# Numerical Methods for Time Inconsistency, Private Information and Limited Commitment

Antonio Mele

July 2017

# CONTENTS

# LISTINGS

# TODO LIST

# 1 INTRODUCTION

Most macroeconomic models have a **recursive** structure, and they can be represented as a **functional equation** in some functional space. In the first part of the course we will look at the basic dynamic programming theory and its practical implementation. This is the more general approach for solving dynamic models in macroeconomics. In particular, it **always** works: given enough time, the algorithms that we will see (value function iteration and policy function iteration) converge to the solution of the model at hand. The crucial words are “given enough time": these algorithms can be very slow also for relatively simple models.

In the second part of the course, we will then look at techniques that are designed to solve functional equations in general, called **projection methods**. This approach is much faster than standard algorithms based on basic dynamic programming results. However, these algorithms have not clear convergence properties, and many times the researcher has to “help" the code to converge with tricks. In some sense, working with projection methods is more an art than an exact science. These methods are more sophisticated and therefore we need to discuss some **basic numerical concepts** before talking about projection methods.

Finally, in the third and fourth part of the course, we will study techniques for models with non-standard recursive properties. These are models in which expectations of future choices affect current choices. Hence, they are not recursive in the standard sense (i.e., in these models the past states are not a sufficient statistic for determining current choices). The literature has shown two possible ways for “recursifying” these problems. The first uses agent’s future lifetime utilities (aka **promised values** or continuation values) as state variables, which we will see in the third part. The second technique uses (combinations of) past Lagrange multipliers as state variables, and it is therefore called **the Lagrangean approach**; we will see the details in the fourth part. Both these techniques then rely on extending dynamic programming theory to show that a recursive solution exists in the newly created state space. We will see a few examples and solve some of them with projection techniques.

At the end of this course, you should be able to solve model with time inconsistency, limited commitment and private information with a combination of dynamic programming, projection methods and set approximations techniques.

## 1.1 MATERIAL

All the material for these lectures is on a Github repository: **https://github.com/CSS17-week1/nmtipilm**

IF YOU DON'T HAVE A GITHUB ACCOUNT:

1. Follow these simple steps to create one:
   a) Create a Github account at this link
   b) Choose the free plan
   c) Go to the organization page at this link
2. or, go to the organization page at this link and download the zip file

IF YOU ALREADY HAVE A GITHUB ACCOUNT OR YOU JUST CREATED ONE: Fork the repository by clicking on "Fork".

Please send me an email with your Github nickname and I will add you to the organization.

The material contains all the code used for the course, including exercises and solutions. To make your life easy, the external libraries used in the code are included. Finally, slides are in the corresponding folder. Files will be updated if needed. If you find some error or you think there is a better way to solve a problem, please open an issue on Github, describing what the problem is. If you want to send a pull request that solves it, even better!

# 2 RECURSIVE METHODS

## 2.1 INTRODUCTION

We will approach the dynamic programming techniques as the basic way of solving a macroeconomic model. In fact, every other technique (including log-linearization, perturbation, etc.) is taking the recursivity of the problem as given, and using it to solve it in a simpler way.

However, notice that some frameworks can be particularly “hostile" to the researcher, and simplified algorithms that use some specific properties of the model cannot be implemented. In these cases, it is **always** possible to apply basic implementations of the value function iteration (VFI) algorithm to a dynamic optimization problem.

To give you a few examples in which dynamic programming can be the only feasible strategy, here are two now widely known cases:

1. Strong presence of non-linearities can induce a very different behaviour of the system when far from the steady state. This is true for example in models with ZLB, as shown in the work of Braun et al. (2012).

2. Non-convexities imply that first order conditions are not sufficient for a solution. This is true for example in heterogeneous agents models with discrete choices in labour (work/not work decision) as in Chang and Kim (2007).

For these problems, global techniques are more suited, and VFI is a very powerful algorithm, since it can potentially solve any problem that can be stated as a Bellman equation. However, it becomes computationally burdensome quite fast as the number of state variables increases. There are many tricks to deal with this problem, and VFI can be combined with other numerical procedures that would help speed up the algorithm (interpolation, projections methods, etc.).

VFI is appropriate for models in which time inconsistency is not present. However, many authors have worked on extending the same idea of dynamic programming to time inconsistent problems (we will talk about the two main

approaches in the chapters 5 and 6). With some modifications, the same algorithms can be used also for optimal policy problems and New Keynesian models, bearing in mind that the number of endogenous states is a certain source of troubles.

For the purpose of this course we will present the technique with a very simple example of an RBC model.

## 2.2 THE BASICS OF DYNAMIC PROGRAMMING

Dynamic programming became the standard method in macroeconomics during the 70s. Currently, all macroeconomists use models that can be solved with the tools of dynamic programming. In order to explain the main ideas of it, we can analyze the deterministic neoclassical growth model. Everything we say will carry on with uncertainty under the assumption of Markovian exogenous shocks and a few more adjustments.

The neoclassical growth model can be solved by looking at the competitive equilibrium of the economy. We can solve both the firms' and the agents' problems as dynamic programs and then find the equilibrium of the economy. In many cases in which the economy presents distortions, this is the way to go. However, since there are no market failures, the competitive equilbrium is Pareto efficient, and therefore allocations can be found by solving a planner problem.

We will make a few assumptions to make sure our problem is well-behaved. The production function is

$$Y_t = F(A_t, h_t, K_{t-1}) \tag{2.1}$$

while the utility function of the representative household is given by

$$U_t = U(C_t, L_t) \tag{2.2}$$

We also assume that

**Assumption 1 (Utility function).** *$U(C_t, L_t) = U(C_t)$, $U : R_+ \to R$ is bounded, continuously differentiable, strictly increasing, strictly concave and* $\lim_{C \to 0} U'(C) = \infty$.

This assumption guarantees that there is no disutility of work, and hence agents work all the time, i.e. labor supply is equal to the maximum amount of available hours, which we normalize to 1. The rest of the assumption guarantees interiority and uniqueness of the solution.

**Assumption 2 (Production function).** *The production function $F : R_+^3 \to R_+$ is continuously differentiable, strictly increasing, homogeneous of degree 1 and strictly quasi-concave, with*

$$F_K > 0; F_{KK} < 0; F_h > 0; F(A, 0, L) = 0 \qquad \forall K, L > 0$$

$$\lim_{K \to 0} F_K(A, 1, K) = \infty, \lim_{K \to \infty} F_K(A, 1, K) = 0 \qquad \text{(Inada conditions)}$$

These assumptions are useful in making the problem "well-behaved", i.e. the constraint set does not have any pathology. We also assume for the moment that the productivity shock $A_t$ is not a shock, but remains constant: $A_t = A$.

Finally, the resource constraint of the economy is

$$C_t + K_t - (1 - \delta) K_{t-1} \leq F(A, 1, K_{t-1}) \tag{2.3}$$

where we have assumed that there is no government, i.e. $G_t = 0$.

We can therefore consider the problem of a benevolent social planner with the objective of maximizing the expected discounted utility of the representative household by choosing sequences $\{C_t, K_t\}_{t=0}^{\infty}$ subject to the resource constraint (2.3), taking $K_{-1}$ as given. It can never be optimal to waste output, therefore equation (2.3) holds with equality and we can use it to eliminate consumption from the problem. We can define

$$f(K) \equiv F(A, 1, K) + (1 - \delta) K$$

and rewrite the problem as[1]:

$$\max_{\{K_t\}_{t=0}^{\infty}} \quad \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - K_t\right) \tag{2.4}$$

$$s.t. \quad 0 \leq K_t \leq f(K_{t-1}), \quad t = 0, 1, ... \tag{2.5}$$

$$K_{-1} \text{ given}$$

Let's assume for a moment that we want to solve this problem but we have a finite horizon $T$. In this case, we have to find a sequence $\{K_t\}_{t=0}^{T}$ that solves our problem. Notice that this is a standard concave program, since the set of sequences $\{K_t\}_{t=0}^{T}$ that satisfy (2.5) is a closed, bounded and convex subset of $\mathbf{R}^{T+1}$, and the objective in (2.4) is continuous and strictly concave, therefore there is a unique solution and we can characterize it with Kuhn-Tucker conditions. Now, $f(0) = 0$ and $U'(0) = \infty$, therefore constraint (2.5) only binds

1 The constraint comes from the fact that we want consumption to be non-negative.

for $K_T$, and of course $K_T = 0$. Therefore we can get the following first-order conditions:

$$\beta f'(K_t) U'\left(f(K_t) - K_{t+1}\right) = U'\left(f(K_{t-1}) - K_t\right) \tag{2.6}$$

which is a standard Euler equation. Taking into account the boundary conditions

$$K_T = 0, \quad K_{-1} > 0 \text{ given}$$

we can solve for the optimal sequence.

Now, let's go back to infinite horizon. How do we solve for the optimal (infinite!) sequence? A natural way is to solve for the finite horizon case as we just did, using first order conditions, and then take the limit of the solution as $T$ goes to infinity. However, we can follow a different approach, due to Richard Bellman, which is more general and works also in the case in which the problem's first order conditions are not sufficient for characterizing the solution.

We can guess that a solution must take the form $K_t = g(K_{t-1})$ with $g : R_+ \to R_+$. Why this functional form? In each period, the planning problem is always the same, and only the capital that we have at the beginning of each period changes, so the choice of future capital stock and consumption must be a function of current capital stock. So now the question becomes: how do we get the function $g$?

Imagine for a second that we already solved the program (2.4) for all possible values of initial capital $K_{-1}$. Therefore we can define a function $V : R_+ \to R$ such that for any possible $K_{-1} > 0$ given,

$$\begin{aligned} V(K_{-1}) \equiv \max_{\{K_t\}_{t=0}^{\infty}} & \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - K_t\right) \\ & s.t. \quad 0 \leq K_t \leq f(K_{t-1}), \quad t = 0, 1, ... \end{aligned}$$

The function $V$ is called the *value function* of the problem 2.4. We can therefore say that $V(K_0)$ is the value of the utility from period 1 on, obtained with an initial capital $K_0$. Then, let's rewrite the planner problem at time 0 as:

$$\begin{aligned} & \max_{C_0, K_0} \left[U(C_0) + \beta V(K_0)\right] \\ s.t. \quad & C_0 + K_0 \leq f(K_{-1}) \\ & C_0, K_0 \geq 0, \quad K_{-1} > 0 \quad \text{given} \end{aligned} \tag{2.7}$$

If we knew $V$, we could get $g$ from (2.7), by defining $g : R_+ \to R_+$ in the following way : $K_0 = g(K_{-1})$ and $C_0 = f(K_{-1}) - g(K_{-1})$ are the maximizers

in (2.7). Therefore, $K_t = g(K_{t-1})$ completely describes the dynamics. However, we don't know $V$! What can we do?

Remember that $V$ is the maximized objective function for program (2.4). Therefore, if program (2.7) also solves the same problem, it must be that

$$V(K_{-1}) = \max_{0 \leq K_0 \leq f(K_{-1})} \left\{ U\left(f(K_{-1}) - K_0\right) + \beta V(K_0) \right\}$$

But under this formulation, we don't need time subscripts, this is a static problem! We can therefore write it as

$$V(K) = \max_{0 \leq y \leq f(K)} \left\{ U\left(f(K) - y\right) + \beta V(y) \right\} \tag{2.8}$$

This is an equation in which the unknown is a function (the value function $V$), and this is the reason why it is called a *functional equation*[2]. The solution therefore is also a function. It may seem that transforming the initial sequential problem (in which the solution is an infinite sequence) into a functional equation is making the problem more complicated. However, this is not the case: it turns out that it is much easier to solve the functional equation. Once we solve the equation, we get also a maximizer function $y = g(K)$[3]. We can use this function to calculate the optimal sequence of capital accumulation starting from $K_{-1}$.

The Bellman equation has an important property: in each period, all the information about the past is contained in the current level of capital. This is why we can eliminate time subscripts. We call this property **stationarity**. We usually call **state variables** those variables that summarize information about the past.

### 2.2.1 Equivalence of Sequential and Functional Equation Solutions

We have established that we can analyze problems like (2.4) by solving a functional equation like (2.8). However, we need to be sure that solutions of the sequential problems are also solutions of the functional equation. We need to establish if a solution exists and if it is unique. Moreover, we need a method to solve for the optimal $V$ (and $g$ of course).

2 This equation is often referred to as *Bellman equation*, in honor of Richard Bellman that first formulated it in 1957.

3 More generally, we get a correspondence, since there may be more than one solution $y$ for each $K$.

The equivalence can be established by using the Contraction Mapping Theorem (CMP). This theorem shows that, under general assumption that our model satisfies, there is a unique continuous, increasing and concave value function $V$ that solves the Bellman equation. Moreover, the policy function $g$ is also continuous. This result can be easily generalized to cases with more than one endogenous state variable. The reader is advised to read Stokey et al. (1989) for a detailed treatment.

Let us start by formulating our problem in a more general way. We are interested in optimization problems with the following structure:

$$\max_{\{u_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t r\left(x_t, u_t\right) \tag{2.9}$$

$$\begin{aligned} s.t. \quad & x_{t+1} = h\left(x_t, u_t\right), \quad t = 0, 1, ... \\ & x_0 \in X \text{ given} \end{aligned} \tag{2.10}$$

where $x_t \in X$ is a vector of state variables, $u_t \in U$ is a vector of control variables, function $r\left(x_t, u_t\right)$ in (2.9) is a return function, and equation (2.10) is called the **transition equation** or the **law of motion** for state variables. We can always rewrite the problem in the following way (for an example see the previous section):

$$\max_{\{x_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t f\left(x_t, x_{t+1}\right) \tag{2.11}$$

$$\begin{aligned} s.t. \quad & x_{t+1} \in \Gamma\left(x_t\right), \quad t = 0, 1, ... \\ & x_0 \in X \text{ given} \end{aligned}$$

$\Gamma : X \to X$ is a correspondence (i.e., a mapping from each point of set $X$ to subsets of $X$) that describes feasibility constraints and the law of motion for state variables. To this sequential problem we can associate a Bellman equation:

$$V\left(x\right) = \max_{y \in \Gamma(x)} f\left(x, y\right) + \beta V\left(y\right) \qquad \forall x \in X \tag{2.12}$$

We need few assumptions to be sure that the solution of sequential and functional problem are the same.

**Assumption 3.** *$\Gamma\left(x\right)$ is nonempty $\forall x \in X$*

**Assumption 4.** *For any feasible plan $\{x_t\}_{t=0}^{\infty}$, $\lim_{n\to\infty} \sum_{t=0}^{n} \beta^t f\left(x_t, x_{t+1}\right)$ exists*

Then define

$$V^*(x_0) \equiv \max_{\{x_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t f(x_t, x_{t+1})$$
$$s.t. \quad x_{t+1} \in \Gamma(x_t), \quad t = 0, 1, ...$$

We have the following:

**Theorem 1.** *Let Assumptions 3-4 be satisfied. Then $V^*$ satisfies the functional equation (2.12).*

A partial converse is:

**Theorem 2.** *Let Assumptions 3-4 be satisfied. If $V$ is a solution to the functional equation (2.12) and satisfies*

$$\lim_{n\to\infty} \beta^n V(x_n) = 0$$

*for any feasible plan $\{x_t\}_{t=0}^{\infty}$, then $V = V^*$.*

We can also show that, under Assumptions 3-4, a feasible plan $\{x_t^*\}_{t=0}^{\infty}$ that attains the maximum in program (2.11) is such that

$$V^*(x_t^*) = f(x_t^*, x_{t+1}^*) + \beta V^*(x_{t+1}^*), \quad t = 0, 1, ... \tag{2.13}$$

A partial converse is true: if a feasible plan $\{x_t^*\}_{t=0}^{\infty}$ satisfies equation (2.13) and a limit condition[4], then this plan attains the maximum of the sequential problem. These theorems establish that both the sequential and the functional problems have the same solution.

### 2.2.2 The case of bounded returns

We only look at the case in which returns are bounded, but the interested reader can check Stokey et al. (1989) for other interesting cases. In this section we are going to assume

**Assumption 5.** *$X$ is a convex subset of $\mathbb{R}^l$, and the correspondence $\Gamma : X \to X$ is non-empty, compact-valued and continuous*

**Assumption 6.** *The function $f(\cdot, \cdot)$ is bounded and continuous, and $0 < \beta < 1$*

4 The condition states that $\limsup_{t\to\infty} \beta^t V^*(x_t^*) \leq 0$

In order to use these two assumptions, we need to define an operator, i.e. a mapping from a certain space of functions to another space of functions. Let $C(X)$ be the space of continuous bounded real-valued functions and let $T : C(X) \to C(X)$ be an operator defined as

$$(TW)(x) \equiv \max_{y \in \Gamma(x)} f(x,y) + \beta W(y) \tag{2.14}$$

Therefore we can rewrite our Bellman equation (2.12) as $V(x) = TV(y)$. We can also define the policy function correspondence as

$$G(x) \equiv \left\{ y \in \Gamma(x) : V(x) = f(x,y) + \beta V(y) \right\}$$

We can then state the following result:

**Theorem 3.** *Let Assumptions 3-6 be satisfied. Therefore, there exists a unique function $V$ that solves the Bellman equation (2.12).*

We will not get into the mathematical details of the proof[5], but we can give an intuitive sketch of it. Notice that, thanks to our definition of the operator $T$ in (2.14), we can write our Bellman equation as:

$$V = TV$$

and therefore we can solve our problem by finding the fixed point of the operator $T$. To find the fixed point, we apply the Contraction Mapping Theorem, or Banach-Cacciopoli Theorem. This theorem make use of the properties of contraction mappings, i.e. operators that "squeeze" functions. The theorem establishes the uniqueness of the fixed point of a contraction mapping. In order to establish if our Bellman operator is a contraction, we can use the two Blackwell's conditions. The first Blackwell condition is **monotonicity**: if we have two functions such that $V(x) \geq W(x)$, then it must be that $TV(x) \geq TW(x)$. It is easy to show this property for our operator, because it is a maximization operator. The second Blackwell condition is **discounting**: for any constant $k$, we must have that $T(W+k)(s) \leq T(W)(s) + \beta k$ for all $s \in S$ where $\beta \in [0,1)$. In words, this means that adding a constant to $W$ leads $T(W)$ to increase by less than this constant. Also this property is trivially satisfied since we have $\beta \in (0,1)$. Once we have established that Blackwell's conditions are satisfied, we know our operatoris a contraction mapping, and therefore we

5 If you are interested, you can read the proof of Theorem 4.6 in Stokey et al. (1989).

can apply the Contraction Mapping Theorem that says contraction mappings have a unique fixed point.

The Contraction Mapping Theorem also tells us how to find the solution. In particular, it is possible to show that , starting from an arbitrarily chosen value function, and applying many times the operator $T$ to get a new value function, we converge to the fixed point. This procedure is called **value function iteration** and it is the basic numerical method that economists use to solve a Bellman equation.

Theorem 3 doesn't tell us anything about the properties of value function and policy correspondence. In order to characterize our functions in a more accurate way, we need more assumptions on the primitives of the problem.

**Assumption 7.** *For each $y$, $f\left(\cdot, y\right)$ is strictly increasing in its first arguments*

**Assumption 8.** $\Gamma$ *is monotone:* $x \leq x' \Rightarrow \Gamma\left(x\right) \subseteq \Gamma\left(x'\right)$

With these assumptions, we can prove that the value function is strictly increasing.

**Theorem 4.** *Let Assumptions 5-8 be satisfied, and $V$ be the unique solution of (2.12). Then $V$ is strictly increasing.*

An additional property we will like to check is the single-valuedness of the policy correspondence, i.e. if it is a policy **function**. This can be established under two additional conditions.

**Assumption 9.** $f\left(\cdot,\cdot\right)$ *is strictly concave, i.e.*

$$f\left[\alpha f\left(x,y\right)+\left(1-\alpha\right)\left(x',y'\right)\right] \geq \alpha f\left(x,y\right)+\left(1-\alpha\right)f\left(x',y'\right)$$
$$\forall\left(x,y\right),\left(x',y'\right)\in A \quad \forall\alpha\in\left(0,1\right)$$

*and with strict inequality if $x \neq x'$.*

**Assumption 10.** $\Gamma$ *is convex, i.e.* $\forall\alpha\in\left[0,1\right]$ *and* $\forall x, x'\in X$

$$y\in\Gamma\left(x\right) \text{ and } y'\in\Gamma\left(x'\right) \text{ implies}$$
$$\alpha y+\left(1-\alpha\right)y'\in\Gamma\left[\alpha x+\left(1-\alpha\right)x'\right]$$

With these assumptions we can establish the strict concavity of the value function and the continuity of the policy function (i.e. the policy correspondence is single-valued).

**Theorem 5.** *Let Assumptions 5-6 and 9-10 be satisfied. Then $V$ is strictly concave and $G$ is a continuous function.*

### 2.2.3 Stochastic Models

Everything we said applies to stochastic models where the shocks $A_t$ follows a Markov process. The mathematics behind this extension is quite complicated and involves measure theory, but as before we are going to skip all tedious technical details. What we need to do is rewrite all the previous theorems for the stochastic case. In order to do that, we need to make assumptions about the probability distribution we use.

For our purposes, we will just state the extended problem and claim that we can use functional equations to solve it. Again, the reader interested in the details can refer to Stokey et al. (1989). For simplicity, we can assume that the possible realizations of the shock $A_t$ are a finite number: $A_t \in \left\{\widehat{A_1}, ..., \widehat{A_S}\right\}$. This is very restrictive and absolutely not needed: the set of possible realizations could be countable or continuous. However, we need to make a crucial assumption about the probability distribution of $A_t$.

**Definition 1.** *A stochastic process $\{A_t\}$ is said to have the **Markov property** if for all $j \geq 1$ and all $t$ ,*

$$Prob(A_{t+1}|A_t, A_{t-1}, ..., A_{t-j)} = Prob(A_{t+1}|A_t).$$

We assume the vector of shocks $\{A_t\}$ satisfies the Markov property. Indicate the conditional probabilities of state $A'$ tomorrow, given state $A$ today, with $\pi\left(A'|A\right)$. We can basically repeat the whole analysis in the previous section and prove that we can solve the sequential problem by analyzing a slightly different version of the Bellman equation:

$$V\left(K, A\right) = \max_{0 \leq y \leq f(K,A)} U\left(K, y, A\right) + \beta \sum_{A'} \pi\left(A'|A\right) V\left(y, A'\right) \qquad \forall \left(K, A\right) \tag{2.15}$$

## 2.3 THE VALUE FUNCTION ITERATION ALGORITHM

The Contraction Mapping Theorem provides an algorithm for finding the value function. In particular, it is possible to show that, starting from an arbitrarily chosen value function, and iteratively applying the Bellman equation to get a new value function, we converge to the true value function associated with the sequential problem. The convergence is pointwise, i.e. for every point in the state space we converge to the true value function for that point. This

procedure is called **value function iteration** (VFI) and it is the basic numerical method that economists use to solve a Bellman equation. In this section we show how to write a simple MATLAB code that implements VFI.

### 2.3.1 VFI on a Computer

The CMP guarantees that the VFI converges to the unique value function that solves the problem. In particular, it is possible to show that, given a guess for the value function, we always converge to the true value function by iterating on the Bellman equation. Notice that the initial guess can be arbitrary, but of course if we start with a good guess (i.e., a guess close to the solution), we can converge faster to the true value function. Therefore in many situations it might be useful to find a good initial guess by using some of the local techniques studied in this course[6].

First of all, we cannot work on a computer with continuous variables: we need to discretize the number of possible choices we have. We therefore define a grid $G \equiv \{x_1, ..., x_m\}$ (where each point $x_i$ is a combination of $K$ and $A$ in our simple stochastic growth model), and we use the values contained in this grid as our state space. Obviously, the more grid points we have, the more accurate the solution will be. We can therefore proceed in the following way:

---

**Value function iteration algorithm**

1. Form an initial guess for the value function for each point on the grid: $V_i^0 = V^0(x_i)$

6 Finding good guesses is not always as easy as it sounds. However, there are a few techniques that can help. One is homotopy: imagine that the model you want to solve is a more general version of a well known setup for which we can easily find a solution, where this generality can be measured by a set of parameters. For example, we can easily solve the basic neoclassical growth model with log utility, no depreciation and Cobb-Douglas production function. A more general version would be the same model with CRRA utility, where the difference lies in the relative risk aversion parameter. We can therefore use the solution of the log utility setup as an initial guess for the more general version of the model, and iteratively solve CRRA versions with risk aversion parameter changing at each step, using as guess the previous iteration's solution.

2. For any $n \geq 0$, get a new guess for the value function by computing $V^{(n+1)}(\cdot) = TV^{(n)}(\cdot)$ for any point in the grid, where the operator $T$ is the **Bellman operator**:

$$(T)(V(K,A)) = \max_{0 \leq K' \leq f(K,A)} U(f(K,A) - K') + \sum_{A'} \pi\left(A'|A\right) V\left(K', A'\right) \tag{2.16}$$

3. Stop if $\left\| V^{(n+1)} - V^{(n)} \right\| < \epsilon$, where $\|\cdot\|$ is the sup norm. Otherwise, go back to 1.

---

The algorithm is very simple. However, we need to make a few more assumptions before writing the code. First of all, we need to make assumptions on the various functional forms that we want to use. In the basic stochastic growth model, for example, we must specify the utility function and the production function. We also need to choose the parameters of the model: in our example, we must a choose a value for the discount factor $\beta$, the parameters of the utility function (for example, relative risk aversion), and the depreciation rate of capital. Another crucial parameter is the criterion for convergence ($\epsilon$).

On top of these choices, we also need to define the grid for the state variables. This can also be a complicated task: in many case with more than one state variable, restricting yourself to look for a solution on a very small grid is useful as a start. Typically, a good idea is to generate a grid that includes the deterministic steady state among the grid points. However, in some cases you may be interested on what happens when you are far from the steady state[7] and therefore you may need a very wide grid. This method is quite robust and usually wide grids do not impose much of a burden.[8]

Since we are going to use MATLAB, we need to develop a code that exploits the main advantages of the software while avoiding its drawbacks. MATLAB is not very good in loops, therefore it is very important to try to avoid loops as much as possible. The code proposed below does this by making use of vector and matrices to store values for utility, policy and value functions. The code closely follows the steps in Ljungqvist and Sargent (2012), chapter 4.

