UNIVERSITY OF
CAMBRIDGE

Cambridge Working Papers in Economics

Inequality Constraints and Euler Equation based Solution Methods

Pontus Rendahl

Forthcoming in the Economic Journal

CWPE 1320

Inequality Constraints and Euler Equation Based Solution Methods*

Forthcoming in the Economic Journal

Pontus Rendahl

October 23, 2013

Abstract

Solving dynamic models with inequality constraints poses a challenging problem for two major reasons: dynamic programming techniques are reliable but often slow, while Euler equation based methods are faster but have problematic or unknown convergence properties. This paper attempts to bridge this gap. I show that a common iterative procedure on the first-order conditions – usually referred to as *time iteration* – delivers a sequence of approximate policy functions that converges to the true solution under a wide range of circumstances. These circumstances extend to a large set of endogenous and exogenous state variables as well as a very broad spectrum of occasionally binding constraints.

1. Introduction

Dynamic models with inequality constraints are of considerable interest to a wide range of economists. The workhorse consumption-savings framework, for instance, is commonly augmented to include liquidity constraints that limit the extent to which agents can borrow; non-cooperative dynamic games frequently observe enforcement constraints that bound the set of subgame perfect equilibria; and in the wake of the recent financial crisis, there has been a surge of interest in models incorporating collateral constraints that may propagate shocks by way of the "financial accelerator".[1]

*University of Cambridge, Faculty of Economics, Austin Robinson Building, Sidgwick Avenue, CB3 9DD, Cambridge, United Kingdom. Email: pontus.rendahl@gmail.com. This paper is a substantially revised version of a chapter in my Ph.D dissertation written at the European University Institute, previously circulated under the title "Inequality Constraints in Recursive Economies". The author would like to thank Wouter den Haan, Dirk Krueger, Albert Marcet, Karel Mertens, Morten Ravn, Sanne Zwart, and two anonymous referees for helpful comments and suggestions. The usual disclaimer applies.

[1]See, for instance, Deaton (1991), Aiyagari (1994), and Ludvigson and Michaelides (2001) for examples of liquidity constraints; Kehoe and Perri (2002), Cooley *et al.* (2004), and Jermann and Quadrini (2012) of enforcement constraints; and Gertler and Kiyotaki (2010), Gertler and Karadi (2011), and de Groot (2011) of collateral constraints. Studies that incorporate collateral constraints frequently assume that these always bind in order to simplify computation. The "financial accelerator" is commonly attributed to Kiyotaki and Moore (1997) and Bernanke *et al.* (1999).

Yet, solving dynamic models with occasionally binding inequality constraints is no trivial matter. Dynamic programming techniques are reliable but often slow, while Euler equation based methods are known to be faster but have problematic or unknown convergence properties.[2] This paper aims to bridge this gap.

I show that a common iterative procedure applied on the Euler equation – *time iteration* – delivers a sequence of approximate policy functions that converges to the true solution under a wide range of circumstances. The proposition extends to a large set of endogenous and exogenous state variables, and includes a broad spectrum of occasionally binding constraints.[3] I exemplify the potential advantages of the method to alternative techniques by solving a simple real business cycle model with irreversible investments (cf. McGrattan (1996); Christiano and Fisher (2000)). The proposed method is generally faster than state-of-the-art dynamic programming algorithms, and at least equally accurate.

In the context of dynamic programming, dealing with occasionally binding constraints is generally straightforward. In one popular approach, the state- *and* the choice space is confined to belong to a discrete grid, which is (almost trivially) delimited to rule out any violations of the constraint set (e.g. Imrohoroğlu (1989); Hansen and Imrohoroğlu (1992)). The resulting procedure has the advantage of being simple, robust, and essentially arbitrarily accurate, but also comes at a very high cost in terms of computational power (Santos and Vigo-Aguiar, 1998).

To ease this computational burden, it has become increasingly common to treat the choice set as (if) convex (e.g. Johnson *et al.* (1993); Krusell and Smith (1998)). Function values in-between grid points are then evaluated using some interpolation, or approximation, routine, and the maximisation step is carried out by constrained optimisation. Whereas these improvements have led to some significant gains in both speed and accuracy (e.g. Judd and Solnick (1994)), they are still not without complications of their own. An efficient implementation requires the approximation method to preserve certain desirable properties of the value function, such as differentiability and concavity, which adds complexity to the computations. And while most numerical optimisation routines are far more reliant on the slope- rather than the level of the value function, dynamic programming infers the former from an approximation of the latter, which leaves the procedure vulnerable to the possibility of self-propagating errors.[4] A slope inferred from an approximation of

[2] To be clear, *dynamic programming techniques* refers to the method of successive approximations of the value function, with its various numerical implementations.

[3] *Occasionally binding-* or *inequality* constraints are used interchangeably throughout this paper.

[4] It should be noted here that the choice of approximation- and optimisation method is not independent. For instance, if one would use a gradient based optimisation routine, which is normally very fast, it would also be advisable to employ an (at least) once continuously differentiable and concave approximation method, such as a shape-preserving spline (see for instance Judd and Solnick (1994) and Judd (1998)). Alternatively, a search-based optimisation routine, which instead is quite slow, would leave greater freedom in terms of suitable approximation methods.

a function can be orders worse than an approximation of the slope of the function, and errors may easily accumulate.[5]

Parts of these difficulties can be circumvented by operating directly on the Euler equation – or more generally – on the first-order conditions.[6] The optimisation step is then replaced by a collection of nonlinear equations, and the choice of approximation method is given a much larger degree of freedom. A monotone and continuous approximation of the slope of the value function, for instance, is isomorphic to a once continuously differentiable and concavity preserving approximation of the level of the value function, but comes at a much smaller computational cost. However, these approaches do not come without their own issues. Euler equation based methods often have problematic or unknown convergence properties and, without an educated initial guess for the optimal policy functions, convergence may fail.[7]

This paper addresses some of these concerns. I show that any element in the sequence of approximate value functions defined by dynamic programming is differentiable when a general class of inequality constraints is considered, and I provide analytical expressions of their respective derivatives. Using these theoretical insights, an iterative procedure on the first-order conditions, commonly known as time iteration, is derived. Given that this procedure is equivalent to value function iteration, it is, under mild initial conditions, a globally convergent method of finding the equilibrium functions for recursively defined, Pareto optimal problems. And due to the concavity of the problem, this turns out to be a very convenient and efficient technique from a computational perspective.

The ideas developed in this paper relate foremost to those of Coleman (1989; 1990; 1991), and Deaton (1991) and Deaton and Laroque (1992). In influential work, Coleman (1989; 1990; 1991) argues that a certain iterative procedure applied on the Euler equation converges to the true solution of an infinite horizon problem. The proposed procedure, which Coleman refers to as "policy function iteration", is identical to that explored in this paper, but the results are obtained using a different approach and, therefore, also carry different implications.[8] In particular, Coleman shows that under the right conditions the Euler equation defines a "monotone map", and Tarski's fixed-point

[5] See for instance Judd (1998), p. 213 and p. 438 for a discussion of these issues.

[6] See Bizer and Judd (1989), Coleman (1990), den Haan and Marcet (1990), and Baxter (1991) for early applications of this approach. Davig (2004), Davig and Leeper (2007), Kumhof and Ranciere (2011), and Malin *et al.* (2011) provide some recent examples. See McGrattan (1996), Judd (1998), and Christiano and Fisher (2000) for a detailed discussion.

[7] Christiano and Fisher (2000), for instance, use the solution to a log-linearised version of their problem as an initial guess for the policy function. Despite this, some of the algorithms they explore fail to converge.

[8] Time iteration, or policy function iteration, can be described as the solution to a finite horizon problem using the first-order conditions while letting the horizon approach infinity. As in Judd (1998), I prefer the term "time iteration" to "policy function iteration" to avoid confusion with Howard's improvement algorithm (Howard, 1960), which is sometimes referenced under the latter name (see e.g. Santos and Rust (2003)).

theorem applies.[9] Convergence is therefore ensured even for non-Pareto optimal economies, but is also restricted to a relatively simple one-dimensional class of problems without occasionally binding constraints (see in particular Coleman (1991)). This paper, in contrast, abstracts from the analysis of suboptimal economies, but instead allows for a much larger state- and choice space, including multiple first-order conditions and a broad collection of possibly binding inequality constraints.

Deaton, on the other hand, does consider the possibility of occasionally binding constraints. In particular, Deaton (1991) studies a standard stochastic consumption-savings problem in the presence of liquidity constraints that preclude borrowing. By exploiting theoretical results developed in Deaton and Laroque (1992), Deaton shows that the same iterative procedure as advocated in Coleman (1989; 1990; 1991) defines a contraction mapping, and convergence follows from the Banach fixed-point theorem. This approach has received quite some attention in the literature and is particularly widely applied in consumption-theoretic studies.[10] I generalise Deaton's (1991) results in some dimensions, but specifise it in some other. In particular, I expand the analysis to cover a much larger state- and choice space, and consider a richer set of occasionally binding constraints. In contrast to Deaton (1991), however, this paper abstracts from the difficulties that arise when the return function may be unbounded.

2. Analysis

This section presents the main propositions and discusses their implications. In the first part, I set up a general stochastic optimisation problem formulated as a Bellman equation. The problem is cast such that it includes the possibility of occasionally binding inequality constraints, subject to some standard constraint qualifications. I then proceed by showing that under some commonly imposed conditions on the primitives of the problem, any element in the sequence of successive approximations of the value function, $\{v_n\}$, is differentiable with respect to the (vector of) endogenous state variables, and analytic expressions for their respective derivatives is provided. Lastly I show that as these derivatives solely depend on the policy and Lagrange multiplier functions associated with each v_n, the first order conditions allow these functions to be updated by only using their values in the previous iteration. The method of successive approximations that maps v_{n-1} to v_n can therefore be recast as a procedure that maps past policy- and multiplier functions to current policy- and multiplier function, and with identical convergence properties. This latter procedure, it turns out, is identical to that of Coleman (1989; 1990; 1991), Deaton (1991), and Judd (1998, pp. 553-555 and 601-602), and is referred to as time iteration.

[9] For these reasons, Coleman's method is sometimes referred to as the "monotone map method" (see Davig (2004), Davig and Leeper (2007), Davig *et al.* (2012)).

[10] See for instance Ludvigson (1999), Ludvigson and Michaelides (2001), Szeidl (2002), Haliassos and Michaelides (2003), Carroll (2006), and Carroll (2009).

The last part of the section discusses the practical implications of these results, and compare the resulting iterative procedure to standard value function iterations as well to other Euler equation based solution methods.

2.1. A General Problem

This paper is concerned with problems that can be formulated according to the following Bellman equation

$$v(x,z) = \max_{y \in \Gamma(x,z)} \{F(x,y,z) + \beta \int_Z v(y,z')Q(z,dz')\}. \tag{1}$$

Where $x \in X$ is the endogenous state, and $z \in Z$ is the exogenous state with a law of motion determined by the stationary transition function Q. I will employ the following standard assumptions.

(i) X is a convex Borel set in $\mathbb{R}^\ell$ with Borel subsets $\mathcal{X}$, and Z is a compact Borel set in $\mathbb{R}^k$ with Borel subsets $\mathcal{Z}$. Denote the (measurable) product space of $(X,\mathcal{X})$ and $(Z,\mathcal{Z})$ as $(S,\mathcal{S})$.

(ii) The transition function, $Q(Z,\mathcal{Z})$, has the Feller property.[11]

(iii) The feasibility correspondence $\Gamma(x,z) : X \times Z \to 2^X$ is, nonempty, compact-valued, and continuous. Moreover, the set $A = \{(y,x) \in X \times X : y \in \Gamma(x,z)\}$ is convex in x, for all $z \in Z$.

