Value Function Iteration as a Solution Method for the Ramsey Model

By Burkhard Heer[a,b] and Alfred Maußner[c]

[a] Free University of Bolzano-Bozen, School of Economics and Management, Via Sernesi 1, 39100 Bolzano-Bozen, Italy, Burkhard.Heer@unibz.it
[b]CESifo
[c] University of Augsburg, Department of Economics, Universitätsstraße 16, 86159 Augsburg, Germany, alfred.maussner@wiwi.uni-augsburg.de

March 31, 2008

Abstract

Value function iteration is one of the standard tools for the solution of the Ramsey model. We compare six different ways of value function iteration with regard to speed and precision. We find that value function iteration with cubic spline interpolation between grid points dominates the other methods in most cases. For the initialization of the value function over a fine grid, modified policy function iteration over a coarse grid and subsequent linear interpolation between the grid points provides a very efficient way to reduce computational time.

JEL classification: C63, C68, E32

Keywords: Value Function Iteration, Policy Function Iteration, Howard's Algorithm, Acceleration, Cubic Interpolation, Stochastic Ramsey Model

1 Introduction

Value function iteration is among the most prominent methods in order to solve Dynamic General Equilibrium (DGE) models. It is often used as a reference case for the comparison of numerical methods because of its known accuracy as in the seminal work by Taylor and Uhlig (1990) on the solution methods of nonlinear stochastic growth models or in later studies on the computation of the standard real business cycle model with flexible labor supply by Aruoba et al. (2006) and Heer and Maußner (2008a). Value function iteration is safe, reliable, and easy to implement. As one of its main disadvantages, it is slow in speed. Therefore, it is often applied in models where the dimension of the state space is low, usually one or two dimensions. In this paper, we will analyze various forms of value function iteration and consider the implications for speed and efficiency. We find that computational time of ordinary value function iteration can be reduced significantly by a factor of 10^3-10^4 if one applies cubic spline interpolation between grid points and uses the results from modified policy function iteration over a coarse grid for the initialization of the value function on the finer grid.

In this paper, we use value function iteration to compute the infinite-horizon Ramsey model with a representative agent. The consideration of value function iteration and possible ways to increase the computational speed for it, however, is also very important for the computation of heterogeneous-agent economies where agents may differ with regard to their individual state variable, for example assets or age. In these cases, value function iteration may be one of the very few feasible solution method since local methods like perturbation methods, which are most often applied to the solution of business cycle models in practise, break down.[1] Similarly, the use of non-local methods like projection methods or parameterized expectations may not be applicable either because the underlying interval of the individual state space is simply too large to allow for a good approximation of the policy function by a polynomial function.[2] The latter approximation methods are particularly vulnerable to a change of behavior in the policy function if a constraint becomes binding. For example, the labor supply of households may become zero if wealth exceeds a certain treshhold value. As a consequence, the optimal labor supply function displays a kink at this point and function approximation methods may behave poorly.[3]

[1]Another discrete-space method that may be applied in these cases is the finite-element method. For this method, see McGrattan (1999).

[2]For an introduction to and a discussion of the different numerical solution methods see Judd (1998) or Heer and Maußner (2008b).

[3]Christiano and Fisher (2000) have studied the use of projection methods in the case of a non-

In addition, the application of value function iteration methods may not be confined to one- or two-dimensional problems: 1) With the advance of computer technology, three or four-dimensional problems may soon be solvable with value function iteration for acceptable accuracy. 2) In many applications, the curvature of the value function with respect to some state variables may be so small that few grid points in some dimensions of the state variable will be sufficient.[4] 3) Often, we need a good initial value for methods that rely upon the approximation of a function over a large interval. In our own work, we find in a model of the equity premium that projection methods do not find the solution if the initialization is not very close to the solution and we therefore had to apply time-consuming genetic search algorithms (see Heer and Maußner (2008b), Chapter 6.3). 4) The dimension of the individual state space may sometimes be larger than the dimension of the variables that are actually needed for the arguments of the value function. For example, Erosa and Ventura (2002) consider a household optimization problem where the individual household has the three-dimensional state variable consisting of his individual productivity, real money, and capital. They solve the problem in two steps. In the first step, they compute the value function as a function of wealth, which is the sum of money and capital, and individual productivity. In a second stage, they solve the optimal portfolio problem how to allocate wealth on money and capital.

