Notes on Numerical Dynamic Programming in Economic Applications

Moritz Kuhn*

CDSEM Uni Mannheim

preliminary version

18.06.2006

*These notes are mainly based on the article *Dynamic Programming* by John Rust(2006), but all errors in these notes are mine. I thank the participants of the joint seminar on *Optimal Control in Economic Applications* of the *Institute of Scientific Computing* at the University of Heidelberg and of the Economics Department at the University of Mannheim for their helpful comments. I would be very thankful for further comments: mokuhn@rumms.uni-mannheim.de

Contents

1 Introduction and Motivation

Dynamic Programming is a recursive method for solving *sequential decision problems* .
In economics it is used to find optimal decision rules in deterministic and stochastic environments[1], e.g. to identify *subgame perfect equilibria* of dynamic multiplayer games, and to find *competitive equilibria* in dynamic market models[2].
The term *Dynamic Programming* was first introduced by Richard Bellman, who today is considered as the inventor of this method, because he was the first to recognize the common structure underlying most *sequential decision problems* . Today *Dynamic Programming* is used as a synonym for *backward induction* or *recursive*[3] *decision making* in economics.
Although *Dynamic Programming* is a more general concept it is most of the time assumed that if there is an underlying stochastic process that the process has the *Markov property.* This is only due to tractability of the problem, especially if we move to infinite horizon problems. The crucial limitation for *Dynamic Programming* is the exponential growth of the state space, what is also called the *curse of dimensionality.*

2 Problem formulation

To emphasize the generality of the method of *Dynamic Programming* I start by formulating a very general class of problems to which *Dynamic Programming* as a solution method can be applied. The problem formulation allows for both *deterministic* and *stochastic sequential decision problems* . First I will only consider finite horizon problems and deterministic states. I will also make explicit that this class of problems also contains the problem of finding *subgame perfect equilibria* in extensive form games[4].

[1]Following Rust(2006) I use for stochastic environments sometimes the phrase *games against nature.* I will hopefully become clearer later, why this term is sometimes used in economics.

[2]Although some reader might not yet be familiar with these terms it will become clear later what these terms mean.

[3]The term recursive will show up throughout the discussion of this topic. Economists usually capture by the term recursive that the state space is time invariant. In stochastic problems people also sometimes assume at the same time that the underlying stochastic process is *Markovian.*

[4]See appendix A for definition of an extensive form game.

2.1 Finite horizon problems

The problem we want to solve is the following

$$\max_{u_t} \quad \sum_{t=0}^{T} \beta^t F(u_t, x_t) + \beta^T H(u_T, x_T) \tag{1}$$

$$s.t. \quad u_t \in \Gamma(x_t) \ \forall\, t \tag{2}$$

$$x_{t+1} \in \mathbb{L} \forall\, t$$

$$x_{t+1} = \psi(u_t, x_t) \tag{3}$$

$$x_0 \ given$$

$$\tag{4}$$

We assume that there is an underlying *state space* X and we assume that $u_t \in U \subset \mathbb{R}^m$.
This problem formulation describes a family of boundary value problems, because we want to solve the problem for the set $\{x_0 \in X : \exists \{u_t\}_{t=0}^{T} : u_t \in \Gamma(x_t) \forall t\}$. Remember that for a recursive problem we assume that the state space is time invariant. Furthermore the *constraint correspondence* (2) is assumed to be time invariant too. The *objective function* in (1) is defined on the graph of Γ[5]

$$F : gr(\Gamma) \to \mathbb{R}$$

The β in the *objective function* is a time invariant constant and it is assume to be in the open set $(0, 1)$. I will put all variables and functions in an economic context below.
As long as we assume that T is finite we speak of the problem as a *finite horizon problem*.

2.2 Infinite horizon problems

Now I want to discuss briefly the changes if we assume that T is infinite. This case is then respectively called an *infinite horizon problem*.
The problem now reads as follows

$$\max_{u_t} \sum_{t=0}^{\infty} \quad \beta^t F(u_t, x_t) \tag{5}$$

$$s.t. \quad u_t \in \Gamma(x_t) \ \forall\, t \tag{6}$$

$$x_{t+1} \in \mathbb{L} \forall\, t$$

$$x_{t+1} = \psi(u_t, x_t) \tag{7}$$

$$x_0 \ given$$

$$\tag{8}$$

[5]This may yield some problems for the sufficiency of first order conditions also with strict concavity of the objective function, but we will nevertheless stick to this assumption, because it is mainly a differentiability issue, that we are not concerned with anyway in the *Dynamic Programming* approach I discuss here.

There are now some additional issues concerning the existence of a solution, because somehow it seems to be reasonable to impose that a solution should have a value of the objective function that is not infinity, because there are some obvious problems, if there are several solutions that yield infinity as value of the objective function. Furthermore it should be emphasized that going from a finite horizon to an infinite horizon problem is much more involved than it might seem at a first glance. The theory one is used to for finite dimensional optimization has to be extended to infinite dimensional spaces. But since these notes are not about the theory of optimization in infinite dimensional spaces I do not want to go into details about this issue here.

