. So, when it is applicable it should be your first choice. The problem is that it is not always applicable, perhaps most importantly, when the problem features more than one endogenous state variable. And these are the problems that are especially challenging in terms of overall speed requirements. In such cases, the modified policy iteration algorithm and the MacQueen-Porteus error bounds can be especially helpful in speeding up the solution. The bottom line is that all three of these methods should be part of your computational toolbox.

Policy iteration was first proposed Bellman (1957)’s seminal book (which he called “iteration in policy space”), but was substantially generalized by Howard (1960) and later by Denardo (1967), who also proved convergence results in increasingly more general frameworks.

Talk about Powell’s and Karatzas and Shreve’s methods that also rely on a different timing for the Bellman equation.

The idea of parameterizing the conditional expectation as a function of tomorrow’s state variables goes back to Wright and Williams (1984), who used a parameterized expectations approach with current period decision (and next period’s state). The endogenous grid method is originally due to Carroll (2006) and makes use of a similar idea but in a very effective way.