In the following, we consider various value function iteration methods for the computation of the infinite-horizon Ramsey model.[5] In section 2, we describe our six different methods of computation. As an illustration, we will apply these value function iteration methods to the computation of the deterministic Ramsey model. In section 3,

negativity constraint on investment. The constraint is accommodated by the use of a parameterized Lagrange multiplier function and can be handled successfully. The method is fast and accurate. In this case, however, the treshhold value of the individual state variable capital at which the constraint becomes binding is known. In the example of the non-negativity constraint on endogenous labor supply, on the contrary, the exact wealth value for the kink may not be known in advance and can only be found iteratively, which may cause significant computational problems.

[4]Heer (2007), for example, considers the business cycle dynamics of the income distribution in an Overlapping Generations model. The value function of the individual is also a function of the aggregate capital stock. He finds that a grid of 7 points over this variable is sufficient.

[5]We conjecture that our main result also carries over to finite-horizon models like the Overlapping Generations model. In these models, value function iteration is usually much faster than in infinite-horizon models as the value function is found in one iteration starting in the last period of the agent's life, even though there is a trade-off as the value function has to be computed and stored for each age. Our results suggest that value function iteration with cubic spline interpolation is a very fast and accurate method for these kinds of models as well.

we present our findings: 1) The modified policy function iteration scheme is found to be superior to Howard's algorithm for fine grids over the state space. 2) We also show that value function iteration with cubic spline interpolation dominates the other algorithms. In section 4, we extend our analysis to the two-dimensional case of a stochastic economy. In this case, simple value function iteration is no longer feasible as the computational time becomes prohibitive. We find that value function iteration with cubic spline interpolation is still the dominant algorithm in the case of high accuracy. For more moderate levels of accuracy, modified policy function iteration is a viable alternative. In addition, we show that a good initial guess for the value function vitally improves the computational speed by an order of 10. We use modified policy iteration over a coarse grid to come up with a good initial value for the value function over the finer grid using linear interpolation between grid points. Section 5 concludes.

2 Description of the Value Function Iteration Algorithms

In this section, we present the following six different forms of the value function iteration algorithm that we will analyze with regard to speed and accuracy:

1. Simple value function iteration,
2. Value function iteration that exploits the monotonicity of the policy function and the concavity of the value function,
3. Policy function iteration,
4. Modified policy function iteration,
5. Value function iteration with linear interpolation between grid-points,
6. Value function iteration with cubic interpolation between grid-points.

The algorithms are best explained by means of an example. We choose the deterministic infinite-horizon Ramsey model that serves as the basic structure for most business-cycle and growth models.

The Deterministic Infinite-Horizon Ramsey Model We assume that a fictitious planer[6] equipped with initial capital K_0 chooses a sequence of future capital stocks

[6]Equally, we could have considered the decentralized economy where the household optimizes his intertemporal consumption and supplies his labor and capital in competitive factor markets.

$\{K_t\}_{t=1}^{\infty}$ that maximizes the life-time utility of a representative household

$$U_0 = \sum_{t=0}^{\infty} \beta^t u(C_t), \quad \beta \in (0,1),$$

subject to the economy's resource constraint

$$f(K_t) \geq C_t + K_{t+1},$$

and non-negativity constraints on consumption C_t and the capital stock K_{t+1}. The utility function $u(C_t)$ is strictly concave and twice continuously differentiable. The function $f(K_t) = F(N, K_t) + (1-\delta)K_t$ determines the economy's current resources as the sum of output $F(N, K_t)$ produced from a fixed amount of labor N and capital services K_t and the amount of capital left after depreciation, which occurs at the rate $\delta \in (0,1)$. The function f is also strictly concave and twice continuously differentiable.

Value function iteration rests on a recursive formulation of this maximization problem in terms of the Bellman equation:

$$v(K) = \max_{0 \leq K' \leq f(K)} \quad u(f(K) - K') + \beta v(K'). \tag{1}$$

This is a functional equation in the unknown value function v. Once we know this function, we can solve for K' as a function h of the current capital stock K. The function $K' = h(K)$ is known as the policy function.