2.3 Optimal solutions

The optimal solution to the problems described in (4) and (8) is a sequence $\{u_t\}_{t=0}^{T}$, where T might be infinity. There we already see that the object that solves the infinite dimensional problem is element in an infinite dimensional space, whereas the solution to the finite dimensional problem is an element of $\mathbb{R}^{T\times m}$. If we exploit the recursivity of the problem, we can formulate the optimal control as a function of the states, i.e. we want to find an optimal *policy function* $g : X \to U$, such that it generates the optimal solution $\{u_t\}_{t=0}^{T}$ for every $x_0 \in X$.
Several things have to be discussed concerning this reformulation. First of all this formulation defines what is called an optimal *feedback* or *closed loop* control function in engineering, but as long as we consider non stochastic environments, we could also consider what is called an *open loop* optimal control function in engineering, i.e. we do not define the optimal control function as a mapping from states to controls, but from time to controls $\tilde{g}(t) : \mathbb{Z}_+ \to U$. There are two reasons why I already want to use the *feedback control* formulation: First it is the kind of representation common in economics and second later on when there are stochastic environments this is the kind solution we are going for anyway. Hence let me use it already here. In deterministic environments there is a one-to-one mapping form time to states along the optimal controlled path, therefore it is possible to choose the optimal control just time depended, but this one-to-one relationship breaks down as soon as we introduce uncertainty as I will do below. Put it differently, in environments with no uncertainty the initial condition together with the optimal control function allows to forecast any future states of the system. This is no longer true if there enter stochastic shocks to the system and affect the path of the state variables.
But finally it should be pointed out, that an *open loop control* would still be possible if there is uncertainty, but since *open loop schemes* are a strict subset of *closed loop schemes* one can achieve better solutions, if one allows for the possibility of state dependent control schemes. Hence we want to

search in the set of closed loop schemes, whenever this is possible.

2.4 Sequential Decision Problems in economic applications

After I have presented the problem formulation in great generality, I am now going to give some meaning to the objects encountered in the two problem formulations (4) and (8).
The constant β is called the *(pure) discount factor*. Sometimes the adjective *pure* is added to distinguish it from the *stochastic discount factor* or *pricing kernel* that is common in finance, and the *intertemporal marginal rate of substitution*.
The objective function is what is called the *utility function* economics. The function $F : U \times X \to \mathbb{R}$ is called *period utility function*, *instantaneous utility function*, *Bernoulli function* or *felicity function*. I introduced the function $H : U \times X \to \mathbb{R}$ in the finite horizon problem to emphasize the special structure of extensive form games that have special payoffs in the last period, but it might for example also capture things like utility from bequests.
To remain in line with the standard economic notation I revise my notation an I will denote the control variable by $c_t \in C$ and the *felicity function* by $u(c_t, x_t)$. Therefore the *utility function* becomes

$$U(\{c_t, x_t\}_{t=0}^{T}) = \sum_{t=0}^{T} \beta^T u(c_t, x_t) \tag{9}$$

Let me for the purpose of notational convenience assume that $H(\cdot) \equiv 0$ and let me treat the special case for finite games with no intermediate payoffs differently if I consider these problems.
It should be furthermore noted that I allow explicitly for state dependency in the felicity function. An assumption implicitly made is that I will only consider *time separable utility functions*. It would be possible to consider the broader class of *recursive utility functions*, but for this introduction I will restrict the utility functions to be in the class of *time separable utility functions*. There is a good discussion about different concepts of utility specifications in Rust's paper. But the bottom line to remember is that *Dynamic Programming* only works with the class of *recursive utility functions*, where the time seperable utility specification is a special case of.
An other standard assumption I will make throughout is that the felicity function is *concave*.
The *constraint correspondence* in (2) and (6) captures different constraints in economic problems. It describes the *feasible choice set* for the agent, like the *budget set for an agent or the set of prices for a firm*, but it also captures constraints like a *debt constraint* or non-negativity of consumption. For games it describes the *action set* for the agent at the current node. We

already encountered some simple problems that fit in this framework like the neocalssical growth model, the consumption saving problem. But the formulation allows also for problems, where the action sets are discrete. Generally this formulation describes any kind of intertemporal decision problems.
After this little motivation of the problem we can finally start to solve the problem at hand using *Dynamic Programming* , but not without introducing *uncertainty* about future state first.