Assume for simplicity that we have only two possible values for the productivity shock $[A_1, A_2]$. We store the transition probabilities of these two

7 For example, when you want to analyze what happens after big shocks that bring the economy far from the steady state. This is important when analyzing deep recessions or sovereign defaults episodes where GDP might fall by two-digit percentage figures.

8 For a systematic discussion about how to choose grid points, see Judd (1998).

realizations in a matrix $\mathcal{P}$, where for example $\mathcal{P}_{12} \equiv \pi(A_2|A_1)$. Let there be $n$ grid points for capital $[k_1, k_2, ..., k_n]$. Define two matrices $U_j$ such that:

$$U_j(i,h) = U\left(f\left(k_i, A_j\right) - k_h\right), \quad i = 1, ..., n, \quad h = 1, ..., n$$

These matrices store the value of utility of consumption for any possible combination of capital today and tomorrow.

Define also two $n \times 1$ vectors $V_j, j = 1, 2$ such that $V_j(i) = V_j\left(k_i, A_j\right), i = 1, ..., n$. Let $\mathbf{1}$ be a $n \times 1$ vector of ones. We can define an operator $T\left([V_1, V_2]\right)$ that maps couples of vectors $[V_1, V_2]$ into a couple of vectors:

$$TV_1 = \max\{U_1 + \beta\mathcal{P}_{11}\mathbf{1}V_1^{'} + \beta\mathcal{P}_{12}\mathbf{1}V_2^{'}\}$$
$$TV_2 = \max\{U_2 + \beta\mathcal{P}_{21}\mathbf{1}V_1^{'} + \beta\mathcal{P}_{22}\mathbf{1}V_2^{'}\}$$

We can write these equations in compact form:

$$\begin{bmatrix} TV_1 \\ TV_2 \end{bmatrix} = \max\left\{\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} + \beta\left(\mathcal{P} \otimes \mathbf{1}\right)\begin{bmatrix} V_1^{'} \\ V_2^{'} \end{bmatrix}\right\} \tag{2.17}$$

where $\otimes$ is the Kronecker product. We can solve by iterating on the operator $T$ until convergence.

### 2.3.2 The Code

In this section we go step by step through the code. It is assumed the reader has a basic knowledge of MATLAB fundamentals. The code is named `vfi_AM.m`. We use log utility of consumption, Cobb-Douglas production function, and we assign standard values to parameters. We choose a wide grid in order to show that the method does not pose particular problems in this case. For speed purposes, all the matrices in the code are transposed with respect to the previous section [9].

The file `parameters.m` sets the parameters values, and the grid for capital (stored in the column vector `kgrid` ). Before executing the main file `vfi_AM.m` we ALWAYS need to call this script, otherwise we get an error message (given that some parameters would not be defined). We store the parameters and the grid in this file to make easier our comparative statics exercises.

9 this is faster due to the way in which MATLAB operates. MATLAB works faster through columns.

**Listing 2.1**: vfi_AM.m, Part 1

```
cons1 = bsxfun(@minus, A_high*kgrid'.^alpha  + delta*kgrid' ,kgrid);
cons2 = bsxfun(@minus, A_low*kgrid'.^alpha  + delta*kgrid' ,kgrid);

cons1(cons1<=0) = NaN;
cons2(cons2<=0) = NaN;

util1 =  log(cons1);
util2 =  log(cons2);

util1(isnan(util1)) = -inf;
util2(isnan(util2)) = -inf;
```

The file `vfi_AM.m` generates the big matrices $U_j$. We first calculate consumption allocation implied by all the possible combinations of $K$ and $K'$, for each value of the realization of the shock[10]. Every column is therefore consumption implied by any possible choice for $K'$, given a particular value for $K$. Some values for consumption will be negative, therefore we can set them up to `NaN` to make sure that when we take the logs we don't get an error message. We then calculate utility of consumption by taking logs and we set utility for those consumption values that were negative at minus infinity (in this way, the max operator that we are about to use will never consider them to be optimal). We also clear variables that are not needed and take lots of memory space (this is not true in this model, however it might be important in more complicated setups with many state variables)[11]. This part of the code is reported on listing 2.1.

The following lines (see the code in listing 2.2) initialize variables in order to make space in the memory for them. The initial guess for the value function is zero for every gridpoint.

10 This is most of the time a big waste of memory and computational speed. There are many combinations of capital today and tomorrow that would never occur in the optimum (in technical terms, they are outside the ergodic set). Hence, we could potentially economize on the memory needs and the computational time by eliminating them. There are several techniques that can help with this task. For example, Maliar et al. (2011) and Judd et al. (2012) develop ergodic set methods for endogenously choosing the grid.

11 In the code we make use of `bsxfun` command, which is an efficient way to arithmetically combine vectors of different dimensions into one matrix. In recent versions of MATLAB, this is no more necessary: writing `k' - k` would work as well.

**Listing 2.2:** vfi_AM.m, Part 2

```
v       = zeros(nk,2);   % initial guess for value function:
                  % set to zero for simplicity
decis   = zeros(nk,2);   % initial value for policy function
metric  = 10;               % initial value for the convergence
   metric
iter = 0;
tme = cputime;
[rs,cs] = size(util1);
```

The main loop (a `while` loop) iterates over the Bellman equation, and it is reported in listing 2.3. It uses the max operator to find the $K'$ in the grid that maximizes the value for the agent, taking the value function from the previous iteration as given. The max operator delivers two numbers for each realization of $A_t$: the first (for example, `tv1`) is the value of the maximized object, the second (`tdecis1`) is the position of the maximizer in the grid (i.e., if the value of `tdecis1` is 100, it means that the maximizer is the 100th value in `kgrid`). An important point is that we don't need to modify the matrices `util1` and `util2` at each iteration: they are calculated and stored in memory before the main loop, thus saving computational time at each iteration. We calculate `metric` as the *sup norm*, which is our concept of distance for determining convergence. We keep iterating until `metric` becomes small enough. We also save the value of `metric` in a vector for generating a graph of the convergence process. Finally, after convergence is achieved, we transform the variable `decis` of positions into actual capital values over the grid.

The last part of the code does some simulations, generating a series for consumption and investment. In order to generate a Markov chain for $A_t$, it uses a subroutine called `markov.m` (we are not going to give details about it). The file `figures.m` plots both the value and the policy function over the grid, and the series. The last graph is a $log_{10}$ scale plot of `metric` for each iteration (i.e., on the vertical axis we get the order in terms of powers of 10 for the *sup norm*. Each graph is saved as a PostScript file.

**Listing 2.3:** vfi_AM.m, Part 3

```
while metric > convcrit;

  contv= beta*v*prob'; % continuation value
    [tv1,tdecis1]=max(bsxfun(@plus,util1,contv(:,1)) );
  [tv2,tdecis2]=max(bsxfun(@plus,util2,contv(:,2)) );

  tdecis=[tdecis1' tdecis2'];
  tv=[tv1' tv2'];

  metric=max(max(abs((tv-v)./tv)));
  v=   tv;
  decis= tdecis;%
  iter = iter+1;
  metric_vector(iter) = metric;
  disp(sprintf('iter = %g ; metric = %e', iter,metric));
end;
disp('  ');
disp(sprintf('computation time = %f', cputime-tme));

% transform the decision index in capital choice
decis=(decis-1)*ink + mink;
```

## Exercise 2.1 Playing with the code

### *Part I — Running the code*

Run the code `do_vfi_AM.m`. Familiarize with the output and the graphs. Now, after the line which loads parameter values, change the value for the discount factor to 0.995, by adding the following line:

```
betta = 0.995
```

This line changes the value set in the file `parameters.m` to a new value (this is a general way to do comparative statics by using a baseline set of parameters.) Save this file as `do_vfi_AM_beta.m` and run it.

1. What do you notice in the convergence process? What happens to computational time? (Why?)

### *Part II — Accuracy*

We now want to see what happens if we change the number of gridpoints. Change the code so that the grid has 100 gridpoints. (**Hint**: notice that after you set `nk=100`, then you also have to modify the variables `kgrid` and `ink`, therefore you need to add those lines too!!!) Save the file as `do_vfi_AM_smallgrid.m` and run it.

1. What can you notice? Now try with 2000 gridpoints, save the file as `do_vfi_AM_largegrid.m`, and run it. Do you see any change?

### *Part III — Comparative statics*

We want to do a series of comparative statics exercises. In order to do that, we can modify the file `do_compstat`, which is a basic structure for this task. Open the file `do_compstat.m`. You should see the following lines (I have excluded the lines that plot the simulated results for brevity):

```
%% Generate solution and simulation for case A
% load parameters and grid
parameters;
% solve the model via VFI
vfi_AM;
% store results in few new variables
v_A = v;
decis_A = decis;
controls_A = controls;

%% Generate solution and simulation for case B
```

```
% load parameters and grid
parameters;
% modify parameters here

% solve the model via VFI
vfi_AM;
% store results in few new variables
v_B = v;
decis_B = decis;
controls_B = controls;
```

This file compares a case A and a case B, where the case A is the benchmark (i.e. the solution with default parameters), and case B is the one with the updated parameter value. Notice the line that says:

```
% modify parameters here
```

We can just put the new value we want to consider there. Then by running the file we should get the graphs of case A and B and compare them.

1. As a first try, set `delta = 0.8` (notice that this implies a larger depreciation rate for capital, given the definition of $\delta$ in the code). Save the file as `do_compstat_delta.m`. Run the do file, what changes with respect to the benchmark case?
2. Now let's change the values for the shock realizations `A_high` and `A_low`. In particular, let's have a mean-spread transformation such that the the mean is the same given the iid hypothesis for the transition matrix. Set `A_high=1.25` and `A_low=0.75`, save the file as `do_compstat_shock.m` and run the code. What can you notice?
3. Let's now relax the assumption of iid shocks. In order to do that, we need to change the transition matrix `prob` by inducing some persistence in the stochastic process. Modify the matrix in such a way that each realizations has a probability of repeating itself in the following period of 95%, i.e. the probability of a high (low) realization tomorrow given that the realization today is high (low) is 0.95. Save the file as `do_compstat_persistent.m` and run the code. What conclusions can you draw from the graphs?

## Exercise 2.2 Irreversible investment

VFI makes easy to incorporate inequality constraints. In order to see this, modify the code by introducing an irreversibility constraint for investment, i.e. $k_{t+1} \geq (1-\delta)k$.

1. Repeat the previous exercise for the model with irreversible investment. The relevant codes are `do_vfi_AM_irrinv.m` and `do_compstat_irrinv.m`.

## Exercise 2.3 Convex adjustment costs

It is also quite straightforward to introduce convex adjustment costs. Modify the code by adding the adjustment costs in the form $AC(k_{t+1}, k_t) \equiv \zeta(k_{t+1} - k_t)^2$, where $\zeta$ is a parameter. Choose $\zeta = .25$ to begin with, then play with it and see what happens.

## Exercise 2.4 Reversible vs irreversible investmen

Now let's compare the reversible investment model with the one with irreversible investment. In order to do that, we use the file `do_compare.m`. (Notice that in this case we will have to change parameters in two points of the code!)

1. Run the code as it is, and look at the graphs. What are the main differences between the reversible and irreversible investment models? Check in particular the simulated series of consumption and investment.
2. Now let's how the depreciation rate is crucial. First change the depreciation rate of capital to `delta=0.99`. You can do this by inserting `delta=0.99` in the appropriate spaces (remember: you have two do it twice for this code!!!). Save the new file as `do_compare_delta.m` and run it. What do you observe? Can you explain it intuitively? Now set `delta=0.5` (in both the appropriate spaces!!!) and run the code again. What now?
3. Let's see how the model with irreversible investment reacts when the variance of the shocks is reduced. Set `A_high=1.25` and `A_low=0.75`, save the file as `do_compare_shock.m` and run the code. This must be surprising for you! (Is it?)
4. Finally, what about persistence? Set the transition matrix `prob` such that each realization has a probability of repeating itself in the following period of 95%, i.e. the probability of a high (low) realization tomorrow given that the realization today is high (low) is 0.95. Save the file as `do_compare_persistent.m` and run the code.

### Exercise 2.5 Many possible realizations of the technology shock

We can adapt the code to deal with more than two realizations of the shocks. This is a slightly more involved change since we need to modify the code in several points:

1. we need a new variable that stores the number of realizations of the shocks (call it `num_realiz`) in the parameters' file
2. we need to provide a transition matrix `prob` which adapts to `num_realiz` (for the moment we stick to the iid case for simplicity) in the parameters' file
3. we need a vector where we store the actual values for technology shock realizations in the parameters' file
4. we need to adapt the way in which we create the matrices $U_j$ (hint: use the Kronecker product, use Matlab help for the `kron` command)
5. we need to adjust the way in which we compute the Bellman operator, in particular for the expectations' part
6. finally, we must adapt the simulations to the generic case with many possible realizations of the Markov chain

We can of course do the same for the model with irreversible investment. The two solution codes are respectively `vfi_AM_general.m` and `vfi_AM_irrinv_general.m`.

## 2.4 POLICY FUNCTION ITERATION

The value function iteration algorithm is quite slow, and for many economic problems computational speed is important. We therefore would like to have a similar approach that has the same convergence properties, but it is faster. We can have a much faster computation of the solution by using the policy function iteration algorithm, also known as Howard's improvement algorithm.

Remember that our problem is

$$V(K_{-1}) \equiv \max_{\{K_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - K_t\right)$$
$$s.t. \quad 0 \leq K_t \leq f(K_{t-1}), \quad t = 0, 1, ...$$

The policy function algorithm has the following steps:

---

**Policy function iteration algorithm**

1. Start with a guess for the policy function $K' = g^{(0)}(K)$
2. Calculate the value associated with this policy function:

$$V^{(0)}(K) = \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - g^{(0)}(K_{t-1})\right)$$

3. For any $n \geq 0$, get a new policy function $K' = g^{(n+1)}(K)$ by solving

$$\max_{K'} \left\{ U\left(f(K) - K'\right) + \beta V^{(n)}\left(K'\right) \right\}$$

4. Calculate the value associated with this policy function:

$$V^{(n+1)}(x) = \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - g^{(n+1)}(K_{t-1})\right)$$

5. Iterate over 3-4. Stop if $\left\| V^{(n+1)} - V^{(n)} \right\| < \epsilon$

---

To simplify notation, we can think of the system as residing in one of $N$ predetermined positions, call them $x_i$ for $i = 1, 2, ..., N$. Therefore, each $x_i$ is a combination of $A$ and $k$ on the grid. Our task is to find what is the $x_j$ on the grid that we choose as state tomorrow, when starting from $x_i$ today. We can do this by defining a class of transition matrices $\mathcal{M}$ of size $N \times N$, with elements denoted as $P$ among which we will choose. In other words, we will choose the $P_{ij} = Prob\{x_{t+1} = x_j \mid x_t = x_i\}$ for each $i$ and $j$ that solve our maximization problem. Each matrix $P$ is a "combination" of a stochastic part (defining transition for the stochastic variable '$A$) and a deterministic part defininf transition to a future capital $k'$. We can write the Bellman equation as:

$$v(x_i) = \max_{P \in \mathcal{M}} \left\{ u(x_i) + \beta \sum_{j=1}^{N} P_{ij} v\left(x_j\right) \right\}$$

or in a more compact form:

$$v = \max_{P \in \mathcal{M}} \left\{ u + \beta P v \right\} \tag{2.18}$$

We can rewrite it as as

$$v = Tv$$

where $T$ is the operator corresponding to the RHS of (2.18). Define another operator $B \equiv T - I$ such that

$$Bv = \max_{P \in \mathcal{M}} \{u + \beta P v\} - v$$

and therefore we can see the policy function iteration as solving $Bv = 0$. In other words, we can rewrite the policy function iteration algorithm in terms of the $B$ operator.

---

**Policy function iteration algorithm, version 2**

1. Given $P^{(n)}$, get $v^{(n)}$ from

$$\left(I - \beta P^{(n)}\right) v^{(n)} = u^{(n)} \tag{2.19}$$

2. Find $P^{(n+1)}$ such that

$$u^{(n+1)} + \left(\beta P^{(n+1)} - I\right) v^{(n)} = Bv^{(n)} \tag{2.20}$$

---

The first step is a linear algebra problem that can be solved by posing:

$$v^{(n)} = \left(I - \beta P^{(n)}\right)^{-1} u^{(n)}$$

You can interpret the policy iteration algorithm as a form of the Newton's method to find the zeroes of $Bv = 0$. Notice that, by using (2.19) you can rewrite equation (2.20) as:

$$\left(I - \beta P^{(n+1)}\right) v^{(n+1)} + \left(\beta P^{(n+1)} - I\right) v^{(n)} = Bv^{(n)}$$

or in other terms

$$v^{(n+1)} = v^{(n)} + \left(I - \beta P^{(n+1)}\right)^{-1} Bv^{(n)} \tag{2.21}$$

Since from equation (2.19) we can interpret $\left(I - \beta P^{(n+1)}\right)$ as the gradient of $Bv^{(n)}$, therefore Howard's improvement algorithm is a version of the Newton's method[12] to find the roots of $Bv = 0$.

12 More on Newton's method in the next chapter.

How do we implement it numerically? Say we have only two possible values for the shock as in the previous Section, and define the two $n \times n$ matrices:

$$J_h\left(k_i, k_j\right) = \left\{ \begin{array}{ll} 1 & if \quad g\left(k_i, A_h\right) = k_j \\ 0 & o/w \end{array} \right\}$$

These matrices are the policy function that we have to solve for. They are slightly different from the matrices $P$ in the previous discussion, since they don't include the stochastic part. One can always decompose the stochastic and the deterministic part of $P$ as we are doing here. For a given policy function, $k' = g\left(k, A\right)$, define two $n \times 1$ vectors $U_h$ such that:

$$U_h\left(k_i\right) = u\left(A_h f\left(k_i\right) + \left(1-\delta\right) k_i - g\left(k_i, A_h\right)\right)$$

Assume the policy function is used forever. We can associate the two vectors $V_h\left(k_i\right)$ as the values associated with starting from state $\left(k_i, A_h\right)$. Therefore:

$$\left[ \begin{array}{c} V_1 \\ V_2 \end{array} \right] = \left[ \begin{array}{c} U_1 \\ U_2 \end{array} \right] + \beta \left[ \begin{array}{cc} \mathcal{P}_{11} J_1 & \mathcal{P}_{12} J_1 \\ \mathcal{P}_{21} J_2 & \mathcal{P}_{22} J_2 \end{array} \right] \left[ \begin{array}{c} V_1 \\ V_2 \end{array} \right]$$

We can therefore solve for $V_h$ by means of elementary linear algebra and have:

$$\left[ \begin{array}{c} V_1 \\ V_2 \end{array} \right] = \left[ I - \beta \left( \begin{array}{cc} \mathcal{P}_{11} J_1 & \mathcal{P}_{12} J_1 \\ \mathcal{P}_{21} J_2 & \mathcal{P}_{22} J_2 \end{array} \right) \right]^{-1} \left[ \begin{array}{c} U_1 \\ U_2 \end{array} \right] \tag{2.22}$$

Now, we can do the following:

---

**Policy function iteration algorithm, version 3**

1. Given an initial feasible policy function, compute equation (2.4) and use it to solve for the value functions with equation (2.22)
2. Do one iteration on the Bellman equation (2.17) by using value functions found in step 1
3. Iterate until convergence

---

This algorithm is much faster than value function iteration, it converges in fewer iterations. We can speed up even more by slightly changing step 2: instead of doing just one iteration on the Bellman equation, do $m$ iteration, where $m$ is a reasonably small integer (I would say up to 50)[13].

13 If we understand that PFI is like a Newton method for a particular set of nonlinear equations, then we also understand why it is much faster. For VFI, convergence is linear in logs, while for PFI is quadratic, as for standard Newton methods.

*The Code*

The PFI code is very similar to the VFI, of course. In fact, the code for VFI must be changed in a few critical spots. The file `pfi_AM.m` is substantially identical to `vfi_AM.m`, except in the main loop.

**Listing 2.4:** pfi_AM.m

```
while metric > convcrit;

  contv= beta*v*prob'; % continuation value

  [tv1,tdecis1]=max(bsxfun(@plus,util1,contv(:,1)) );
  [tv2,tdecis2]=max(bsxfun(@plus,util2,contv(:,2)) );

  tdecis=[tdecis1' tdecis2'];

  % Build return vectors
  r1 = zeros(cs,1);
  r2 = zeros(cs,1);
  for i=1:cs
    r1(i) = util1(tdecis1(i),i);
    r2(i) = util2(tdecis2(i),i);
  end

  % create matrices Js (see lecture notes)
  g2=sparse(cs,cs);
  g1=sparse(cs,cs);
  for i=1:cs
    g1(i,tdecis1(i))=1;
    g2(i,tdecis2(i))=1;
  end
  % This is the marix P (see lecture notes)
  trans=[ prob(1,1)*g1 prob(1,2)*g1; prob(2,1)*g2 prob(2,2)*g2];

  % Linear algebra step to get the value function associated with
      P
  tv(:) = ((speye(2*cs) - beta.*trans))\[ r1; r2 ];

```

**Listing 2.4 (continued):** pfi_AM.m

```
  metric=max(max(abs((tv-v)./tv)));
  v=   tv; % .15*tv+.85*v; %
  decis= tdecis;%
  iter = iter+1;
  metric_vector(iter) = metric;
  disp(sprintf('iter = %g ; metric = %e', iter,metric));
end;
disp('  ');
disp(sprintf('computation time = %f', cputime-tme));

% transform the decision index in capital choice
decis=(decis-1)*ink + mink;
```

We need to calculate the return matrices implied by the new policy, and then the matrices $J$ that contain the policy function in matrix form. We then perform the linear algebra step to find the new value function. Notice that the code can be written in a more efficient way, however it is already more than 20 times faster than the VFI code, and converges in substantially fewer iterations. This is a general property of the Newton's method, since it has a quadratic convergence, while VFI has linear convergence[14].

## Exercise 2.6

Modify the PFI code to analyse the model with irreversible investment and the model with investment adjustment costs seen in the section about value function iteration. (Hint: are the modifications done for the VFI code enough?)

14 The interested reader can consult Judd (1998) for more details.

# 3 WHEN RECURSIVE METHODS FAIL

## 3.1 INTRODUCTION

Till now, we have seen problems of the form:

$$\begin{aligned} \max_{\{u_t\}_{t=0}^{\infty}} \quad & \sum_{t=0}^{\infty} \beta^t r\left(x_t, u_t\right) \\ s.t. \quad & x_{t+1} = h\left(x_t, u_t\right), \quad t = 0, 1, ... \\ & x_0 \in X \text{ given} \end{aligned}$$

However there are many situations in (macro)economics that cannot be represented in the previous form. All these situations involve cases in which (the expectations of) future behavior influence today's choices. Once we introduce this complication, the mathematical problem we are facing does not satisfy a standard Bellman equation, in which the policy function only depends on the state variables. There are mainly three categories in which this problem can arise: problems of optimal policy choice by a benevolent government which suffer from time inconsistency, models of limited commitment, and private information environments.

These problems seem therefore impossible to solve. However, it turns out that we can use a generalization of the Bellman theory ("dynamic programming squared", as Ljungqvist and Sargent (2012) calls it), where we rewrite the problem in order to introduce a new auxiliary state variable (sometimes more than one), and we can show that this new formulation of the problem has a recursive structure.

There are two main approaches to these more complicated problems. The first was introduced by Abreu, Pearce and Stacchetti in late 80s, and it is the most popular tool in problems with private information and limited commitment, since it directly relies on a game-theoretic formulation of the technique. The second is the approach by Marcet and Marimon, which is still on its way to publication, and it is very popular in optimal policy and limited commitment models. More recently, Marcet and Marimon's approach has been extended to private information environments by the work of Sleet and Yeltekin, and to repeated moral hazard framework by my own work.

In the next section, we give some examples of economic models for which we cannot use standard dynamic programming. The next two chapters will be devoted to the description of the two approaches. In chapter 5 we will present the main ideas in Abreu et al. (1990) (APS from here on) and we will show how to rewrite the examples in a recursive way. Then we will devote our attention to Marcet and Marimon (2017)'s approach in chapter 6.

## 3.2 SOME EXAMPLES WHERE STANDARD DYNAMIC PROGRAMMING FAILS

We present three examples. The first is an optimal fiscal policy problem. The second is a case of one-sided limited commitment. The third is a dynamic private information environment.

### 3.2.1 Optimal fiscal policy

Imagine the economy is populated by a representative agent and a benevolent government. The government must finance exogenous random expenditure by means of distorsive taxation and debt. The agent solves his own maximization problem:

$$\begin{aligned} \max_{\{c_t, b_t, l_t\}_{t=0}^{\infty}} \quad & E_0 \sum_{t=0}^{\infty} \beta^t \left[ u(c_t) + v(l_t) \right] \\ s.t. \quad & c_t + p_t^b b_{t+1} = l_t (1 - \tau_t) + b_t \end{aligned} \tag{3.1}$$

where $c_t$ is consumption, $b_{t+1}$ are one-period bonds that pay one unit of consumption tomorrow and cost $p_t^b$ today to buy, $l_t$ is labor supply and $\tau_t$ is a proportional tax on labor income. Substitute the budget constraint in the utility function. The first order conditions for this maximization problem are:

$$/b_{t+1}: \qquad p_t^b u'(c_t) = \beta E_t u'(c_{t+1}) \tag{3.2}$$

$$/l_t: \qquad -\frac{v'(l_t)}{u'(c_t)} = (1 - \tau_t) \tag{3.3}$$

The government wants to maximize the utility of representative agent subject to its own budget constraint, resource constraint, and taking as given the optimal choices of the agent summarized by equations (3.2)-(3.3) and the indivi-

dual budget constraint (3.1). Therefore we can write the government problem as:

$$\begin{aligned}
\max_{\{c_t,b_t,l_t\}_{t=0}^{\infty}} \quad & E_0 \sum_{t=0}^{\infty} \beta^t \left[u\left(c_t\right) + v\left(l_t\right)\right] \\
s.t. \quad & (3.1), (3.2), (3.3) \\
& c_t + g_t = l_t && (3.4) \\
& g_t + p_t^b b_{t+1}^g = b_t^g + \tau_t l_t && (3.5) \\
& b_t^g + b_t = 0 && (3.6)
\end{aligned}$$

where it is understood that all constraints must be satisfied for any $t$.