(iv) The return function $F(\cdot,\cdot,z) : A \to \mathbb{R}$ is once continuously differentiable, jointly (strictly) concave in x and y, and bounded on A for all $z \in Z$.

(v) The discount factor, β, is in the interval $(0,1)$.

It is important to note that the above definition of the feasibility correspondence includes the possibility of inequality constraints.

Then for any weakly concave and bounded v_0, it follows that (Stokey *et al.* (1989), pp. 263-266)

(i) The sequence of functions defined by

$$v_{n+1}(x,z) = \max_{y \in \Gamma(x,z)} \{F(x,y,z) + \beta \int_Z v_n(y,z')Q(z,dz')\}, \tag{2}$$

converges uniformly to the unique fixed point v. And

$$g_{n+1}(x,z) = \operatorname*{argmax}_{y \in \Gamma(x,z)} \{F(x,y,z) + \beta \int_Z v_n(y,z')Q(z,dz')\}, \tag{3}$$

converges pointwise to the unique fixed point g.[12]

[11] Alternatively one may assume that Z is countable and $\mathcal{Z}$ contains all subsets of Z.

[12] Following standard notation, g is the argmax of (1). It is important to note that if X is compact, then convergence of g_n is uniform.

(ii) v and v_n are strictly concave.

(iii) g and g_n are continuous functions.

To more explicitly introduce the presence of inequality constraints, I will assume that all *equality* constraints have been substituted into the instantaneous return function, F. The remaining restriction on feasibility will then solely be summarised by the collection of functions $m_j(x, y, z)$, such that $m_j(x, y, z) \leq 0$, for $j = 1, \ldots, r$.

ASSUMPTION 1. *The feasibility correspondence can be formulated as*

$$\Gamma(x, z) = \{y \in X : m_j(x, y, z) \leq 0, j = 1, \ldots, r\},$$

and the functions $m_j(x, y, z)$, $j = 1, \ldots, r$, are, once continuously differentiable in x and y, and jointly (weakly) convex in x and y.

Let $\mathbf{J}_x(x, y, z)$ denote the Jacobian matrix of the constraint vector $\mathbf{m} = (m_1(\cdot), m_2(\cdot), \ldots, m_r(\cdot))'$ with respect to the first ℓ arguments (the x's). And let $\mathbf{J}_y(x, y, z)$ denote the Jacobian with respect to the ensuing ℓ arguments (the y's).[13]

ASSUMPTION 2. *Linear Independence Constraint Qualification (LICQ): The Jacobian of the $p \leq r$ binding constraints has full rank; i.e. $rank(\mathbf{J}_y(x, g_n(x, z), z)) = p$, for each $n \in \mathbb{N}$.*

ASSUMPTION 3. *The following hold*

(i) $\Gamma(x, z) \subset int(X)$, or

(ii) *X is compact and $g(x, z) \in int(X)$.*

A few things ought to be noted here. Assumption 1 is actually quite weak. While it is assumed that all *equality* constraints have been substituted into the instantaneous return function, none of the results hinge on this expositional simplification. Together with Assumption 1, Assumption 2 ensures that the duality gap is zero and that the Kuhn-Tucker (KT) conditions are sufficient. There are other, weaker, constraint qualifications that ensures the sufficiency of the KT conditions, but LICQ is the weakest constraint qualification that ensures *both* the existence and uniqueness of the Lagrange multipliers (Wachsmuth, 2013). Together with Assumption 3, Assumption 2 also implies that there exists a $\hat{y}$ such that $m_j(x, \hat{y}, z) < 0$, for all x, z and j. This is normally known as Slater's condition. Part (i) of Assumption 3 implies part (ii), but the converse is not necessarily true. And lastly, the purpose of Assumption 3 is to ensure that it is the collection of constraints that restricts feasibility, and not the edge of the state space itself.

[13]The Jacobian is here defined as an $(\ell \times r)$-matrix, where ℓ is the number of endogenous state variables and r is the number of inequality constraints.

Thus, while Assumptions 1 and 3 are largely expositional, Assumption 2 is more substantive and merits a further discussion. First, and quite trivially, p, the number of binding constraints, must fall short of, or equal, the number of endogenous state variables, ℓ.[14] One example of such a violation would be a standard neoclassical growth model with an irreversibility constraint on investments, $k_{t+1} - (1-\delta)k_t \geq 0$, *and* with a liquidity constraint on capital $k_{t+1} \geq \phi$.[15] Then for $k_t = \phi/(1-\delta)$ both constraints simultaneously bind and LICQ is violated. While the KT conditions may still remain sufficient, the Lagrange multipliers are no longer unique as only their sum, and not their individual values, would distinctly influence the first order conditions.

This argument can be made more general. If $\boldsymbol{\mu}$ denotes a vector of Lagrange multipliers associated with some generic optimisation problem, and $\mathbf{J}_y$ denotes the Jacobian of the associated constraint vector, then *the vector* $\mathbf{v} = \mathbf{J}_y\boldsymbol{\mu}$ plays an integral part in the KT conditions; but the individual values of $\mathbf{J}_y$ and $\boldsymbol{\mu}$ do not. However, if the rank of $\mathbf{J}_y$ falls short of the number of binding constraints, there are infinitely many vectors $\hat{\boldsymbol{\mu}}$ that satisfies the system of equations $\mathbf{v} = \mathbf{J}_y\hat{\boldsymbol{\mu}}$, and the Lagrange multipliers are not unique.

2.2. Results

Define the operator T on $C^1(S)$, the space of bounded, strictly concave, and once continuously differentiable functions, as

$$(Tf)(x,z) = \max_{y\in\Gamma(x,z)} \{F(x,y,z) + \beta \int_Z f(y,z')Q(z,dz')\}. \tag{4}$$

That is, T is an operator that takes a function $f \in C^1(S)$ as an argument and maps it into a new function Tf. While it is obvious that under the above conditions, T maps functions in $C^1(S)$ into functions in $C(S)$ – the space of bounded, strictly concave, and continuous functions – it will be shown that $T : C^1(S) \to C^1(S)$, which follows less straightforwardly.[16]

Under all the aforementioned assumptions it is possible to express the mapping in (4) as

$$(Tf)(x,z) = \min_{\boldsymbol{\mu}\geq 0} \max_{y\in X} L(x,y,z,\boldsymbol{\mu}) = \max_{y\in X} \min_{\boldsymbol{\mu}\geq 0} L(x,y,z,\boldsymbol{\mu}), \tag{5}$$

where L denotes the Lagrangian (or saddle function)

$$L(x,y,z,\boldsymbol{\mu}) = F(x,y,z) + \beta \int_Z f(y,z')Q(z,dz') - \boldsymbol{\mu}'\mathbf{m}(x,y,z), \tag{6}$$

and $\boldsymbol{\mu}$ is a $(r \times 1)$ column vector of Lagrange multipliers associated with the constraint vector $\mathbf{m}$ (see for instance Rockafellar (1970)).

The ultimate goal of this section is to show that time iteration yields a convergent sequence of policy functions. The following definition of time iteration will be used.

[14] The rank of the Jacobian of the constraint vector must, by construction, always fall short of, or equal, the number of endogenous state variables.

[15] See Section 3 for a clarification of the notation used here. The parameter ϕ is an arbitrary positive constant.

[16] See, for instance, Stokey *et al.* (1989) for the former claim.

DEFINITION 1. *Let $F_x(x,y,z)$ denote the ℓ-vector consisting of the partial derivatives of F with respect to its first ℓ arguments. And let $F_y(x,y,z)$ denote the corresponding vector with respect to the ensuing ℓ arguments. Then,* ***time iteration*** *is the iterative procedure that finds the sequence $\{h_n(x,z)\}_{n=0}^{\infty}$ as $y = h_{n+1}(x,z)$ such that*

$$0 = F_y(x,y,z) - \mathbf{J}_y(x,y,z)\boldsymbol{\mu}_{n+1}(x,z) + \beta \int_Z [F_x(y,h_n(y,z'),z') - \mathbf{J}_x(y,h_n(y,z'),z')\boldsymbol{\mu}_n(y,z')]Q(z,dz'). \quad (7)$$

While the notation may appear esoteric, time iteration can be thought of as using the Euler equation to find today's optimal policy, h_{n+1}, given the policy of tomorrow, h_n.[17]

PROPOSITION 1. *The n-step value function, v_n, is (once) continuously differentiable with respect to $x \in int(X)$ and its vector of partial derivatives is given by*

$$v_{x,n}(x,z) = F_x(x,g_n(x,z),z) - \mathbf{J}_x(x,g_n(x,z),z)\boldsymbol{\mu}_n(x,z).$$

Proof. In Appendix A. □

Since the space $C^1(S)$ is not complete in the sup-norm, Proposition 1 does not imply that the limiting value function, v, is differentiable.[18] Moreover, in the proposition above, strict concavity of the problem and full rank of J_y is assumed. This simplifies the proof given in Corollary 5, p. 597, in Milgrom and Segal (2002), which essentially is equivalent for $x \in [0,1]$.

The final proposition will show that the sequence of policy functions obtained by time iteration converges to the true policy function.

PROPOSITION 2. *If the function $y = h_{n+1}(x,z)$ satisfies equation (7) with $h_n = g_n$, and the function $g_{n+1}(x,z)$ satisfies equation (3) with $v_n = Tv_{n-1}$, then $h_{n+1}(x,z) = g_{n+1}(x,z)$.*

Proof. In Appendix A. □

Thus, Proposition 2 reveals that if h_n is the nth solution of time iteration and g_n is the nth solution of dynamic programming, then h_n and g_n must coincide. And since it is known that for all $\varepsilon > 0$ there exist an N_s such that $\sup_s |g(s) - g_n(s)| < \varepsilon$ for all $n \geq N_s$, it follows from Proposition 2 that $\sup_s |g(s) - h_n(s)| < \varepsilon$ for all $n \geq N_s$ (see, for instance, Theorem 3.8 in Stokey *et al.* (1989)). As a consequence the sequence $\{h_n\}_{n\in\mathbb{N}}$ converges to the unique function g.[19]

Finally there is one additional remark to be made. As, g_n converges to g it follows that $F_x(x,g_n(x,z),z)$ converges to $F_x(x,g(x,z),z)$. Thus if the constraint vector $\mathbf{m}(x,y,z)$ is independent of x, $v_{x,n}(x,z)$ is independent of $\boldsymbol{\mu}_n$ and $v_{x,n}(x,z) \to F_x(x,g(x,z),z)$.[20] Hence, if convergence

[17] In the theoretical part of this paper I will use h to denote the policy function obtained through time iteration, and g as the policy function obtained through dynamic programming. The goal is to show that $h_n = g_n$.

[18] See Rincon-Zapatero and Santos (2009) for a result related to the limiting value function.

[19] If X is compact, N_s is independent of s and convergence is uniform.

[20] This particular case covers *state-independent* inequality constraints, such as borrowing or short-selling constraints.

of g_n is *uniform*, then $v(x, z)$ is, under these additional conditions, indeed differentiable and its derivative is given by $F_x(x, g(x, z), z)$.[21] Thus, as a byproduct, this paper provides a simple proof of the differentiability of the limiting value function, v, under state-independent constraints, such as debt limits.[22]

2.3. Discussion

The propositions presented above establish an equivalence between value function- and time iteration and, as such, neither method has any advantage over the other. It may therefore not be obvious how these ideas may help us in numerically computing dynamic models with occasionally binding constraints. Thus, to appreciate the usefulness of the techniques advocated in this paper, it is important to develop an understanding of the challenges involved in numerical solution methods, and how time iteration may help to alleviate some of these difficulties.