3 Uncertainty

The general formulation of the *sequential decision problems* allows for a straightforward incorporation of uncertainty in the decision problem. I start with formulating a general approach of introducing uncertainty to the model. For this general introduction I do not impose any structure on the *stochastic process* but for the discussion to follow I do restrict myself to stochastic processes that are Markovian. Although it is a restriction on the underlying process it is commonly used in economics mainly for tractability reasons, especially in an infinite horizon setting. But let us start with a general formulation for uncertainty entering the model.
The uncertainty in the model is a sequence of random variables[6] $\{S_t\}_{t=0}^{T}$ that evolve according to a history dependent density functions $\phi(S_t|s_{t-1}, s_{t-2}, \ldots, s_1)$. For notational convenience I will from now on denote a history of shocks up to point t by $s^{t-1} := \{s_{t-1}, s_{t-2}, \ldots, s_1\}$ therefore I will denote the conditional density function $\phi(S_t|s_{t-1}, s_{t-2}, \ldots, s_1)$ by $\phi(S_t|s^{t-1})$. Since I did not specify what the elements of the state vector x_t are, I can incorporate uncertainty by just assuming that the shock history s^t is part of the state vector x_t. But this means further that decisions in period t are taken after uncertainty about the state in t has been resolved. Therefore the agent can condition his action on the current state. This is the fact why we want to look for optimal *closedloop* control schemes.
Furthermore one has to take a stand on what the objective function now is. In almost all economic applications it is assumed that agents are *expected utility maximizers*. As discussed in Rust (2006) *Dynamic Programming* depends crucially on the linear structure of the conditional expected value operator. Like with the different specifications for the utility functions the important thing to remember is that *Dynamic Programming* only works if the expected value is maximized in the objective function.
In this case the objective function (9) becomes

$$E\left[U(\{c_t, x_t\}_{t=0}^{T})\middle|\, x_0\right] = E\left[\sum_{t=0}^{T} \beta^T u(c_t, x_t)|x_0\right] \tag{10}$$

[6]The notational convention that I will follow is that a random variable is denoted by capital letters whereas a realization of the random variable is denoted by lower letters.

where $E[\cdot|x_0]$ denotes the expected value operator conditional on x_0. To avoid any measurability issues let me assume that the support of the random variable S_t is finite and let me denote the probability of history s^t by $\Pi(s^t)$ and by $\Pi(s_{t+1}|s^t)$ the conditional probability or *transition probability* given history s^t. If I assume the process to be Markovian, the conditional probability simplifies to $\Pi(s_{t+1}|s_t)$.
The idea of the a *game against nature* comes from the fact that one can interpret the stochastic process as a history of actions taken by nature, where nature plays a *mixed strategy* over the action set. Therefore an agent's optimal strategy can be interpreted as a best response to the strategy play by nature.

4 Theory of Dynamic Programming

The key idea behind *Dynamic Programming* is the *Principle of Optimality* formulated by Bellman (1957)

> *An optimal policy has the property that whatever the initial state is, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

In other words an optimal path has the property that whatever the initial conditions and the control variables over some preceding period are, the control for the remaining period must be optimal for the remaining problem, i.e. for every state the *continuation value is maximized*.
At this stage I want to point out two distinct characteristics of the *Principle of Optimality* . First of all we see again that the problem is solved not only for a unique initial state but for a whole family of initial states, because the solution must be optimal for any state of the problem. Actually what we will see later is that we neglect the initial condition altogether and solve the problem, such that no matter what the initial state is the solution will be optimal for every $x \in X$. This leads to the second point to mention here, because it should be emphasized that *Dynamic Programming* solves for optimal solutions for all states and not only for the optimal solutions in the support of the controlled process.
It is of interest for economists to recognize that this is exactly the difference between a *Nash equilibrium* and a *subgame perfect equilibrium* of a sequential game. The *Nash equilibrium* only yields optimal behavior on the equilibrium path, whereas in a *subgame perfect equilibrium* behavior is also optimal off the equilibrium path. But it should be clear that the value of the objective function is unaffect by the fact that the optimal solution is not optimal also off the optimal path, because all other paths have measure zero. Hence the optimal solution is characterized as follows

An optimal decision has to be optimal only on the optimal path, but not at node that have probability zero, i.e. that are off the optimal path. Therefore there is a multiplicity of optimal solutions if the solution is changed on the complement of the support of the controlled process.

To conclude, one can say that *Dynamic Programming* solves actually a more general problem, because in optimal control we require a solution only to be optimal along the controlled process.

4.1 Backward induction

After this introduction I want to outline an algorithm to solve the problem at hand, but first let me introduce an additional concept.
Let me define the *value function* $v : X \to \mathbb{R}$ as

$$v(x_0) = \max_{\{c_t\}_{t=0}^T} E\left[U(\{c_t, x_t\}_{t=0}^T)|x_0\right] \tag{11}$$

and denote the optimal policy function by

$$\{c^*(x_t)\}_{t=0}^T = \arg\max_{\{c_t\}_{t=0}^T} E\left[U(\{c_t, x_t\}_{t=0}^T)|x_0\right] \tag{12}$$

