Chapter 5

A Quick Introduction to Numerical Methods

One of the main advantages of the recursive approach is that we can use the computer to solve numerically interesting models. There is a wide variety of approaches. Each method uses a different characteristic of optimality involved in the dynamic programming method. In this chapter, we only consider - briefly - a couple of approaches which both use very heavily the Bellman equation (BE) we studied in the previous chapters. For a more exhaustive analysis we suggest Marimon and Scott (1999), Judd (1998), and Santos (1999).

5.1 Value Function Iteration

Let X the state space. Recall that the BE in the deterministic case takes the following form:

$$V(x) = \max_{x' \in \Gamma(x)} F(x, x') + \beta V(x').$$

The theoretical idea behind the value function iteration approach is to use the contraction mapping generated by the Bellman operator $\mathbf{T}$ associated to the dynamic programming problem to derive the value function $V = \mathbf{T}V$.

The algorithm recalls the 'guided guesses' approach we saw in Chapter 1, and it works as follows:

1. Pick an initial guess V_0 for the value function: $V_0 : X \to \mathbb{R}$.
2. Apply the $\mathbf{T}$ operator to it and get a new function $V_1 = \mathbf{T}V_0$

3. Then apply again $\mathbf{T}$ to V_1 and get $V_2 = \mathbf{T}V_1$ and so on until the fixed point of the operator: $V^* = \mathbf{T}V^*$ is reached.

Note few things:

First, the maximization required by $\mathbf{T}$ should be done by using *numerical algorithms*. There are very many numerical algorithms available, each with its own advantages and disadvantages. We will not analyze this specific aspect of the problem here. We demand the interested reader to Judd (1998) and the literature cited there.

Second, the operator $\mathbf{T}$ works in the space of functions. Each time the state space is not a finite set the space of functions constitute an infinite dimensional space. Clearly, a computer cannot deal with infinite dimensional spaces. We are hence forced to reduce the mapping $\mathbf{T}$ to map finite dimensional spaces into itself. This is done by *approximating the (value) function*. The two most commonly used methods of dealing with such infinities are the discretization and the smooth approximation methods. We will briefly analyze both of them in turn. We will then analyze another method of approximating the value function which uses directly the functional equation: the collocation method.

Third, the computer cannot perform an infinite amount of iterations in finite time. The exact fixed point condition $V^* = \mathbf{T}V^*$ is however very difficult to obtain in finitely many iterations. We are hence forced to allow for a degree of *tolerance*.

5.1.0.1 Discretization

The idea of this method is to substitute the value function with a discrete version of it by discretizing the state space. In this way, we solve two problems at once. First, the (approximated) Bellman operator $\hat{\mathbf{T}}$ maps finite dimensional vectors into finite dimensional vectors. Second, the maximization procedure at each step is particularly simple. The algorithm works as follows:

1. Take the state space X, and discretize it, say $\hat{X} = \left[x^1,\ x^2,\ x^3, ..., x^N\right]$.
2. Pick an initial guess V_0 by associating one value to each state level, say $\hat{V}_0 = \left[v_0^1,\ v_0^2, ..., v_0^N\right]$.
3. For each $x^i \in \hat{X}$, look for the $x^j (= x') \in \hat{X}$ which solves

$$\max_{x^j \in \hat{X},\ x^j \in \hat{\Gamma}(x^i)} F\left(x^i, x^j\right) + \beta v_0^j$$

where $\hat{\Gamma}$ is possibly an approximated version of the feasibility due to discretization. Sometimes we do not need to relax Γ as for example we can simply substitute it into

F and solve for an unconstrained problem. Notice that this a very simple procedure as it involves the choice of the maximal element among finitely many.

4. Denote v_1^i the value associated to $F\left(x^i, x^{j^*}\right)+\beta v_0^{j^*}$ where j^* is the index that delivers the maximal value at the previous stage.

5. If we do it for all x^i we get a new N dimensional vector of values which constitutes our new (discretized) function $\hat{V}_1 = \left[v_1^1,\ v_1^2, ..., v_1^N\right]$.