To this end, I will proceed in two steps. First, I will discuss the challenges involved in numerical dynamic programming, and how Euler equation based methods can provide some relief. Second, I will briefly discuss various ways of solving the Euler equation, and to which extent time iteration carries some advantages over alternative approaches.

2.3.1. Dynamic programming vs. Euler equation based methods

Many applications of dynamic programming rely on a discretised state- and choice space, and such a formulation makes any inequality constraint easy to implement. The resulting grid is simply delimited such that any violation of the constraint-set is made impossible (see, for instance, Hansen and Imrohoroğlu (1992)). However, in order to achieve any reasonable degree of accuracy, the discretisation must be made on a very fine grid and this causes the procedure to suffer immensely from the curse of dimensionality.[23]

Parts of these concerns can be avoided by relying on sophisticated approximation-, or interpolation, methods. In particular, by defining the value function on a relatively coarse grid, function values in-between grid points are then approximated and the choice space can be treated as convex (see, for example, Krusell and Smith (1998)). In the context of the standard neoclassical growth model, Judd and Solnick (1994) showed that approximating the value function with a shape-

[21] This follows as for any sequence of functions such that f_n converges pointwise to f, and f'_n exists, is continuous, and converges uniformly to the function g, then f is differentiable with derivative g (see for example Schroder (2008)).

[22] In fact, this result holds under weaker assumptions than previously stated; in particular, the uniqueness of the Lagrange multipliers are no longer necessary and LICQ could be replaced by a less strict constraint qualification, such as Slater's condition.

[23] Some numbers may illustrate this point quite clearly. Suppose that a grid consisting of a thousand nodes in each endogenous state-variable is known to produce an accurate solution to some dynamic problem. Then a reasonably small-scaled problem with three state variables suggests a meshed grid of $1,000^3$, or one billion, nodes.

preserving spline on a grid consisting of 12 nodes performs as well as a discretisation technique using 1200 nodes.

However, these methods are still no panacea. First, efficient optimisation routines exploit information regarding the slope of the value function, and not its level. An incredibly accurate approximation of the level of the value function can still be orders off track when it comes to the derivative. Second, for an efficient implementation the approximation procedure is usually confined to a computationally expensive class of methods. For instance, linear interpolations – which are known to be very efficient – create kinks in the value function, and discontinuities in its derivative. These discontinuities may ultimately feed into the associated policy function and give rise to misleading results. Alternative approximation methods, such as orthonormal polynomials, are smoother but can on the other hand display pronounced internodal oscillations. These oscillations can easily translate to quite wild swings in the derivative of the value function (and even alter its sign), and may therefore pave the ground for erroneous or diverging solutions. Thus, the choice of approximation methods should therefore be confined to a relatively expensive class which preserves certain desirable properties of the value function, such as concavity and continuous differentiability.

So in what way can Euler equation based methods circumvent these issues? Any approximation of the Euler equation – whether it is the policy function itself as in standard projection methods (Judd, 1992), or the entire conditional expectation as in the parameterised expectations algorithm (den Haan and Marcet, 1990) – is tantamount to an approximation of the *derivative* of the value function. Thus, an accurate approximation of the Euler equation will provide much more precise information of the *slope* of the value function than would an accurate approximation of its *level*. Furthermore, simple and efficient approximation methods, as those mentioned above, are much less capable of causing mischief when applied directly on the Euler equation. Linear interpolations, for instance, are not only continuous, but also known to preserve monotonicity and positiveness, even in arbitrarily many dimensions (Judd, 1998). A continuous, positive, and monotonically increasing approximation of the derivative of the value function is therefore as accurate as a once continuously differentiable and concavity-preserving approximation of its level. But the former comes at a *much* smaller computational cost than the latter, and there are therefore potentially large efficiency gains to be made.

As a final remark in this section, it appears appropriate to point out that what has been proven is a *theoretical* equivalence between time- and value function iteration, and not a practical *numerical* equivalence. Thus once numerical approximations replace precise mathematical operations – i.e. the maximisation step and/or function evaluations – the contraction property may be lost. Stachurski (2008) shows how "nonexpansive" numerical approximation methods preserves the contraction property under value function iteration. This result does not immediately carry over to time iteration.

2.3.2. Solution methods based on the Euler equation

A substantial share of dynamic economic models can be summarised by a collection of first-order conditions following

$$x_{t+1} = f(x_t, x_{t+1}, x_{t+2}), \tag{8}$$

were $x_t \in X$ represents the state of the system. An Euler equation based solution technique attempts to find a policy function $x_{t+1} = g(x_t)$ such that

$$g(x_t) = f(x_t, g(x_t), g(g(x_t))), \tag{9}$$

for all $x_t \in X$. There are several ways to accomplish this. In what follows I will briefly discuss three classes of solution methods – one direct approach, and two iterative procedures – under which, I believe, most practical implementations can be categorised, and I juxtapose their respective advantages.

Under the direct approach, the researcher decides on some approximation method for $g(x)$, and finds the associated coefficients such that equation (9) holds according to some metric.[24] This procedure is normally very fast, but also quite fragile. In particular, if $g(x)$ is approximated using N coefficients, the direct approach amounts to solving an N-dimensional system of nonlinear equations. This is a non-trivial task, in particular in higher dimensions, and convergence can easily fail.

Iterative methods can to some extent avoid these issues. *Fixed point iteration* is initiated by some guess for the policy function, call this $g_n(x)$, which is then updated according to[25]

$$g_{n+1}(x_t) = f(x_t, g_n(x_t), g_n(g_n(x_t))), \tag{10}$$

until g_{n+1} does not differ significantly from g_n. In many applications, such as the simple case illustrated here, this procedure carries some immediate advantages over the direct approach as the updated guess, g_{n+1}, can be found without resorting to any root-finding operation at all. The cumbersome system of non-linear equations is replaced by a simple, but repeated, evaluation of the Euler equation.[26] Despite these salient properties there are no guarantees that the sequence of successive guesses obtained under fixed point iteration will eventually converge to the solution g, and oscillating or exploding sequences are frequent.[27]

[24] For example, (9) should hold exactly on a finite grid of points in X according to the collocation method. In the Galerkin method, (9) holds as a weighted average on the entire state space, X.

[25] See for instance Judd (1998) pp. 599-601.

[26] Fixed point iteration nests the ideas of the Parameterised Expectations Algorithm by den Haan and Marcet (1990). In particular, as g_{n+1} only shows up in the left-hand side, one could instead approximate the entire right-hand side as $\hat{f}_n(x_t) = f(x_t, g_n(x_t), g_n(g_n(x_t)))$, which is then updated as $\hat{f}_{n+1}(x_t) = f(x_t, \hat{f}_n(x_t), \hat{f}_n(\hat{f}_n(x_t)))$.

[27] A remedy to this issue is to consider a dampening parameter $\eta \in (0, 1)$. In particular, let $\hat{g}_{n+1}(x)$ denote the left-hand side of (10), then the "dampened" update guess is given by $g_{n+1}(x) = \eta\hat{g}_{n+1}(x) + (1 - \eta)g_n(x)$. However, in some cases η must be set at a low value close to zero which may slow the algorithm down considerably.

Time iteration avoids these convergence issues. In particular, starting with some initial guess, $g_n(x)$, the sequence of successive approximations is then given by

$$g_{n+1}(x_t) = f(x_t, g_{n+1}(x_t), g_n(g_{n+1}(x_t))), \tag{11}$$

And following Proposition 2, $g_{n+1} \to g$, and time iteration is therefore a globally convergent solution method. However, these benefits do not come without costs. As the updated guess, g_{n+1}, appears both in the left- and right-hand side of (11), some root-finding operation is therefore unavoidable. But in contrast to the direct approach, the root-finding operation is confined to repeatedly solving a small-scale system of nonlinear equations (of dimensionality at most equal to the number of first-order conditions), instead of solving a much larger system only once.[28] And in the one-dimensional case, the method of endogenous grid points (Carroll, 2006) can avoid any numerical root-finding operations altogether, and each iteration of time iteration is equally fast as that of fixed point iteration, but with superior convergence properties.

3. A Numerical Example

This section illustrates the ideas developed in this paper by numerically solving a standard stochastic growth model with irreversible investment. The example is partly chosen on the basis that the underlying structure represents a fundamental building-block of most modern macroeconomic models, and partly as it includes a nontrivial occasionally binding constraint. In addition, the model is parsimonious enough to be solved very accurately using standard dynamic programming techniques, which will therefore be used as a benchmark for comparisons.

It should be emphasised already at this point that this section is by no means intended as a large scale comparison of various solution algorithms.[29] Instead, the purpose is to illustrate the applicability of the results derived in the preceding section, and how these ideas can be implemented in practice. As a byproduct of this exercise, however, some numerical results do emerge.

3.1. Model

I follow Christiano and Fisher (2000) and McGrattan (1996) and consider a standard stochastic growth model with irreversible investment. As this economy is Pareto optimal, the model can be summarised by the Bellman equation

$$v(k,z) = \max_{k' \in \Gamma(k)} \{u(zf(k) + (1-\delta)k - k') + \beta \int_{z' \in Z} v(k', z') Q(z, dz')\}, \tag{12}$$

[28] Malin *et al.* (2011), p. 237, state this difference as "Thus we trade off solving non-linear systems of much smaller size by the necessity to having to solve the systems many times (as opposed to only once)".

[29] See Christiano and Fisher (2000) or Aruoba *et al.* (2006) for comprehensive work in this direction.

with feasibility correspondence

$$\Gamma(k) = \{k' \in K : k' \geq (1-\delta)k\}. \tag{13}$$

The function u denotes the momentary utility function, and f the production function. Both are assumed to satisfy standard regulatory and Inada conditions. The parameter δ represents the depreciation rate of capital. And the feasibility correspondence in (13) reflects the irreversible nature of investment.[30]

The state space K belongs to an interval, $[\underline{k}, \overline{k}]$, such that $0 < \underline{k} < \overline{k} < \infty$, and $g(k,z) \in int(K)$. This conjecture will later be numerically verified, but holds quite naturally due to the Inada conditions imposed on u and f. In the numerical implementation the state space is discretised into N uniformly spaced elements, $\hat{K} = \{k_1, k_2, \ldots, k_N\}$, and function values in-between grid points will be found by some approximation routine (see the more detailed description below). The stochastic element z follows a two-state Markov process, $z \in Z = \{z_b, z_g\}$, with law of motion described by transition matrix P. As a consequence any integral of the type $\int_{z' \in Z} f(z')Q(z, dz')$ will from hereon be written as $\sum_{z' \in Z} f(z')P_{z,z'}$, where $P_{z,z'}$ denotes the probability of the economy transitioning from state z in period t to state z' in period $t+1$.

3.2. Solution Methods

I solve the above model using two different solution methods: value function- and time iteration. The two following subsections briefly discuss their precise numerical implementations. As any numerical implementation can only deal with the discretised state space $\hat{K} \times Z$, the qualification *for* $(k,z) \in \hat{K} \times Z$ is considered implied and is therefore omitted.