The idea of *backward induction* is to solve the problem from the end and working backwards towards the initial period. Let me consider a simple problem of a *game against nature*, with a finite action set and finite support of the stochastic process. This problem can easily be represented by an *event tree* (See figure 1). It is a very simple example what is mainly due to the fact that I did not want to draw an even bigger event tree. But nevertheless let us apply *backward induction* to solve this little problem. To give it a name, think of action I as 'Invest' and D as 'Don't invest' and the shock are the income realization. The numbers on the lines denote the conditional probabilities of the shock occuring given the current history. The numbers at the end of the tree and the two numbers above and below the first two nodes are the values I want to assign for the evaluated *felicity function* at these nodes[7].
If we now work backward we start at the last decision state and begin by determining the optimal behavior if the agent were at the top point. As one can easily see, the optimal behavior there is D. If we go to the next node, again the optimal behavior is D. If we do this for all nodes in the last period we know the continuation value at these nodes, i.e. 4.0 at the top node, 2.0 at the next node below and so on... When we now move backwards through

[7] You may have already noticed that we have to assume a distinct initial value for the problem, if we want to evaluate the felicity function and do not consider cases, where the problem is independent of the initial value. Hence we solve for a distinct problem out of the family of problems.

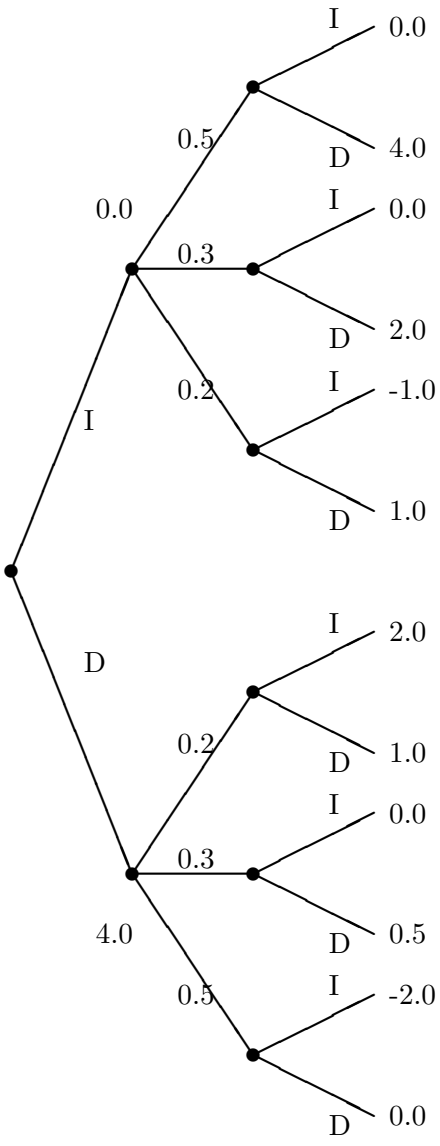

Figure 1: event tree

the tree nature moves and therefore we have to determine the conditional expected continuation value for an agent at the node before nature moves[8]. The only choice left is the choice of the agent in the initial period. To determine the optimal behavior there we consider current payoff indicated at the first node plus the discounted continuation value from future choices. Remember the *Principle of Optimality* that tells us that given any history of shocks and choices future choices must constitute an optimal policy, i.e. maximize the expected continuation value. Hence if we evaluate the value function[9] for x_0 we see immediately that the optimal choice in the first period is I.
Hence we have by simple comparison of expected continuation values found the value function $v(x_0)$ for the initial state x_0 and the optimal policy for every state.
Now it can be easily seen what I tried to emphasize above, namely that the optimal policy is not only optimal in the support of the optimally controlled process, but also at the nodes that are never reached with positive probability like all the nodes in the lower branch of the event tree in our simple example. This is due to the fact that when one starts solving the problem from the last period, one does not know ex ante which path will turn out to be optimal because I is the optimal choice in the first period and therefore the problem has to be solved for optimal solutions at every possible state of the process, i.e. also for optimal solutions in states that are never reached once the optimal policy is conducted. We will see later for the infinite horizon case, that this causes some problems for the numerical implementation if one does not impose some additional restrictions on the problem. I will discuss the numerical implementation of the backward induction in section 6.3.

4.2 Fixed point problem

As I already mentioned above, it is a substantial step going from finite to infinite time horizon[10].
Let us reconsider the problem in 8 and define the *value function* appropriately

$$v(x_0) = \max_{\{c_t\}_{t=0}^{\infty}} E\left[U(\{c_t, x_t\}_{t=0}^{\infty})|x_0\right] \tag{13}$$

[8]Remember that we have assumed expected utility maximization of agents

[9]This is a slight abuse of terminology, but I will call the continuation value induced by a certain policy value function as well although the policy function might not maximize the expected continuation value.

[10]In the appendix I discuss briefly the case of a stochastic end point.