6. And so on until the vectors $\hat{V}_n$ and $\hat{V}_{n+1}$ are 'close' enough to each other. Where 'close' is defined according to some metric. For example:

$$d\left(\hat{V}_n, \hat{V}_{n+1}\right) = \sum_i \varpi^i \left|v_n^i - v_{n+1}^i\right| < \varepsilon,$$

where ϖ^i are pre-specified weights and ε is the tolerance level.

Example: The Deterministic Optimal Growth Model In order to see more precisely how this method works, we will consider its application to the optimal growth model.

First of all, we need to parametrize the problem. We use the Cobb-Douglas production function and a CRRA utility:

$$f(k) = k^\alpha + (1-\delta)\,k \ \text{ and } u(c) = \frac{c^{1-\sigma}}{1-\sigma},$$

where δ is the depreciation rate, α is the capital share, and $\frac{1}{\sigma}$ is the intertemporal elasticity of substitution.

We will consider the special case where $\sigma = 1$. In this case, $u(c) = \ln c$. In order to compare our approximated solution to the exact one, we start by assuming a 100% depreciation rate, i.e. $\delta = 1$.

The problem to be solved is hence

$$V(k) = \max_{0\le k'\le k^\alpha} \ln\left(k^\alpha - k'\right) + \beta V\left(k'\right),$$

We know from Chapter 1 that in this case $g(k) = \alpha\beta k^\alpha$. We now compute it by using the discretization method. We shall open the `Matlab` program and write our computation code:[1]

[1] I thank Liam Graham for lending me his simple Matlab code.

Initialize the problem

```
clear all;
close all;
```

Define parameters

```
beta=.9; % β = .9
alpha=.35; % α = .35
NumPoints =100;
```

Discretize the state space around the steady state capital stock

```
k_bar = (alpha*beta)^(1/(1-alpha)); % Recall that k* = (αβ)^(1/(1-α))
k_lo = k_bar*0.5;
k_hi = k_bar*2;
step = (k_hi-k_lo)/NumPoints;
K = k_lo:step:k_hi;
n=length(K);
```

Since loops in Matlab are very slow, while matrix manipulations are very fast, we first build an $n \times n$matrix whose columns are output at each value of k

```
Y= K.^alpha;
YY = ones(n,1)*Y;
```

Then another $n \times n$matrix whose columns are capital

```
KK = ones(n,1)*K;
```

Consumption at each level of $k'$is then given by

```
C=YY-KK';
```

Calculate the utility arising from each level of consumption

```
U=log(C);
```

Take an initial guess at the value function

```
V = zeros(n,1);
```

Apply the operator:

$$W = U + \beta \begin{bmatrix} 0 \\ 0 \\ .. \\ 0 \end{bmatrix} [1\ 1\ .....1]$$

```
VV=V*ones(1,n);
W=U+beta*VV;
```

Given a k, we want to find the k' that solves

$$\begin{bmatrix} TV\left(k^1\right) \\ TV\left(k^2\right) \\ .. \\ TV\left(k^N\right) \end{bmatrix} = \max\ W$$

```
V=max(W)';
```

Main iteration loop for the value function.

```
flag=1;
while (flag > 10^ (-2))
    VV=V*ones(1,n);
    W=U+beta*VV;
    V1=max(W)';
    flag = max(abs(V1-V));
    V=V1;
end
```

When the value function has converged, find the policy function i.e. the k' that gives the maximum value of the operator for each k. In order to accomplish this, we first find the vector of indices where W takes its maximum value:

```
[val,ind]=max(W);
```

Then we use the indices to pick out the corresponding values of k.

```
k_star = K(ind);
```

Finally, let's keep track of the analytical optimal policy for comparison purposes:

```
k_true = K.*alpha*beta;
```

We can then plot the two policies in the same figure.[2]

Exercise 45 *Open the file* `discrete.m` *in* `Matlab`*. Compute and plot the policy for increasing levels of the discount factor, say for* β *between* .9 *and* .99*. Comment on the different computation times needed for convergence. Now set back* $\beta = .9$ *and increase the range of values of* k *between* $0.4k^*$ *and* $5k^*$ *and perform the same computations. Comment your results. [Hint: Be careful that a too wide grid might create negative consumption, and obviously you do not want that!] Now modify the code so that to allow for a depreciation rate* δ *below 100%. Produce figures for different values of* α *and* δ *: Say,* α *between* .2 *and* .5 *and* δ *between* .05 *and* .3*. Comment your results from an economic point of view.[Warning: Recall that consumption must be nonegative.]*

[2]Creating plots with `Matlab` is quite easy. Have a look at the file: `discrete.m`

5.1.0.2 Smooth Approximation

This method reduces substitutes the value function V with a parametrized one V_θ where $\theta \in \Theta$ and Θ is a subset of an $k-$dimensional space.