3.2.1. Value function iteration

I initiate the value function iteration algorithm using the conjecture[31]

$$v_0(k,z) = u(zf(k)), \tag{14}$$

and iterate according to the familiar procedure in equation (2). Following Proposition 1, the optimisation step can be carried out by finding a $\tilde{g}_n(k,z)$ which satisfies the (unrestricted) first-order condition

$$-u'(zf(k) + (1-\delta)k - \tilde{g}_n(k,z)) + \beta \sum_{z' \in Z} v'_{n-1}(\tilde{g}_n(k,z), z')P_{z,z'} = 0, \tag{15}$$

[30] An alternative formulation consistent with the notation in the previous section would be $\Gamma(k) = \{k' \in K : m(k,k',z) \leq 0\}$, with $m(k,k',z) = (1-\delta)k - k'$.

[31] In the numerical implementation this conjecture proved to be simpler for the nonlinear equation solver than the more commonly used $v_0(k,z) = u(zf(k))/(1-\beta)$.

where v'_{n-1} refers to the derivative of v_{n-1} with respect to its first element. The optimal policy choice is then given by $g_n(k,z) = \max\{\tilde{g}_n(k,z), (1-\delta)k\}$. And $v_{n-1}(k,z)$ is updated according to

$$v_n(k,z) = u(zf(k) + (1-\delta)k - g_n(k,z)) + \beta \sum_{z' \in Z} v_{n-1}(g_n(k,z), z')P_{z,z'}. \tag{16}$$

Outside of any $k \in \hat{K}$, the value of $v_n(k,z)$ is given by a Piecewise Cubic Hermite Interpolating Polynomial (PCHIP). PCHIP sets the derivative at a "knot" equal to the harmonic mean of the line-secants between the knot itself and its two neighboring knots.[32] This procedure preserves monotonicity and prevents overshooting of the interpolant in-between grid points. This latter property significantly reduces the risk of internodal oscillations.

The unrestricted first-order condition is solved using Newton's method with an analytically provided Jacobian. The procedure displays no issues of convergence for any parameterisation. The value function iteration algorithm is terminated once $\varepsilon < 1(-6)$, where

$$\varepsilon = \max_{(k,z) \in \hat{K} \times Z} |\varepsilon(k,x)|, \tag{17}$$

and

$$\varepsilon(k,z) = -u'(zf(k) + (1-\delta)k - \tilde{g}_n(k,z)) + \beta \sum_{z' \in Z} v'_n(\tilde{g}_n(k,z), z')P_{z,z'}. \tag{18}$$

Lastly, following the discussion in Appendix B, I also consider a version of Howard's improvement algorithm (Howard, 1960). While the precise details of the procedure can be found in the appendix, the algorithm proceeds similarly to that of value function iteration in finding g_n above. But in contrasts to merely updating v_{n-1}, it "improves" upon the value function through the iterative scheme

$$v_{n,h+1}(k,z) = u(zf(k) + (1-\delta)k - g_n(k,z)) + \beta \sum_{z' \in Z} v_{n,h}(g_n(k,z), z')P_{z,z'}, \tag{19}$$

with $v_{n,h}(g_n(k,z), z') = v_{n-1}(g_n(k,z), z')$, for $h = 0$. This improvement step is terminated once $h = H$, or the distance between $v_{n,h+1}$ and $v_{n,h}$ is small – whichever comes first. The value function is then updated as $v_n = v_{n,H}$, and the process repeats itself.[33]

Apart from significantly enhancing the convergence rate (Santos and Rust, 2003), Howard's improvement algorithm has an intuitive appeal. Given a candidate value and policy function, v_n and g_n, satisfying equation (16), the algorithm improves on v_n by calculating the value function that emerges if the agent would operate on policy function g_n, H number of times. The resulting

[32] A *knot* is the same as a node; i.e. an element of $\hat{K}$. Derivatives at endpoints are set by the one sided line-secant. For more information about this approximation method see Moler (2004) or Kahaner *et al.* (1989).

[33] For simplicity I let $v_{n,H}$ denote the improved value function at the iteration at which the procedure was terminated even if less than H iterations were executed.

value, here called $v_{n,H}$, is then used in place of v_n in the subsequent iteration of the standard algorithm, and the process repeats itself. The enhanced convergence rate stems from the fact that the (computationally costly) maximisation step is only carried out infrequently, while the value function itself is "improved" upon repeatedly.

As a rule of thumb, I terminate the improvement step once $H = 20$ or the sup-norm between $v_{n,h+1}$ and $v_{n,h}$ is less than $1(-6)$. However, there are some variations across parameterisations in terms of which value of H is optimal, and I alter H in order to attain a procedure as efficient as possible. The improvement algorithm is terminated using the same criterion as that of value function iteration.

3.2.2. Time iteration

The time iteration algorithm is initiated using the conjecture

$$v_0'(k,z) = zf'(k)u'(zf(k)). \tag{20}$$

Similarly to value function iteration I find $\tilde{g}_n(k,z)$ as the solution to the (unrestricted) first-order condition

$$-u'(zf(k) + (1-\delta)k - \tilde{g}_n(k,z)) + \beta \sum_{z' \in Z} v_{n-1}'(\tilde{g}_n(k,z), z')P_{z,z'} = 0. \tag{21}$$

And the optimal policy choice is again given by $g_n(k,z) = \max\{\tilde{g}_n(k,z), (1-\delta)k\}$. The Lagrange multiplier, $\mu_n(k,z)$, is trivially found as the "residual"

$$\mu_n(k,z) = u'(zf(k) + (1-\delta)k - g_n(k,z)) - \beta \sum_{z' \in Z} v_{n-1}'(g_n(k,z), z')P_{z,z'}, \tag{22}$$

and the derivative v_{n-1}' is then updated following Proposition 1

$$v_n'(k,z) = (1 + zf'(k) - \delta)u'(zf(k) + (1-\delta)k - g_n(k,z)) - (1-\delta)\mu_n(k,z). \tag{23}$$

Outside of any $k \in \hat{K}$, the value of $v_n'(k,z)$ is given by linear interpolation. In many aspects linear interpolation is a suitable choice of approximation to juxtapose the two solution methods: While PHCIP provides a C^1 approximation, linear interpolation is C^0. This will give some obvious speed advantages to time iteration, although without any *a priori* expected loss in accuracy. For the sake of further comparison, however, I will also report the results when $v_n'(k,z)$ is approximated using PCHIP.

As with value function iteration, the unrestricted first-order condition is solved using Newton's method with an analytically provided Jacobian. The procedure displays no issues of convergence for any parameterisation. Again, the time iteration algorithm is terminated once $\varepsilon < 1(-6)$, where ε is defined in exactly the same way as in equation (17).

I also employ the improvement algorithm discussed in Appendix B. The improvement algorithm proceeds very similarly to that of time iteration in finding g_n above. In contrasts to merely updating v'_{n-1}, as is done in equation (23), I "improve" upon the derivative through the iterative scheme

$$\begin{aligned} v'_{n,h+1}(k,z) = (1 + zf'(k) - \delta)u'(zf(k) + (1-\delta)k - g_n(k,z)) \\ + g'_n(k,z) \times [-u'(zf(k) + (1-\delta)k - g_n(k,z)) + \beta \sum_{z' \in Z} v'_{n,h}(g_n(k,z), z')P_{z,z'}], \end{aligned} \quad (24)$$

with $v'_{n,h}(g_n(k,z),z') = v'_{n-1}(g_n(k,z),z')$, for $h = 0$, and where g'_n denotes the derivative of the policy function g_n with respect to its first argument (see Appendix B for a further discussion).[34] This improvement step is terminated once $h = H$, or the distance between $v'_{n,h+1}$ and $v'_{n,h}$ is small – whichever occurs first. The derivative of the value function is then updated as $v'_n = v'_{n,H}$, and the process repeats itself.[35] The disclaimer discussed in the previous section in setting H applies also here. But as a rule of thumb, $H = 20$ performs generally well.

Two final remarks ought to be noted. First, whenever $H = 1$ the second term in equation (24) is equal to $g'_n(k,z) \times \mu_n(k,z)$. When the irreversibility constraint is binding, $\mu_n > 0$ and $g'_n = (1-\delta)$, and equation (24) collapses into equation (23). Thus, with $H = 1$, the improvement algorithm is identical to that of time iteration. Second, the time iteration algorithm described above may appear different to that of Definition 1. From a theoretical perspective this difference is merely cosmetic: Substituting equation (23) into the definition of the multiplier reinstates the procedure in Definition 1. From a practical, or applied, perspective there may, however, be a difference. By approximating the derivative of the value function we avoid approximating both the policy- and Lagrange multiplier function. This gives one functional approximation less to worry about. The procedure also has the pedagogical advantage of nesting the improvement algorithm described above.

3.3. Accuracy of Solutions

Let $q(k,z)$ denote an arbitrary and feasible policy, such that $q(k,z) \in \Gamma(k,z)$, for all $(k,z) \in K \times Z$.[36] The fixed point associated with q is then defined as

$$v^q(k,z) = u(zf(k) + (1-\delta)k - q(k,z)) + \beta \sum_{z' \in Z} v^q(q(k,z),z')P_{z,z'}. \quad (25)$$

In the case in which q indeed is optimal, q must equal to g, and v^q must satisfy the Bellman equation in (12). However, for *any* two feasible policies q and $\hat{q}$ (which may or may not be optimal),

[34] An approximation for the derivative of $g_n(\cdot,z)$ on $\hat{K}$ is computed using forward difference for the first node, backward difference for the last node, and the central difference for all other nodes.

[35] As previously I let $v'_{n,H}$ denote the improved derivative at the iteration at which the procedure was terminated even if less than H iterations were executed.

[36] I denote an arbitrary and feasible policy as q in order to underline its distinction from the optimal policy g.

$v^q(k,z) > v^{\hat{q}}(k,z)$ implies that $q(k,z)$ must be strictly preferred to $\hat{q}(k,z)$. As a consequence, we may assess the relative desirability – and therefore the relative accuracy – of any two feasible policies by simply comparing their associated fixed points. Evaluating accuracy in this way carries three important properties.

First, the relative accuracy between any two arbitrary feasible policies q and $\hat{q}$ can be assessed without any reference to the "true" or "optimal" policy, g. Second, the relative accuracy of a policy is expressed in terms of the actual utility cost associated with its error, and not by its distance to the "true" policy function.[37] Lastly, the cost of operating under policy $\hat{q}$ relative to policy q can be expressed in terms of consumption equivalents, which has a very natural interpretation. In particular, following Lucas (1987), the value associated with switching from policy $\hat{q}$ to policy q can be expressed in terms of the % increase in consumption in each period for perpetuity according to,

$$\eta(k,z;q,\hat{q}) = \frac{\ln v^q(k,z) - \ln v^{\hat{q}}(k,z)}{1-\gamma} \times 100, \tag{26}$$

where γ denotes the (constant) coefficient of risk aversion.[38]

To calculate a precise estimate of η, it is important to have a precise estimate of the fixed points v^q and $v^{\hat{q}}$. To this end, I linearly interpolate a given policy function $q : \hat{K} \times Z \to \Gamma$ onto a fine grid $\bar{K}$ containing 1,000,000 nodes such that $q : \bar{K} \times Z \to \Gamma$. Lastly, I use nearest neighbour interpolation to arrive at the policy function $q : \bar{K} \times Z \to \bar{\Gamma}$, with

$$\bar{\Gamma}(k) = \{k' \in \bar{K} : k' \geq (1-\delta)k\}. \tag{27}$$

As this expanded policy function maps values of the very fine grid $\bar{K} \times Z$ onto $\bar{K}$ there is no need for any additional interpolation methods, and a very accurate approximation to the fixed point in (25) is easily computed. I find an approximation by iterating 2,000 times on equation (25) using the initial guess

$$v_0^q(k,z) = \frac{u(zf(k))}{1-\beta}, \quad (k,z) \in \bar{K} \times Z. \tag{28}$$

It should be noted that while the above method provides a very precise procedure to assess the *relative* accuracy of two policies q and $\hat{q}$, it does so at the cost of not providing any information about the *absolute* accuracy of any given policy. To attain the dual objective of assessing both relative as well as absolute accuracy of the suggested solution methods, I also solve the Bellman equation in (12) with Γ replaced by the fine grid $\bar{\Gamma}$ and for $(k,z) \in \bar{K} \times Z$ (see Appendix C for more details). Considering the large cardinality of $\bar{K}$, I interpret this benchmark solution to be exact.[39]

[37] I use the words "true" and "optimal" in quotation marks here as virtually all comparative studies on this topic use some stand-in for the true function, which is never recovered (e.g. Christiano and Fisher (2000)).