We can rewrite the value function as follows

$$\begin{aligned} E\left[v(x_0)|x_0\right] &= \max_{\{c_t\}_{t=0}^{\infty}} U(\{c_t, x_t\}_{t=0}^{\infty}) \\ &= \max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t E\left[u(c_t, x_t)|x_0\right] \\ &= \max_{\{c_t\}_{t=0}^{\infty}} u(c_0, x_0) + \sum_{t=1}^{\infty} \beta^t E\left[u(c_t, x_t)|x_0\right] \\ &= \max_{c_0} u(c_0, x_0) + \beta \max_{\{c_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} E\left[u(c_t, x_t)|x_0\right] \\ &= \max_{c_0} u(c_0, x_0) + \beta E\left[v(x_1)|x_0\right] \end{aligned} \tag{14}$$

It has to be proven that this problem yields the same solution as the one in (4), but since it is not the subject of this talk I will not do it here[11]. Just believe me for the moment that this can be done.[12]. We know that this condition is true for every initial value and therefore it is due to the *Principle of Optimality* true for every pair of consecutive periods and we can replace x_0 by x and x_1 by x' and do the same for the control variables, then the definition of the value function becomes

$$v(x) = \max_{c} u(c, x) + \beta E\left[v(x')|x\right] \tag{15}$$

This equation is also called the *Bellman equation* in economics and it is no longer an algebraic equation, but it is a functional equation, because the unknown is the value function $v(x)$. To illustrate this[13] one can replace the max operator and define a new operator $\Psi : V \to V$

$$v(x) = \Psi(v(x'))$$

This operator $\Psi(\cdot)$ that maps from the space of bounded functions in the space of bounded functions can be shown to be a *contraction mapping* with modulus β. This can be done using *Blackwell's sufficient conditions*. From the *contraction mapping theorem*, which is also called *Banach's fixed point theorem*, we know that a solution exists and that is is unique. Therefore the approach to find a solution to the *Bellman equation* becomes a fixed point problem and the algorithm to solve this problem can exploit the contraction property of the *Bellman equation*. I want to mention also that all optimal policy functions that solve this problem have a recursive structure.

[11] The interested reader can find the proof in Stokey/Lucas (1989)

[12] There are some additional issues if the value function is unbounded, but also these can be resolved in most cases if we choose an appropriate norm. See also Rust (2006) for further discussion and references.

[13] Let me assume for the further discussion that $v(x)$ is a bounded function just to avoid additional issues.

5 Multiplayer games and Competetive Equilibrium

So far I only considered single agent optimization problems, where an agent tries to find a feasible policy that maximizes his expected utility.
Things become a bit more involved, if there are several agents or as in competetive equilibrium models a continuum of agents, because then the agents' optimal decisions have not only be optimal for the different states of nature but also for the actions taken by the other agents. Therefore we have to add an outer loop to the problem to solve an additional fixed point problem.
The agent's maximization problem becomes the inner loop of the composed problem, but the state space has to be enlarged, because it also contains the strategies of the other players and the distribution of their characteristics like capital holdings or age. Sometimes it might be sufficient to include some sufficient statistics about the distribution of the other agents in the state space instead of the whole distribution.
The algorithm to solve the composed problem goes back and forth between the two problems until it converges to a set of policies/strategies such that the policies/strategies of agents are optimal given the other players policies/strategies, i.e. the strategies are mutual best responses, and the assumed behavior of the other agents by agents solving their maximization problem is the behavior that the other agents will choose in the end. This is also called *rational expectations* in economics.
Furthermore we require that in a *competetive equilibrium* these optimal strategies yield *market clearing* at given prices.
There is an additional issue arising in the multiple agent settings, because there might be a multiplicity of equilibria. Although there are multiple optimal solutions to the single agent optimization problem as discussed above, the problem in the multiple agent setting is that the agents might have different payoffs across different equilibria.
There are especially in game theory a variety of equilibrium refinements to rule out some of the equilirbia, like the concept of *Markov-perfect equilibria*, but I do not want to discuss them here.

6 Numerical Methods

After I have presented the theory underlying *Dynamic Programming* I am now going to present some numerical implementations to solve the general problem based on this theory. First I consider the finite horizon case that we solved using *backward induction* and then I discuss several closely related approaches to solve the infinite horizon problem.

6.1 Function approximation methods

It is important to distinguish two concepts for function approximation. One approach which I will call the *local approach* tries to approximate the function by a large number of function values over a fine grid and then approximate the function by interpolation between these function values.
The other approach I will call the *global approach*[14] tries to approximate the function by a set of basis functions whose shape is governed by a finite set of parameters.

6.2 Grid Choice

In many problems the *state space* X and the feasible set for the control $\Gamma(X)$ contain a continuum of elements.
Since the numerical routine can not handle infinite dimensional objects directly, some discretization is needed. Different methods have been proposed of how to do this discretezation. From the above discussion it should be noted that the state space might contain infinite dimensional objects like the distribution functions or best response functions of other plyaers. These objects have to be either discretized using the methods below or parameterized using the methods described later. I will call the set of discretized values the *grid*.
The problem of grid choice is discussed here only briefly and not in the extent it might require. The problem is closely related to the issue of numerical integration, because the same problems arise there. Since I do not have the time I will not cover this topic here and the interested reader is referred some book on numerical methods like Judd (1998).
The outline of the methods I give here is very basic and by no means exhaustive.

6.2.1 Equally spaced grids

The first approach is to choose an equally spaced grid in each dimension of the single state space and then form the Cartisian product of the state grids. This approach has the problem, that the set of grid points grows exponentially in the grid points of the single state directions and this is called the *Curse of Dimensionality* and is maybe the major drawback of *Dynamic Programming* .