We can simplify this problem. First of all, we can use (3.6) in (3.1), obtaining:

$$c_t - p_t^b b_{t+1}^g = l_t\left(1 - \tau_t\right) - b_t^g \tag{3.7}$$

Second, by Walras law, we can eliminate (3.5) from the problem. We can use (3.3), (3.4) and (3.2) in (3.7) to get:

$$c_t - \beta \frac{E_t u'\left(c_{t+1}\right)}{u'\left(c_t\right)} b_{t+1}^g = -\left(c_t + g_t\right) \frac{v'\left(c_t + g_t\right)}{u'\left(c_t\right)} - b_t^g \tag{3.8}$$

Therefore, the maximization problem we want to solve is:

$$\begin{aligned}
\max_{\{c_t,b_t,l_t\}_{t=0}^{\infty}} \quad & E_0 \sum_{t=0}^{\infty} \beta^t \left[u\left(c_t\right) + v\left(l_t\right)\right] \\
s.t. \quad & c_t - \beta \frac{E_t u'\left(c_{t+1}\right)}{u'\left(c_t\right)} b_{t+1}^g = -\left(c_t + g_t\right) \frac{v'\left(c_t + g_t\right)}{u'\left(c_t\right)} - b_t^g
\end{aligned}$$

where again the constraint must be satisfied for any $t$. Notice that in this problem, the optimal choice of, e.g. $c_t$ depends, through the constraint, on the expected value of $c_{t+1}$. Therefore the problem is not recursive, since information about the past state variables is not sufficient to determine the current period optimal choices. This means that we cannot use standard dynamic programming.

### 3.2.2 One-sided lack of commitment

Imagine there are two individuals in our economy. The first is risk neutral, and we will call it the principal. The second is risk averse and we will call it the agent.

The agent receive a stochastic endowment in each period $t$, which we will indicate with $y_t$. The principal is able to borrow and lend at a constant risk-free interest rate $R = \beta^{-1}$, while the agent cannot save and has no access to any credit market. The principal and the agent want to share the risk associated with this endowment. Therefore, they want to write a contract at time 0, that defines the sharing rule for each period $t$ and each possible contingency determined by the realization of the endowment process.

We can solve this model by analyzing the following dynamic programming problem:

$$\begin{aligned} \max_{\{c_t\}_{t=0}^{\infty}} \quad & E_{-1} \sum_{t=0}^{\infty} \beta^t \left( y_t - c_t \right) \\ s.t. \quad & E_{-1} \sum_{t=0}^{\infty} \beta^t u \left( c_t \right) \geq U_0 \end{aligned} \tag{3.9}$$

This problem fits in the standard dynamic programming theory: therefore the policy function will depend only on state variables (in our case, the endowment of the agent), as well as the value function. In particular, notice that, if we call $\gamma$ the Lagrange multiplier associated with the constraint (3.9), and take first-order conditions of the problem, we get:

$$u' \left( c_t \right) = \frac{1}{\gamma}$$

i.e., the optimal consumption for the agent is constant, and the entire risk is beard by the principal.

Unfortunately, this situation is not very realistic. Imagine that, while the Principal is a nice guy and can commit forever to stay in the risk-sharing contract, the Agent is not a loyal guy, and can leave the arrangement at any time, being at that point on his own and having to consume just his individual endowment. Therefore, the principal would like to write a contract such that the risk-sharing rule gives the agent always a sufficient incentive to keep the agreement.

We can write the problem as before, but we need to add a so called **participation constraint** that makes sure the value of staying in the contract for the

agent is always larger than the value of leaving in the contract and staying in autarky (i.e., consuming only his own endowment):

$$\begin{aligned}
&\max_{\{c_t\}_{t=0}^{\infty}} E_{-1} \sum_{t=0}^{\infty} \beta^t \left(y_t - c_t\right) \\
&s.t. \quad E_{-1} \sum_{t=0}^{\infty} \beta^t u\left(c_t\right) \geq U_0 \\
&\qquad u\left(c_t\right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left(c_{t+j}\right) \geq \\
&\qquad\qquad \geq u\left(y_t\right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left(y_{t+j}\right), \forall t \qquad (3.10)
\end{aligned}$$

Notice that in this formulation future choice variables affect current choices, through the participation constraint (3.10). Therefore, we cannot use standard dynamic programming.

### 3.2.3 Private information

I show two examples. The first is a case in which agents have private information on their endowment. The second is a framework with hidden effort.

#### *Endowment is private information*

We remove the assumption of lack of commitment introduced in the previous section, but we assume the agent's endowment is private information, and evolves according to the same transition probability function as in the previous example with lack of commitment. Define $y^t \equiv \left\{y_j\right\}_{j=0}^{t} \in Y^{t+1}$ as the history of realizations of $y_t$. We can invoke the revelation principal, and restrict ourselves to the analysis of truthful revelation mechanisms. The principal chooses transfers $\tau_t\left(y^t\right) \equiv c_t\left(y^t\right) - y_t$ trying to minimize the expected discounted cost of implementing an incentive compatible allocation. We assume there are $N$ pos-

sible realizations for the income process $y_t$. Let $y \in \{\overline{y}_i\}_{i=1}^N$ such that $\overline{y}_i < \overline{y}_{i+1}$ for any $i$. Define the local downward incentive-compatibility constraints as

$$DIC_t\left(y^{t-1}, \overline{y}_i; \overline{y}_{i-1}\right) = \left\{ \beta^t \left[ u\left(c_t\left(y^{t-1}, \overline{y}_i\right)\right) + \beta U_{t+1}\left(y^{t-1}, \overline{y}_i\right) \right.\right.$$
$$\left.\left. - u\left(c_t\left(y^{t-1}, \overline{y}_{i-1}\right)\right) - \beta U_{t+1}\left(y^{t-1}, \overline{y}_{i-1}\right) \right] \right\}$$
$$\forall y^{t-1}, \quad \forall \overline{y}_i, \quad i = 2, ..., N$$

where

$$U_{t+1}\left(y^{t-1}, \overline{y}_i\right) \equiv \sum_{j=1}^{\infty} \sum_{y^{t+j} \in Y^{t+j+1}} \beta^{j-1} u\left(c_{t+j}\left(y^{t-1}, \overline{y}_i, y^{t+j}\right)\right) \pi\left(y^{t+j} | \left(y^{t-1}, \overline{y}_i\right)\right)$$

We can solve a relaxed Pareto-constrained problem for which we only impose the local downward incentive-compatibility constraints. In general, we can find sufficient conditions under which the relaxed problem delivers the solution of the original problem. Then the Pareto-constrained problem of the principal can be written as

$$\max_{\left\{\tau_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[-\tau_t\left(y^t\right)\right] \pi\left(y^t | y_{-1}\right)$$

$$s.t. \qquad \sum_{t=0}^{\infty} \sum_{y^t \in y^{t+1}} \beta^t u\left(c_t\left(y^t\right)\right) \pi\left(y^t | y_{-1}\right) \geq U_0$$

$$DIC_t\left(y^{t-1}, \overline{y}_i; \overline{y}_{i-1}\right) \geq 0 \qquad \forall y^{t-1}, \quad \forall \overline{y}_i, \quad i = 2, ..., N$$

In this case, we can see that the dependence of current choices on future variables enters in the problem through DIC constraints. Therefore, standard dynamic programming cannot be used.

*Hidden effort*

Everything is as in the previous example, except that now we assume the endowment is observable and i.i.d., but its future distribution is affected by an

unobservable action $a_t$ through the transition function $\pi\left(y_{t+1} \mid a_t\right), s = 1, ..., S$. This action is costly for the agent, therefore his instantaneous utility is given by:

$$u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)$$

with $u(\cdot)$ strictly increasing, strictly concave and satisfying Inada conditions, while $v(\cdot)$ is strictly increasing and strictly convex; both are twice continuously differentiable. We also assume the instantaneous utility is uniformly bounded.

We can characterize the optimal contract by solving:

$$\begin{aligned}
&\max_{\left\{\tau_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[-\tau_t\left(y^t\right)\right] \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \\
&s.t. \quad \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)\right] \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \geq U_0 \\
&\left\{a_t\left(y^t\right)\right\}_{t=0}^{\infty} \in \arg \max_{\left\{a_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)\right] \times \\
&\qquad \times \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \qquad (3.11)
\end{aligned}$$

Here the dependence of current choices on future variables comes from the constraint (3.11). Once again, Bellman equation does not hold in its standard formulation.

# 4 NUMERICAL TECHNIQUES

This chapter explains how the CompEcon Toolbox work, and then introduces projection methods as a powerful method to solve functional equations.

## 4.1 NUMERICAL METHODS

This section introduces the reader to the CompEcon Toolbox. In particular, we will illustrate a few numerical techniques that we will later use when presenting the projection methods. Instructions on how to add the CompEcon Toolbox on your local MATLAB are in the appendix.

### 4.1.1 Integrals

The codes used in the dynamic programming chapters assumed discrete shocks. However, they can be easily adapted to continuous shocks. In order to do this, we need to find a way to calculate expectations over a continuous supports, which is equivalent to calculating an integral. There are different ways to do it, and all methods approximate the integral with a finite sum (this is called *numerical quadrature*). For example, if we want to calculate the integral over a set $I$ of a function $f(x)$ weighted by a weight function $w(x)$, we can use

$$\int_I f(x)w(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i \tag{4.1}$$

and different methods usually differ in the way we choose the nodes $x_i$ and the weights $w_i$. Newton-Cotes methods approximate the $f$ between nodes using low order polynomials, then sum the integrals of those polynomials (which are easy to calculate). There are two widely used versions: the trapezoid rule, which approximates the function with piecewise linear interpolants, and the Simpson's rule which uses piecewise quadratic interpolants.

Gaussian quadrature chooses nodes and weights by matching some moments of the distribution. Imagine you have a weighting function $w : I \to R$,

where $I \in R$ is an interval. Then the Gaussian quadrature of order $n$ chooses $n$ quadrature nodes that satisfy the $2n$ moment matching conditions:

$$\int_I x^k w(x)dx = \sum_{i=1}^{n} w_i x_i^k,\ k = 0, ..., 2n-1 \tag{4.2}$$

When the weighting function is $w(x) = 1$, we call this method Gauss-Legendre quadrature, and it turns out to be well suited for calculating integrals of smooth (i.e. with continuous derivatives) functions.

Monte Carlo methods randomly choose nodes. However, there are a lot of issues with those randomly chosen nodes, the biggest being that every random number generator is quite imprecise. There are therefore so called pseudo-Monte Carlo methods that try to improve the reliability of the integral.

Depending on the problem, there may be one method that outperforms others in terms of accuracy and/or computational time. However, it's worth mentioning that there are various ways of replicating a continuous process with one with discrete support, for example Tauchen's method is one of them[1].

COMPECON IMPLEMENTATION All these methods are included in the CompEcon Toolbox. These routines allow to compute integrals of real-valued function over bounded intervals. Trapezoid rule is implemented by using

```
[x,w] = qnwtrap(n,a,b);
```

where $n$ is the number of nodes, and $a$ and $b$ are the extrema of the interval. Notice that all these parameters can be vectors, in case you want to calculate multidimensional integrals. Simpson's rule and Gauss-Legendre integration have the same syntax with commands respectively `qnwsimp` and `qnwlege`. For example, we can calculate the definite integral of $\log(x)$ on $[3,7]$ with 20 nodes trapezoid rule by using

```
[x,w] = qnwtrap(20,3,7);
integral = w'*log(x);
```

## Exercise 4.1 Integrals

1. Calculate the integral of $e^{-x}$ on the interval $[-1,1]$ using the trapezoid rule.

1 By looking for Tauchen in Google, you may find a few different MATLAB routines that implement this method, if you are interested.

2. Calculate the integral of $|x|^{\frac{1}{2}}$ on the interval $[-1,1]$ using Simpson's rule.

3. Calculate the integral of $(1+25x^2)^{-1}$ on the interval $[-1,1]$ using Gauss-Legendre method.

## Exercise 4.2 Compare the methods

Write a short script that compares the three methods. Calculate the integral of $e^{-x}$, $|x|^{\frac{1}{2}}$ and $(1+25x^2)^{-1}$ on the interval $[-1,1]$ by hand (these are pretty easy to calculate). Then use CompEcon commands and calculate the integrals with 10,20,30,100 nodes for each method. Save the results in a matrix and then compare the accuracy, i.e. the difference between the numerical integral and the correct integral calculated by hand. What can you notice?

CompEcon contains routines for computing Gaussian nodes and weights of the most common distributions. For our purpose, the most important command is `qnwnorm` which calculates the nodes and weights for multidimensional normal distributions with the syntax

```
[x,w] = qnwnorm(n,mu,var);
```

where `n` is a vector containing the number of nodes in each dimension, `mu` is the mean vector, and `var` is the variance-covariance matrix. For example, if we want to calculate the expectations of $e^{-x}$, with $x \sim \mathbf{N}(0,1)$, and 30 nodes, we can use

```
[x,w] = qnwnorm(30,0,1);
expectations = w'*exp(-x);
```

## Exercise 4.3 Univariate normal

Calculate the expected value of $x^{-\sigma}$, for $\sigma = \{0.5,1,2,10\}$ and $x \sim \mathbf{N}(0,1)$. Use 30 nodes.

## Exercise 4.4 Multivariate normal

Calculate the expected value of $e^{x_1+x_2}$, with $x_1$ and $x_3$ jointly normal with $Ex_1 = 3$, $Ex_2 = 4$, $Var(x_1) = 2$, $Var(x_2) = 4$, $Cov(x_1,x_2) = -1$. Use 10 nodes in the $x_1$ direction and 15 nodes in the $x_2$ direction.

Similar functions exist for Gamma, lognormal, Beta distributions, for example. There are more sophisticated ways of choosing nodes and weights, and

CompEcon has more routines dedicated to it. The interested reader can refer to Miranda and Fackler (2004) for more details.

### 4.1.2 Nonlinear equations

In many cases, the optimization step in the VFI or PFI algorithm can be substituted with a nonlinear equation problem given by the first-order conditions. This of course saves time, and it often makes the problem easier to solve. Moreover, nonlinear equations are essential in projection methods, which are a method for solving functional equations. In the following, we illustrate a few algorithms that are helpful for this task. All the following methods are implemented in Matlab as "functions of functions", i.e. one of the input is a function that contains the equation we want to solve, in the format $f(x) = 0$. Therefore, we can either write down the function with the equations we want to solve as an inline function, or we can create a file. As it is customary in Matlab, the nonlinear solver solves for the first input, and we have to pass other additional input values as parameters.

#### *Bisection method*

Bisection is the simplest method for finding the solution of a nonlinear equations of the form

$$f(x) = 0$$

where $f(x)$ is a continuous real-valued function on a real interval $[a, b]$. The algorithm is very simple:

---

**Bisection algorithm**

1. Start with two points $a_1, b_1$ such that $f(a_1) < 0$, $f(b_1) > 0$
2. Choose a new point $x_2 \in [a_1, b_1]$ and calculate the sign of $f(x_2)$. If positive, now restrict your search to $[a_1, x_2]$, if negative to $[x_2, b_1]$.

---

Coding the algorithm is very easy. This method works well in case there is a unique solution, however it might create some problems when the equation at hand has more than one root. The function `bisect` in CompEcon implements the bisection method. We can use it in the following way:

```
1 x = bisect(fun, a, b)
```

where the function `fun` is either an inline function, a handle for a function, or a string containing the function file name. The interval over which we look for the solution is $[a, b]$. For example, if we want to find a zero for the function $x^3 - 5$ in the interval $[1, 2]$, we can write

```
f = inline('x^3 - 5');
x = bisect(f,1,2)
```

The function should be defined with a function file when one needs to find the roots of a parameterized version of the objective function. `bisect` works with vectorial or matricial functions too.

## Exercise 4.5 Bisection

1. Find the roots of $e^{-x^2} - \cos(x)$ over the interval $[-4, 6]$.
2. Plot the function over the interval. What can you observe?

*Function iteration*

This method is useful for equations of the type

$$f(x) = 0$$

where $f : R^n \to R^n$. We first rewrite the equation as $x = g(x)$ for some function $g : R^n \to R^n$, and then, given a guess $x^{(0)}$

---

**Function iteration algorithm**

1. Calculate $x^{(1)} = g(x^{(0)})$
2. Iterate over the recursion $x^{(n+1)} = g(x^{(n)})$ until convergence, i.e. until $\left\| x^{(n+1)} - x^{(n)} \right\| < \epsilon$

---

Rewriting the equation in the form $x = g(x)$ is always possible since $x = x - f(x)$. This method is widely used to solve Euler equations. Function iteration is guaranteed to converge to a fixed-point of $g$ if $g$ is differentiable and if the initial guess is close enough to the solution where $g'(x) < 1$. However, convergence happens quite often even if this condition is not satisfied. The routine `fixpoint` implements this algorithm in CompEcon. The syntax for `fixpoint` is:

```
x = fixpoint(fun, guess)
```

where the considerations about the function made in the bisection case hold here too. As an example, to solve the equation $x + x^0.5 = 0$ with initial guess 0.4, we write

```
g = inline('x^0.5');
x = fixpoint(g,0.4)
```

where the second argument of the command is an initial guess for the solution.

## Exercise 4.6 Function iteration

Find the roots of $e^{-x^2} - \cos(x)$ over the interval $[-4, 6]$. (Hint: first you have to transform the equation from the form $f(x) = 0$ into $x = x - f(x)$).

*Newton methods*

Newton methods are the most widely used for nonlinear equations. In practice, this method amounts to solving a sequence of linear problems that converge to the nonlinear problem's solution. Given the nonlinear equation $f(x) = 0$, take the first order Taylor approximation around $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0$$

Newton methods solve the linear equation above, finding a new guess $x^{(k+1)}$ as $x^{(k+1)} = x^{(k)} - [f'(x^{(k)})]^{-1} f(x^{(k)})$, where $[f'(x^{(k)})]^{-1}$ is the Jacobian of the function $f$, and iterating over this computation until convergence. Newton's method converges if $f$ is continuously differentiable and if the initial guess is "sufficiently" close to a root of $f$ at which $f'$ is invertible. However, what "sufficiently" close means is subject to trial and error. If the method does not converge, one may need to choose a better guess, or possibly change technique. Another issue is robustness. If $f$ is well behaved, there are usually no problems in the choice of the initial guess. However, for strange functions the solution might be highly dependent on the initial guess. Moreover, if $f'$ is invertible but ill-conditioned, then the procedure can deliver a badly approximated solution. The CompEcon Toolbox includes a routine called `newton` that computes the root of a function using the Newton's method. The syntax is slightly more complicated. We need to create a function file in the form:

```
function [fval,fjac] = myfun(x, parameters)
```

where `fjac` is the Jacobian of the function. This file calculates the function and its **analytical** Jacobian. The command syntax is

```
x = newton(myfun, x0, parameters)
```

where `x0` is an initial guess.

## Exercise 4.7 Newton method

Find the roots of $e^{-x^2} - \cos(x)$ using the Newton method.

*Quasi-Newton methods*

For many problems, computing the Jacobian for implementing the Newton's method is quite difficult, or possibly computationally expensive. Quasi-Newton methods are modifications of the Newton method in which the Jacobian is computed in an efficient way. A very popular one is Broyden's method. This amounts to guessing an initial value $x^{(0)}$, and a guess for the Jacobian $A^{(0)}$, and then calculate $x^{(1)} = x^{(0)} - [A^{(0)}]^{-1} f(x^{(0)})$. Then the Jacobian is updated with the smallest possible change that satisfies the *secant condition*: $f(x^{(n+1)}) - f(x^{(n)}) = A^{(n+1)}(x^{(n+1)} - x^{(n)})$ (for speed, most of the implementation of the method update the inverse of the Jacobian). Then we iterate over the recursion $x^{(n+1)} = x^{(n)} - [A^{(n)}]^{-1} f(x^{(n)})$ until convergence. Many time the first guess $A^{(0)}$ is the numerical Jacobian at $x^{(0)}$. This method is subject to all caveats for Newton's method, plus the fact that Jacobian's guesses must be "close" to the actual values. In fact, there is no guarantee that $A^{(n)}$ will converge to the true value of the Jacobian at the solution, and it usually does not. CompEcon contains its own quasi-Newton routines, and in particular it has an implementation of the Broyden's method. The routine `broyden` has a simple syntax. We have to write a function file in the form:

```
function fval = myfun(x, parameters)
```

and then call the solver with

```
[x, fval] = broyden(myfun , x0 , parameters)
```

The material also include another implementation of the Broyden's method included in the library LIBM. This routine is called `broydn` and it is very efficient.[2] Assuming we have created a function file in the format

```
function fval = myfun(x, parameters)
```

2 This routine has been coded by Michael Reiter (IHS Wien).

we can call this solver with

```
[x, check] = broydn(myfun, x0, tol,  iadmat, iprint, parameters)
```

where `tol` is the tolerance level for convergence, `iadmat` is 1 if the library `iadmat` is available, zero otherwise, `iprint` is 1 if you want to see information about each iteration, 0 otherwise. The output `check` is 0 if convergence has been achieved, 1 if there were problems.

## Exercise 4.8 Broyden method

1. Find the roots of $e^{-x^2} - \cos(x)$ using the Broyden's method. Play with the initial guess to see how the solution changes.
2. Imagine you have a two-periods economy with an initial amount of savings $s_0$. Therefore, the equations that describe the intertemporal decision of the agent are given by:

$$c_1^{-\sigma} = \beta(1+r)c_2^{-\sigma}$$
$$c_1 + \frac{c_2}{1+r} = y_1 + \frac{y_2}{1+r} + s_0$$

   where $r = 0.05$, $\sigma = 2$, $\beta = .99$, and $y_1 = y_2 = 1$. Solve for the optimal allocation for different values of the initial savings. (Hint: write a function that takes as second input a vector of initial savings, and solve these equations in a vectorized way.

A WORD ON `fsolve` The function `fsolve` is included in MATLAB and it is a complicated routine that uses combinations of the methods described above and more sophisticated algorithms. It is quite general in his applicability, but most of the time it could be an overkill. Moreover, since it is quite sophisticated, it may be slow for some problems.

### 4.1.3 Optimization

The most intensive computational task of the VFI is the maximization step. In our code we keep it as simple as possible given that we only have to choose over one decision variable, and hence we can easily use the discretized state space for our purpose. However, this task might be more involved and sometimes troublesome if we have more than one decision variable. We can still discretize all the possible choices, however this procedure becomes more and

more burdensome and memory intensive the larger the number of choice variables we have. Therefore, we have to rely on other numerical optimization techniques. In some cases, it is possible to use a first order condition approach and therefore we can treat the maximization step as finding the solution of a system on non-linear equations, using one of the techniques illustrated above. However many complicated problems are non-convex by assumption, and we might need to rely on global numerical optimization techniques, like line search, pattern search or more sophisticated procedures as genetic algorithms, simulated annealing and swarm particle search. Many times it is possible to use more sophisticated techniques just to get a good initial guess for the value function (and policy function) and then simpler (faster) methods can deliver a solution which is accurate enough. There is always a trade-off between accuracy and speed that must be taken into consideration.

*Derivative-free methods*

These methods are very similar in spirit to bisection for non-linear equations. They in fact look for the highest point in a series of smaller and smaller intervals. The most widely used is golden search:

---

**Golden search algorithm**

1. Start with two points $x_1, x_2$ in the interval, such that $x_1 < x_2$, and evaluate $f(x_i)$
2. If $f(x_1) \geq f(x_2)$, then the new interval is $[a, x_2]$, otherwise is $[x_1, b]$
3. Iterate until convergence

---

CompEcon contains the routine `golden` that implements golden search for univariate functions. The syntax is

```
x = golden(myfun, a, b)
```

However, since we many times want to work pointwise with functions from $R^n$ to $R^n$, the library LIBM contains the routine `goldsvec` which can be used in a vectorized problem. This routine works in the same way as `golden`:

```
x = goldsvec(f,a,b)
```

## Exercise 4.9 Golden Search

1. Use the `golden` command to mazimize the MATLAB function `humps` on the interval $[-10, 10]$
2. Use the `golden` command to mazimize the MATLAB function `humps` on the interval $0.2, 2]$. Comment.
3. *(Difficult)* Imagine you have a two-periods economy with an initial amount of savings $s_0$, per-period CRRA utility function $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$, and discount factor $\beta$. The intertemporal budget constraint of the agent is:

$$c_1 + \frac{c_2}{1+r} = y_1 + \frac{y_2}{1+r} + s_0$$

where $r = 0.05$, $\sigma = 2$, $\beta = .99$, and $y_1 = y_2 = 1$. Solve for the optimal allocation for different values of the initial savings using the `goldsvec` routine.

*Newton-Raphson method*

These routines are similar to Newton's methods for nonlinear equations. The idea is to start from the second order Taylor expansion of the maximand:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T f''(x^{(k)})(x - x^{(k)})$$

From here we can take the first order conditions:

$$f'(x^{(k)}) + f''(x^{(k)})(x - x^{(k)}) = 0$$

Hence a solution can be computed by iterating over

$$x^{(n+1)} = x^{(n)} - [f''(x^{(n)})]^{-1} f'(x^{(n)})$$

until convergence. This algorithm converges if $f$ is twice continuously differentiable and if the initial guess is close to a local maximum of $f$ at which the Hessian $f''$ is negative definite. Again, a good guess is crucial for this method, and ill-conditioning of the Hessian might deliver bad approximation. Analogously to non-linear equations techniques, there are of course Quasi-Newton methods that use a computable Hessian. CompEcon includes the routine `qnewton` for this purpose. This routine assumes that the function file is written in the form

```
1 function [fval, fjac]=myfun(x, parameters)
```

and then the method can be called with

```
[x, A]=qnewton(myfun, x0, A0, parameters)
```

where `x0` and `A0` are initial guesses for respectively the maximizer and the Hessian of the function.

Many of the routines in the MATLAB Optimization Toolbox (e.g. those in `fmincon`) also are from this family, although more sophisticated.

*"I-am-in-deep-shit" problems*

Once in a while, you will face a very irregular optimization problem. Either the problem is highly non-convex, or it features several local maxima, or it is in general highly irregular. In this cases, MATLAB has a Global Optimization Toolbox which contains global optimization routines. There is also a lot of software available online (either free or commercial) which can be used for these problems. A non-exhaustive list of possible algorithms is the following:

- Line search
- Pattern search
- Genetic algorithms
- Simulated annealing
- Swarm search optimization

Most of the time, you need to use one of these methodologies just to find an initial good guess, and then you can revert to quasi-Newton methods.