[38] If utility is logarithmic, i.e. if $\gamma \to 1$, the relevant metric is $\eta = (1-\beta)(v^q - v^{\hat{q}}) \times 100$.

[39] Christiano and Fisher (2000) consider the same solution exact when $\bar{K}$ contains 40,000 nodes.

If I let g denote the policy function obtained from the "exact" solution, and q denote the policy function obtained from any other solution, I will report accuracy following the three measures

$$\eta_{max} = \max_{(k,z)\in \bar{K}\times Z} \eta(k,z;g,q), \quad \eta_{min} = \min_{(k,z)\in \bar{K}\times Z} \eta(k,z;g,q), \quad \eta_{mean} = \sum_{(k,z)\in \bar{K}\times Z} \frac{\eta(k,z;g,q)}{2,000,000},$$

where, under the hypothesis that g indeed is optimal, $\eta(k,z;g,q) \geq 0$ for all feasible policy function q, and all $(k,z) \in \bar{K} \times Z$.

Lastly, notice that the linearity of η implies that

$$\eta(k,z;q,\hat{q}) = \eta(k,z;g,q) - \eta(k,z;g,\hat{q}), \tag{29}$$

and that the mean consumption equivalent cost of policy $\hat{q}$ relative to policy q is therefore given by their differences in η_{mean}.[40]

3.4. Functional Forms and Parameterisations

The momentary utility function belongs to the class of constant relative risk aversion, and the production function is of the standard Cobb-Douglas type

$$u(c) = \frac{c^{1-\gamma}}{1-\gamma}, \qquad f(k) = k^{\alpha}, \tag{30}$$

with $u(c)$ interpreted as $\ln(c)$ if γ is equal to one. The transition matrix P is given by

$$P = \begin{pmatrix} \frac{1+\rho}{2} & \frac{1-\rho}{2} \\ \frac{1-\rho}{2} & \frac{1+\rho}{2} \end{pmatrix}, \tag{31}$$

where a typical element, $P_{i,j}$, is interpreted as the probability of state j occurring in the subsequent period given that the current state is i; $Pr(z' = z_j | z = z_i)$. The state space for the exogenous state is given as $Z = \{e^{\sigma}, e^{-\sigma}\}$.

Table 1 summarises the various sets of parameterisations considered, together with the associated bounds on the endogenous state space, $[\underline{k}, \overline{k}]$. The parameterisations follow those of Christiano and Fisher (2000).

I solve the models (1)-(7) using the two algorithms described above. The grid for capital contains $N = 10$, $N = 100$, and $N = 1,000$ equidistant nodes.

3.5. Numerical Results

Tables 2 and 3 display the main results of the numerical exercise. The first three cells of each cluster provide the maximum, minimum, and mean error, respectively. The fourth cell provides

[40] As an example, suppose that $\eta_{mean}(\hat{q}) = 5$, and $\eta_{mean}(q) = 3$, then $\eta_{mean}(q,\hat{q}) = -2$. That is, the superiority of policy q relative to $\hat{q}$ is worth a 2 % rise in consumption for perpetuity. Notice however that the same logic does not apply for η_{max} or η_{min}.

Table 1: *Parameterisations*

	Parameters							
Model	β	γ	α	δ	σ	ρ	$\overline{k}/k_{ss}$	$\underline{k}/k_{ss}$
(1)	$1.03^{-\frac{1}{4}}$	1	0.3	0.02	0.23	0	1.9	0.3
(2)	$1.03^{-\frac{1}{4}}$	10	0.3	0.02	0.23	0	3.8	0.005
(3)	$1.03^{-\frac{1}{4}}$	1	0.05	0.02	0.0382	0	1.2	0.8
(4)	$1.03^{-\frac{1}{4}}$	1	0.3	0.5	0.675	0	3.8	0.3
(5)	$1.03^{-\frac{1}{4}}$	1	0.3	0.02	0.23	0.95	1.7	0.6
(6)	$1.03^{-\frac{1}{4}}$	1	0.3	0.02	0.4	0	2.3	0.2
(7)	$1.03^{-\frac{1}{4}}$	10	0.1	0.02	0.23	0.95	5.9	0.4

the computation time in seconds, with and without the improvement algorithm (with the latter in brackets). The fifth cell provides the number of iterations executed until convergence, again with and without the improvement algorithm. If we start by comparing value function iteration with time iteration using linear interpolation, a few noticeable features emerge. Time iteration is as accurate as value function iteration for all parameterisations except for model (2) and (5), and is also markedly faster.[41] It is an order of magnitude more accurate for model (4) at $N = 10$, but performs slightly worse across the board for model (5) – although with a sustained speed advantage.[42] With respect to model (2), time iteration performs very poorly at small values of N, but remains roughly on par with value function iteration at higher values. However, even value function iteration performs intolerably bad for this particular model at the coarser grids, with the maximum error peaking at a striking 4.5 %. There are two reasons underlying these inaccuracies. First, the ergodic set of capital for model (2) covers a very wide interval, ranging from 0.5 % of the steady-state level of capital, to 380 %. A grid consisting of 10 nodes is therefore arguably far too coarse for any reasonable degree of accuracy. Second, a coefficient of risk aversion of 10 brings forth some significant nonlinearities in the model which appear difficult to approximate. These nonlinearities seem to cause more serious complications for the derivative of the value function – which is heavily exploited in time iteration – than they do for its level. At least at the very small values of N.

When piecewise cubic hermite interpolation replaces linear interpolation in time iteration, some of these results change. For $N = 10$, time iteration consistently outperforms value function iteration – roughly by one order or two – for all parameterisations apart from model (2). It performs no worse than value function iteration at the finer grids, apart from, again, model (2) where it consistently,

[41] With the only exceptions of $N = 100$ for model (4) and $N = 1,000$ for model (7) for which value function iteration is slightly faster.

[42] Speed should not be neglected here. For instance, time iteration with 100 nodes is both faster and more accurate for model (5) than value function iteration with 10 nodes.

Table 2: *Comparison of Solution Methods, Models (1)-(4)*

	Value Function Iteration			Time Iteration (linear)			Time Iteration (pchip)		
Grid size, N	10	100	1,000	10	100	1,000	10	100	1,000
Model (1)									
Max error	5.2(-3)	3.7(-5)	3.7(-5)	6.2(-3)	3.7(-5)	3.7(-5)	4.1(-4)	3.7(-5)	3.7(-5)
Min error	1.3(-3)	1.7(-6)	1.6(-6)	1.7(-3)	2.9(-6)	1.6(-6)	8.4(-5)	1.7(-6)	1.6(-6)
Mean error	2.6(-3)	1.4(-5)	1.4(-5)	2.4(-3)	1.5(-5)	1.4(-5)	1.3(-4)	1.4(-5)	1.4(-5)
CPU time	1.35 (5.76)	1.32 (5.89)	1.80 (8.02)	0.95 (3.37)	0.92 (3.49)	1.22 (5.17)	1.03 (4.08)	1.12 (4.03)	1.55 (5.83)
Iterations	14 (371)	17 (361)	17 (361)	9 (287)	9 (280)	9 (280)	8 (278)	9 (280)	9 (280)
Model (2)									
Max error	4.5	2.8	8.2(-2)	36.3	2.85	3.0(-3)	21.3	9.6(-1)	3.6(-3)
Min error	2.0(-2)	2.8(-6)	9.9(-7)	7.5(-1)	1.4(-3)	1.0(-6)	9.0(-4)	1.0(-6)	9.9(-7)
Mean error	5.3(-1)	3.4(-3)	5.4(-5)	1.3	4.2(-3)	4.1(-5)	1.2(-1)	7.5(-4)	4.1(-5)
CPU time	3.56 (19.9)	2.77 (28.9)	2.43 (9.08)	2.19 (8.05)	1.71 (7.11)	2.71 (11.1)	2.09 (6.68)	2.13 (7.18)	3.8 (11.88)
Iterations	28 (578)	20 (668)	19 (625)	36 (773)	20 (636)	19 (625)	24 (622)	19 (625)	19 (625)
Model (3)									
Max error	2.7(-7)	7.3(-8)	7.4(-8)	2.5(-6)	7.3(-8)	7.5(-8)	2.2(-7)	7.3(-8)	7.5(-8)
Min error	2.3(-7)	2.6(-8)	2.7(-8)	2.2(-6)	2.6(-8)	2.7(-8)	1.8(-7)	2.6(-8)	2.7(-8)
Mean error	2.5(-7)	3.4(-8)	3.5(-8)	2.3(-6)	3.4(-8)	3.5(-8)	1.9(-7)	3.4(-8)	3.5(-8)
CPU time	0.87 (1.89)	0.89 (2.04)	2.76 (2.76)	0.53 (1.71)	0.56 (1.78)	2.47 (2.47)	0.62 (1.99)	0.70 (2.02)	2.82 (2.82)
Iterations	9 (134)	10 (135)	135 (135)*	10 (130)	10 (129)	129 (129)*	10 (128)	13 (129)	129 (129)*
Model (4)									
Max error	5.9(-2)	4.4(-6)	2.4(-6)	2.1(-3)	2.6(-6)	2.4(-6)	1.8(-3)	2.5(-6)	2.4(-6)
Min error	4.6(-2)	3.0(-6)	2.0(-6)	1.9(-3)	2.2(-6)	2.0(-6)	1.6(-3)	2.2(-6)	2.0(-6)
Mean error	4.7(-2)	3.1(-6)	2.1(-6)	1.9(-3)	2.3(-6)	2.1(-6)	1.7(-3)	2.3(-6)	2.1(-6)
CPU time	0.22 (0.50)	0.29 (0.48)	0.59 (0.59)	0.20 (0.39)	0.41 (0.39)	0.55 (0.55)	0.21 (0.41)	0.42 (0.42)	0.62 (0.62)
Iterations	7 (29)	7 (27)	27 (27)*	8 (28)	11 (28)	28 (28)*	7 (28)	28 (28)*	28 (28)*