6.2.2 Random grid

An alternaitve would be to choose not grids in each state direction but do random sampling in the state space, i.e. we draw from some random

[14] I choose this term to distinguish it from the *local approach*, although it might be a bit misleading, because we still approximate the function only on a subset of the domain.

distribution the elements of the state space. Here the algorithm designer can set the number of grid points. If this is done appropriately it can help to get around the *Curse of Dimensionality* .

6.2.3 Adaptive grids

A third possibility I want to present briefly is the possibility of an adaptive grid choice. The idea of this approach is to choose a coarse grid in the beginnig and solve the problem once on for this coarse grid. After the solution has been obtained there are successively grid points added at regions in the state space, where the function seems to have a lot of curvature.

6.2.4 Interpolation

The function approximation by a finite number function values over a set of grid points has the problem that the function is still unknown at most of the points in its support. Therefore one needs methods to derive the function value at points, which are not in the grid. This is done using interpolation methods.
Since these nodes are on *Dynamic Programming* and are supposed to be only an introduction I am not going to discuss interpolation methods here and refer to Judd(1998).

6.3 Backward induction algorithm

Backward induction is an easy to implement algorithm. The only issue that arises is in the case of continuous state and control variables. To implement continuous variables on a computer, one has to choose some kind of discretization. The other issue is the *Curse of Dimensionality* . I will discuss these problems in section 7.
Suppose we are able to appropriately take care of these two issues, then the *backward induction algorithm* consists mainly of a grid search over the feasible set of policies for each state.

Outline of the algorithm:

1. Determine the value function for $v(x_{T+1})$ for all $x_{T+1} \in X$.
2. Start for $t = T$ for very $x \in X$ evaluate for every $c_t \in \Gamma(x_t)$ the current utility from the choice, i.e. calculate

$$\tilde{v}(x_{t-1}, c_t) = u(c_t, x_t) + \beta E\left[v(\psi(c_t, x_t))\right]$$

3. Choose for every x_t

$$\begin{aligned} c_t^*(x_t) &= \arg\max_{c_t \in \Gamma(x_t)} u(c_t, x_t) + \beta E\left[v(\psi(c_t, x_t))\right] \\ \textit{and} \quad v(x_{t-1}) &= \max_{c_t \in \Gamma(x_t)} \tilde{v}(x_{t-1}, c_t) \end{aligned}$$

4. If $t > 0$ go to period $t-1$ and go back to step 2 for the period $t-1$ instead of $t = T$, otherwise stop.

Performing these steps is exactly the procedure I described in the theory section above[15].

6.4 Value function iteration

As the *backward induction algorithm* the *value function iteration* is an easy to implement but nevertheless powerful and reliable algorithm.

[15] If we solve for subgame perfect equilibria in extensive form games, there are usually no intermediate payoffs and only a finite payoff, but nevertheless it should be clear that the whole *backward induction* procedure has to be done to determine an optimal strategy.

Outline of the algorithm:

1. Choose a convergence criterion ε for $\|v_i(x) - v_{i-1}(x)\| < \varepsilon$
2. Start with an initial guess for the value function $v_0(x)$, e.g. $v_0(x) = 0$
3. For very $x \in X$ evaluate for every $c \in \Gamma(x)$ the current utility from the choice if the value function $v_i(x)$ were the correct value function, i.e. calculate

$$\tilde{v}(x_{t-1}, c_t) = u(c, x) + \beta E\left[v_i(\psi(c, x))\right]$$

4. Choose for every x

$$\begin{aligned} c^*(x) &= \arg\max_{c\in\Gamma(x)} u(c,x) + \beta E\left[v_i(\psi(c,x))\right] \\ \textit{and}\quad v_{i+1}(x) &= \max_{c\in\Gamma(x)} \tilde{v}(x_{t-1}, c_t) \end{aligned}$$

5. Check convergence. If the value function has not yet converged go back to step 3 and use $v_{i+1}(x)$ as guess for the value function, otherwise stop.

It relies on the *contraction mapping property* of the *Bellman equation* in (15). Since the *Bellman equation* has an unique fixed point this algorithm will converge to the true solution and uses the method of successive approximations. Starting from $v_0(x)$ it can be shown that

$$\lim_{t\to\infty} \Psi^t\left(v_0(x)\right) = v(x)$$

Moreover it can be shown that

$$\|v(x) - v_i(x)\| \leq \frac{1}{1-\beta}\|v_{i+1}(x) - v_i(x)\|$$

The inequality can also be used to pick the stopping criterion.
There is a nice intuition why this approach works, namely because we assume that agents discount future utility, the effect of future periods on the value function decreases with the distance of the future date from the current date of the decision. Remember that we have $\beta \in (0, 1)$ and therefore $\lim_{t\to\infty} \beta^t = 0$. There will be some date in the future where the effect on the value function is almost neglectable. If one compares the *value function iteration* to the *backward induction algorithm* one sees that essentially the same is done, i.e. starting from some period and going back in time, but this time the index is i and not t but otherwise the same procedure is conducted. The only difference is that this time the algorithm does not stop in $t = 0$

but iterates until the value function has converged. This can be thought of as increasing the agent's time horizon until it is almost infinity. Therefore we have approximated the infinite time horizon by a finite time horizon that is sufficiently large.