In order to have a theoretically justified procedure, we require that V_θ is 'potentially' able to approximate V very well. Formally, we require:

$$\lim_{k\to\infty} \inf_{\theta\in\Theta\subset\Re^k} \|V_\theta - V\|_\infty = 0$$

where $\|\cdot\|_\infty$ is the sup norm, that is, $\|V\|_\infty = \sup_{x\in X} |V(x)|$. This kind of approximation are said *dense.*

One of the most commonly used dense sets is the set of polynomials, which is dense in the space of continuous functions by the Weierstrass theorem. In this case, V is approximated (or interpolated) by

$$V_\theta(x) = \sum_{i=1}^{k} \theta^i p^i(x), \text{ for all } x \in X,$$

where $p^i(x)$ is the $i-th$ order polynomial,[3]or the Chebyshev polynomial $p^i(x) = \cos(i\ arc\cos(x))$, Legendre, Hermite polynomials, Splines, etc... . The number of (independent) polynomials is called the *degree of approximation.*

We will see, that also in this case we use a discretized version $\hat{X}$ of the state space. In this case, for a different propose. The value function will indeed be defined on all the original state space X.

The numerical algorithm works as above:

1. Discretize the state space to $\hat{X}$, fix the size k and the type p^i of polynomials you want to consider (say the set of the first 20 Chebyshev polynomials). Where $\sharp\hat{X} > k+1$.

2. Start with a vector of weights $\theta_0 = \left[\theta_0^1,\ \theta_0^2,\ \theta_0^3,\ ....,\ \theta_0^N\right]$. This gives you the initial guess: $\hat{V}_0(x) = \sum_{i=1}^{k} \theta_0^i p^i(x)$ for all $x \in X$.

3. For each $x^i \in \hat{X}$, define

$$\widehat{\mathbf{T}}\left(V_{\theta_0}(x^i)\right) = \max_{x'\in\Gamma(x^i)} F\left(x^i, x'\right) + \beta\hat{V}_0(x')$$

where Γ is the original correspondence since x' is now allowed to vary in the whole X. In this case, the maximization stage requires the use of a numerical algorithm

[3]A i-th order polynomial takes the form $p^i(x) = \sum_{s=1}^{i} a_s x^s$, where the superscript s indicates the power s of x.

as x varies over a continuum. This is obviously more demanding. The advantage is that, all values of x are somehow evaluated when obtaining the value function.

4. Compute the new vector of weights θ_1 by minimizing some error function, for example a weighted least square criterion as follows:

$$E_N(\theta_1;\theta_0) \equiv \sqrt{\sum_{i=1}^{N} \varpi^i \left| V_{\theta_1}(x^i) - \widehat{\mathbf{T}}\left(V_{\theta_0}(x^i)\right)\right|^2} \tag{5.1}$$

where $x^i \in \hat{X}$ belong to the pre-specified grid of points, and ϖ^i are appropriate weights. Some points in the grid might indeed be more important the others (for example, consider the points close to the steady state for optimal growth models).

5. Do so until a vector of weights θ_n is reached such that for example $\sum_{i=1}^{k} \left|\theta_n^i - \theta_{n-1}^i\right| < \varepsilon$, where ε is the tolerance level. We then can evaluate the approximation error via $E_N(\theta_n;\theta_n)$.

5.2 Solving Directly the Functional Equation: Projection Methods

We saw that the BE also constitutes a functional equation of the form: $\mathbf{T}V - V = 0$. Another possibility is hence to directly look for a solution of the functional equation.