*Parallelization, curse of dimensionality and interpolation*

A general point is that both global and quasi-Newton techniques are usually non-vectorized, at least in their basic MATLAB version. Hence, we have to perform one maximization for each gridpoint, which is clearly highly inefficient. However VFI is an embarrassingly parallelizable algorithm, i.e. the maximization step can be performed gridpoint by gridpoint, hence each of those maximization problems can be send to a different processor. In problem with high-dimensional state spaces this helps in speeding up the convergence.

Moreover, VFI relies on setting up a grid for the state variables. This grid needs to have many points to get a good approximation, but in a unidimensional problems this is not a big issue. The problem can arise when we have more

than one state variable. Say $N$ is the number of state variables, and you want to discretize each state in $m$ grid points. Therefore, your value function needs to be evaluated in $m^N$ points. This is a big problem for a computer, especially it is a memory-intensive task and you might easily run out of memory. This feature of the VFI algorithm is called the **curse of dimensionality**. For this reasons, economists have developed several methods that can help in dealing with dimensionality issues. One way to reduce the computational burden is to use interpolation of the value function and the choice variables between grid points. We will talk about interpolation and function approximation in the next section when discussing projection methods. Another way is a smart choice of gridpoints, putting more in regions where the problem is highly non-linear, and less elsewhere where linearity is a good approximation[3].

Finally, combining VFI with other procedures may help in speeding up the convergence. Projections methods are widely used in economics, and they can be easily applied to Bellman equations. Projections methods are a general way to solve functional equations (i.e. equations where the unknown is a function), and in principle they can be applied also to the non-linear system of functional equations that is the set of first order conditions. They typically need substantially fewer grid points to get the same approximation (for example, in our stochastic growth model you would get about the same accuracy with basic VFI over 1000 gridpoints and collocation over 10 gridpoints). In models with many state variables this property makes a big difference on computational speed.

## 4.2 PROJECTION METHODS

Projection methods are widely used in engineering, physics and mathematics to solve ordinary or partial differential equations and other functional equations. Their application to economics is therefore straightforward, since economists mostly have to solve functional equations too. In this section, we introduce the technique, and we solve the basic stochastic growth model as an example.

3 See for example the recent work on Smolyak algorithm by Malin et al. (2011), or the work on ergodic sets by Judd et al. (2012) and Maliar et al. (2011).

### 4.2.1 Solving first-order conditions

In many cases we don't need to look at the Bellman equation to find the solution of a model, since under some regularity conditions we are able to get the solution from first order conditions. In particular, we can generally do this if the value function is differentiable. Let us write our deterministic dynamic optimization problem in a more general way

$$\begin{aligned} \max_{\{u_t\}_{t=0}^{\infty}} \quad & \sum_{t=0}^{\infty} \beta^t r\left(x_t, u_t\right) \\ s.t. \quad & x_{t+1} = h\left(x_t, u_t\right), \quad t = 0, 1, ... \\ & x_0 \in X \text{ given} \end{aligned}$$

where $r$ is a return function, $x$ is a vector of states, $u$ is a vector of controls, and $h$ describes the law of motion for the state vector. Notice that, when we know the optimal policy $g(x)$, we can always write by definition:

$$V\left(x\right) = r\left(x, g\left(x\right)\right) + \beta V\left[h\left(x, g\left(x\right)\right)\right]$$

Therefore, assuming differentiability of $V$, we can get the following first-order conditions:

$$/u: \quad r_u\left(x, g\left(x\right)\right) + \beta V'\left(h\left(x, g\left(x\right)\right)\right) h_u\left(x, g\left(x\right)\right) = 0 \tag{4.3}$$

Moreover, we can get the envelope condition:

$$\begin{aligned} /x \quad : \quad & V'\left(x\right) = r_x\left(x, g\left(x\right)\right) + r_u\left(x, g\left(x\right)\right) g'\left(x\right) + \\ & + \beta V'\left[h\left(x, g\left(x\right)\right)\right] \cdot \left\{h_x\left(x, g\left(x\right)\right) + h_u\left(x, g\left(x\right)\right) g'\left(x\right)\right\} \end{aligned} \tag{4.4}$$

If we can write the law of motion of the state variables only as a function of the controls (which happens in many economic models), i.e. $x' = h\left(u\right)$, then using (4.3) in (4.4) we get

$$V'\left(x\right) = r_x\left(x, g\left(x\right)\right) \tag{4.5}$$

Equation (4.5) is a version of the Benveniste and Scheinkman (1979) formula. Of course we need to make some assumptions on the primitives of the problem to be able to get a differentiable value function. It is possible to show that, the following assumption, together with Assumptions 5, 6, 9 and 10, guarantee differentiability of the value function in the interior of its domain $A$:

**Assumption 11.** *The return function $r$ is continuously differentiable with respect to the first argument in the interior of $A$.*

Now, if we can write the law of motion of the state variables as $x' = h(u)$, I can use equations (4.3) and (4.5) to get the following expression:

$$r_u(x_t, u_t) + \beta r_x(x_{t+1}, u_{t+1}) h'(u_t) = 0, \quad x_{t+1} = h(u_t) \tag{4.6}$$

Equation (4.6) is called **Euler equation**. We can solve for the optimal policy function by using the Euler equation. Notice that the same kind of analysis can be done for the stochastic case, with the obvious adaptations.

### 4.2.2 The basics of projection methods

To solve Euler equations, and in general functional equations related to economic models, we can use projection methods. Their application is nowadays widespread in macroeconomics. Judd (1998) has a very good treatment of the method, and the interested reader should look at this book for an exhaustive analysis of the technique and its applications. The paper that introduced the technique to economists is Judd (1992), and it reports many examples that can be used as exercises. Miranda and Fackler (2004) illustrate the approach and describe in details the CompEcon tools that help working with these techniques.

The basic idea of projection methods is very simple. We want to solve a functional equation of the type:

$$\mathcal{N}(g(x)) = 0 \tag{4.7}$$

where $\mathcal{N}$ is an operator (for example, the Bellman operator, or the FOCs operator) and $g(x)$ is a function of some space $X$. If we solve this functional equation numerically, then we can get an approximation of our solution $\widehat{g}(x)$.

Judd (1992) suggests to follow a sequence of steps. The **first step** is to choose how we approximate the solution, and an appropriate concept of distance in order to measure the accuracy of our calculated solution. The easiest way to proceed is to assume that our function can be approximated as a linear combination of some simple functions (we call them *basis functions*, or simply basis), like polynomials for example.

The **second step** is to choose a *degree of approximation* $n$, i.e. how many basis functions we want to use. We also need to choose a computable approximation $\widehat{\mathcal{N}}$ for $\mathcal{N}$, if the exact operator is not computable directly. For example, if the

operator is an integral, we need to use quadrature methods to compute an approximated version of the operator. We also must select some functions $p_i, i = 1, \ldots, n$ that we will use later to calculate the projections. We have now our approximated guess/solution $\widehat{g}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$ for any $x$. Therefore, any solution can be summarized by a vector of coefficients $\mathbf{a}$.

In the **third step**, we numerically compute the approximated policy function $\widehat{g}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$ for a particular guess of $\mathbf{a}$, and we also compute the so called *residual function*

$$R(x; \mathbf{a}) \equiv \left(\widehat{\mathcal{N}}\left(\widehat{g}\right)\right)(x)$$

The first guess for $\mathbf{a}$ can be important. Depending on the application, it is crucial to start with a good guess to achieve convergence.

The **fourth step** is the calculation of the projections

$$P_i(\cdot) \equiv \langle R(\cdot; \mathbf{a}), p_i(\cdot) \rangle, i = 1, \ldots, n \tag{4.8}$$

where $\langle \cdot \rangle$ is the inner product.

Finally, in the **fifth step**, we iterate over steps 3 and 4 to get a vector of coefficients $\mathbf{a}$ that sets the projections (4.8) to zero.

### *Choice of basis functions*

There are few details to keep in mind. First, the typical choice for the inner product used in the calculation of the projections is, given a weighting function $w(x)$:

$$\langle f(x), h(x) \rangle \equiv \int f(x) h(x) w(x) dx$$

Second, there are various possible choices for basis functions. We may use ordinary polynomials $1, x, x^2, x^3, \ldots$. However, it turns out that they are problematic in many situations. In general, we can divide the type of basis we choose in two broad categories: spectral methods and finite element methods. Spectral methods have basis functions that are almost everywhere nonzero, and are continuously differentiable as many time as needed, therefore imposing smoothness on the approximated function (which sometimes is not a desirable feature). A very popular choice in spectral methods is Chebychev polynomials. They are defined as

$$T_n(x) \equiv \cos(n \arccos x), \quad x \in [-1, 1]$$

and it is possible to generate them with the following recursive law:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$
$$T_0(x) = 1, \quad T_1(x) = x$$

It is possible to show that those polynomials satisfy an orthogonality condition for a particular inner product:

$$\int_{-1}^{1} T_i(x) T_j(x) \left(1 - x^2\right)^{-\frac{1}{2}} dx = 0, \quad i \neq j$$

Now, let $z_l^n \equiv \cos\left(\frac{(2l-1)\pi}{2n}\right), l = 1, \ldots, n$ be the zeroes of $T_n$. We can show that

$$\Sigma_{l=1}^{n} T_i\left(z_l^n\right) T_j\left(z_l^n\right) = 0, \quad i \neq j$$

therefore we want to use the roots of the Chebychev polynomials for interpolation. Moreover, there are theorems that show that, using zeroes as evaluation points, convergence is achieved faster. In general,, Chebychev polynomials are very useful if the function we want to approximate is smooth, and therefore can be used when we have the proof or the reasonable suspicion that our solutions are differentiable and regular.

Finite element techniques use instead basis functions that are zero everywhere except for a small support. A very popular choice is tent functions, aka piecewise linear basis, Take an approximation with support in $[a, b]$ and be $h = (a - b)/n$. Then, for $i = 0, 1, \ldots, n$:

$$\phi_i(x) = \begin{cases} 0 & a \leq x \leq a + (i-1)h \\ (x - (a + (i-1)h))/h & a + (i-1)h \leq x \leq a + ih \\ 1 - (x - (a + (i-1)h))/h & a + ih \leq x \leq a + (i+1)h \\ 0 & a + (i+1)h \leq x \leq b \end{cases}$$

A generalization of those bases is piecewise $k$-degree polynomials, like Hermite polynomials and cubic splines. Cubic splines are indeed very popular, and you will use them often for problems that are irregular.

As a rule of thumb, if the function we want to approximate is smooth, we use Chebychev polynomials; otherwise we can use piecewise polynomial interpolation of different orders, usually perfecting the order by trial and error, trying to use the lowest possible order that delivers a good approximation.

If our state space is multidimensional, we can use tensor products to build multidimensional approximation from unidimensional basis: if $\{\phi_i(x)\}_{i=1}^{\infty}$

is the basis for a function in one variable, we can use the tensor product basis $\{\phi_i(x)\,\phi_j(y)\}_{i,j=1}^{\infty}$ for functions of two variables, and so on. The main problem is that the number of elements increases exponentially with the dimension. There are various ways to overcome this problem: one is to use complete polynomials of order $k$:

$$\mathcal{P}_k \equiv \left\{ x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} \,\middle|\, \sum_{l=1}^{n} i_l \leq k, 0 \leq i_1, \ldots, i_n \right\}$$

CompEcon includes the most popular choices for basis functions (Chebychev polynomials, piecewise linear function and splines of any order). By default, it uses tensor product for multidimensional interpolation. However, complete polynomials are not implemented.

*Choice of projections*

There are various possibilities for the choice of which particular projections we want to use. The difference will depend generally on the functions $p_i$ that we use in the calculation of the projection. One possibility is the least-squares approach, which minimizes the weighted sum of squared residuals:

$$\min_{\mathbf{a}} \langle R(x;\mathbf{a}), R(x;\mathbf{a}) \rangle$$

The Galerkin method uses the basis functions for projections, and solves the following equations:

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), \phi_i(x) \rangle = 0, \quad i = 1, \ldots, n$$

The method of moments uses the first $n$ polynomials:

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), x^{i-1} \rangle = 0, \quad i = 1, \ldots, n$$

The subdomain method solves

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), \mathbf{I}_{D_i} \rangle = 0, \quad i = 1, \ldots, n$$

where $\{D_i\}$ is a sequence of intervals covering the entire domain of the function, and $\mathbf{I}_{D_i}$ is the indicator function for $D_i$.

A very popular method is collocation. It is popular because it is easy to implement and very fast compared to alternative projection choices (see Miranda and Fackler (2004) for an explanation of how the method is implemented with

matrix algebra). The collocation method chooses $n$ points $\{x_i\}_{i=1}^n$ in the domain and solves

$$R\left(x_i; \mathbf{a}\right) = 0, \quad i = 1, \ldots, n$$

This is equivalent to solve

$$P_i\left(\mathbf{a}\right) \equiv \left\langle R\left(x; \mathbf{a}\right), \delta\left(x - x_i\right)\right\rangle = 0, \quad i = 1, \ldots, n$$

where $\delta(\cdot)$ is the Dirac delta function that is equal to zero everywhere but in zero, where it takes value 1. Orthogonal collocation chooses collocation nodes $\{x_i\}_{i=1}^n$ as the zeroes of the basis function, making the computations even faster thanks to orthogonality conditions.

In general, the main computational burden of projection methods is the calculation of the projections. Collocation is very fast, since other methods would require the computation of an integral.

### 4.2.3 CompEcon routines for projection methods

CompEcon has a series of functions that implement projection methods in an easy and accessible way. Many of these routine are coded in C and then MEX-ified for MATLAB use, making them very fast. It is therefore a good idea to learn how to use them for day-to-day research.

GENERATING FUNCTIONAL SPACES WITH `fundefn` The function `fundefn` creates a MATLAB structured variable that characterizes the functional space of the basis functions chosen by the programmer. The syntax is:

```
fspace = fundefn(bastype,n,a,b,order);
```

The inputs set the basis functions. `bastype` is a string that identifies which type of basis function we want to use. The possible options are given by Chebichev polynomials (`'cheb'`), splines (`'spli'`), or linear spline basis with finite difference derivatives (`'lin'`). The input `n` is a vector indicating the degree of approximation along each dimension (this routines automatically uses tensor product for multidimensional interpolation). The values of `a` and `b` identify the left and right endpoints for interpolation intervals for each dimension. Finally, `order` is used to specify the spline order (and it is therefore optional in all other basis type's choice), where the default is cubic splines.

For example, imagine that we want to create a functional basis space for 5th degree Chebychev polynomials for a univariate function in the interval $[-5, 6]$. We use the command:

```
fspace = fundefn('cheb',5,-5,6);
```

If instead we are interested in generating a cubic spline space in two dimensions, with 10 basis functions in the first dimension and 8 in the second on the interval $\{(x_1, x_2) : -5 \leq x_1 \leq 6, 2 \leq x_2 \leq 9\}$, we write:

```
fspace = fundefn('spli',[10 8],[-5 2], [6 9]);
```

If we prefer a 5th order spline, then we use:

```
fspace = fundefn('spli',[10 8],[-5 2], [6 9],5);
```

COMPUTING APPROXIMATED FUNCTIONS WITH `funeval` One crucial point of the projection methods is the computation of the approximated function $\widehat{g}(x) = \sum_{i=1}^{n} c_i \phi_i(x)$, given a set of coefficients $\mathbf{c}$ and basis functions $\{\phi_i(x)\}_{i=1}^{n}$. The routine `funeval` performs this task for us. Having defined a set of points `x` over which we want to evaluate our approximated function, and a vector of coefficients `c`, we can then write

```
y = funeval(c,fspace,x);
```

The set of points `x` needs to be defined as a $m \times k$ matrix, where $m$ is the number of points over which we want to evaluate the function, and $k$ is the dimensionality of the space. For example, if we want to evaluate our approximated function over 100 points in a 3-dimensional basis space, then `x` must be a $100 \times 3$ matrix.

OTHER USEFUL ROUTINES: `funbas` AND `funnode` In our code for solving the stochastic growth model with collocation, we will make use of two more commands. The first is `funbas`, which calculates the matrix of values of the basis functions in a particular set of points `x`:

```
Basis = funbas(fspace,x);
```

This function is useful when we want to evaluate a function at the same points but with different coefficients' values (for example, because of our iterative procedure to find the coefficients vector $\mathbf{c}$). We can then use

```
Basis = funbas(fspace,x);
y = Basis*c;
```

which is equivalent to using `funeval`.

Finally, `funnode` computes standard nodes for the basis functions included in CompEcon. For example, if the basis functions are Chebychev polynomials, then the command

```
x = funnode(fspace);
```

returns the $1 \times k$ cell array of Chebychev zeros.

### 4.2.4 How to solve the stochastic growth model with collocation

In order to solve the stochastic growth model (SGM) with collocation, we need to make a few choices. First choice is, what will be our operator $\mathcal{N}$? Second choice, what function will we approximate, i.e. which one is our function $g(\cdot)$?

We will give an example working with first order conditions, hence our operator $\mathcal{N}$ will be the set of FOCS. In the example, we will use $k'$ as our function $g(k, A)$. Another possible choice is to use consumption $c$, however it does not make a big difference in this simple model. In general, the choice of which operator and which functions we approximate can make a **big** difference in terms of speed, accuracy and number of "tricks" needed to get the code converging.

A general principle is to reduce the number of equations and functions to include in $\mathcal{N}(g(x)) = 0$ to the minimum. If you can substitute some variables in some equations and reduce the number of equations/unknowns, then do it. Moreover, equations can be highly non-linear in some variables, while linear or quasi-linear in others: this is the typical case in Kuhn-Tucker conditions, where the equations are linear in the Lagrange multipliers. It is then sometimes helpful to exploit this feature.

For the SGM, the social planner problem is

$$\begin{aligned} \max_{k_{t+1}, c_t} & E_0 \sum_{t=0}^{\infty} \beta^t u(c_t) \\ & s.t. \quad c_t + k_{t+1} - (1-\delta)k_t \leq A_t k_t^{\alpha} \end{aligned}$$

Substituting the resource constraint into the objective function, we have

$$\max_{k_{t+1}} E_0 \sum_{t=0}^{\infty} \beta^t u(A_t k_t^{\alpha} - k_{t+1} + (1-\delta)k_t)$$

The first order conditions for the social planner problem are:

$$u'(c_t) = \beta E_t \left[ u'(c_{t+1}) \left( \alpha A_{t+1} k_{t+1}^{\alpha - 1} + 1 - \delta \right) \right] \tag{4.9}$$

We can calculate the steady state in the deterministic case, normalizing $A^{ss} = 1$:

$$1 = \beta \left[ \left( \alpha k^{\alpha - 1} + 1 - \delta \right) \right]$$

from which we get

$$k^{ss} = \left[ \frac{1}{\alpha \beta} - \frac{1 - \delta}{\alpha} \right]^{\frac{1}{\alpha - 1}}$$

We will create a grid around the steady state capital $k^{ss}$. However, let's focus for one moment on equation (4.9). We can rewrite it highlighting the dependencies on the state variables as

$$u'(c(k, A)) = \beta E \left[ u'(c(k', A')) \left( \alpha A'(k')^{\alpha - 1} + 1 - \delta \right) \right] \tag{4.10}$$

and given our decision to approximate next period's capital, we can rewrite it by using the function $g(\cdot)$ as

$$\begin{aligned} u'(\max(0, Ak^{\alpha} + (1 - \delta)k)) = \beta E \big[ & u'(\max(0, A'(g(k, A))^{\alpha} + \\ & (1 - \delta) g(k, A))) \left( \alpha A'(g(k, A))^{\alpha - 1} + 1 - \delta \right) \big] \end{aligned} \tag{4.11}$$

Equation (4.11) is our operator $\mathcal{N}(g(x))$, where $x = (k, A)$ is a point in the state space. We use Gaussian quadrature to calculate the expectations. We use Chebychev polynomials as basis functions, and we implement orthogonal collocation by choosing grid point to be the zeroes of the basis functions.

### 4.2.5 The code

The code is composed of several files. The main file to run is `solveSGM.m`, which is reported in Listing 4.1. The general scheme of the code is the following:

1. Set parameters values and grid for states
2. Generates the functional space for the basis functions, and a "good" initial guess for the coefficients
3. Find coefficients that put the residual function as close to zero as possible (this is in fact done by solving nonlinear equations with Broyden's method)
4. Test solution accuracy and then simulate the model

This file calls two other files: `mainSGM.m` contains the collocation algorithm, and calls the file in which we stored the residual function `focsSGM.m`. In the following, we illustrate each file in details.

*`solveSGM.m`*

This file assigns values to parameters, then creates lower and upper bounds for the capital grid around the deterministic steady state. It then sets the parameters for calculating expectations with quadrature techniques, using the function `qnwnorm` from CompEcon, and fixes the range for shocks by setting lower and upper bound for the productivity level $A$.

Then we choose the parameters for the collocation algorithm. First we choose how may rounds of approximation we want to do. This is a common practice when solving models with projection methods. Typically, it is quite easy to find a solution on a coarse grid with a low degree of accuracy. However, it is more complicated to get a good accurate solution when working with finer grids. Therefore the practice is to first run collocation over a coarse grid, find a rough solution and then use this solution as an initial guess over a finer grid. It may be necessary to do two or more rounds of approximation to get accurate solutions if the problem is highly non-linear or presents irregularity of sort.

The next line sets the order of the approximation for each round. In this example, we use order 5 in both dimensions (capital and productivity shock) for the first round, and 10 for the second round.

The following section sets the type of basis function we want to use (notice that the code has commented out all the possible options, if the reader wants to experiment with them and compare results). Then the code calls the file `mainSGM.m`, which contains the collocation algorithm. The last part of the code is then devoted to simulations (calling the file `simul_SGM.m`) and figures.

**Listing 4.1:** solveSGM.m

```
1 global alpha betta rho sig sigma delta sigeps ;
2 global nQuadr QuadrWeights QuadrPoints;
3 global  RoundAppr rounds_approx  ;
4
5 %% PARAMETERS:
6 alpha =.4; % production function coefficient
7 delta = .1; % depreciation rate for capital
8 betta =  .95;  % discount factor
```

**Listing 4.1 (continued):** pfi_AM.m

```
sig = 1; % CRRA utility parameter (c^(1-sig))/(1-sig)
sigma = .05;  % S.D. of productivity shock
rho = 0; % persistence of productivity shock

%% create a grid for capital
kstar = (1/(alpha*betta) - ...
  (1-delta)/alpha )^(1/(alpha-1)); % det. steady state
k_min = .5*kstar;
k_max = 2*kstar;

%% parameters for quadrature
nQuadr = 50; %number of quadrature points;
% we choose nQuadr high to get smoothness;
[QuadrPoints,QuadrWeights] = qnwnorm(nQuadr,0,sigma^2);

%% Range for shock
sigeps = sigma/sqrt(1-rho^2);
% Range for shock:
A_max = 3*sigeps;
A_min =  -3*sigeps;

%% Parameters for the collocation algorithm
rounds_approx = 2; % number of rounds of approximation
Order_vector = [[5;5] [10; 10]];% n. of grid points for each
    round
ntest = 100; % n. of grid points for testing (for each dimension)

%% Approximation type for CompEcon
% approxtype = 'lin';  % piecewise linear
approxtype = 'cheb'; % chebychev polynomials
% approxtype = 'spli'; % splines
splineorder = []; % splines' order, default are cubic splines

%% parameters for simulations
number_series =1;    % number of series
```

**Listing 4.1 (continued):** pfi_AM.m

```
periods_simulation =  100; % number of periods for the simulation
k0 = k_min.*ones(number_series,1);

%% Run main file
mainSGM; % solves the model

% deliver an accuracy statistic (the smaller the better)
max_test

% save the solution
save solutionSGM.mat park fspace ;

%% SIMULATIONS AND FIGURES

% simulate the series
[A, k,  c, y ,epsilon] = simul_SGM(number_series, ...
  periods_simulation, k0, park, fspace);

figure(1);
subplot(2,1,1);
plot(1:periods_simulation, k);
xlabel('time'); ylabel('k'); axis([1 periods_simulation 0 7]);
title('Simulation: capital'); legend('k_t');

subplot(2,1,2);
plot(1:periods_simulation-1,c(2:end));
xlabel('t'); ylabel('c');
title('Simulation: consumption'); legend('c_t');
```

*mainSGM.m*

The file `mainSGM.m` executes the collocation algorithm. First, it generates the functional space for the basis functions with `fundefn` from CompEcon, by using parameters defined in code 4.1. Then generates a grid in the state space $(k, A)$ by choosing standard interpolation nodes related to the basis' type choice, with

`funnode`. We use the function `gridmake` which stacks gridpoints in a $n_k n_A \times 2$ matrix, where $n_k$ and $n_A$ are the number of gridpoints in each dimension. In order to set initial conditions for the basis' coefficients, we guess a solution and then regress this solution on the basis functions (defined using `funbas`). In the first round, we just guess that capital tomorrow is equal to capital today, while in the second round we use the solution found on a coarse grid/approximation as a guess. Finally, we find basis' coefficients `park` such that the residual function is set to zero, by solving the set of nonlinear first order conditions (set in the function file `focs.SGM` described below) with the Broyden's method. Notice that there are as many equations as gridpoints, hence we solve in the first round a system of 25 equations, while in the second the system has 100 equations. This method is well suited for up to 500 equations, but then it may cause some ill-conditioning. In those cases, it is better to use a combination of nonlinear solvers and fixed-point methods[4].