6.5 Howard's improvement algorithm

The Howard's improvement algorithm is also known as *policy function iteration*. It works similar to the *value function iteration*, but this time the guess is for the optimal policy.
First notice that a policy function can also be represented by a transition function. It is a mutation of the transition function of the stochastic process, because next period's states are a function of the current state and the control this period. If one changes the control this period, it just shifts the probability distribution next period. So if we guess a policy $c_0(x)$, we can represent it by a transition function $\Omega(c)$ and we get the following *Bellman equation*

$$v(x) = u(c(x), x) + \beta\Omega(c)v(x)$$

But this can be easily solved[16] for $v(x)$

$$v(x) = (I - \beta\Omega(c))^{-1} u(c(x), x)$$

and we get an implied value function $v(x)$. With this trick at hand I can now outline the algorithm.

[16]The inverse of $(I - \beta\Omega(c))$ exists always because the matrix is *diagonally dominant* and therefore the inverse exists.

Outline of the algorithm:

1. Choose a convergence criterion ε for $\|c_i(x) - c_{i-1}(x)\| < \varepsilon$

2. Start with an initial guess for the policy function $c_0(x)$, that is feasible. Derive the implied transition function $\Omega(c)$.

3. Perform the value function updating

$$v(x) = (I - \beta\Omega(c))^{-1} u(c(x), x)$$

4. For very $x \in X$ evaluate for every $c \in \Gamma(x)$ the current utility from the choice if the value function $v(x)$ were the correct value function, i.e. calculate

$$\tilde{v}(x_{t-1}, c_t) = u(c, x) + \beta E\left[v(\psi(c, x))\right]$$

5. Choose for every x

$$\begin{aligned} c_{i+1}(x) &= \arg\max_{c\in\Gamma(x)} u(c, x) + \beta E\left[v_i(\psi(c, x))\right] \\ \textit{and} \quad v(x) &= \max_{c\in\Gamma(x)} \tilde{v}(x_{t-1}, c_t) \end{aligned}$$

6. Check convergence. If the policy function has not yet converged go back to step 2 and use $c_{i+1}(x)$ as guess for the policy function, otherwise stop.

The advantage of the *policy function iteration algorithm* is that it in general converges faster. If it converges faster than the *value function iteration algorithm* depends on the modulus of the contraction mapping. If it is close to one the *policy function iteration* is much faster, but the time advantage decreases with decreasing modulus and the *value function iteration* might even become faster, because it does not require any matrix inversion.

6.6 Modified policy function iteration

An other method used in applied work, that has proven to yield a substantial increase in convergence speed without requiring matix inversion, is what I will call *modified policy function iteration*.
The algorithm works along the same lines as the *value function iteration*, but instead of just one updating step as before it does several updating steps once a new policy has been determined.
It does not like in the *Howard's improvement algorithm* compute the inverse and uses the updating formula to get a new policy, but it just applies the new policy several times. Therefore we add an additional step to the algorithm

6. Repeat N times the following updating procedure starting with $j = 1$

$$v_i^j(x) = u(c_i(x), x) + E\left[v_i^{j-1}(\psi(c_i(x), x))|x\right]$$

if $j = N$ stop and set $v_{i+1}(x) = v_i^N(x)$.

The number of the updating steps N can be choosen arbirarily. From my experience value for N between 15 and 30 depending on the problem, turned out to yield the best time-speed trade-off.

6.7 Function approximation method

There is a well establish theory about function approximation. These methods are also called *collocation methods* or *projection methods*.
Common to all methods is that they try to approximate the unknown function by some basis functions[17]. The shape of the approximating function is determined by a set of parameters. The parameters are chosen such that a certain approximation criterion is met. Approximation criteria are usually that functions go exactly through some points, this is only possible in a system where there are as least as much degrees of freedom as points to be matched.
The other possibility is to minimize some squared distance like in a regression procedure. This is usually done in *over determined* systems, i.e. when there are less degrees of freedom than points to be matched.
The approximation methods are usually distinguish by the set of basis functions they use, e.g. *Taylor*, *Chebychev*, *Hermite*.
These methods use far less points to approximate the function than do the *local* approximation methods. Therefore there is also a well established theory about how to choose the *collocation nodes*, i.e. the points at which the function is evaluated.
But since these nodes are about *Dynamic Programming* I will not go into the details of these methods and refer to Judd(1998) for further reference.

6.8 Value function iteration collocation

The main difference of the *value function iteration collocation* method is that this time the optimal value function is not approximated using fine grids of the states space, but using the *collocation methods* that I just very briefly discussed.

[17]Sometimes also some additional criteria like derivatives should be matched. The approach were also derivatives are matched is called *splines*.