Again, the Bellman functional equation typically imposes an infinite number (in fact possibly a continuum) of conditions, namely:

$$(\mathbf{T}V)(x) - V(x) = 0 \text{ for all } x \in X.$$

And a computer cannot deal with such huge number of equations. One must therefore settle for an approximate solution that satisfies the functional equation closely. Projection methods approximate the function V with V_θ and then look for the vector of parameters θ^* which minimizes the distance between $V_\theta(x) - \widehat{\mathbf{T}}\left(V_\theta(x)\right)$, such us the error $E_N(\theta;\theta)$ for example. Note however that there is another complication involved with the above equation. The function $\widehat{\mathbf{T}}\left(V_\theta(x)\right)$ is not easy to compute. In particular, there is no analytical way of getting it. This is the reason why Projection methods are combined with policy function iteration.

The Euler equation though is also a characteristic of the optimal program. In particular, if $g(x)$ is the value function of our problem, then in the optimal growth model, with

$u = \ln$ we have

$$\frac{1}{k^\alpha - g(k)} = \beta\alpha \left(g\left(k\right)\right)^{\alpha-1} \frac{1}{\left(g\left(k\right)\right)^\alpha - g\left(g\left(k\right)\right)}.$$

In general we have a functional equation of the form

$$H\left(x, g\left(x\right)\right) = 0$$

where the unknown g is our target, while $H : \mathbb{R}^2 \to \mathbb{R}$ is a known function.

Projection methods then approximate the function g with $\hat{g}\left(x;\theta\right)$ and look for the vector of weights θ^* that minimizes a given error function based on H and on a nonnegative function ϕ:

$$\theta^* \in \arg\min_\theta \int \phi\left(x\right) \left|H\left(x, \hat{g}\left(x;\theta\right)\right)\right|,$$

with $\phi\left(x\right) \geq 0$ for all x. When ϕ takes positive values only at finitely many points in X, we obviously only evaluate the function at few points in a given grid $\hat{X}$ ad we get an error function such as that in (5.1). In this case, the method is called *collocation method*.[4] In practice, these techniques constitute other methods to approximate functions. The collocation method for example, is a generalization of the so called *interpolation methods*.

Example The collocation method applied to the optimal growth model looks for parameters $\theta = (\theta^1, \theta^2, ...\theta^q)$ that minimize

$$\sqrt{\sum_{i=1}^{N} \varpi^i \left| \frac{1}{k_i^\alpha - \hat{g}(k_i;\theta)} - \beta\alpha \left(\hat{g}\left(k_i;\theta\right)\right)^{\alpha-1} \frac{1}{\left(\hat{g}\left(k_i;\theta\right)\right)^\alpha - \hat{g}\left(\hat{g}\left(k_i;\theta\right);\theta\right)} \right|^2},$$

where k_i, ϖ^i $i = 1, 2, ..N$, $N > q + 1$ are the points in a pre-specified grid of capital levels and given weights respectively, and $\hat{g}$ is for exmple a polynomial of the form

$$\hat{g}(k;\theta) = \theta^0 + \theta^1 k + \theta^2 k^2 + ... + \theta^q k^q.$$

[4] See McGrattan's chapter 6 in Marimon and Scott's book for several other methods.

Bibliography

[1] [AC] Adda J., and Cooper (2003) *Dynamic Economics: Quantitative Methods and Applications,* MIT Press.

[2] Judd, K.L. (1998), *Numerical Methods in Economics*, Cambridge, MA, MIT Press.

[3] Marimon, R., and A. Scott (1999) *Computational Methods for the Study of Dynamic Economies*, Oxford University Press.

[4] Santos, M. S. (1999) "Numerical Solutions of Dynamic Economic Models," in J. B. Taylor, and M. Woodford eds., *Handbook of Macroeconomics*, Vol. 1A, Elsevier: 311-386.

[5] Sargent, T. (1987) *Dynamic Macroeconomic Theory*, Harward University Press.

[6] [LS] Ljungqvist, L., and T. J. Sargent (2004), *Recursive Macroeconomic Theory*, Second Edition, MIT Press.

[7] [SLP] Stokey, N., R. Lucas and E. Prescott (1991), *Recursive Methods for Economic Dynamics*, Harvard University Press.