Table 3: *Comparison of Solution Methods, Models (5)-(7)*

	Value Function Iteration			Time Iteration (linear)			Time Iteration (pchip)		
Grid size, N	10	100	1,000	10	100	1,000	10	100	1,000
Model (5)									
Max error	6.9(-4)	8.4(-7)	9.7(-7)	3.7(-4)	1.0(-6)	9.0(-7)	1.7(-6)	8.5(-7)	1.1(-6)
Min error	6.5(-5)	8.7(-8)	8.3(-8)	2.5(-4)	2.0(-7)	1.1(-7)	6.9(-7)	8.4(-8)	1.9(-7)
Mean error	1.1(-4)	1.6(-7)	1.6(-7)	3.2(-4)	3.0(-7)	1.9(-7)	1.0(-6)	1.6(-7)	2.9(-7)
CPU time	1.15 (4.87)	1.16 (5.08)	6.34 (6.34)	0.76 (3.20)	0.89 (3.26)	2.9 (4.59)	0.93 (3.49)	1.07 (3.69)	4.96 (4.96)
Iterations	13 (314)	13 (312)	312 (312)*	10 (256)	11 (251)	135 (251)	10 (251)	11 (252)	251 (251)*
Model (6)									
Max error	2.2(-3)	1.4(-4)	1.4(-4)	8.2(-3)	1.4(-4)	1.4(-4)	6.6(-4)	1.4(-4)	1.4(-4)
Min error	1.2(-4)	9.4(-5)	9.4(-5)	1.9(-3)	9.6(-5)	9.4(-5)	1.2(-4)	9.4(-5)	9.4(-5)
Mean error	2.8(-4)	1.2(-4)	1.2(-4)	2.9(-3)	1.2(-4)	1.2(-4)	1.7(-4)	1.2(-4)	1.2(-4)
CPU time	1.30 (5.37)	1.35 (5.76)	1.75 (7.69)	1.05 (3.52)	0.96 (3.56)	1.25 (5.09)	1.14 (3.83)	1.18 (4.10)	1.62 (5.9)
Iterations	10 (368)	10 (368)	10 (368)	10 (308)	9 (294)	9 (294)	9 (293)	9 (294)	9 (294)
Model (7)									
Max error	2.5(-1)	5.7(-4)	3.0(-5)	1.1(-1)	3.5(-4)	3.0(-5)	2.7(-3)	3.0(-5)	3.0(-5)
Min error	5.4(-2)	1.6(-5)	2.3(-6)	4.7(-2)	1.1(-4)	2.3(-6)	1.2(-3)	2.4(-6)	2.3(-6)
Mean error	1.1(-1)	2.2(-5)	1.0(-5)	9.7(-2)	2.5(-4)	1.0(-5)	1.8(-3)	1.0(-5)	1.0(-5)
CPU time	1.45 (3.93)	1.42 (4.03)	1.84 (5.79)	1.31 (4.56)	1.31 (4.6)	2.07 (7.33)	1.39 (4.36)	1.58 (5.07)	2.52 (7.73)
Iterations	20 (399)	19 (387)	19 (386)	24 (422)	22 (386)	22 (385)	18 (379)	19 (385)	19 (385)

* For models *(3)*, *(4)*, and *(5)* the improvement algorithm did not converge for $N = 1,000$ for either algorithm. Similar difficulties was encountered for model *(4)* using Time Iteration (pchip) with $N = 100$.

although moderately, actually performs better. For model (7) time iteration is roughly an order more accurate than value function iteration when N is equal to 100.

These gains in accuracy associated with hermite interpolation are nontrivial. But they do also come at the expense of computational speed. Piecewise cubic hermite interpolation is slower than linear interpolation and computation time for time iteration increases across all models and for all grids (with $N = 10$ for model (2) being the only exception). Yet, even under this more demanding interpolation scheme time iteration still remains faster than value function iteration for all models apart from (4) and (7), and for almost all grid sizes ($N = 1,000$ for models (2) and (3) is the exception). In general, time iteration is faster than value function iteration in 15 out of the 21 possible cases, and equally or more accurate in 20. The speed advantage of time iteration relative to value function iteration under this interpolation scheme stems from the superior performance of the nonlinear equation solver: PCHIP renders a continuous (albeit kinked) Jacobian for time iteration, but imposes some challenging discontinuities for the Jacobian associated with value function iteration.[43] It is therefore unclear whether these speed gains would remain in models that rely on search-based, rather than gradient based, solution algorithms.

Lastly, an interesting, although disconcerting, observation is that accuracy barely changes as N increases from 100 to $1,000$. With the notable exception of model (2), this pattern emerges across all parameterisations and all solution methods.[44] I can see two possible explanations for this phenomenon. Firstly, there might be something inherent in the numerical implementations which prevents them from ever approaching the exact solution, even as the mesh size approaches zero. This would be a serious concern. Alternatively, and secondly, the "exact" solution may be further from the truth than we initially thought – or at least further from the truth than the alternative solution methods. To see how the latter explanation could generate the above observation, recall that the obtained policy functions have been interpolation and discretised onto the very same grid that defines the "exact" solution. Thus, any measure of accuracy – i.e. the cost of operating on *any* policy other than the "exact policy" – will, by construction, always be positive, and the "exact policy" will always outperform *any* other alternative. But if the "exact" solution is not exact enough, any other policy that is actually closer to the truth will *appear* inferior, even when it is not.

To discriminate between these two hypotheses I interpolate and discretise *all* policy functions (including the "exact") on an even finer grid consisting of 10,000,000 nodes in capital. I subsequently calculate the associated fixed points as described in Section 3.3, and recalculate the associated

[43] I have also tried using a cubic spline as an alternative to PCHIP. The C^2 property of the spline alleviates the issues raised here, but causes some other due to internodal oscillations; indeed for some parameterisations the derivative of the value function took on negative values on parts of the state space, which led the solver to a complex (absorbing) terrain. In general, PCHIP performed quite a lot better than a cubic spline across all models.

[44] Accuracy actually declines for model (5) when using time iteration with piecewise hermite interpolation. I have also tried increasing N to 10,000 with unaltered results.

accuracy measures. The results for the parameterisations where the problem is the most pronounced are provided in Table 4.

Table 4: *Comparison of different solution methods, models (5) and (6)*

	Value Function Iteration			Time Iteration (linear)			Time Iteration (pchip)		
Grid size, N	10	100	1,000	10	100	1,000	10	100	1,000
Model (5)									
Max error	6.9(-4)	-1.8(-7)	-1.9(-7)	3.7(-4)	-2.1(-8)	-1.9(-7)	1.3(-6)	-1.9(-7)	-1.9(-7)
Min error	6.4(-5)	-3.8(-6)	-3.8(-6)	2.5(-4)	-3.7(-7)	-3.8(-6)	-3.1(-6)	-3.8(-6)	-3.8(-6)
Mean error	1.1(-4)	-5.0(-7)	-5.0(-7)	3.2(-4)	-3.6(-7)	-5.0(-7)	3.7(-7)	-5.0(-7)	-5.0(-7)
Model (6)									
Max error	2.1(-3)	-6.1(-5)	-6.2(-4)	8.0(-3)	-6.0(-5)	-6.2(-5)	5.0(-4)	-6.2(-5)	-6.2(-5)
Min error	-9.5(-5)	-9.9(-5)	-9.9(-5)	1.7(-3)	-9.9(-5)	-9.9(-5)	-9.5(-5)	-9.9(-5)	-9.9(-5)
Mean error	8.0(-5)	-7.9(-5)	-7.9(-5)	2.7(-3)	-7.8(-5)	-7.9(-5)	-3.0(-5)	-7.9(-5)	-7.9(-5)

Notes. This table does not include the computational time nor the number of iterations as these are identical to those in Table 3. The disclaimer that applies for Tables 2 and 3 applies also here.

Perhaps unsurprisingly the *relative* performances of the solutions are largely unchanged. Time iteration performs roughly as well as value function iteration for $N = 100$ and $N = 1,000$ irrespective of interpolation method. Linear interpolation loses somewhat in relative accuracy to value function iteration at $N = 10$, while hermite interpolation instead makes some headway. But more importantly, the *absolute* performances of the solutions have changed quite dramatically. Both value function iteration and time iteration deliver a more accurate solution using a grid consisting of 100 nodes than discretised value function iteration on a grid using 1,000,000 nodes.

I have repeated this exercise for all parameterisations in Appendix D with very similar results: Value function iteration and time iteration using linear interpolation outperform the discretisation algorithm in four out of seven cases when $N = 100$, and in six out of seven cases when PCHIP is used. While these numerical results are by no means a centrepiece of the analysis in this paper, they are quite surprising and suggest that the grid size must be much larger than what has been previously used in the literature when using discretisation methods as a benchmark for accuracy.[45]

4. Concluding Remarks

Dynamic models with inequality constraints pose a challenging problem for two major reasons: dynamic programming techniques are reliable but often slow, while Euler equation based methods

[45]Christiano and Fisher (2000), for instance, treat the discretised value function iteration algorithm using 40,000 nodes as "exact" for precisely the same parameterisations as those explored here. McGrattan (1996) uses $2^{14} = 16384$ nodes for a very similar exercise.

are faster but have problematic or unknown convergence properties. This paper has, at least partly, addressed these concerns.

I have shown that a common iterative procedure applied on the first-order conditions – known in the literature as *time iteration* – delivers a sequence of approximate policy functions that converges to the true solution under a wide range of circumstances. These circumstances extend to a finite, but large set of endogenous and exogenous state variables, and include a broad spectrum of occasionally binding constraints. I exemplified the potential advantages of the method to alternative techniques by solving a simple real business cycle model with irreversible investments. In general time iteration turns out to be as accurate as state of the art value function iteration algorithms, but almost indiscriminately faster.

University of Cambridge, Centre for Macroeconomics (CFM), and CEPR.

Manuscript submitted September 3, 2012.

References

Aiyagari, R.S. (1994). 'Uninsured idiosyncratic risk and aggregate saving', *Quarterly Journal of Economics*, vol. 109(3).

Aruoba, S.B., Fernández-Villaverde, J. and Rubio-Ramirez, J.F. (2006). 'Comparing solution methods for dynamic equilibrium economies', *Journal of Economic Dynamics and Control*, vol. 30(2377-2508).

Baxter, M. (1991). 'Approximating suboptimal dynamic equilibria: An euler equation approach', *Journal of Monetary Economics*, vol. 28(2), pp. 173–200.

Bernanke, B., Gertler, M. and Gilchrist, S. (1999). 'The financial accelerator in a quantitative business cycle framework', in (J. B. Taylor and M. Woodford, eds.), *The Handbook of Macroeconomics*, pp. 1341–1393, vol. 1, chap. 21, Elsevier, 1 edn.

Bizer, D.S. and Judd, K.L. (1989). 'Taxation and uncertainty', *The American Economic Review*, vol. 79(2), pp. 331–336.

Carroll, C.D. (2006). 'The method of endogenous gridpoints for solving dynamic stochastic optimization problems', *Economics Letters*, vol. 91(3), pp. 312–320.

Carroll, C.D. (2009). 'Precautionary saving and the marginal propensity to consume out of permanent income', *Journal of Monetary Economics*, vol. 56(6), pp. 780–790.

Christiano, L.J. and Fisher, J.D.M. (2000). 'Algorithms for solving dynamic models with occasionally binding constraints', *Journal of Economic Dynamics and Control*, vol. 24(8), pp. 1179–1232.

Coleman, W.J. (1989). 'An algorithm to solve dynamic models', Federal Reserve Board, Washington, DC.

Coleman, W.J. (1990). 'Solving the stochastic growth model by policy-function iteration', *Journal of Business and Economic Statistics*, vol. 8(1), pp. 27–29.

Coleman, W.J. (1991). 'Equilibrium in a production economy with an income tax', *Econometrica*, vol. 59(4), pp. 1091–1104.

Cooley, T., Marimon, R. and Quadrini, V. (2004). 'Aggregate consequences of limited contract enforceability', *Journal of Political Economy*, vol. 112(4), pp. 817–847.

Davig, T. (2004). 'Regime-switching debt and taxation', *Journal of Monetary Economics*, vol. 51(4), pp. 837–859.

Davig, T. and Leeper, E.M. (2007). 'Generalizing the taylor principle', *American Economic Review*, vol. 97(3), pp. 607–635.