**Listing 4.2:** mainSGM.m

```
for oo = 1: rounds_approx
  RoundAppr = oo;     % this tells you at which round of
      approximation we are

  % Range on which we approximate the solution:
  LowerBound = [k_min A_min ];
  UpperBound = [k_max A_max];

  % Approximation order
  Order = Order_vector(:,oo);

  % for reference, need this for guess after round 1
  if oo >= 2
    fspace_old = fspace;
  end

  disp(' '); disp(' ' );
  disp(sprintf('RoundAppr %d, # gridpoints = [%d %d]',...
    RoundAppr, Order(1), Order(2)));
  disp(' ');
```

4 See Judd (1998) and Miranda and Fackler (2004) for a discussion.

**Listing 4.2 (continued):** pfi_AM.m

```
20
21  % the following lines generate basis function space
22  % we can choose among chebychev polynomials, splines of
        different
23  % orders and piecewise linear functions
24
25  if(strcmp(approxtype,'spli'))
26    fspace = fundefn(approxtype,Order,LowerBound,UpperBound,
          splineorder);
27  else
28    fspace = fundefn(approxtype,Order,LowerBound,UpperBound,[]);
29  end;
30
31  % the following commands create gridpoints
32  nodes = funnode(fspace);
33  Grid = gridmake(nodes);
34
35  % Set initial conditions
36  if (RoundAppr == 1)
37    knext = Grid(:,1);
38  else    % if we are at second approx round, we use the solution
        of the first round
39    % as initial conditions on the new larger grid
40    knext = funeval(park, fspace_old, Grid);
41  end;
42
43  % generate basis functions Basis at Grid :
44  Basis = funbas(fspace,Grid);
45
46  % set initial value for parameters of the approximation
47  park =  Basis\knext;
48
49  % solve FOCs with Broyden method for nonlinear equations
50  [park,info] =  broydn('focsSGM',park,1e-8,0,1,Grid,fspace);
```

**Listing 4.2 (continued):** pfi_AM.m

```
51   disp(sprintf(' info = %d',info)); % if info=0, everything went
        fine, o/w the Broyden algorithm didn't converge
52   disp(sprintf('    '));
53
54 end;
```

*focsSGM.m*

The function `focsSGM.m` contains the residuals to be put to zero by the non-linear solver. Expectations are calculated using quadrature parameters set in `solveSGM.m`. The equation to be solved is scaled such that it is unit free, i.e. the value is in fact in percentage. Hence if the algorithm gets say $10^{-8}$, the error is in the order of $10^{-8}\%$ of the marginal utility of consumption.

**Listing 4.3:** focsSGM.m

```
1  %% FOCS for the stochastic growth model
2
3  function equ = focsSGM(park, Grid,fspace);
4
5  global alpha betta sig rho delta A_bar
6  % global LowerBound UpperBound
7  global nQuadr QuadrWeights QuadrPoints
8
9  LowerBound = fspace.a;
10 UpperBound = fspace.b;
11
12 %rename grid
13 k = Grid(:,1);
14 A = Grid(:,2);
15
16 % evaluate policy functions
17 knext = funeval(park,fspace, Grid);
18 fofk = exp(A).*(k.^alpha) + (1-delta).*k;
19 % c = fofk - knext;
```

**Listing 4.3 (continued):** pfi_AM.m

```
c = max(fofk - knext, zeros(length(Grid),1));

n = length(k);
% generate nQuadr replications of the Grid, one for each
   realization of shock:
Grid_knext = kron(knext,ones(nQuadr,1));

% Expected value of next period A, corresponding to Grid:
ExpA =  rho*A;
% all realizations of next A:
GridANext = kron(ExpA,ones(nQuadr,1)) + ...
  kron(ones(n,1),QuadrPoints);
% truncate it to state space:
GridANext = min(max(GridANext,LowerBound(2)),UpperBound(2));

GridNext = [Grid_knext GridANext];

% calculate variables at t+1
knextnext = funeval(park,fspace, GridNext);
fofknext = exp(GridANext).*(Grid_knext.^alpha) + (1-delta).*
   Grid_knext;
 cnext = max(fofknext - knextnext, zeros(length(Grid_knext),1));
% cnext = fofknext - knextnext;
mucnext = muc(cnext);
mpknext = mpk(GridNext);
% calculate expectations with quadrature
exp_mucnext = (QuadrWeights'*reshape(mpknext.*mucnext,nQuadr,n))
   ';

% equation to be solved: Euler equation
equ = (muc(c) - betta.*exp_mucnext)./muc(c);

% avoid strange solutions
if (any(cnext<0))  || (any(c<0)) || (any(knext<0))  || (any(
   knextnext<0))
```

**Listing 4.3 (continued):** pfi_AM.m

```
51    equ(1) = 1e100;
52  end;
```

### 4.2.6 Tricks and difficulties

For the purpose of illustrating the methodology, the simple stochastic growth model is helpful. However, being so simple, it does not involve any major difficulty that is usually associated with solving models with projection methods. In the following we illustrate a few of them.

*Convergence*

One problem with projection methods is that there is not a theorem guaranteeing convergence of the procedure. Hence, many times the researcher has to "guide" the convergence process with ad-hoc solutions. One widely used method is *homotopy*, i.e. start from the solution of a simpler model and iteratively modify the $\mathcal{N}$ operator to match the new, more complicated model. Another option is using bounds for the first few iterations, once we are sure we have a good guess.

*Dependence on the initial guess*

Many times the solution we obtain depends on the initial guess, since we often rely on local methods for solving the system of functional equations. Hence, it is important to get a good guess. Moreover, it is crucial to check that we always converge to the same solution even if we start from different initial guesses, keeping in mind what said above about convergence issues.

*Ill-conditioning*

In most cases, the solution of a model with projection methods will make use of non-linear solvers, and consequently either Jacobian or Hessian matrices will have to be computed. Ill-conditioning is a very frequent problem in highly non-linear models, and there are no easy fixes. Sometimes, a smarter choice of the gridpoints helps. Increasing the number of approximation rounds also could make the problem less pronounced.

*Speed considerations and models with large state spaces*

In high-dimensional models, speed may be a concern. However, recent work has shown that projection methods can be used with large state spaces efficiently, with an appropriate choice of the gridpoints. One option is to use the Smoliak algorithm for choosing a sparse grid (see Malin et al. (2011)). Another recent approach is to use the ergodic set of the model and pick gridpoints in it (see Judd et al. (2012)).

### Exercise 4.10 (difficult) Collocation for VFI

We have solved the model under the assumption that first order conditions are necessary and sufficient. There are cases in which this is not possible, therefore one would then think that the only option is to use other dynamic programming techniques. In fact, VFI and PFI can be combined with collocation. We ask you to do it with the RBC model: create a code that solves the Belmann equation with collocation.

## 4.A APPENDIX: INSTALLING THE LIBRARIES

### 4.A.1 LIBM installation

The library LIBM includes a few routines which will be used in the collocation algorithm. You just need to add it to the MATLAB path.

### 4.A.2 CompEcon installation

CompEcon Toolbox is a free set of routines for economists which accompanies Miranda and Fackler (2004), you can download it from the website `http://www4.ncsu.edu/~pfackler/compecon/download.html`. We provide you with the most recent version, including also compiled MEX files. You should just add it to your MATLAB path, and it should work.

However, sometimes using a version of the MEX files installed in another machine does not work. Installation requires a C compiler, since many routines are coded in C and you will need to transform them in MEX files in order to be able to use them with MATLAB. We suggest you use Visual Studio, however any C compiler would do. There are a few freely available, for example MinGW.

First, you download and add the folder to your MATLAB path, making sure to use the option "Add with subfolders". Then, you run the file `mexall`, by just typing it in the MATLAB command window. The file will ask for a C compiler, and it will compile the C files into MEX files. In order to check if your installation worked, please run the file `solveSGM.m` in the collocation folder. Notice that this file requires the library LIBM, therefore the latter must be already installed. If you receive any error message, or have any problem with the installation, please write an email to meleantonio@gmail.com or open a new issue on Github.

# 5 TIME INCONSISTENCY: APS APPROACH

## 5.1 INTRODUCTION

The APS approach is an extension of dynamic programming for non-recursive problems, which makes use of auxiliary costate variables to "recursify" the problem. It is sometimes called "dynamic programming squared". The main idea is the following. Take for example the case of limited commitment. In that situation, the value function will depend on the previous history of the endowment shocks. However, a history will induce a particular continuation value for the agent. APS and subsequent research showed that, in most problems, the continuation value is a sufficient statistic for the history, i.e. knowing the continuation value is equivalent to knowing the previous history of the game. Therefore, the agent's continuation value has the same role of a state variable: it perfectly describes the past information. We can hence add the continuation value of the agent as a costate variable, i.e. another fictitious state variable that make our problem recursive. Another way to think about it is that the continuation value summarizes the promises that a fully committed principal made to the agent in the past. Choosing a future continuation value means choosing a promise to the agent. However, this also means that we need to restrict the set in which we look for new optimal continuation values: they must be consistent with promises made in the past. This latter point is the tricky bit of the APS methodology. In simple terms, we need to characterize the set of feasible continuation values (i.e. consistent with the original problem) before solving for the optimal allocation. The main contribution of APS is to show how to characterize this set by iterating on a set-valued operator, i.e. an operator that maps sets into sets. As the reader can imagine, this procedure is quite complicated even for models without state variables. In presence of state variables, we need to characterize this set for each point in the state space. In other words, we need to characterize a correspondence. There are a few numerical techniques to do it, however they are computationally burdensome and the difficulty increases with the dimensionality. For this reason, the literature features the analysis of models with up to 2 states.

We first give a general version of the APS approach, that involves a formulation in the jargon of game theory, and then we show how to use it in the various examples listed in the previous section. We conclude with some numerical issues.

The general formulation is in terms of a repeated game with imperfect monitoring with $N$ players, and characterizes the set of sequential equilibria in pure strategy. However, the main idea can be explained in a game with perfect information, in terms of subgame perfect equilibria. Imagine a game with $N$ players, that choose an action $a_i$ in each period. Their per-period payoff is given by $u_i(a)$, where $a \equiv \{a_1, \ldots, a_N\}$ is the profile of actions for all players. Each player observes a signal $h_{i,t}$ in each period. History is indicated by $h_i^t \equiv \{h_{i,1}, \ldots, h_{i,t-1}\}$. Since we assume perfect information, the signal includes all actions and realization of shocks for each player, therefore the histories are the same for all players and we will indicate them with $h^t$. The strategy profile is indicated by $\sigma_i$ and it is a mapping from the set of possible histories to the set of actions. Any strategy profile $\sigma \equiv (\sigma_1, \ldots, \sigma_N)$ generates a sequence of action profiles. We can define the discounted payoff as

$$v_i(\sigma) = \sum_{t=0}^{\infty} \beta^t u_i\left(a^t\right)$$

Also define the profile of continuation strategies for the subgame starting at $h^t$ as $\sigma|_{h^t}$, and consequently

$$v_i\left(\sigma|_{h^t}\right) = \sum_{j=t}^{\infty} \beta^{j-t} u_i\left(a^j\right)$$

We can therefore give the following

**Definition 2.** *$\sigma^*$ is a subgame perfect equilibrium if, for any $h^t$, $\sigma|_{h^t}$ is a Nash equilibrium of the subgame starting at $h^t$.*

Define the set of all subgame perfect equilibria as $V$. Denote by $v(a)$ the profile of continuation values that are induced by a subgame perfect equilibrium when the first period profile of actions is $a$.

**Definition 3.** *Let $W \subset \mathbb{R}^{\mathbb{N}}$. Then $(a, v)$ is admissible with respect to $W$ if $a \in A$, $v \in W$ and*

$$u_i(a) + \beta v_i(a) \geq u_i(\widehat{a}_i, a_{-i}) + \beta v_i(\widehat{a}_i, a_{-i}) \quad \forall \widehat{a}_i \quad \forall i$$

Let $B(W)$ be the set of all values of admissible pairs for $W$. It is possible to prove the following Lemma:

**Lemma 1.** *The operator $B$ is monotone and preserve compactness:*

1. *if $W \subseteq W' \subseteq \mathbb{R}^{\mathbb{N}}$, then $B(W) \subseteq B(W')$*
2. *if $W \subseteq \mathbb{R}^{\mathbb{N}}$ is compact, then $B(W)$ is compact.*

We say that $W \subseteq \mathbb{R}^{\mathbb{N}}$ is **self-generating** if $W \subseteq B(W)$. Given this definition, APS (1990) prove the following crucial result:

**Theorem 6.** ***(Self-generation)*** *If $W \subseteq \mathbb{R}^{\mathbb{N}}$ is bounded and self-generating, then $B(W) \subseteq V$.*

The following theorem establish that we can solve any repeated game with the help of the operator $B$ which is the set-valued equivalent of a Bellman operator for functions.

**Theorem 7.** ***(Factorization)*** *$V = B(V)$*

The set of equilibrium values is a fixed point of the set-valued operator $B$. The main point now is how to calculate this set. The previous theorem and the monotonicity of the operator give us a hint. APS prove the following

**Theorem 8.** *Let $W_0 \subseteq \mathbb{R}^{\mathbb{N}}$ be compact and such that $V \subseteq B(W_0) \subseteq W_0$. For $n = 1, 2, \ldots$, let $W_n = B(W_{n-1})$. Then $\{W_n\}$ is a decreasing sequence and $V = \lim_{n\to\infty} W_n$.*

We can follow this procedure:

1. Start with a set $W_0 \subseteq \mathbb{R}^{\mathbb{N}}$ such that $V \subseteq B(W_0) \subseteq W_0$.
2. For any $n = 1, 2, \ldots$, calculate $W_n = B(W_{n-1})$
3. Iterate until convergence.

This algorithm delivers the set of all values of subgame perfect equilibria associated with the game. We are typically interested in the efficient ones. Therefore, the standard application of APS has two stages: first, use the previous algorithm to get the set of continuation values that are subgame perfect, and then look for the efficient ones in that set.

Theorem 6 and 7 can be applied to any set. Therefore, we can generate the set of all payoffs that satisfy certain constraints by applying Theorem 8, starting from a large enough initial set. Therefore, the extension e.g. to games with private information or to optimal policy problems is straightforward.

## 5.2 CHARACTERIZATION OF THE FEASIBLE SET

Even if we now have Theorem 8, for most problems there is no closed form characterization of it. Therefore we need to characterize the set with some numerical approximation. The following is based on Judd et al. (2003) (CJY), which have then produced more general approaches along the same lines.

This approach can be better explained with infinitely repeated games. The actions of player $i$ in the stage game are $A_i$, $i = 1, \ldots, N$. The set $A \equiv A_1 \times \ldots \times A_N$ contains all possible players' actions. Player $i$'s payoff in the stage game will be $\Pi_i : A_i \to \mathbf{R}$. Player $i$ responds optimally to other players' actions, getting a payoff given by

$$\Pi_i^*(a_{-i}) \equiv \max_{a_i \in a_i} \Pi_i(a_i, a_{-i})$$

Define $B(W)$ as

$$B(W) = \bigcup_{(a,w) \in A \times W} \left\{ (1-\delta)\Pi(a) + \delta w \mid \forall i (IR_i \geq 0) \right\} \tag{5.1}$$

where

$$IR_i \equiv [(1-\delta)\Pi(a) + \delta w_i] - [(1-\delta)\Pi^*(a_{-i}) + \delta \underline{w}_i]$$

is the individual rationality constraint for player $i$, and

$$\underline{w}_i \equiv \inf_{w \in W} w_i$$

is player $i$'s worst possible continuation value in $W$ .

The idea in CJY is pretty simple: iteratively find outer and inner approximations of the set $V$ that "sandwich" it. These approximation are parsimoniously representing the set as a convex polytope, in one of two possible ways: 1) the intersection of a finite number of half-spaces:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\leq b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\leq b_m \end{aligned}$$

or, more compactly,

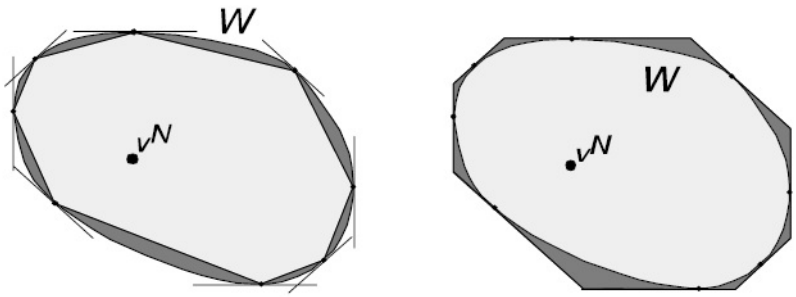
$$Ax \leq b$$

, or 2) the convex hull of a set of vertices $Z$.

CJY characterize two different convex polytope approximations of convex sets $W$ :

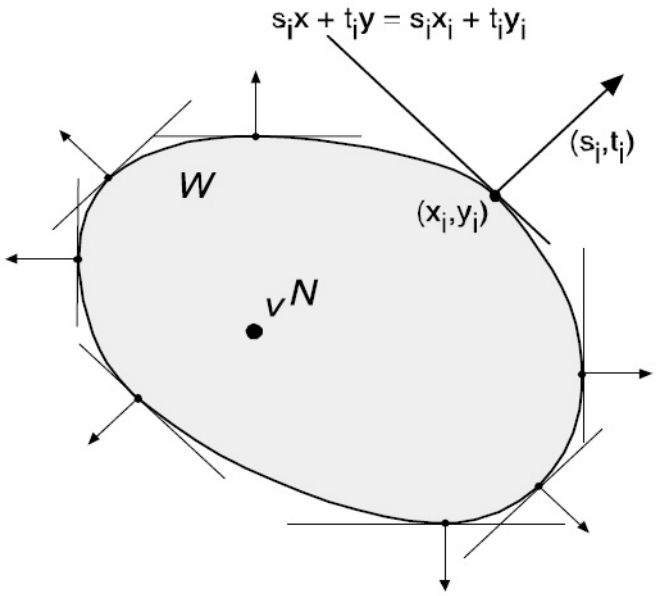
- **Outer Approximation.** $W$ is the convex combination of a finite number of half-spaces which can be represented as a collection of linear inequalities. In this case, $W = \bigcap_{\ell=1}^{L} \left\{ z \in \mathbf{R}^N \mid h_\ell z \leq c_\ell \right\}$ where $h_\ell \in \mathbf{R}^N$ is the gradient orthogonal to the face $\ell$ of $W$, and $c_\ell \in \mathbf{R}$ is a scalar which we call a level.

- **Inner Approximation.** $W$ is approximated as the convex hull of some finite number of vertices $Z$, that is $W = co(Z)$, the convex hull of Z.




(a) Inner Approx. (b) Outer Approx.

More precisely, an outer approximation is obtained in the following way. Suppose we know $N$ points on the boundary of $W$, call them $Z = \{z_1, z_2, \ldots, z_N\}$, and the corresponding subgradients $G = \{g_1, g_2, \ldots, g_N\}$. In other words, the hyperplane $z_i \cdot g_i = z \cdot g_i$ is tangent to $W$ at $z_i$, and the gradients are such that $g_i \cdot w \leqq g_i \cdot z_i$ for $w \in W$, see figure below.




In the graph, $z_i = (x_i, y_i)$, $g_i = (s_i, t_i)$. Each tangent hyperplane generates two half-spaces. The one containing $W$ is called the interior half-space. The outer approximation is therefore the intersection of all the interior half-spaces generated by all $z_i$'s, and it is in fact the smallest convex set containing $W$, given our choice of $Z$ and $G$.

The issue is therefore the computation of these inner and outer approximation. The first step is to choose $Z$ and $G$. For the inner approximation, there are two possibilities. One is the *ray procedure*: choose a point $v^N$ in the set $W$, then let $\theta$ be the polar coordinate angle from $v^N$. Choose a set of $\theta$'s and find rays at angle $\theta$ that intersect the boundary of $W$. This is a linear programming problem. An alternative way (which can be used both for inner and outer approximations) is the *extremal procedure*. For any subgradient $g \in \mathbf{R}^n$, find a point $z_0$ that solves

$$\max_{z \in W} z \cdot g$$

Such a point is on the boundary of $W$. This maximization problem is also a linear programming problem. Repeating the same maximization for several subgradients, we can generate a set of pairs $(Z, G)$. In fact, when $Z$ is available, we can use it for calculating the inner approximation as the convex hull of $Z$.

The algorithm to calculate the outer approximation is the following.

---

**Algorithm 1: Outer Monotone Approximation of** $B(W)$

1. Define $L$ search subgradients $H = \{h_1, h_2, ..., h_L\} \subset R^N$
2. Define $W$ with $M$ approximation subgradients $G = \{g_1, g_2, ..., g_M\} \subset R^N$ and levels $C = \{c_m \mid m = 1, ..., M\} \subset R$ such that $W \equiv \bigcap_{\ell=1}^{M} \{z \mid g_m \cdot z \leqq c_m\}$
3. For each $h_\ell \in H$:
   a) For each $a \in A$, find optimal feasible equilibrium value in the $h_\ell$ direction, by assuming that action $a$ is the current action profile, by solving

$$\begin{aligned} c_\ell(a) = \max_{w} \; & h_\ell \cdot [(1-\delta)\Pi(a) + \delta w] \\ & w \in W \\ & (1-\delta)\Pi^i(a) + \delta w_i \geq (1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i \qquad (5.2) \\ & \qquad i = 1, ..., N \end{aligned}$$

   and set $c_\ell(a) = -\infty$ if there is $w$ that satisfies the constraint (5.2)

   b) Choose the best action profile $a \in A$ by computing $c_\ell^+ = \max\{c_\ell(a) \mid a \in A\}$
4. Get $B^O(W; H) = W^+$, where levels are in $C^+ = \{c_1^+, ...c_L^+\}$, the approximation gradients are in $H$ and $W^+ = \bigcap_{\ell=1}^{L} \{z \mid g_\ell \cdot z \leqq c_\ell^+\}$

---

With this algorithm in mind, we can iterate our approximated operator $B^O(W; H)$ to get closer and closer to the set $V$, with the following algorithm.

---

**Algorithm 2: Outer Hyperplane Algorithm for Approximating $V$**

1. Set initial guess $W^0 \supset \mathcal{W}$ and the elements of the approximation:
   a) Set $L$ search subgradients $H = \{h_1, h_2, ..., h_L\} \subset R^N$
   b) Select boundary points $Z^0 = \{z_1^0, ..., z_L^0\} \subset R^N$
   c) Compute hyperplane levels $c_\ell^0 = g_\ell^0 \cdot z_\ell^0$, $\ell = 1, ..., L$ and collect them in $C^0$
   d) Let $W^0 = \bigcap_{\ell=1}^{L} \{z \mid g_\ell \cdot z \leqq c_\ell^0\}$

2. Generate $W^{k+1} = B^O(W^k; H)$, where $C^{k+1} = \{c_1^{k+1}, \ldots, c_L^{k+1}\}$ define $W^{k+1} = \bigcap_{\ell=1}^{L} \{z \mid g_\ell \cdot z \leqq c_\ell^{k+1}\}$

3. Stop if $W^{k+1}$ is close to $W^k$, i.e. if $\max_\ell \left| c_\ell^{k+1} - c_\ell^k \right| < \epsilon$.

---

In this algorithm we make use of Algorithm 1 in the second step. We crucially need to start from a set large enough.

For the inner approximation, the analogous two algorithms are the following.

---

**Algorithm 3: Monotone Inner Hyperplane Approximation of** $B(W)$

1. Define $L$ search subgradients $H = \{h_1, h_2, \ldots, h_L\} \subset R^N$

2. Define $W$ with $M$ approximation subgradients $G = \{g_1, g_2, \ldots, g_M\} \subset R^N$ and levels $C = \{c_m \mid m = 1, \ldots, M\} \subset R$ such that $W \equiv \bigcap_{\ell=1}^{M} \{z \mid g_m \cdot z \leqq c_m\}$

3. For each $h_\ell \in H$:

   a) For each $a \in A$, find optimal feasible equilibrium value in the $h_\ell$ direction, by assuming that action $a$ is the current action profile, by solving

$$\begin{aligned} c_\ell(a) = \max_w \; & h_\ell \cdot [(1-\delta)\Pi(a) + \delta w] \\ & w \in W \\ & (1-\delta)\Pi^i(a) + \delta w_i \geq (1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i \qquad (5.3) \\ & \qquad i = 1, \ldots, N \end{aligned}$$

   and set $c_\ell(a) = -\infty$ if there is no $w$ that satisfies the constraint (5.3)

   b) Choose the best action profile $a \in A$ and the corresponding continuation value:

$$\begin{aligned} a_\ell^* &= \arg\max\{c_\ell(a) \mid a \in A\} \\ z'_\ell &= (1-\delta)\Pi^i(a_\ell^*) + \delta w_\ell(a_\ell^*) \end{aligned}$$

   c) Collect set of vertices of convex hull $Z' = \{z'_\ell \mid \ell = 1, \ldots, L\}$ and define $W^+ = co(Z')$

4. Compute $Z^+ = \{z' \in Z' \mid z' \in \partial W^+\}$, and find subgradients $G^+ = \{g_1^+, g_2^+, ..., g_M^+\} \subset R^N$ and levels $C^+ = \{c_m^+ \mid m = 1, ..., M\} \subset R$ such that $co(Z^+) \equiv \bigcap_{\ell=1}^{M} \{z \mid g_m^+ \cdot z \leqq c_m^+\} = W^+$

5. Get $B^I(W; H) = W^+$, where levels are in $C^+ = \{c_1^+, ...c_L^+\}$, the approximation subgradients are in $G^+$ and vertices are in $Z^+$

---

To define convergence for the inner approximation iterations, CJY use the following distance (called the Hausdorff distance):

$$d(Z_1, Z_2) = \max \left\{ \max_{z_1 \in Z_1} \min_{z_2 \in Z_2} \|z_1 - z_2\| , \max_{z_2 \in Z_2} \min_{z_1 \in Z_1} \|z_1 - z_2\| \right\} \tag{5.4}$$

The following algorithm iterates over successive inner approximations:

---

**Algorithm 4: Inner Hyperplane Algorithm for Approximating $V$**

1. Set initial elements of the approximation:
   a) Set $L$ search subgradients $H = \{h_1, h_2, ..., h_L\} \subset R^N$
   b) Select vertices $Z^0 = \{z_1^0, ..., z_M^0\} \subset R^M$ for initial guess $W^0 = co(Z^0)$
   c) Define the gradients $G = \{g_1^0, ..., g_M^0\} \subset R^M$ and levels $C^0 = \{c_1^0, ..., c_M^0\} \subset R^M$ which define $W^0 = \bigcap_{m=1}^{M} \{z \mid g_m^0 \cdot z_m^0 \leqq c_m^0\}$
2. Generate $W^{k+1} = B^I(W^k; H)$, with vertices $Z^{k+1} = \{z_1^{k+1}, ..., z_M^{k+1}\}$
3. Stop if $d(Z^{k+1}, Z^k) < \epsilon$

---

### 5.2.1 An example: the repeated prisoner's dilemma

Let the discount factor be $\delta = 0.8$. The payoffs of the prisoner's dilemma[1] are

1 This part builds on material by Dean Corbae, and a code by Pablo D'Erasmo

| | | player 2 | |
|---|---|---|---|
| | Actions | C | D |
| player 1 | C | (4,4) | (0,6) |
| | D | (6,0) | (2,2) |

To calculate the outer approximation, we can use `fmincon` to solve the linear program, after having specified the search subgradients $h_\ell$ and levels $c_\ell$ for $\ell = 1, \ldots, L$. To construct the initial guess for the levels, we can use the search subgradients and define extreme points $z_\ell^0$, then setting $c_\ell^0 = h_\ell \cdot z_\ell^0$.