Outline of the algorithm:

1. Choose the set of basis functions to approximate the value function. Determine the set of parameters θ.
2. Choose a convergence criterion ε for $\|\theta_i - \theta_{i-1}\| < \varepsilon$
3. Start with an initial guess for the parameters of the value function θ_0 that imply a certain shape of $v_{\theta_0}(x)$
4. At the collocation points $x \in X$ evaluate for every $c \in \Gamma(x)$ the current utility from the choice if the value function $v_{\theta_i}(x)$ were the correct value function, i.e. calculate

$$\tilde{v}(x_{t-1}, c_t) = u(c, x) + \beta E\left[v_{\theta_i}(\psi(c, x))\right]$$

5. Choose for all nods x

$$v(x) = \max_{c \in \Gamma(x)} \tilde{v}(x_{t-1}, c_t)$$

   and calculate the new values for θ_{i+1}
6. Check convergence. If the parameters have not yet converged go back to step 4 and use θ_{i+1} as parameters for the value function, otherwise stop.

7 Curse of dimensionality

A crucial limitation to *Dynamic Programming* is the exponential growth of the number of histories. If there are no restrictions on the form of the policy function, then the agent in principle would need an optimal policy for every history.
In the case when there are S possible states each period and suppose that there are C elements in the choice, then the number of histories is of order $(ND)^t$.
The other problem is if there are many state variables and every combination of the single states variables must be considered, then the number of overall states also grow exponentially. This problem also arises, if we impose recursivity on the policy function. This problem was already discovered by Bellman and Dreyfus and they referred to it as the *curse of dimensionality*.

8 Conclusions and further reading

These notes hopefully showed that *Dynamic Programming* is a reliable and powerful method to solve *sequential decision problems* . The advantage is

that it has a well established theory.
Especially the easy incorporation of uncertainty make it a widely used approach in economics. *Dynamic Programming* can be applied in a large variety of economic problems, e.g. *general equilibrium theory*, *industrial organization*, *game theory* or *mechanism design*.
Although I emphasized the numerical application in these notes, *Dynamic Programming* is also of theoretical interest for economic research.
Numerically *Dynamic Programming* is an easy and straightforward to implement method, that yields accurate results and is insensible to numerical inaccuracy.
Finally I should mention that these notes are only a short introduction to *Dynamic Programming* and are by far not an exhaustive introduction to all issues of *Dynamic Programming* . People how want to apply the methods should also be familiar with the related fields of numerical work like function approximation or integration.

References

[1] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] Kenneth L. Judd. *Numerical Methods in Economics*. Cambridge University Press, 1998.

[3] John Rust. Dynamic programming. Entry for consideration by the New Palgrave Dictionary of Economics.

[4] Nancy L. Stokey and Robert E. Lucas. *Recursive Methods in Economic Dynamics*. Harvard University Press, 1989.

A Extensive form games

An **extensive form game** $\Psi(\mathcal{I}, \mathcal{T}, \{S_i\}_{i\in\mathcal{I}}, \{\Xi_i\}_{i\in\mathcal{I}}, \{u_i(\cdot)\}_{i\in\mathcal{I}}, \phi(S))$ is a collection of players indexed by $i \in \mathcal{I}$, a time structure for players' actions $\mathcal{T}$, strategy sets S_i, information sets Ξ_i and a payoff function for each player that maps histories of actions into payoffs $u_i : \times_{i\in\mathcal{I}} \times_{t\in\mathcal{T}} S_i \to \mathbb{R}$.

These games are commonly described by a *game tree*.

B Markov Chain

Let us assume that X_t is a random variable with finite support. Let us denote the probability of $X_t = x_t$ given the history $x^{t-1} = \{x_{t-1}, x_{t-2}, \ldots, x_0\}$ as $\Pi(x_t|x^{t-1})$. The stochastic process is a *Markov chain* if $\Pi(x_t|x^{t-1}) = \Pi(x_t|x_{t-1})$, i.e. the probability of x_t only depends on the realization of the last period x_{t-1}.

C Undertermined end time

These problems can numerically be treated like finite horizon problems. The only restriction one has to impose is that there is a T such that survival probability is zero. But this assumption does not seem to be too unreasonable. The only difference is that we now have an additional survival probability entering the problem, since it is uncertain, if the agent will survive and be still alive in the next period. The way to adjust for that is that we multiply the transition probabilities by a survival probability. Therefore let me define the *discounted transition probability* by

$$\tilde{\Pi}(s_{t+1}|s_t) := \rho_{t+1}\Pi(s_{t+1}|s_t)$$

where $\Pi(s_{t+1}|s_t)$ denotes the conditional probability of going from state s_t to state s_{t+1} and ρ_{t+1} denotes the survival probability from period t to $t+1$. We can now solve the problem as before, but use the transition probabilities $\tilde{\Pi}(\cdot)$ instead of $\Pi(\cdot)$. But keep in mind that the *discounted transition probability* will not sum to 1 anymore but to ρ.