Davig, T., Leeper, E.M. and Walker, T.B. (2012). 'Unfunded liabilities and uncertain fiscal financing', *Journal of Monetary Economics*, vol. 57(5), pp. 600–619.

de Groot, O. (2011). 'Coordination failure and the financial accelerator', University of Cambridge.

Deaton, A. (1991). 'Saving and liquidity constraints', *Econometrica*, vol. 59, pp. 1221–1248.

Deaton, A. and Laroque, G. (1992). 'On the behaviour of commodity prices', *The Review of Economic Studies*, vol. 59(1), pp. 1–23.

den Haan, W.J. and Marcet, A. (1990). 'Solving the stochastic growth model by parameterizing expectations', *Journal of Business and Economic Statistics*, vol. 8(1), pp. 31–34.

Gertler, M. and Karadi, P. (2011). 'A model of unconventional monetary policy', *Journal of Monetary Economics*, vol. 58(1), pp. 17–34.

Gertler, M. and Kiyotaki, N. (2010). 'Financial intermediation and credit policy in business cycle analysis', in (B. M. Friedman and M. Woodford, eds.), *Handbook of Monetary Economics*, vol. 3, Elsevier, 1 edn.

Haliassos, M. and Michaelides, A. (2003). 'Portfolio choice and liquidity constraints', *International Economic Review*, vol. 44(1), pp. 143–177.

Hansen, G.D. and Imrohoroğlu, A. (1992). 'The role of unemployment insurance in an economy with liquidity constraints and moral hazard', *Journal of Political Economy*, vol. 100(1), pp. 118–142.

Howard, R.A. (1960). *Dynamic Programming and Markov Processes*, Cambridge, MA: MIT Press.

Imrohoroğlu, A. (1989). 'Cost of business cycles with indivisibilities and liquidity constraints', *Journal of Political Economy*, vol. 97(6), pp. 1364–1383.

Jermann, U. and Quadrini, V. (2012). 'Macroeconomic effects of financial shocks', *American Economic Review*, vol. 102(1), pp. 238–271.

Johnson, S.A., Stedinger, J.R., Shoemaker, C.A., Li, Y. and Tejada-Guibert, J. (1993). 'Numerical solution of continuous-state dynamic programs using linear and spline interpolation', *Operations Research*, vol. 41(3), pp. 484–500.

Judd, K.L. (1992). 'Projection methods for solving aggregate growth models', *Journal of Economic Theory*, vol. 58(2).

Judd, K.L. (1998). *Numerical Methods in Economics*, The MIT Press.

Judd, K.L. and Solnick, A. (1994). 'Numerical dynamic programming with shape-preserving splines', Stanford University.

Kahaner, D., Moler, C.B. and Nash, S. (1989). *Numerical Methods and Software*, New Jersey: Prentice-Hall.

Kehoe, P.J. and Perri, F. (2002). 'International business cycles with endogenous incomplete markets', *Econometrica*, vol. 70(3), pp. 907–928.

Kiyotaki, N. and Moore, J. (1997). 'Credit cycles', *Journal of Political Economy*, vol. 105(2), pp. 211–248.

Krusell, P. and Smith, A.A. (1998). 'Income and wealth heterogeneity in the macroeconomy', *Journal of Political Economy*, vol. 106(5), pp. 867–896.

Kumhof, M. and Ranciere, R. (2011). 'Inequality, leverage and crises', Manuscript, Paris School of Economics.

Lucas, R.E. (1987). *Models of Business Cycles*, New York: Blackwell.

Ludvigson, S. (1999). 'Consumption and credit: A model of time-varying liquidity constraints', *Review of Economics and Statistics*, vol. 81(3), pp. 434–447.

Ludvigson, S.C. and Michaelides, A. (2001). 'Does buffer-stock saving explain the smoothness and excess sensitivity of consumption?', *The American Economic Review*, vol. 91(3), pp. 631–647.

Malin, B.A., Krueger, D. and Kubler, F. (2011). 'Solving the multi-country real business cycle model using a smolyak-collocation solving the multi-country real business cycle model using a smolyak-collocation method', *Journal of Economic Dynamics and Control*, vol. 35(2), pp. 229–239.

McGrattan, E.R. (1996). 'Solving the stochastic growth model with a finite element method', *Journal of Economic Dynamics and Control*, vol. 20, pp. 19–42.

Milgrom, P. and Segal, I. (2002). 'Envelope theorems for arbitrary choice sets', *Econometrica*, vol. 70(2), pp. 583–601.

Moler, C.B. (2004). *Numerical Computing with MATLAB*, SIAM.

Puterman, M.L. and Brumelle, S.L. (1979). 'On the convergence of policy iteration in stationary dynamic programming', *Mathematics of Operations Research*, vol. 4(1), pp. 60–69.

Rincon-Zapatero, J.P. and Santos, M.S. (2009). 'Differentiability of the value function without interiority assumptions', *Journal of Economic Theory*, vol. 144(5), pp. 1948–1964.

Rockafellar, R.T. (1970). *Convex Analysis*, Princeton University Press.

Santos, M.S. and Rust, J. (2003). 'Convergence properties of policy iteration', *SIAM Journal on Control and Optimization*, vol. 42(6), pp. 2094–2115.

Santos, M.S. and Vigo-Aguiar, J. (1998). 'Analysis of a numerical dynamic programming algorithm applied to economic models', *Econometrica*, vol. 66(2), pp. 409–426.

Schroder, B.S.W. (2008). *Mathematical Analysis: A Consise Introduction*, Hoboken, New Jersey: John Wiley & Sons, Inc.

Stachurski, J. (2008). 'Continuous state dynamic programming via nonexpansive approximation', *Computational Economics*, vol. 31(2), pp. 141–160.

Stokey, N.L., Lucas, J.R.E. and Prescott, E.C. (1989). *Recursive Methods in Economic Dynamics*, Harvard University Press.

Szeidl, A. (2002). 'On ergodic distributions and buffer stock saving models', Manuscript, Harvard University.

Wachsmuth, G. (2013). 'On licq and the uniqueness of lagrange multipliers', *Operations Research Letters*, vol. 41(1), pp. 78–80.

A Proofs

LEMMA 1. *The minimiser, $\mu(x,z)$, of (5) is a continuous function with respect to x and z.*

Proof. By the definition of a saddle function, the fact that $\mu \geq 0$ and $m_j(x,\hat{y},z) < 0$, for all x, z and j, it follows that

$$(Tf)(x,z) \geq L(x,\hat{y},z,\mu) \geq F(x,\hat{y},z) + \beta \int_Z f(\hat{y},z')Q(z,dz') - \mu' m(x,\hat{y},z). \tag{A.1}$$

Which implies that μ is bounded. Denote the upper bound as $\bar{\mu}$.

Given a certain $\mu \in [0,\bar{\mu}]^r$, define $\tilde{g}(x,z,\mu)$ as

$$\tilde{g}(x,z,\mu) = \underset{y \in X}{\operatorname{argmax}}\, L(x,y,z,\mu). \tag{A.2}$$

By Berge's Theorem of the Maximum, $L(x,\tilde{g}(x,z,\mu),z,\mu)$ is a continuous function. Hence, the set of minimisers $\mu(x,z)$ that solve the dual problem

$$\min_{0 \leq \mu \leq \bar{\mu}} L(x,\tilde{g}(x,z,\mu),z,\mu), \tag{A.3}$$

is an upper hemicontinuous correspondence in x and z. By Assumptions 2 and 3, $\mu(x,z)$ is single valued and consequently a continuous function in x and z (Wachsmuth, 2013). □

A.1 Proof of Proposition 1

Proof. It is sufficient to show that $T : C^1(S) \to C^1(S)$.

Define the saddle function

$$L(x,g(x,z),z,\mu(x,z)) = (Tf)(x,z) = F(x,g(x,z),z) + \beta \int_Z f(g(x,z),z')Q(z,dz') - \mu(x,z)' m(x,g(x,z),z). \tag{A.4}$$

Pick an $x \in int(X)$ and an $\hat{x} \in X$ such that $x_j = \hat{x}_j$ for all $j \neq i$ and $\hat{x}_i = x_i + \varepsilon$, where x_i denotes the ith element of the vector x. For notational convenience, denote the policy and multiplier functions from (5) as g, μ and $\hat{g}$, $\hat{\mu}$ for (x,z) and $(\hat{x},z)$ respectively.

The definition of a saddle function implies

$$L(\hat{x},g,z,\hat{\mu}) \leq L(\hat{x},\hat{g},z,\hat{\mu}) \leq L(\hat{x},\hat{g},z,\mu), \tag{A.5}$$

and

$$L(x,\hat{g},z,\mu) \leq L(x,g,z,\mu) \leq L(x,g,z,\hat{\mu}). \tag{A.6}$$

Combine these two expressions and divide by $\hat{x}_i - x_i$

$$\frac{L(\hat{x},g,z,\hat{\mu}) - L(x,g,z,\hat{\mu})}{\hat{x}_i - x_i} \leq \frac{(Tf)(\hat{x},z) - (Tf)(x,z)}{\hat{x}_i - x_i} \leq \frac{L(\hat{x},\hat{g},z,\mu) - L(x,\hat{g},z,\mu)}{\hat{x}_i - x_i}. \tag{A.7}$$

By Lemma 1 and the results on page 5, the functions g and μ are continuous. Consequently the limits of $\hat{g}$ and $\hat{\mu}$ exist and equal $\lim_{\hat{x}\to x} \hat{g} = g$, $\lim_{\hat{x}\to x} \hat{\mu} = \mu$. Hence

$$\lim_{\hat{x}\to x} \frac{L(\hat{x},g,z,\hat{\mu}) - L(x,g,z,\hat{\mu})}{\hat{x}_i - x_i} = \lim_{\hat{x}\to x} \frac{L(\hat{x},\hat{g},z,\mu) - L(x,\hat{g},z,\mu)}{\hat{x}_i - x_i} = \frac{\partial L(x,g,z,\mu)}{\partial x_i}. \tag{A.8}$$

By the Pinching (Squeeze) Theorem

$$\lim_{\hat{x}\to x} \frac{(Tf)(\hat{x},z) - (Tf)(x,z)}{\hat{x}_i - x_i} = \frac{\partial L(x,g,z,\mu)}{\partial x_i} = \frac{\partial (Tf)(x,z)}{\partial x_i}. \tag{A.9}$$

Thus

$$(Tf)_x(x,z) = L_x(x,g,z,\mu) = F_x(x,g,z) - J_x(x,g,z)\mu(x,z). \tag{A.10}$$

If v_0 is a weakly concave and differentiable function, the desired result is achieved. □

A.2 Proof of Proposition 2

Proof. A sufficient condition for a maximum is a saddle point of the Lagrangian

$$L(x,y,z,\mu) = F(x,y,z) + \beta \int_Z v_n(y,z')Q(z,dz') - \mu'_{n+1} m(x,y,z). \tag{A.11}$$

By Proposition 1, the value function $v_n(y,z')$ is differentiable and by Assumption 3, given minimisers μ_{n+1}, sufficient conditions for a saddle point are thus

$$0 = F_y(x,y,z) + \beta \int_Z v_{x,n}(y,z')Q(z,dz') - J_y(x,y,z)\mu_{n+1}. \tag{A.12}$$

By Proposition 1, this can be rewritten as

$$0 \;=\; F_y(x,y,z) \;-\; J_y(x,y,z)\mu_{n+1}(x,z) \;+\; \beta \int_Z [F_x(y,h_n(y,z'),z') \;-\; J_x(y,h_n(y,z'),z')\mu_n(y,z')]Q(z,dz'). \tag{A.13}$$

Due to strict concavity the solution is unique and $h_{n+1}(x,z) = g_{n+1}(x,z)$. □

B An Improvement Algorithm

Value function iterations are considered prohibitively costly in many practical applications as convergence occur at a linear rate. Howard's improvement algorithm (Howard, 1960) can at least partly circumvent this issue, as convergence can be shown to be quadratic (see Puterman and Brumelle (1979), and Santos and Rust (2003)). This part of the Appendix shows, albeit somewhat heuristically, that some of the ideas of Howard (1960) can be extended to the current setting of time iteration.