As an example, in our game, if the number of the search subgradients is 4, then $H$ is

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$$

and the polytope will be a rectangle in this case. For each $a \in A$, the incentive compatibility constraints can be included in the set of linear constraints (they are linear in $w_i$). For each subgradient $h_\ell$ and action profile $a$, we need to find the continuation value $w$ which makes $a$ individually rational and maximizes the weighted sum of player payoffs where the weights are given by $h_\ell$. Note that the search subgradients and the approximation subgradients can be the same.

To solve for the inner approximation, the steps are the same as in the outer approximation method. However, the set $W$ is defined as the convex hull ofthe vertices. To construct an initial guess, we can shrink the set obtained by the outer approximation subroutine by 2-3% (CJY suggest this rule-of-thumb). Given points $z \in Z$, the convex hull of these points can be found by using the function `convhull` in Matlab. This function will return the indices of the points in the convex hull. Then, you need to convert those indices in actual points, setting $W = co(Z)$.

To solve the maximization problem, a procedure similar to a grid search can be used, where the points in the grid are a finite number of convex combinations between consecutive points in the convex hull (like connecting the points in the vertices). This is quite inefficient, and there are better ways to do it, but it is a first step in the right direction.

The material includes a code by Pablo D'Erasmo which implements the inner and outer approximations. We also included the code provided by the authors of the paper.

### 5.2.2 New developments

There has been a lot of work on these techniques in the last few years. The procedure we described above from Judd et al. (2003) can be extended to accommodate models with finite number of states. Sleet and Yeltekin (2016) show how to deal with the case of uncountable endogenous states in a compact set. This case is more complicated, since there are nonconvexities that make the problem non-standard.

## 5.3 ONE-SIDED LACK OF COMMITMENT

Now that we have a computational method for finding the set of possible continuation values consistent with a sequential problem, we can go back to our examples. The limited commitment case was the following

$$\begin{aligned}
\max_{\{c_t\}_{t=0}^{\infty}} \quad & E_{-1} \sum_{t=0}^{\infty} \beta^t \left( y_t - c_t \right) \\
s.t. \quad & E_{-1} \sum_{t=0}^{\infty} \beta^t u\left( c_t \right) \geq U_0 \\
& u\left( c_t \right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left( c_{t+j} \right) \geq \\
& \qquad \geq u\left( y_t \right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left( y_{t+j} \right), \forall t \quad (5.5)
\end{aligned}$$

Assume $y_t$ is i.i.d. and its support is $\mathbf{Y} \equiv \left\{ \overline{y}_s \right\}_{s=1}^{S}$. Call its distribution function $\pi_s, s = 1, \ldots, S$.

The RHS of (5.5) is a time invariant function of the realization of the shock at time $t$, therefore we can define:

$$U_{aut} \equiv E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left( y_{t+j} \right)$$

and we can rewrite the participation constraint as:

$$u\left( c_t \right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left( c_{t+j} \right) \geq u\left( y_t \right) + \beta U_{aut} \quad (5.6)$$

Now, define the *agent's promised expected discounted value* as:

$$U_t \equiv E_{t-1} \sum_{j=0}^{\infty} \beta^j u\left(c_{t+j}\right) \tag{5.7}$$

This variable summarizes the promises that the principal makes to the agent in each period. Therefore, we can formulate the contract recursively in the following way: the principal enters period $t$ with a promise $U_-$ made in $t-1$, before the endowment shock is realized. Then he complies with his previous promise by choosing state-contingent consumption $c_s, s = 1, \ldots, S$ and a new promise $U_s, s = 1, \ldots, S$. We therefore are using the expected discounted value of the agent as a state variable. If we call $P(U_-)$ the value function of the principal, we can write the following functional equation:

$$P(U_-) = \max_{\{c_s, U_s\}_{s=1}^{S}} \sum_{s=1}^{S} \pi_s \left[\overline{y}_s - c_s + \beta P(U_s)\right] \tag{5.8}$$

$$s.t. \quad \sum_{s=1}^{S} \pi_s \left[u(c_s) + \beta U_s\right] \geq U_- \tag{5.9}$$

$$u(c_s) + \beta U_s \geq u\left(\overline{y}_s\right) + \beta U_{aut} \quad s = 1, \ldots, S \tag{5.10}$$

$$c_s \in C \tag{5.11}$$

$$U_s \in \mathcal{U} \tag{5.12}$$

The previous characterization deserves some explanation. First of all, notice that constraint (5.12) restricts the choice of continuation values $U_s$ to be in the set $\mathcal{U}$, which is obtained by repeatedly applying the APS operator $B$ defined as:

$$B(W) = \left\{ \begin{array}{l} U \in W : \sum_{s=1}^{S} \pi_s \left[u(c_s) + \beta U_s\right] \geq U, \\ u(c_s) + \beta U_s \geq u\left(\overline{y}_s\right) + \beta U_{aut} \quad s = 1, \ldots, S \\ c_s \in C \quad s = 1, \ldots, S \end{array} \right\} \tag{5.13}$$

Apart from this set, the functional equation looks like a Bellman equation in which we have the agent's value function as a state variable. Therefore, we can characterize the optimal contracts in two steps:

1. Find the set $\mathcal{U}$ by repeatedly applying the operator $B$ defined in (5.13)
2. Solve the Bellman equation as usual, either with value function iteration or Howard's improvement algorithm.

We can also use collocation, either pairing it with VFI, or by using it on the first order conditions of the problem.

## 5.4 PRIVATE INFORMATION

### 5.4.1 Endowment is private information

Remember our problem:

$$\begin{aligned}
\max_{\left\{\tau_t\left(y^t\right)\right\}_{t=0}^{\infty}} \quad & \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[-\tau_t\left(y^t\right)\right] \pi\left(y^t | y_{-1}\right) \\
s.t. \quad & \sum_{t=0}^{\infty} \sum_{y^t \in y^{t+1}} \beta^t u\left(c_t\left(y^t\right)\right) \pi\left(y^t | y_{-1}\right) \geq U_0 \\
& DIC_t\left(y^{t-1}, \overline{y}_i; \overline{y}_{i-1}\right) \geq 0 \qquad \forall y^{t-1}, \quad \forall \overline{y}_i, \quad i = 2, \ldots, N
\end{aligned}$$

We can rewrite the problem with a functional equation that uses promised utilities as state variables:

$$P\left(U_-\right) = \max_{\{\tau_s, U_s\}_{s=1}^{S}} \sum_{s=1}^{S} \pi_s \left[-\tau_s + \beta P\left(U_s\right)\right] \tag{5.14}$$

$$s.t. \quad \sum_{s=1}^{S} \pi_s \left[u\left(\tau_s + \overline{y}_s\right) + \beta U_s\right] = U_- \tag{5.15}$$

$$u\left(\tau_s + \overline{y}_s\right) + \beta U_s \geq u\left(\tau_s + \overline{y}_{s-1}\right) + \beta U_{s-1} \quad s = 2, \ldots, S \tag{5.16}$$

$$c_s \in C \tag{5.17}$$

$$U_s \in \mathcal{U} \tag{5.18}$$

where $\mathcal{U}$ is the fixed point of the operator:

$$B\left(W\right) = \left\{ \begin{array}{l} U \in W : \sum_{s=1}^{S} \pi_s \left[u\left(\tau_s + \overline{y}_s\right) + \beta U_s\right] \geq U, \\ u\left(\tau_s + \overline{y}_s\right) + \beta U_s \geq u\left(\tau_s + \overline{y}_{s-1}\right) + \beta U_{s-1} \quad s = 2, \ldots, S \\ c_s \in C \quad s = 1, \ldots, S \end{array} \right\} \tag{5.19}$$

Also in this case, we can solve for the optimal contract in two steps: first, characterize the set $\mathcal{U}$; second, solve the Bellman equation.

### 5.4.2 Hidden effort

Our problem was:

$$\max_{\left\{\tau_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[-\tau_t\left(y^t\right)\right] \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right)$$

$$s.t. \quad \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)\right] \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \geq U_0$$

$$\left\{a_t\left(y^t\right)\right\}_{t=0}^{\infty} \in \arg \max_{\left\{a_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)\right] \times$$

$$\times \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \tag{5.20}$$

We can reformulate the problem recursively as:

$$P\left(U_-, \overline{y}_i\right) = \max_{\left\{c, \{U_s\}_{s=1}^S, a^*\right\}} \left[\overline{y}_i - c + \beta \sum_{s=1}^{S} \pi_s\left(a^*\right) P\left(U_s, \overline{y}_s\right)\right] \tag{5.21}$$

$$s.t. \quad u\left(c\right) - v\left(a^*\right) + \beta \sum_{s=1}^{S} \pi_s\left(a^*\right) U_s = U_- \tag{5.22}$$

$$a^* = \arg\max_{a \in A} \left\{u\left(c\right) - v\left(a\right) + \beta \sum_{s=1}^{S} \pi_s\left(a\right) U_s\right\} \tag{5.23}$$

$$c \in C, a \in A \tag{5.24}$$

$$U_s \in \mathcal{U} \tag{5.25}$$

where $\mathcal{U}$ is the fixed point of the operator:

$$B\left(W\right) = \left\{ \begin{array}{l} U \in W : u\left(c\right) - v\left(a^*\right) + \beta \sum_{s=1}^{S} \pi_s\left(a^*\right) U_s = U, \\ a^* = \arg\max_{a \in A} \left\{u\left(c\right) - v\left(a\right) + \beta \sum_{s=1}^{S} \pi_s\left(a\right) U_s\right\} \\ c \in C, a \in A \quad s = 1, ..., S \end{array} \right\} \tag{5.26}$$

This form is slightly different from the previous problem with private information, mainly because here we have chosen a different timing convention: effort affects the probability of the state tomorrow. Another timing convention is that effort is exerted before the realization of the shock today, and therefore

affects the distribution of the endowment today. In that case, we can write the recursive formulation as:

$$P(U_-) = \max_{\left\{\{c_s,U_s\}_{s=1}^S,a^*\right\}} \sum_{s=1}^{S} \pi_s(a^*)\left[\overline{y}_s - c_s + \beta P(U_s)\right] \tag{5.27}$$

$$s.t. \quad \sum_{s=1}^{S} \pi_s(a^*)\left[u(c_s) + \beta U_s\right] - v(a^*) = U_- \tag{5.28}$$

$$a^* = \arg\max_{a\in A} \sum_{s=1}^{S} \pi_s(a)\left[u(c_s) + \beta U_s\right] - v(a) \tag{5.29}$$

$$c_s \in C, a \in A \tag{5.30}$$

$$U_s \in \mathcal{U} \tag{5.31}$$

where $\mathcal{U}$ is the fixed point of the operator:

$$B(W) = \left\{ \begin{array}{l} U \in W : \sum_{s=1}^{S} \pi_s(a^*)\left[u(c_s) + \beta U_s\right] - v(a^*) = U, \\ a^* = \arg\max_{a\in A} \sum_{s=1}^{S} \pi_s(a)\left[u(c_s) + \beta U_s\right] - v(a) \\ c_s \in C, a \in A \quad s = 1, \ldots, S \end{array} \right\} \tag{5.32}$$

This formulation is more similar to the one in the previous section. Under some assumptions on the probability distribution, we can avoid to use constraint (5.29) and use instead the first-order condition of the agent's maximization problem. These conditions are known as Rogerson's conditions. The first is the following:

**Condition 1** (Monotone Likelihood-Ratio Condition (MLRC)). $\widehat{a} \leq \widehat{\widehat{a}} \Longrightarrow \frac{\pi\left(\overline{y}_s|\widehat{a}\right)}{\pi\left(\overline{y}_s|\widehat{\widehat{a}}\right)}$ *is nonincreasing in* $s$.

The above property can be restated in a simpler way: if $\pi(\cdot)$ is differentiable, then MLRC is equivalent to $\frac{\pi_a\left(\overline{y}_s|a\right)}{\pi\left(\overline{y}_s|a\right)}$ being nondecreasing in $s$ for any $a$, where $\pi_a\left(\overline{y}_s \mid a\right)$ is the derivative of $\pi(\cdot)$ with respect to $a$. An important consequence of the MLRC is the following: let $F(\cdot)$ be the cumulative distribution function of $\pi(\cdot)$; then MLRC implies that the density function $F'\left(\overline{y}_s \mid a\right)$ is nonpositive for any $s$ and every $a$. Therefore, more effort implies a first order stochastic dominance shift of the distribution (see Rogerson (1985)).

**Condition 2** (Convexity of the Distribution Function Condition (CDFC)). $F''\left(\overline{y}_s \mid a\right)$ *is nonnegative for any* $s$ *and every* $a$.

This condition combined with MLRP implies that more effort increases the probability of observing a high output, but as we get closer to the highest realizations this effect becomes smaller and smaller: CDFC is a sort of decreasing marginal returns for effort in stochastic terms.

## 5.5 OPTIMAL FISCAL POLICY

In this case, we need to make a small departure from the standard APS approach. Instead of using the continuation value of the agent as a state variable, we are going to use the marginal utility of consumption. The problem was:

$$\begin{aligned} \max_{\{c_t, b_t, l_t\}_{t=0}^{\infty}} \quad & E_0 \sum_{t=0}^{\infty} \beta^t \left[ u\left(c_t\right) + v\left(l_t\right) \right] \\ s.t. \quad & c_t - \beta \frac{E_t u'\left(c_{t+1}\right)}{u'\left(c_t\right)} b_{t+1}^g = -\left(c_t + g_t\right) \frac{v'\left(c_t + g_t\right)}{u'\left(c_t\right)} - b_t^g \end{aligned}$$

First of all, notice that we have a state variable in this problem: debt holdings. Therefore, we have to characterize not a feasible set for continuation values but a correspondence: the feasible set changes if the debt holdings change.

Phelan and Stacchetti (2001) analyze this problem for the case of no commitment. We use their ideas to analyze the problem with full commitment of the government. Let $m \equiv u'\left(c\right)$, and assume the government expenditure shock can take $S$ values and is i.i.d. as in previous examples. We can write the problem recursively as:

$$P\left(m_-, \overline{g}_s, b_-\right) = \max_{\left\{c, b, l, \{m_s\}_{s=1}^S\right\}} u\left(c\right) + v\left(l\right) + \beta \sum_{s=1}^{S} P\left(m_s, \overline{g}_s, b\right) \tag{5.33}$$

$$s.t. \quad c - \beta \frac{\sum_{s=1}^S m_s}{m_-} b = -\left(c + \overline{g}_s\right) \frac{v'\left(c + \overline{g}_s\right)}{m_-} - b_- \tag{5.34}$$

$$c \in C, l \in L, b \in B \tag{5.35}$$

$$m_s \in \mathcal{U}\left(b_-\right) \tag{5.36}$$

where the correspondence $\mathcal{U}\left(b_-\right)$ is the fixed point of the operator

$$B\left(W\left(b_-\right)\right) = \left\{ \begin{array}{c} m \in \left(b_-\right) : c - \beta \frac{\sum_{s=1}^S m_s}{m_-} b = -\left(c + \overline{g}_s\right) \frac{v'\left(c + \overline{g}_s\right)}{m_-} - b_- \\ c \in C, l \in L, b \in B \end{array} \right\} \tag{5.37}$$

## 5.6 THE CURSE OF DIMENSIONALITY SQUARED

We said the approach of APS can be called "dynamic programming squared". Unfortunately, this approach also brings with the advantages a big disadvantage: what I call "curse of dimensionality squared".

We have seen that any application can be solved by a simple two-step procedure: first, characterize the set of continuation values, and second, solve the Bellman equation. However, the first step causes many troubles if there are other state variables in the model, or if there is more than one agent. We have seen such a case in the optimal fiscal policy problem. However, this problem arises frequently for simple extensions of the basic examples we have seen here.

Take the case in which there is lack of commitment. Imagine that instead of having a stochastic endowment, there is a production function that uses capital as input, and it is subject to a productivity shock. This implies, first of all, that agent's continuation value in autarky will depend on the capital accumulated up to that period. Second, this implies that the continuation values in general depend on the capital accumulated up to that period, and therefore, the set of continuation values that are admissible will depend on capital. In other words, the set of admissible solutions that we have to characterize in the first step is two-dimensional: for any feasible level of capital, there is a set of admissible continuation values. Therefore, the set of admissible continuation values is a correspondence that maps from the set of feasible capital stock to the set of continuation values. As the number of natural state variables (i.e., state variables already present in the problem before we apply APS arguments) increases, computing the admissible set becomes an impossible task (I don't know any paper that works with more than 2 natural states). Even with just capital accumulation, the problem is challenging.

This is one reason why many researchers prefer the recursive Lagrangian techniques developed by Marcet and Marimon (2017): one does not have to characterize the feasible set for the auxiliary recursive program, therefore saving time and computational power. There are many cases in which, due to the number of states and/or agents, the MM approach is the only feasible method. We will analyze it in the next chapter.

# 6 TIME INCONSISTENCY: MM APPROACH

## 6.1 INTRODUCTION

In the previous chapter we have seen how we can use the APS methodology to solve problems which are not recursive in the Bellman sense. APS suggest to "recursify" the problem by using the continuation values as state variables. The continuation utilities summarize the promises made to agents. There is another approach, developed almost at the same time of APS, which relies on the work by Marcet and Marimon (2017)[1]. This approach makes use of a Lagrangian formulation of the dynamic program, and derives a recursive form by using Lagrange multipliers (or functions of Lagrange multipliers) as costates. There are many advantages in this method, the main one being that we do not need to characterize the feasible set for the auxiliary state variables as in APS: Lagrange multipliers live in the non-negative real line by definition. Additional advantages include the fact that the number of decision variables that one has to solve for is much smaller under this approach.

In this chapter we first give a general treatment of the MM technique. We base our explanation on the latest versions of the paper (2011, 2017) which are more general and have fixed a few problems that previous version had. We then show how to solve the examples by using this technique. We provide a detailed numerical implementation of the repeated moral hazard model based on Mele (2014), which uses collocation and several techniques seen in chapter 4.

1 The first version of the paper is dated 1992 and a widely available version is dated 1998, but the latest two versions (2011, 2017) have solved a few problems of the theory. In the meantime, people have extensively applied the methodology to problems of optimal policy and limited commitment, and more recently to private information setups.

## 6.2 MM IN A NUTSHELL

MM start from the consideration that many dynamic problems we face in (macro)economics can be described with the following non-standard dynamic programming problem:

$$V(x,s) = \sup_{a_t} E_0 \sum_{t=0}^{\infty} \beta^t r(x_t, a_t, s_t) \qquad \textbf{PP}$$

$$s.t. \quad x_{t+1} = \ell(x_t, a_t, s_{t+1}), \qquad p(x_t, a_t, s_t) \geq 0 \qquad (6.1)$$

$$E_t \sum_{n=1}^{N_j+1} \beta^n h_0^j(x_{t+n}, a_{t+n}, s_{t+n}) + h_1^j(x_t, a_t, s_t) \geq 0 \qquad (6.2)$$

$$j = 1, ..., l; \quad t \geq 0, \quad x_0 = x, s_0 = s,$$

$$N_j = \infty \quad for \quad j = 1, ..., k \quad N_j = 0 \quad for \quad j = k+1, ..., l$$

As we said before, the Bellman equation cannot hold here, since constraints like (6.2) make today's choices depend on future choices. Notice that those constraints (6.2) with $N_j = \infty$ are of the same class of participation constraints or downward incentive compatibility constraint, while those constraints with $N_j = 0$ are of the same class of the constraint in our optimal policy example. MM work on a generalized version of (**PP**) which relies on some algebraic trick to make sure the objective function is homogeneous of degree one:

$$V_\mu(x,s) = \sup_{\{a_t\}} E\left[\sum_{j=0}^{l} \sum_{t=0}^{N_j} \beta^t \mu^j h_0^j(x_t, a_t, s_t) \mid s\right] \qquad \textbf{PP}_\mu$$

$$s.t. \quad x_{t+1} = \ell(x_t, a_t, s_{t+1}), \qquad p(x_t, a_t, s_t) \geq 0 \qquad (6.3)$$

$$E_t \sum_{n=1}^{N_j+1} \beta^n h_0^j(x_{t+n}, a_{t+n}, s_{t+n}) + h_1^j(x_t, a_t, s_t) \geq 0 \qquad (6.4)$$

$$j = 1, ..., l; \quad t \geq 0, \quad x_0 = x, s_0 = s,$$

$$N_j = \infty \quad for \quad j = 1, ..., k \quad N_j = 0 \quad for \quad j = k+1, ..., l$$

It should be obvious that $\mathbf{PP} = \mathbf{PP}_\mu$ when $\mu = (1, 0, 0, ..., 0)$, and $h_0^0(x,a,s) \equiv r(x,a,s)$, $h_1^0(x,a,s) \equiv r(x,a,s) - R$ where $R > 0$ is a large never binding upper bound. The value function of this problem takes the form $V_\mu(x,s) = \mu v_\mu(x,s)$, i.e. it looks like a social welfare function with Pareto weights given by the vector $\mu$.

Marcet and Marimon show that, by attaching Lagrange multipliers $\gamma^j$ to the first-period constraints (6.2), you can transform ($\mathbf{PP}_\mu$) in the following saddle point problem ($\mathbf{SPP}_\mu$):

$$SV(x,\mu,s) = \inf_{\gamma\in R^l_+}\sup_{\{a_t\}}\sum_{j=0}^{l}\left(\mu^j h_0^j(x,a_0,s)+\gamma^j h_1^j(x,a_0,s)\right)+ \qquad \mathbf{SPP}_\mu$$

$$+\beta E_0\left[\sum_{j=0}^{l}\varphi^j(\mu,\gamma)\sum_{t=0}^{\infty}\beta^t h_0^j(x_{t+1},a_{t+1},s_{t+1})\right]$$

$$s.t.\quad x_{t+1}=\ell(x_t,a_t,s_{t+1}),\qquad p(x_t,a_t,s_t)\geq 0,\quad t\geq 0$$

$$E_t\sum_{n=1}^{N_j+1}\beta^n h_0^j(x_{t+n},a_{t+n},s_{t+n})+h_1^j(x_t,a_t,s_t)\geq 0$$

$$j=1,...,l;\quad t\geq 1$$

What we have done? We have incorporated the period-0 constraints (6.2) in the objective of this $inf\ sup$ problem. The crucial step now is to show that this problem has a recursive structure. We can do this by defining a law of motion $\varphi$ for the weights $\mu$ as follows: $\mu'^j=\varphi(\mu,\gamma,s)=\mu^j+\gamma^j$ if $N_j=\infty$, and $\mu'^j=\varphi(\mu,\gamma,s)=\gamma^j$ if $N_j=0$. Therefore, the solution $\gamma^*$ of $\mathbf{SPP}_\mu$ in state $(x,s)$ generates a new problem $\mathbf{SPP}_{\varphi(\mu,\gamma^*,s)}$ in state $(x^{*\prime},s')$. The recursivity of $\mathbf{SPP}_\mu$ is shown by proving that solutions of $\mathbf{SPP}_{\varphi(\mu,\gamma^*,s)}$ in state $(x^{*\prime},s')$ are one-period ahead solutions of $\mathbf{SPP}_\mu$ in state $(x,s)$. This implies that the optimal solution of $\mathbf{SPP}_\mu$ satisfies the following functional equation:

$$W(x,\mu,s)=\inf_{\gamma\geq 0}\sup_{a}\left\{\mu h_0(x,a,s)+\gamma h_1(x,a,s)+\beta E\left[W(x',\mu',s')\mid s\right]\right\} \qquad \mathbf{SPFE}$$

$$s.t.\quad x'=\ell(x,a,s'),\qquad p(x,a,s)\geq 0$$

$$\mu'=\varphi(\mu,\gamma,s) \tag{6.5}$$

This is a saddle-point functional equation which is the analog of Bellman equation for problems with constraints in the form of (6.2). The recursivity comes from the Lagrange multipliers: we use Lagrange multipliers as record-keeping tools, that incorporate the promises of the contract. For constraints with $N_j=\infty$, the sum of past Lagrange multipliers is the relevant state variable that helps us recover recursivity, while for constraints with $N_j=0$ we use the past Lagrange multiplier as a state variable.

Summarizing: MM show that problems like ($\mathbf{PP}_\mu$) can be solved recursively by using (cumulated) Lagrange multipliers as state variables. Therefore, a standard way to proceed is to write down the Lagrangean of the problem at hand, and solve the functional equation (**SPFE**) associated with it. Value function iteration or Howard's improvement algorithm work well for this methodology, although most papers work on the Lagrangian's first order conditions.

## 6.3 THE RELATIONSHIP BETWEEN $\mathbf{PP}_\mu$, $\mathbf{SPP}_\mu$ AND SPFE

In order to prove their results, MM need to make (fairly general) assumptions on the primitives of the problem. Here I provide a list of all of them. In each theorem we are going to say which ones we need.

**Assumption 12.** *$S$ is a countable set, and the process $\{s_t\}$ is Markov.*

**Assumption 13.** *$X$ and $A$ are convex subsets of $R^n$ and $R^m$. Functions $p(\cdot,\cdot,\cdot)$ and $\ell(\cdot,\cdot,\cdot)$ are continuous. For any $(x,s) \in X \times S$ there exists $\widetilde{a} \in A$ such that $p(x,\widetilde{a},s) > 0$.*

**Assumption 14.** *$\ell(\cdot,\cdot,s)$ is linear and $p(\cdot,\cdot,s)$ is concave.*

**Assumption 15.** *Given $(x,s) \in X \times S$ there exist constants $B > 0$ and $\varphi \in \left(0, \beta^{-1}\right)$ such that if $p(x,a,s) \geq 0$ and $x' = \ell(x,a,s')$, then $\|a\| \leq B\|x\|$ and $\|x'\| \leq \varphi\|x\|$.*

**Assumption 16.** *$h_i^j(\cdot,\cdot,s)$, $i = 0,1$, $j = 0,...,l$, are continuous and uniformly bounded, $h_0^j(\cdot,\cdot,s)$ is non-decreasing and $\beta \in (0,1)$.*

**Assumption 17.** *$h_i^j(\cdot,\cdot,s)$, $i = 0,1$, $j = 0,...,l$, are continuously differentiable on $\{(x,a) : p(x,a,s) > 0\}$.*

**Assumption 18.** *$h_i^j(\cdot,\cdot,s)$, $i = 0,1$, $j = 0,...,l$, are concave.*

**Assumption 19.** *Assumption 18 holds and moreover $h_0^j(\cdot,\cdot,s)$, $j = 0,...,l$, are strictly concave.*

**Assumption 20.** *For all $(x,s)$, there exists a plan $\{\widetilde{a}\}_{n=0}^{\infty}$ with initial conditions $(x,s)$ which satisfies the inequality constraints* (6.3) *and* (6.4) *with strict inequality.*

Assumptions 12-16 are quite standard for many models. Assumption 15 allows for technologies that yield long-run growth (e.g. AK technology in Ramsey growth model).

Under Assumptions 12 - 16 and 18, and given $\mu \in R_+^{l+1}$, there is a solution $\mathbf{a}^* \equiv \{a_t\}_{t=0}^{\infty}$ that solves $\mathbf{PP}_\mu$ with initial conditions $(x, s)$, achieving the value $V_\mu(x, s)$. If Assumption 19 holds, the solution is unique.

The first theorem links the solution of $\mathbf{PP}_\mu$ to $\mathbf{SPP}_\mu$.

**Theorem 9.** $\left(\mathbf{PP}_\mu \Rightarrow \mathbf{SPP}_\mu\right)$ *Let Assumptions 12 - 16, 18 and 20 be satisfied. Given $\mu \in R_+^{l+1}$, let $\mathbf{a}^*$ be a solution of $\mathbf{PP}_\mu$ with initial conditions $(x, s)$. Then there exists $\gamma^* \in R_+^l$ such that $(\mathbf{a}^*, \gamma^*)$ is a solution to $\mathbf{SPP}_\mu$ in state $(x, s)$, and the value of the latter is $V_\mu(x, s)$.*

The second theorem proves the converse result, and it is worth mentioning that it is basically assumption-free:

**Theorem 10.** $\left(\mathbf{SPP}_\mu \Rightarrow \mathbf{PP}_\mu\right)$ *Given $\mu \in R_+^{l+1}$ and initial conditions $(x, s)$, let $(\mathbf{a}^*, \gamma^*)$ be a solution to $\mathbf{SPP}_\mu$. Then $\mathbf{a}^*$ is a solution to $\mathbf{PP}_\mu$ in state $(x, s)$.*

We have established a link between the original problem $\mathbf{PP}_\mu$ and the saddle-point problem $\mathbf{SPP}_\mu$. This is similar to standard duality theory. The following theorems establish a link between $\mathbf{SPP}_\mu$ and **SPFE**. Define the saddle-point policy correspondence as:

$$\Psi(x, \mu, s) = \left\{(a^*, \gamma^*) : W(x, \mu, s) = \mu h_0(x, a^*, s) + \gamma^* h_1(x, a^*, s) + \beta E\left[W\left(x^{*\prime}, \mu^{*\prime}, s'\right) \mid s\right]\right.$$
$$\left. s.t. \quad x^{*\prime} = \ell\left(x, a^*, s'\right), \qquad p(x, a^*, s) \geq 0, \qquad \mu^{*\prime} = \varphi(\mu, \gamma^*, s)\right\}$$

We can therefore state the result:

**Theorem 11.** $\left(\mathbf{SPP}_\mu \Rightarrow \mathbf{SPFE}\right)$ *Let $W(x, \mu, s) \equiv V_\mu(x, s)$ be the value of $\mathbf{SPP}_\mu$ at $(x, s)$, for an arbitrary $(x, \mu, s)$, then the value function $W$ satisfies **SPFE**. In particular, if $(\mathbf{a}^*, \gamma^*)$ is a solution to $\mathbf{SPP}_\mu$ in state $(x, s)$, then $\left(a_0^*, \gamma^*\right) \in \Psi(x, \mu, s)$.*

The converse is also true under differentiability assumptions:

**Theorem 12.** $\left(\mathbf{SPFE} \Rightarrow \mathbf{SPP}_\mu\right)$ *Let $W$ be continuous in $(x, \mu)$, concave in $z$, homogeneous of degree one in $\mu$, and satisfy **SPFE**. Let Assumption 17 hold. Then if $(\mathbf{a}^*, \boldsymbol{\gamma}^*)$ is generated by the saddle-point policy function $\psi$ associated with $W$ from initial conditions $(x, \mu, s)$, and $p\left(x_t^*, a_t^*, s_t\right) > 0$ for any $t$ and any $s_t$, then $(\mathbf{a}^*, \boldsymbol{\gamma}^*)$ is also a solution to $\mathbf{SPP}_\mu$ at $(x, s)$.*

It is important to notice that the conditions under which Theorem 12 holds are quite standard. Homogeneity, continuity and boundedness of $W$ are due to Assumptions 13-16. Concavity and differentiability of $W$ are more restrictive, and due to Assumptions 18 and 17. However, Theorem 12 can be proved also without Assumptions 18 and 17: in that case the only restrictive condition is that $(\mathbf{a}^*,\gamma^*)$ must be generated by a saddle-point policy function, i.e. must be uniquely determined.

In the next result, we will see that **SPFE** is a contraction mapping. This is very important for computational purposes, because given Theorems 11 and 12, we can use iterative methods analogous to dynamic programming to solve for the optimal policy. In order to do that, remember that our value function $W$ is homogeneous of degree one in $\mu$ and we can therefore rewrite it as $W = \mu\omega$. Define the set $M$ as:

$$\begin{aligned} M = \{\omega : X \times R_+^{l+1} \times S \to R^{l+1} \text{ s.t. for } j = 0, .., l \\ \text{i. } \omega_j(\cdot, \cdot, s) \text{ is continuous, and } \omega_j(\cdot, \mu, s) \text{ is bounded if } \|\mu\| \leq 1 \\ \text{ii. } \omega_j(\cdot, \mu, s) \text{ is concave, and} \\ \text{iii. } \omega_j(x, \cdot, s) \text{ is convex and homogeneous of degree zero } \} \end{aligned}$$

Marcet and Marimon define the Dynamic Saddle-Point Problem as:

$$\inf_{\gamma \geq 0} \sup_{a} \left\{ \mu h_0(x, a, s) + \gamma h_1(x, a, s) + \beta E\left[\mu' \omega(x', \mu', s') \mid s\right] \right\} \qquad \mathbf{DSPP}_{(x,\mu,s)}$$

$$s.t. \quad x' = \ell(x, a, s'), \qquad p(x, a, s) \geq 0$$
$$\mu' = \varphi(\mu, \gamma, s)$$

In order to have a well defined program in $\mathbf{DSPP}_{(x,\mu,s)}$, we need to make the following interiority assumption:

**Assumption 21.** *For any $(x, \mu, s) \in X \times R_+^{l+1} \times S$, there exists $\widetilde{a} \in A$ satisfying Assumption 13 such that for any $\mu' \in R_+^{l+1}$, $\|\mu'\| \leq 1$, and $j = 0, \ldots, l$ the following holds: $h_1^j(x, \widetilde{a}, s) + \beta E\left[\omega^j(\ell(x, \widetilde{a}, s), \mu', s') \mid s\right] > 0$.*

This assumption is satisfied whenever Assumption 20 is and $\mu'\omega(\ell(x, \widetilde{a}, s), \mu', s')$ is the value function of $\mathbf{SPP}_{(\ell(x,\widetilde{a},s),\mu',s')}$.

**Proposition 1.** *Let $\omega \in M$ and assume Assumptions 12-16, 18 and 21 hold. Then there exists $(a^*, \gamma^*)$ that solves $\mathbf{DSPP}_{(x,\mu,s)}$. If Assumption 19 is satisfied, then $a^*(x, \mu, s)$ is unique.*

Let us assume, for the next results, that the solution is unique:

**Assumption 22.** *The action* $a^*\left(x,\mu,s\right)$ *that solves* $\mathbf{DSPP}_{\left(x,\mu,s\right)}$ *is unique.*

Notice that $\mathbf{DSPP}_{\left(x,\mu,s\right)}$ defines a **SPFE** operator similar to the one we used in the Bellman equation, $T: M \to M$, such that [2]:

$$\omega_j\left(x,\mu,s\right) = h_0^j\left(x,a^*\left(x,\mu,s\right),s\right) + \beta E\left[\omega_j\left(x'\left(x,\mu,s\right),\mu'\left(x,\mu,s\right),s'\right) \mid s\right] \quad \text{if } j=0,...,k, \text{ and} \tag{6.6}$$

$$\omega_j\left(x,\mu,s\right) = h_0^j\left(x,a^*\left(x,\mu,s\right),s\right) \quad \text{if } j=k+1,...,l \tag{6.7}$$

Let $M^j$ be the $j$-th projection of $M$, $j=0,...,l$. Let $T^j: M^j \to M^j$ be the "individual" operator defined by (6.6)-(6.7) under Assumption 22. Therefore it is possible to show:

**Lemma 2.** *Let Assumptions 12-16, 18, 19, 21 and 22 hold. Then* $T^j: M^j \to M^j$ *is a contraction mapping.*

By using Lemma 2, and Theorems 10 and 12, we obtain the final result:

**Theorem 13.** $\left(\mathbf{DSPP}_{\left(x,\mu,s\right)} \Rightarrow \mathbf{PP}_\mu\left(x,s\right)\right)$ *Let Assumptions 12-16, 18, 19, 21 and 22 hold. Then* $T: M \to M$ *has a unique solution* $\omega$*, which defines a value function* $W\left(x,\mu,s\right) = \mu\omega\left(x,\mu,s\right)$ *and a saddle-point policy function* $\psi$*, such that if* $(\mathbf{a}^*,\boldsymbol{\gamma}^*)$ *is generated by* $\psi$ *from* $\left(x,\mu,s\right)$*, then* $\mathbf{a}^*$ *is the unique solution to* $\mathbf{PP}_\mu$ *at* $(x,s)$*.*

For many applications, concavity is a strong assumption. Luckily, we can have unique solutions also without it. Define $\widetilde{M}$ as the set $M$ without the concavity assumption (i.e., assumption ii.). We have the following Corollary:

2 The definition of the operator $T$ is based on the fact that we can write **SPFE** as

$$\mu\omega\left(x,\mu,s\right) = \mu h_0\left(x,a^*,s\right) + \gamma^* h_1\left(x,a^*,s\right) + \beta E\left[\varphi\left(\mu,\gamma^*,s\right)\omega\left(x^{*\prime},\varphi\left(\mu,\gamma^*,s\right),s'\right) \mid s\right]$$

and also the fact that, at the solution, the following relation holds:

$$\gamma^{*j}\left\{h_1^j\left(x,a^*,s\right) + \beta E\left[\omega_j\left(x^{*\prime},\varphi\left(\mu,\gamma^*,s\right),s'\right) \mid s\right]\right\} = 0$$

**Corollary 1.** ***(Bounded returns)*** *Let Assumptions 12-16 and that for all* $(x, \mu, s)$ *Assumption 22 hold. Then* $T : \widetilde{M} \to \widetilde{M}$ *has a unique solution* $\omega$*, which defines a value function* $W(x, \mu, s) = \mu\omega(x, \mu, s)$ *and a saddle-point policy function* $\psi$*, such that if* $(\mathbf{a}^*, \gamma^*)$ *is generated by* $\psi$ *from* $(x, \mu, s)$*, then* $\mathbf{a}^*$ *is the unique solution to* $\mathbf{PP}_\mu$ *at* $(x, s)$*.*

## 6.4 HOW TO SOLVE OUR EXAMPLES WITH MM

As in the APS section, we will see how to analyze a model with MM technique. Before going into the details, it is better to have a discussion of the numerical techniques we want to use.

### 6.4.1 Numerical approaches

As we have seen, at the end of the previous section, we can get a solution of the non-standard dynamic programming problem by solving a saddle-point functional equation which looks very similar to a Bellman equation (except for the min max operator). We also showed that the operator defined by the functional equation is a contraction mapping, therefore we can solve it with an iterative procedure in the same spirit of value function iteration or Howard's improvement algorithm, adapted to the min max operator.

However, this procedure is quite slow [3]. In almost all applications the researchers did not exploit the functional equation, but instead use an approach that solves first-order conditions of the Lagrangean associated with the dynamic optimization. In this case, there are various options in terms of numerical tools: one is projection methods, and we will see an example later.

Another approach is Parameterized Expectations (PEA). The idea is quite similar to projection methods [4]. In general, we can think of expectations as a function of the state variables. PEA approximates these expectations with low order polynomials, and finds the coefficients of these polynomials by means of a nonlinear least squares regression over the functional equation we want to solve. We leave details to appendix 6.A. This method is however complicated

3 Adam and Billi (2004) solve an optimal (monetary) policy problem with value function iteration. Their problem is more complex than our simple fiscal policy example, but it is worth noticing that their code converges in around 40 hours.

4 Christian Haefke and Michael Reiter have written PEA codes that implement it as a collocation algorithm. Their codes are available on their homepage.

to implement: it rarely converges without additional tricks, and it is often very slow. We therefore will focus on projection methods.

### 6.4.2 Optimal policy

The problem was:

$$\begin{aligned} \max_{\{c_t,b_t,l_t\}_{t=0}^{\infty}} \quad & E_0 \sum_{t=0}^{\infty} \beta^t \left[ u\left(c_t\right) + v\left(l_t\right) \right] \\ s.t. \quad & c_t - \beta \frac{E_t u'\left(c_{t+1}\right)}{u'\left(c_t\right)} b_{t+1}^g = -\left(c_t + g_t\right) \frac{v'\left(c_t + g_t\right)}{u'\left(c_t\right)} - b_t^g \end{aligned}$$

First of all, define:

$$y_t \equiv \frac{-\left(c_t + g_t\right) v'\left(c_t + g_t\right) + b_t u'\left(c_t\right) - c_t u'\left(c_t\right)}{b_{t+1}} \tag{6.8}$$

Notice that we can transform it in a $\mathbf{PP}_\mu$ problem in the following way. Let $l \equiv 1$, $N_0 = \infty$ and $N_1 = 0$, $s \equiv g$, $x \equiv b$, $a \equiv (c, l)$, $p\left(x, a, s\right) \equiv l - \left(c + g\right)$, $h_0^0\left(x, a, s\right) \equiv u\left(c\right) + v\left(c + g\right)$, $h_0^1\left(x, a, s\right) \equiv u'\left(c\right)$, $h_1^0\left(x, a, s\right) \equiv h_0^0\left(x, a, s\right) - R$, $R$ big, $h_1^1\left(x, a, s\right) \equiv -y$, and $\ell\left(x, a, s\right) \equiv \frac{-\left(c+g\right)v'\left(c+g\right)+bu'(c)-cu'(c)}{y}$. Therefore we can write it as:

$$\max_{\{c_t,b_t\}_{t=0}^{\infty}} E\left[ \left\{ \sum_{t=0}^{\infty} \beta^t \mu^0 \left[ u\left(c_t\right) + v\left(c_t + g_t\right) \right] \right\} + \mu^1 \left[ u'\left(c_0\right) \right] | s \right] \quad \mathbf{PP}_\mu\mathbf{OP}$$

$$s.t. \quad b_{t+1} = \frac{-\left(c_t + g_t\right) v'\left(c_t + g_t\right) + b_t u'\left(c_t\right) - c_t u'\left(c_t\right)}{y_t}$$

$$E_t \sum_{n=1}^{\infty} \beta^n \left[ u\left(c_n\right) + v\left(c_n + g_n\right) \right] + \left[ u\left(c_t\right) + v\left(c_t + g_t\right) - R \right] \geq 0 \qquad t \geq 0 \tag{6.9}$$

$$\beta E_t \left[ u\left(c_{t+1}\right) \right] - y_t \geq 0 \qquad t \geq 0 \tag{6.10}$$

In order to find a solution, it is useful to write down the Lagrangean of $\mathbf{PP}_\mu\mathbf{OP}$. Assign Lagrange multipliers $\gamma_t^j$ to constraints (6.9) and (6.10). To simplify algebra, it better to use constraints (6.10) multiplied by $b_{t+1}$. However, notice that the constraint (6.9) is always not binding because of our interiority assumpti-

ons. Therefore, $\gamma_t^0 = 0$ for any $t$. We can consider the relaxed program where there is no the sequence of constraints (6.9). We can write the Lagrangean as:

$$\mathcal{L}\left(\left\{c_t, g_t, b_t, \mu_t, \gamma_t\right\}_{t=0}^{\infty}\right) = E\left[\sum_{t=0}^{\infty}\beta^t\left\{\mu^0\left[u\left(c_t\right)+v\left(c_t+g_t\right)\right]+\right.\right.$$
$$\left.\left.+\mu_t^1\left[u'\left(c_t\right)\right]b_t-\gamma_t^1\left[-\left(c_t+g_t\right)v'\left(c_t+g_t\right)+b_tu'\left(c_t\right)-c_tu'\left(c_t\right)\right]\right\}|s\right]$$

where $\mu_{t+1}^1 = \gamma_t^1$, with $\mu_0^1 = \mu^1 = 0$, and $\mu^0 = 1$. The easiest strategy is now to solve for the first-order conditions of the Lagrangean by using collocation method. If we take derivatives of the Lagrangean we get:

$$/c_t: \quad u'\left(c_t\right)+v'\left(c_t+g_t\right)+\mu_t^1u''\left(c_t\right)b_t-\gamma_t^1\left[-v''\left(c_t+g_t\right)-\right.$$
$$\left.-\left(c_t+g_t\right)v''\left(c_t+g_t\right)+b_tu''\left(c_t\right)-u'\left(c_t\right)-c_tu''\left(c_t\right)\right]=0 \tag{6.11}$$

$$/b_{t+1}: \quad E_t\left[\left(\mu_{t+1}^1-\gamma_{t+1}^1\right)u'\left(c_{t+1}\right)\right]=0 \tag{6.12}$$

plus all the constraints of the original problem. Given that we can apply MM, we know that the policy functions will be:

$$\left(c_t, b_{t+1}, \gamma_t^1\right) = \psi\left(b_t, g_t, \mu_t^1\right)$$

therefore we can approximate the function $\psi$ with Chebichev polynomials, substitute this approximation in the first order conditions (6.11)-(6.12) and all the constraints of the original problem, and iterate until the approximated policy function solves them.

## Exercise 6.1 Optimal policy

Modify the code for collocation over the first order conditions of the stochastic growth model to solve the Ramsey taxation problem.

### 6.4.3 Lack of commitment

The problem was:

$$\max_{\{c_t\}_{t=0}^{\infty}} E_{-1} \sum_{t=0}^{\infty} \beta^t \left(y_t - c_t\right)$$

$$s.t. \quad E_{-1} \sum_{t=0}^{\infty} \beta^t u\left(c_t\right) \geq U_0 \tag{6.13}$$

$$u\left(c_t\right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left(c_{t+j}\right) \geq u\left(y_t\right) + \beta U_{aut}, \quad t = 0, 1, \ldots \tag{6.14}$$

First of all, notice we can get rid of (6.13): this constraint only requires the principal to give a weight $\alpha$ (which depends on the minimum value $U_0$) to agent's utility in the principal's maximization problem. Therefore we can rewrite our problem as:

$$\max_{\{c_t\}_{t=0}^{\infty}} E_{-1} \sum_{t=0}^{\infty} \beta^t \left(y_t - c_t + \alpha u\left(c_t\right)\right)$$

$$s.t. \quad u\left(c_t\right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left(c_{t+j}\right) \geq u\left(y_t\right) + \beta U_{aut}, \quad t = 0, 1, \ldots$$

We can see this problem is already in the form $\mathbf{PP}_{\mu}$, by letting $l \equiv 1$, $N_0 = \infty$ and $N_1 = \infty$, $s \equiv y$, $a \equiv c$, $h_0^0\left(x, a, s\right) \equiv y - c + \alpha u\left(c\right)$, $h_0^1\left(x, a, s\right) \equiv u\left(c\right)$, $h_1^0\left(x, a, s\right) \equiv h_0^0\left(x, a, s\right) - R$, $h_1^1\left(x, a, s\right) \equiv h_0^1\left(x, a, s\right) - u\left(y\right) - \beta U_{aut}$, and using $\mu_0 = (1, 0)$. We can write it as:

$$\max_{\{c_t\}_{t=0}^{\infty}} E_{-1} \sum_{t=0}^{\infty} \beta^t \left[\mu^0 \left(y_t - c_t + \alpha u\left(c_t\right)\right) + \mu^1 u\left(c_t\right)\right]$$

$$s.t. \quad u\left(c_t\right) + \beta E_t \sum_{j=1}^{\infty} \beta^{j-1} u\left(c_{t+j}\right) \geq u\left(y_t\right) + \beta U_{aut}, \quad t = 0, 1, \ldots$$

$$E_t \sum_{n=0}^{\infty} \beta^t \left[y_{t+n} - c_{t+n} + \alpha u\left(c_{t+n}\right)\right] - R \geq 0 \tag{6.15}$$

As before, the best strategy is to go through the Lagrangean first-order conditions. Then assign a Lagrange multiplier $\gamma_t^j$. As before, the constraint (6.15) is never binding and $\gamma_t^0 = 0$ for any $t$. The Lagrangean is:

$$\mathcal{L}\left(\left\{c_t, y_t, \mu_t, \gamma_t\right\}_{t=0}^{\infty}\right) = E\sum_{t=0}^{\infty}\beta^t\left[\mu^0\left(y_t - c_t\right) + \left(\mu^0\alpha + \mu_t^1\right)u\left(c_t\right) + \right.$$
$$\left. + \gamma_t^1\left(u\left(c_t\right) - u\left(y_t\right) - \beta U_{aut}\right)\right]$$

with $\mu^0 = 1$, $\mu^1 = 0$ and $\mu_{t+1}^j = \mu_t^j + \gamma_t^j$ for $j = 0, 1$.

Take first order conditions with respect to consumption:

$$-1 + \left(\alpha + \mu_{t+1}^1\right)u'\left(c_t\right) = 0$$

Rearranging, we get:

$$\frac{1}{u'\left(c_t\right)} = \left(\alpha + \mu_{t+1}^1\right) \tag{6.16}$$

We can use collocation to solve for (6.16) and the participation constraint. Notice that participation constraint is not always binding, therefore we must solve for the two possible cases: the case of binding constraint, and the case with no binding constraint. We can do this in two steps:

1. We solve the problem without taking into account the participation constraint. We calculate the RHS of (6.14) consistent with this solution.
2. For those gridpoints such that $RHS > u\left(y_t\right) + \beta U_{aut}$, our solution is good. For those gridpoints such that $RHS < u\left(y_t\right) + \beta U_{aut}$, assume the participation constraint is binding, and re-solve the problem.

However, it is not so simple! You may need to adapt the grid iteratively, since you don't know where the set of states values where the constraint binds is.

### 6.4.4 Private information

The application of MM arguments to private information problems is quite recent. In their paper, Marcet and Marimon say the theory is still not applicable to private information problems. Recently, however, various attempts have being made to extend their theory to such models. The main contributions on this topic are Sleet and Yeltekin (2010) for unobservable shocks, and Mele (2014) for unobservable actions.

*Endowment is private information*

For this problem, we use the methodology of Sleet and Yeltekin (2010). Remember our maximization problem:

$$\max_{\left\{\tau_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[-\tau_t\left(y^t\right)\right] \pi\left(y^t|y_{-1}\right)$$

$$s.t. \qquad \sum_{t=0}^{\infty} \sum_{y^t \in y^{t+1}} \beta^t u\left(c_t\left(y^t\right)\right) \pi\left(y^t|y_{-1}\right) \geq U_0 \tag{6.17}$$

$$\beta^t \left[u\left(c_t\left(y^{t-1}, \overline{y}_i\right)\right) + \beta U_{t+1}\left(y^{t-1}, \overline{y}_i\right) - \right.$$

$$\left. -u\left(c_t\left(y^{t-1}, \overline{y}_{i-1}\right)\right) - \beta U_{t+1}\left(y^{t-1}, \overline{y}_{i-1}\right)\right] \geq 0 \tag{6.18}$$

$$\forall y^{t-1}, \quad \forall \overline{y}_i, \quad i = 2, \ldots, N$$

where

$$U_{t+1}\left(y^{t-1}, \overline{y}_i\right) \equiv \sum_{j=1}^{\infty} \sum_{y^{t+j} \in Y^{t+j+1}} \beta^{j-1} u\left(c_{t+j}\left(y^{t-1}, \overline{y}_i, y^{t+j}\right)\right) \pi\left(y^{t+j}|y_{-1}\right)$$

First of all, also in this case we can eliminate the constraint (6.17) with the same trick as in lack of commitment:

$$\max_{\left\{c_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[y_t - c_t\left(y^t\right) + \alpha u\left(c_t\left(y^t\right)\right)\right] \pi\left(y^t|y_{-1}\right)$$

$$s.t. \quad \beta^t \left[u\left(c_t\left(y^{t-1}, \overline{y}_i\right)\right) + \beta U_{t+1}\left(y^{t-1}, \overline{y}_i\right) - \right.$$

$$\left. -u\left(c_t\left(y^{t-1}, \overline{y}_{i-1}\right)\right) - \beta U_{t+1}\left(y^{t-1}, \overline{y}_{i-1}\right)\right] \geq 0$$

$$\forall y^{t-1}, \quad \forall \overline{y}_i, \quad i = 2, \ldots, N$$

We can transform it in a $\mathbf{PP}_\mu$ problem in the following way. Let $l \equiv 1$, $N_0 = \infty$ and $N_1 = \infty$, $s \equiv y$, $a \equiv c$, $h_0^0(x, a, s) \equiv y - c + \alpha u(c)$, $h_0^1(x, a, s) \equiv u(c)$, $h_1^0(x, a, s) \equiv h_0^0(x, a, s) - R$, $h_1^1(x, a, s) \equiv h_0^1(x, a, s)$.