Let g_n denote the policy function associated with $T\tilde{v}_{n-1}$. Here $\tilde{v}_{n-1}$ denotes some candidate value function, and T is the operator defined as in (4). Howard's improvement algorithm then suggests to find improvements to $T\tilde{v}_{n-1}$, which I will call $\tilde{v}_n^{h+1}$, through the iterative procedure

$$\tilde{v}_n^{h+1}(x,z) = \{F(x,g_n(x,z),z) + \beta \int_Z \tilde{v}_n^h(g_n(x,z),z')Q(z,dz')\}, \tag{B.1}$$

with $\tilde{v}_n^h = \tilde{v}_{n-1}$, for $h = 0$. This procedure is then repeated until $h = H$, or, say, until the distance between $\tilde{v}_n^{h+1}$ and $\tilde{v}_n^h$ is sufficiently small.[46] The updated $\tilde{v}_n$ is given as $\tilde{v}_n = \tilde{v}_n^H$, and the new policy function g_{n+1} solves $T\tilde{v}_n$:

$$T\tilde{v}_n = \max_{y \in \Gamma(x,z)} \{F(x,y,z) + \beta \int_Z \tilde{v}_n^H(y,z')Q(z,dz')\}. \tag{B.2}$$

The procedure repeats itself until $\{\tilde{v}_n\}$ converges. As previously mentioned, $\tilde{v}_n$ converges to v at a quadratic rate.

Under standard value function iteration any element in the sequence $\{v_n\}$ is known to be concave and differentiable (Proposition 1). However, the same cannot be said about any element in $\{\tilde{v}_n\}$.[47] From a theoretical perspective this may cause problems as concavity and differentiability are properties often exploited to solve the optimisation problem in (B.2). In practice, however, it is quite common to ignore these issues and proceed under the hypothesis that each element $\tilde{v}_n^H$ are known to be concave and differentiable, even if this may not be the case. The optimisation problem in (B.2) is then solved by finding the solution to the first-order conditions

$$0 = F_y(x,y,z) - J_y(x,y,x)\mu_{n+1}(x,z) + \beta \int_Z \tilde{v}_{x,n}^H(y,z')Q(z,dz'), \tag{B.3}$$

where $\tilde{v}_{x,n}^H$ denotes – following the notation used throughout this paper – the ℓ-vector of derivatives of $\tilde{v}_n^H$.

[46] For notational convenience I will let $\tilde{v}_n^H$ denote the outcome of this procedure, irrespective of the stopping criterion.

[47] Unless, of course, $H = 1$.

A sufficient, albeit strong, condition to ensure that each $\tilde{v}_n^{h+1}$ indeed is differentiable – and this is where the heuristic part comes in – is that g_n, the policy function, is differentiable. Let $G_{x,n}$ denote the Jacobian of the policy function g_n. The ℓ-vector of derivatives of $\tilde{v}_n^{h+1}$ is then recursively given by

$$\tilde{v}_{x,n}^{h+1}(x,z) = F_x(x, g_n(x,z), z) + G_{x,n}(x,z)[F_y(x, g_n(x,z), z) + \beta \int_Z \tilde{v}_{x,n}^h(g_n(x,z), z')Q(z, dz')]. \tag{B.4}$$

Interestingly, equations (B.3) and (B.4) suggests an obvious improvement algorithm to augment time iteration. *First*, given a candidate $\tilde{v}_{x,n-1}^H$, we can find g_n as the solution to (B.3). This step would be identical to the updating step in Howard's improvement algorithm – equation (B.2) – under the familiar assumptions of concavity and differentiability. *Second* – defining $\tilde{v}_{x,n} = \tilde{v}_{x,n-1}^H$ and $\tilde{v}_{x,n}^h = \tilde{v}_{x,n}$, for $h = 0$ – we would use equation (B.4) to repeatedly find improvements of the *derivative* of $\tilde{v}_n$, and iterate until $h = H$, or until until the distance between $\tilde{v}_{x,n}^{h+1}$ and $\tilde{v}_{x,n}^h$ is sufficiently small. This step differs from the analogous step in Howard's algorithm in that we never bother to either find, nor improve upon, the value function itself, but only its derivative. While perhaps obvious, it is worth emphasizing that under the same assumptions as those commonly imposed in many practical applications – i.e. differentiability of the policy function g_n, and concavity of $\tilde{v}_n^h$ – the sequence of policy functions $\{g_n\}$ that emerges under the derived improvement algorithm will converge to the limiting function g. And convergence will occur at *exactly* the same rate as that of Howard's improvement algorithm.

C A Benchmark Solution

For the benchmark solution I discretise the state space K into a set $\bar{K}$ containing $1,000,000$ equidistant nodes $\bar{K} = \{k_1, k_2, \ldots, k_N\}$. For each $(k,z) \in \bar{K} \times Z$ and each $n = 0, 1, \ldots$, I solve

$$Tv_{n-1}(k,z) = \max_{k' \in \bar{\Gamma}(k)} \{u(zf(k) + (1-\delta)k - k') + \beta \sum_{z' \in Z} v_{n-1}(k', z')P_{z,z'}\}, \tag{C.1}$$

with feasibility correspondence

$$\bar{\Gamma}(k) = \{k' \in \bar{K} : k' \geq (1-\delta)k\}. \tag{C.2}$$

The solution is then improved upon until $h = 400$ using

$$v_n^h(k,z) = u(zf(k) + (1-\delta)k - g_n(k,z)) + \beta \sum_{z' \in Z} v_n^{h-1}(k', z')P_{z,z'}, \tag{C.3}$$

with $v_n^{h-1} = v_{n-1}$ for $h = 1$, and where $g_n(k,z)$ is the maximiser of (C1). The updated value function is then set to $v_n = v_n^{400}$, and the procedure repeats itself until $\|v_{n+1} - v_n\| < 1(-9)$. The initial guess for the value function is taken from the solution of the algorithm in Section 3.2.1 using 10,000 nodes. The maximisation step is carried out by a rather brute force search algorithm which guarantees global optimality. As this procedure is known to converge to the true solution as the mesh size of the discretisation approaches zero (Santos and Vigo-Aguiar, 1998), I consider the solution as virtually exact.

D Additional accuracy measures

Table D1 presents the same accuracy results as those in Tables 2 and 3 but while using the procedure described in Section 3.5 on page 22.

Table D1: *Comparison of different solution methods, models (5) and (6)*

	Value Function Iteration			Time Iteration (linear)			Time Iteration (pchip)		
Grid size, N	10	100	1,000	10	100	1,000	10	100	1,000
Model (1)									
Max error	5.2(-3)	-1.6(-6)	-1.8(-6)	6.2(-3)	1.3(-6)	-1.8(-6)	4.1(-4)	-1.7(-6)	-1.8(-6)
Min error	1.3(-3)	-3.5(-5)	-3.4(-5)	1.7(-3)	-3.4(-5)	-3.4(-5)	2.2(-5)	-3.4(-5)	-3.4(-5)
Mean error	2.6(-3)	-1.3(-5)	-1.3(-5)	2.3(-3)	-1.2(-5)	-1.3(-5)	1.1(-4)	-1.3(-5)	-1.3(-5)
Model (2)									
Max error	4.5	2.8	8.3(-2)	36.3	2.9	2.3(-3)	21.3	9.6(-1)	2.4(-3)
Min error	2.0(-2)	-1.8(-3)	-3.1(-3)	7.6(-1)	1.4(-3)	-3.0(-3)	8.9(-4)	-1.6(-3)	-3.1(-3)
Mean error	5.3(-1)	3.2(-3)	-7.2(-5)	1.3	4.0(-3)	-8.6(-5)	1.2(-1)	6.2(-4)	-8.6(-5)
Model (3)									
Max error	2.2(-7)	-1.2(-8)	-1.1(-8)	2.4(-6)	-1.1(-8)	-1.1(-8)	1.4(-7)	-1.2(-8)	-1.1(-8)
Min error	1.4(-7)	-5.8(-8)	-5.7(-8)	2.1(-6)	-5.8(-8)	-5.7(-8)	9.0(-8)	-5.8(-8)	-5.7(-8)
Mean error	1.9(-7)	-1.7(-8)	-1.7(-8)	2.3(-6)	-1.7(-8)	-1.7(-7)	1.3(-7)	-1.7(-8)	-1.6(-8)
Model (4)									
Max error	5.9(-2)	1.8(-6)	-5.0(-7)	2.1(-3)	-2.7(-7)	-5.0(-7)	1.8(-3)	-2.4(-7)	-5.0(-7)
Min error	4.6(-2)	1.3(-7)	-8.8(-7)	1.9(-3)	-7.1(-7)	-8.9(-7)	1.6(-3)	-7.3(-7)	-8.9(-7)
Mean error	4.7(-2)	4.7(-7)	-5.6(-7)	1.9(-3)	-3.8(-7)	-5.6(-7)	1.7(-3)	-3.9(-7)	-5.6(-7)
Model (5)									
Max error	6.9(-4)	-1.8(-7)	-1.9(-7)	3.7(-4)	-2.1(-8)	-1.9(-7)	1.3(-6)	-1.9(-7)	-1.9(-7)
Min error	6.4(-5)	-3.8(-6)	-3.8(-6)	2.5(-4)	-3.7(-7)	-3.8(-6)	-3.1(-6)	-3.8(-6)	-3.8(-6)
Mean error	1.1(-4)	-5.0(-7)	-5.0(-7)	3.2(-4)	-3.6(-7)	-5.0(-7)	3.7(-7)	-5.0(-7)	-5.0(-7)
Model (6)									
Max error	2.1(-3)	-6.1(-5)	-6.2(-4)	8.0(-3)	-6.0(-5)	-6.2(-5)	5.0(-4)	-6.2(-5)	-6.2(-5)
Min error	-9.5(-5)	-9.9(-5)	-9.9(-5)	1.7(-3)	-9.9(-5)	-9.9(-5)	-9.5(-5)	-9.9(-5)	-9.9(-5)
Mean error	8.0(-5)	-7.9(-5)	-7.9(-5)	2.7(-3)	-7.8(-5)	-7.9(-5)	-3.0(-5)	-7.9(-5)	-7.9(-5)
Model (7)									
Max error	2.5(-1)	5.6(-4)	-1.2(-5)	1.1(-1)	3.2(-4)	-1.2(-5)	2.7(-3)	-1.2(-5)	-1.2(-5)
Min error	5.4(-2)	-1.3(-4)	-1.4(-4)	4.7(-2)	6.0(-5)	-1.4(-4)	1.2(-3)	-1.4(-4)	-1.4(-4)
Mean error	1.1(-1)	-5.8(-5)	-7.0(-5)	9.7(-2)	1.7(-4)	-7.0(-5)	1.7(-3)	-7.0(-5)	-7.0(-5)

Notes. This table does not including the computational time nor the number of iterations as the solution methods have not changed, and these are therefore identical to those of Tables 2 and 3. The same disclaimer that applies for Tables 2 and 3 also, of course, applies here.