Also in this example assume that shocks are i.i.d., and define:

$$\begin{aligned} p_{ik} &\equiv \frac{\pi(\overline{y}_k)}{\pi(\overline{y}_i)} \\ \varsigma_t^j(y^{t-1}, \overline{y}_i) &\equiv \gamma_t^j(y^{t-1}, \overline{y}_i) \\ \varsigma_t^j(y^{t-1}, \overline{y}_k) &\equiv -\gamma_t^j(y^{t-1}, \overline{y}_i) p_{ik} \end{aligned}$$

where $\gamma_t^j(y^{t-1}, \overline{y}_i)$ is the Lagrange multiplier attached to constraint $j = 0, 1$ as usual, and in particular $\gamma_t^1(y^{t-1}, \overline{y}_i)$ is attached to $DIC_t\left(y^{t-1}, \overline{y}_i; \overline{y}_{i-1}\right)$. We can due our standard algebra, and by defining $\mu_t^j(y^{t-1}, \overline{y}_i) \equiv \mu_{t-1}^j(y^{t-1}) + \varsigma_t^j(y^{t-1}, \overline{y}_i)$ for any $\overline{y}_i \in Y$ with $\mu_{-1}^0(y_{-1}) = \mu_{-1}^0 = 1$, and $\mu_{-1}^1(y_{-1}) = \mu_{-1}^1 = 0$, we can write the Lagrangean as:

$$\begin{aligned} \mathcal{L}\left(c^\infty, \gamma^\infty; \alpha\right) = \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \Bigg[ & \mu^0 \left( y_t - c_t\left(y^t\right) \right) + \\ & + \left( \mu^0 \alpha + \mu_t^1\left(y^t\right) \right) u\left(c_t\left(y^t\right)\right) \Bigg] \pi^t\left(y^t\right) \end{aligned}$$

Now, as long as the problem is convex, we can use KT first-order conditions and solve it with collocation.

*Hidden effort*

The problem of the planner is:

$$\begin{aligned} \max_{\left\{c_t\left(y^t\right)\right\}_{t=0}^{\infty}} & \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[ y_t - c_t\left(y^t\right) \right] \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \\ s.t. \quad & \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[ u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right) \right] \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \geq U_0 \\ & \left\{ a_t\left(y^t\right) \right\}_{t=0}^{\infty} \in \arg \max_{\left\{a_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \sum_{y^t \in Y^{t+1}} \beta^t \left[ u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right) \right] \times \\ & \qquad \times \pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \end{aligned} \tag{6.19}$$

We follow Mele (2014) and therefore we need to restrict the class of models we can solve. In particular, we need to assume that we can use a first-order

approach: instead of using (6.19), we use the first-order conditions of that optimization problem with respect to effort. In order to guarantee that we get the same solution as in the original problem, we assume that two conditions are satisfied. They are known as Rogerson conditions.

**Condition 3** (Monotone Likelihood-Ratio Condition (MLRC)). $\widehat{a} \leq \widehat{\widehat{a}} \Longrightarrow \frac{\pi\left(\overline{y}_s|\widehat{a}\right)}{\pi\left(\overline{y}_s|\widehat{\widehat{a}}\right)}$ *is nonincreasing in* $s$.

The above property can be restated in a simpler way: if $\pi\left(\cdot\right)$ is differentiable, then MLRC is equivalent to $\frac{\pi_a\left(\overline{y}_s|a\right)}{\pi\left(\overline{y}_s|a\right)}$ being nondecreasing in $s$ for any $a$, where $\pi_a\left(\overline{y}_s \mid a\right)$ is the derivative of $\pi\left(\cdot\right)$ with respect to $a$. An important consequence of the MLRC is the following: let $F\left(\cdot\right)$ be the cumulative distribution function of $\pi\left(\cdot\right)$; then MLRC implies that the density function $F'\left(\overline{y}_s \mid a\right)$ is nonpositive for any $s$ and every $a$. Therefore, more effort implies a first order stochastic dominance shift of the distribution (see Rogerson (1985)).

**Condition 4** (Convexity of the Distribution Function Condition (CDFC)). $F''\left(\overline{y}_s \mid a\right)$ *is nonnegative for any* $s$ *and every* $a$.

This condition combined with MLRP implies that more effort increases the probability of observing a high output, but as we get closer to the highest realizations this effect becomes smaller and smaller: CDFC is a sort of decreasing marginal returns for effort in stochastic terms. The two conditions make sure the agent's problem is strictly concave, and therefore first-order conditions are necessary and sufficient to characterize the optimal effort. We can therefore rewrite our problem as:

$$\max_{\left\{c_t\left(y^t\right),a_t\left(y^t\right)\right\}_{t=0}^{\infty}} \sum_{t=0}^{\infty}\sum_{y^t\in Y^{t+1}} \beta^t\left[y_t - c_t\left(y^t\right)\right]\pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right)$$

$$s.t. \quad \sum_{t=0}^{\infty}\sum_{y^t\in Y^{t+1}} \beta^t\left[u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)\right]\pi\left(y^t \mid a^{t-1}\left(y^{t-1}\right)\right) \geq U_0 \tag{6.20}$$

$$v'\left(a_t\left(y^t\right)\right) = \sum_{j=1}^{\infty}\beta^j \sum_{y^{t+j}|y^t} \frac{\pi_a\left(y_{t+1} \mid y_t, a_t\left(y^t\right)\right)}{\pi\left(y_{t+1} \mid y_t, a_t\left(y^t\right)\right)} \times$$

$$\times\left[u\left(c_{t+j}\left(y^{t+j}\right)\right) - v\left(a_{t+j}\left(y^{t+j}\right)\right)\right]\pi\left(y^{t+j} \mid y^t, a^{t+j-1}\left(y^{t+j-1} \mid y^t\right)\right)$$

It is easy to show that (6.20) must be binding in the optimum. This allows us to reinterpret the planner problem as a social welfare function maximization, where the planner assigns weights to principal's and agent's utilities:

$$W^{SWF}(s_0) = \max_{\left\{a_t\left(y^t\right),c_t\left(y^t\right)\right\}_{t=0}^{\infty}} \mu_0^0 \sum_{t=0}^{\infty}\sum_{y^t}\beta^t\left[y\left(y_t\right)-c_t\left(y^t\right)\right]\pi\left(y^t \mid s_0, a^{t-1}\left(s^{t-1}\right)\right) +$$
$$+\mu_0^1\sum_{t=0}^{\infty}\sum_{y^t}\beta^t\left[u\left(c_t\left(y^t\right)\right)-v\left(a_t\left(y^t\right)\right)\right]\pi\left(y^t \mid s_0, a^{t-1}\left(s^{t-1}\right)\right)$$

$$s.t. \quad v'\left(a_t\left(y^t\right)\right) = \sum_{j=1}^{\infty}\beta^j\sum_{y^{t+j}|y^t}\frac{\pi_a\left(y_{t+1} \mid y_t, a_t\left(y^t\right)\right)}{\pi\left(y_{t+1} \mid y_t, a_t\left(y^t\right)\right)}\times$$
$$\times\left[u\left(c_{t+j}\left(y^{t+j}\right)\right)-v\left(a_{t+j}\left(y^{t+j}\right)\right)\right]\pi\left(y^{t+j} \mid y^t, a^{t+j-1}\left(y^{t+j-1} \mid y^t\right)\right)$$

Notice that this is our version of problem $\mathbf{PP}_\mu$. Let $\beta^t\lambda_t\left(y^t\right)\pi\left(y^t \mid s_0, a^{t-1}\left(y^{t-1}\right)\right)$ be the Lagrange multiplier associated to each ICC. I can therefore write the Lagrangian as:

$$L\left(y_0, c^\infty, a^\infty, \lambda^\infty\right) = \sum_{t=0}^{\infty}\sum_{y^t}\beta^t\left\{\mu_0^0\left(y_t - c_t\left(y^t\right)\right)+\right.$$
$$\left.+\mu_0^1\left(u\left(c_t\left(y^t\right)\right)-v\left(a_t\left(y^t\right)\right)\right)\right\}\pi\left(y^t \mid s_0, a^{t-1}\left(y^{t-1}\right)\right)+$$
$$-\sum_{t=0}^{\infty}\sum_{y^t}\beta^t\lambda_t\left(y^t\right)\left\{v'\left(a_t\left(y^t\right)\right)-\sum_{j=1}^{\infty}\beta^j\sum_{y^{t+j}|y^t}\frac{\pi_a\left(y_{t+1} \mid y_t, a_t\left(y^t\right)\right)}{\pi\left(y_{t+1} \mid y_t, a_t\left(y^t\right)\right)}\times\right.$$
$$\left.\times\left[u\left(c_{t+j}\left(y^{t+j}\right)\right)-v\left(a_{t+j}\left(y^{t+j}\right)\right)\right]\pi\left(y^{t+j} \mid y^t, a^{t+j-1}\left(y^{t+j-1} \mid y^t\right)\right)\right\}\times$$
$$\times\pi\left(y^t \mid y_0, a^{t-1}\left(y^{t-1}\right)\right)$$

The Lagrangean can be manipulated with simple algebra to get the following expression:

$$L\left(y_0, c^\infty, a^\infty, \lambda^\infty\right) = \sum_{t=0}^{\infty} \sum_{y^t} \beta^t \left\{ \mu_0^0 \left( y_t - c_t\left(y^t\right)\right) + \right.$$
$$+ \mu_t^1\left(y^t\right)\left[u\left(c_t\left(y^t\right)\right) - v\left(a_t\left(y^t\right)\right)\right] +$$
$$\left. - \lambda_t\left(y^t\right) v'\left(a_t\left(y^t\right)\right)\right\} \pi\left(y^t \mid s_0, a^{t-1}\left(y^{t-1}\right)\right)$$

where

$$\mu_t^1\left(y^{t-1}, y_t\right) = \mu_0^1 + \sum_{i=0}^{t-1} \lambda_i\left(y^i\right) \frac{\pi_a\left(y_{i+1} \mid y_i, a_i\left(y^i\right)\right)}{\pi\left(y_{i+1} \mid y_i, a_i\left(y^i\right)\right)}$$

The intuition is simple. For any $y^t$, the expression $\lambda_t\left(y^t\right) \frac{\pi_a\left(y_{t+1}|y_t, a_t\left(y^t\right)\right)}{\pi\left(y_{t+1}|y_t, a_t\left(y^t\right)\right)}$ is a *planner's promise* about how much she will increase the weight of agent's welfare in the future, depending on which realization of state $y_{t+1}$ is observed. By keeping track of all $\lambda$'s and $\frac{\pi_a}{\pi}$'s realized in the past, $\mu_t^1\left(y^t\right)$ summarizes all the promises made by the planner in previous periods. In this framework, there is straightforward interpretation of $\mu_t^1\left(y^t\right)$: it is the *Pareto-Negishi weight* of the agent's lifetime utility, that evolves *endogenously* in order to track agent's effort with the following recursive law of motion:

$$\mu_{t+1}^1\left(y^t, \widehat{y}_s\right) = \mu_t^1\left(y^t\right) + \lambda_t\left(y^t\right) \frac{\pi_a\left(y_{t+1} = \widehat{y}_s \mid y_t, a_t\left(y^t\right)\right)}{\pi\left(y_{t+1} = \widehat{y}_s \mid y_t, a_t\left(y^t\right)\right)} \quad \forall \widehat{y}_s \in Y$$
$$\mu_0^1\left(y^0\right) = \mu_0^1$$

To better understand the role of $\mu_t^1\left(y^t\right)$, let us assume there are only two possible realizations of the state of nature: $y_t \in \left\{y^L, y^H\right\}$ At time 0, the weight is equal to $\mu_0^1$. In period 1, given our assumption on the likelihood ratio, the Pareto-Negishi weight is higher than $\mu_0^1$ if the principal observes $s_H$, while it is lower than $\mu_0^1$ if she observes $s_L$. Therefore the agent is rewarded by a higher weight in the social welfare function of the principal (i.e., the principal

cares more about him) if a good state of nature is observed, while it is punished by a lower weight (i.e., the principal cares less about him) if a bad state of nature happens.

*How to solve it numerically*

For simplicity, I assume that the Markov process has only two possible realizations ($y^L < y^H$). I also assume there is no persistence across time (i.e., the state is i.i.d.), and I use the simpler notation $\pi(a_t) = \pi\left(y_{t+1} = y^H \mid a_t\right)$. The numerical procedure is a collocation algorithm (see Judd (1998)) over the first-order conditions of the Lagrangean. From the recursive formulation we know that policy functions depend on the natural states of the problem and on the costates (i.e., Pareto weights) that come out from the Lagrangean approach. Let $\varsigma$ be the vector of allocations, $\chi$ be the vector of Lagrange multipliers, $x \in X$ be the vector of natural states, and $\theta \in \Theta$ be the vector of costates, and define $R\left(y, \varsigma, \chi, x, \theta\right)$ as the objective function in the Lagrangean, and $r\left(y, \varsigma, \chi, x, \theta\right)$ as the instantaneous utility function for the agent. We therefore proceed as follows:

1. Fix $\mu_0^1$ and define a discrete grid $G \subset X \times \Theta$ for natural states and costates.

2. Approximate policy functions for allocations $\varsigma$ and Lagrange multipliers $\chi$, the value function of the principal $J$ and the continuation value of the agent $U$ using cubic splines (or Chebychev polynomials, depending on the application), and set initial conditions for the approximation parameters

3. For any $(y, x, \theta) \in G$, use a nonlinear solver [5] to solve for the Lagrangean first order conditions and the following two equations for the continuation value $U$ and the value function $J$:

$$U(y,x,\theta) = r(y,\varsigma,\chi,x,\theta) + \\ + \beta\left[\pi(a)\,U\left(y^H, x'^H, \theta'^H\right) + (1-\pi(a))\,U\left(y^L, x'^L, \theta'^L\right)\right] \tag{6.21}$$

$$J(y,x,\theta) = R(y,\varsigma,\chi,x,\theta) \\ + \beta\left[\pi(a)\,J\left(y^H, x'^H, \theta'^H\right) + (1-\pi(a))\,J\left(y^L, x'^L, \theta'^L\right)\right] \tag{6.22}$$

Steps 1-3 are applied first to a coarse grid and then we get higher accuracy by applying steps 1-3 to a finer grid. In general, a good approximation is obtained with few gridpoints.

*Repeated moral hazard*

In order to make the algorithm clear, I provide a detailed example of the procedure in the case of a standard repeated moral hazard setup. I simplify the notation by writing a generic variable as $x_t$ instead of $x_t\left(y^t\right)$. I assume that the income process has two possible realizations ($y^L$ and $y^H$). I also assume there is no persistence across time (i.e., the state is i.i.d.), and I use the simpler notation $\pi(a_t) = \pi\left(y_{t+1} = y^H \mid a_t\right)$.

The Lagrangean becomes:

$$L = E_0^a \sum_{t=0}^{\infty} \beta^t \left\{(y_t - c_t) + \mu_t^1\left[u(c_t) - v(a_t)\right] - \lambda_t v'(a_t)\right\}$$

with

$$\begin{aligned} \mu_{t+1}^{1,H} &= \mu_t^1 + \lambda_t \frac{\pi_a(a_t)}{\pi(a_t)} \\ \mu_{t+1}^{1,L} &= \mu_t^1 - \lambda_t \frac{\pi_a(a_t)}{1-\pi(a_t)} \\ \mu_0^1\left(y^0\right) &= \mu_0^1 \end{aligned}$$

5 In all applications presented in this paper, I use a version of the Broyden algorithm coded by Michael Reiter.

where $E_t^a$ is the expectation operator over histories induced by the probability distribution $\pi(a_t)$. The first-order conditions can be rewritten as

$$c_t: \quad u'(c_t) = \frac{1}{\mu_t^1} \tag{6.23}$$

$$\begin{aligned} a_t: \quad 0 = & -\lambda_t v''(a_t) - \mu_t^1 v'(a_t) + \\ & + \pi_a(a_t)\beta E_{t+1}^a \left\{ \sum_{j=1}^{\infty} \beta^{j-1} \left\{ \left(y_{t+j} - c_{t+j}\right) - \lambda_{t+j} v'\left(a_{t+j}\right) + \right. \right. \\ & \left. \left. + \mu_{t+j}^1 \left[ u\left(c_{t+j}\right) - v\left(a_{t+j}\right) \right] \right\} \mid y_{t+1} = y^H \right\} + \\ & - \pi_a(a_t)\beta E_{t+1}^a \left\{ \sum_{j=1}^{\infty} \beta^{j-1} \left\{ \left(y_{t+j} - c_{t+j}\right) - \lambda_{t+j} v'\left(a_{t+j}\right) + \right. \right. \\ & \left. \left. + \mu_{t+j}^1 \left[ u\left(c_{t+j}\right) - v\left(a_{t+j}\right) \right] \right\} \mid y_{t+1} = y^L \right\} + \\ & + \beta\lambda_t \pi(a_t) \frac{\partial\left(\frac{\pi_a(a_t)}{\pi(a_t)}\right)}{\partial a_t} \left[ u(c_{t+1}) - v(a_{t+1}) \mid y_{t+1} = y^H \right] + \\ & + \beta\lambda_t \left(1 - \pi(a_t)\right) \frac{\partial\left(\frac{-\pi_a(a_t)}{1-\pi(a_t)}\right)}{\partial a_t} \left[ u(c_{t+1}) - v(a_{t+1}) \mid y_{t+1} = y^L \right] \end{aligned} \tag{6.24}$$

and

$$\begin{aligned} \lambda_t: \quad 0 = & -v'(a_t) + \pi_a(a_t)\beta E_{t+1}^a \left\{ \sum_{j=1}^{\infty} \beta^{j-1} \left[ u\left(c_{t+j}\right) - v\left(a_{t+j}\right) | y_{t+1} = y^H \right] \right\} - \\ & - \pi_a(a_t)\beta E_{t+1}^a \left\{ \sum_{j=1}^{\infty} \beta^{j-1} \left[ u\left(c_{t+j}\right) - v\left(a_{t+j}\right) | y_{t+1} = y^L \right] \right\} \end{aligned} \tag{6.25}$$

Notice that

$$J\left(y^i,\mu_{t+1}^{1,i}\right)=E_{t+1}^{a}\left\{\sum_{j=1}^{\infty}\beta^{j-1}\left\{\left(y_{t+j}-c_{t+j}\right)-\lambda_{t+j}v'\left(a_{t+j}\right)+\right.\right.$$
$$\left.\left.+\mu_{t+j}^{1}\left[u\left(c_{t+j}\right)-v\left(a_{t+j}\right)\right]\right\}\mid y_{t+1}=y^i\right\}$$
$$i=H,L$$

and

$$U\left(y^i,\mu_{t+1}^{1,i}\right)=E_{t+1}^{a}\left\{\sum_{j=1}^{\infty}\beta^{j-1}\left[u\left(c_{t+j}\right)-v\left(a_{t+j}\right)|y_{t+1}=y^i\right]\right\}$$
$$i=H,L$$

Therefore we can rewrite (6.24) and (6.25) as

$$\begin{aligned}a_t\ :\quad 0=&-\lambda_t v''\left(a_t\right)-\mu_t^1 v'\left(a_t\right)+\beta\pi_a\left(a_t\right)\left[J\left(y^H,\mu_{t+1}^1\right)-J\left(y^L,\mu_{t+1}^1\right)\right]+\\&+\beta\lambda_t\left\{\pi\left(a_t\right)\frac{\partial\left(\frac{\pi_a(a_t)}{\pi(a_t)}\right)}{\partial a_t}\left[u\left(c_{t+1}\right)-v\left(a_{t+1}\right)\mid y_{t+1}=y^H\right]+\right.\\&\left.+\left(1-\pi\left(a_t\right)\right)\frac{\partial\left(\frac{-\pi_a(a_t)}{1-\pi(a_t)}\right)}{\partial a_t}\left[u\left(c_{t+1}\right)-v\left(a_{t+1}\right)\mid y_{t+1}=y^L\right]\right\}\end{aligned}\tag{6.26}$$

$$\lambda_t:\quad 0=-v'\left(a_t\right)+\beta\pi_a\left(a_t\right)\left[U\left(y^H,\mu_{t+1}^{1,H}\right)-U\left(y^L,\mu_{t+1}^{1,L}\right)\right]\tag{6.27}$$

I fix $\mu_0^1$ and I choose a discrete grid for $\mu_t^1$ that contains $\mu_0^1$. I approximate with cubic splines $a$, $\lambda$, $U$ and $J$ on each grid node. I get consumption directly from $\mu^1$ by using (6.23): $c=u'^{-1}\left(\left(\mu^1\right)^{-1}\right)$. Finally I solve the system of nonlinear equations that includes (6.26), (6.27), (6.21) and (6.22). The code is included in the material.

## 6.A PARAMETERIZED EXPECTATIONS ALGORITHM (PEA)

We can think of PEA in very general terms. Imagine an economy which can be described by a vector of variables $z_t$ and a vector of exogenous shocks $u_t$.

The process $z_t, u_t$ satisfy various Euler equations, and constraints, and other conditions that we can describe as:

$$g\left(E_t\left[\phi\left(z_{t+1}, z_t\right) \mid x_t\right], z_t, z_{t-1}, u_t\right) = 0 \tag{6.28}$$

where $x_t$ is a subset of $(z_{t-1}, u_t)$ and a vector of state variables. Moreover, we need to assume that the conditional expectations in (6.28) is recursive in the sense that:

$$E_t\left[\phi\left(z_{t+1}, z_t\right) \mid x_t\right] = \mathcal{E}\left(x_t\right) \tag{6.29}$$

This is a more general class than the one we are interested in, but all the models we have seen in our Lectures satisfy these assumptions.

The essence of PEA is an approximation of the conditional expectations in (6.29) by a polynomial or a similar functional form (splines or finite methods are ok too). Call this approximation $\psi\left(\beta; x\right)$, where $\beta$ is a vector of coefficient for the polynomial approximation. We can rewrite (6.28) as:

$$g\left(\psi\left(\beta; x_t\left(\beta\right)\right), z_t\left(\beta\right), z_{t-1}\left(\beta\right), u_t\right) = 0 \tag{6.30}$$

We can solve for a good approximation (i.e., for a vector $\beta$ that solves the previous equation with accuracy) following this algorithm:

1. Write the system of equations (6.28) such that it is invertible in its second argument. Find a set of state variables that satisfy (6.29). Replace the true conditional expectation by the parameterized function $\psi\left(\beta; \cdot\right)$ and get (6.30). Fix initial conditions $(z_0, u_0)$. Generate a series $\{u_t\}_{t=0}^T$ with a random number generator for T large.

2. Given $\beta$, recursively calculate $\left\{z_t\left(\beta\right)\right\}_{t=0}^T$ using (6.30) and the series $\{u_t\}_{t=0}^T$ generated in step 1.

3. Find a new vector of parameters $G\left(\beta\right)$ that solves the following non-linear least squares problem:

$$G\left(\beta\right) = \arg\min_{\xi} \frac{1}{T}\sum_{t=0}^{T}\left\|\phi\left(z_{t+1}\left(\beta\right), z_t\left(\beta\right)\right) - \psi\left(\xi; x_t\left(\beta\right)\right)\right\| \tag{6.31}$$

which is easy to solve if we perform a NLLS regression of $\phi\left(z_{t+1}\left(\beta\right), z_t\left(\beta\right)\right)$ on $\psi\left(\xi; x_t\left(\beta\right)\right)$.

4. Iterate on steps 2-3 until you get the fixed point of $G(\cdot)$, i.e. until $G\left(\beta_f\right) \approx \beta_f$ in numerical terms.

This algorithm can be applied to our examples. The main problem is that convergence is very slow, and sometimes difficult to achieve. My personal experience is that collocation methods outperforms PEA in many dimensions, and therefore my advice is to use collocation for all these problems.

There is a very easy treatment of PEA by Marcet and Lorenzoni (1998), which is freely downloadable online. There is also a version of PEA in which you use so called moving bounds, i.e. the range of parameters among which you search is limited by bounds that shrink overtime. This helps to keep the numerical problem well behaved, consequently making simpler to achieve convergence. This algorithm was developed by Maliar and Maliar (2003), and the MATLAB code is available on their website.

# BIBLIOGRAPHY


Abreu, D., D. Pearce, and E. Stacchetti (1990). Toward a theory of discounted repeated games with imperfect monitoring. *Econometrica 58*(5), pp. 1041–1063.

Braun, T., L. M. Korber, and Y. Waki (2012). Some unpleasant properties of log-linearized solutions when the nominal rate is zero. Federal Reserve Bank of Atlanta working paper 2012-5a.

Chang, Y. and S.-B. Kim (2007). Heterogeneity and aggregation: Implications for labor-market fluctuations. *American Economic Review 97*(5), 1939–1956.

Judd, K. L. (1992, December). Projection methods for solving aggregate growth models. *Journal of Economic Theory 58*(2), 410–452.

Judd, K. L. (1998). *Numerical Methods in Economics*. MIT Press.

Judd, K. L., L. Maliar, and S. Maliar (2012, November). Merging Simulation and Projection Approaches to Solve High-Dimensional Problems. NBER Working Papers 18501, National Bureau of Economic Research, Inc.

Judd, K. L., S. Yeltekin, and J. Conklin (2003). Computing supergame equilibria. *Econometrica 71*(4), 1239–1254.

Ljungqvist, L. and T. J. Sargent (2012). *Recursive Macroeconomic Theory*. MIT Press.

Maliar, L. and S. Maliar (2003, January). Parameterized Expectations Algorithm and the Moving Bounds. *Journal of Business & Economic Statistics 21*(1), 88–92.

Maliar, S., L. Maliar, and K. Judd (2011, February). Solving the multi-country real business cycle model using ergodic set methods. *Journal of Economic Dynamics and Control 35*(2), 207–228.

Malin, B. A., D. Krueger, and F. Kubler (2011). Solving the multi-country real business cycle model using a smolyak-collocation method. *Journal of Economic Dynamics and Control 35*(2), 229 – 239. Computational Suite of Models with Heterogeneous Agents II: Multi-Country Real Business Cycle Models.

Marcet, A. and G. Lorenzoni (1998, June). Parameterized expectations approach; Some practical issues. Economics Working Papers 296, Department of Economics and Business, Universitat Pompeu Fabra.

Marcet, A. and R. Marimon (2017). Recursive contracts. Technical report.

Mele, A. (2014). Repeated moral hazard and recursive Lagrangeans. *Journal of Economic Dynamics and Control 42*(C), 69–85.

Miranda, M. J. and P. L. Fackler (2004). *Applied Computational Economics and Finance*, Volume 1 of *MIT Press Books*. The MIT Press.

Rogerson, W. P. (1985, November). The First-Order Approach to Principal-Agent Problems. *Econometrica 53*(6), 1357–67.

Sleet, C. and S. Yeltekin (2010). The recursive lagrangian method: Discrete time. *Manuscript (available at http://citeseerx.ist.psu.edu/viewdoc/summary*.

Sleet, C. and Ş. Yeltekin (2016, Jun). On the computation of value correspondences for dynamic games. *Dynamic Games and Applications 6*(2), 174–186.

Stokey, N., R. J. Lucas, and E. Prescott (1989). *Recursive methods in economic dynamics*. Harvard University Press.