The optimal sequence of capital stocks monotonically approaches the stationary solution K^* determined from the condition $\beta f'(K^*) = 1$. Thus, the economy will stay in the interval $[K_0, K^*]$ (or in the interval $[K^*, K_0]$ if $K_0 > K^*$). In order to solve the model numerically, we compute its solution on a discrete set of n points. In this way, we transform our problem from solving the functional equation (1) in the space of continuous functions (an infinite dimensional object) to the much nicer problem of determining a vector of n elements.[7]

Our next decision concerns the number of points n. A fine grid $\mathscr{K} = \{K_1, K_2, \dots K_n\}$, $K_i < K_{i+1}$, $i = 1, 2, \dots, n$, provides a good approximation. On the other hand, the number of function evaluations that are necessary to perform the maximization step on the right hand-side (rhs) of the Bellman equation increases with n so that computation time places a limit on n. We will discuss the relation between accuracy and computation time below. For the moment being, we consider a given number of grid-points n.

[7]Note, however, that the stationary solution of this new problem will differ from K^*. For this reason we will use $\bar{K} > K^*$ as an upper bound of the state space.

A related question concerns the distance between neighboring points in the grid. In our applications we will work with equally spaced points $\Delta = K_{i+1} - K_i$ for all $i = 1, 2, \ldots, n-1$. Yet, as the policy and the value function of the original problem are more curved for low values of the capital stock, the approximation is less accurate in this range. As one solution to this problem, one might choose an unequally-spaced grid with more points in the lower interval of state space; for instance $K_i = K_1 + \Delta(i-1)^2$, $\Delta = (K_n - K_1)/(n-1)^2$, or choose a grid with constant logarithmic distance, $\Delta = \ln K_{i+1} - \ln K_i$. However, one can show that neither grid type dominates uniformly across applications.

In our discrete model the value function is a vector $\mathbf{v}$ of n elements. Its ith element holds the life-time utility U_0 obtained from a sequence of capital stocks that is optimal for the given initial capital stock $K_0 = K_i \in \mathscr{K}$. The associated policy function can be represented by a vector $\mathbf{h}$ of indices. As before, let i denote the index of $K_i \in \mathscr{K}$, and let $j \in 1, 2, \ldots, n$ denote the index of $K' = K_j \in \mathscr{K}$, that is, the maximizer of the rhs of the Bellman equation for a given K_i. Then, $\hat{h}^i = j$.

The vector $\mathbf{v}$ can be determined by iterating over

$$\begin{aligned} v_i^{s+1} &= \max_{K_j \in \mathscr{D}_i} \quad u(f(K_i) - K_j) + \beta v_j^s, \quad i = 1, 2, \ldots, n, \\ \mathscr{D}_i &:= \{K \in \mathscr{K} : K \leq f(K_i)\}. \end{aligned}$$

Successive iterations will converge linearly at the rate β to the solution $\mathbf{v}^*$ of the discrete valued infinite-horizon Ramsey model according to the contraction mapping theorem.[8]

Method 1: Simple Value Function Iteration The following steps describe a very simple to program algorithm to compute $\mathbf{v}^*$. First, we initialize the value function. Since we know that the solution to

$$\max_{K'} \quad u(f(K) - K') + \beta \cdot 0$$

is $K' = 0$, we initialize $\mathbf{v}_i^0$ with $u(f(K_i))\, \forall i = 1, \ldots, n$. In the next step we find a new value and policy function as follows: For each $i = 1, \ldots, n$:

Step 1: compute

$$w_j = u(f(K_i) - K_j) + \beta v_j^0, \; j = 1, \ldots, n.$$

[8]See, e.g., Theorem 12.1.1 of Judd (1998), p. 402.

Step 2: Find the index j^* such that

$$w_{j^*} \geq w_j \, \forall j = 1, \dots, n.$$

Step 3: Set $h_i^1 = j^*$ and $v_i^1 = w_{j^*}$.

In the final step, we check if the value function is close to its stationary solution. Let $\|\mathbf{v}^0 - \mathbf{v}^1\|_\infty$ denote the largest absolute value of the difference between the respective elements of $\mathbf{v}^0$ and $\mathbf{v}^1$. The contraction mapping theorem implies that $\|\mathbf{v}^1 - \mathbf{v}^*\| \leq \epsilon(1-\beta)$ for each $\epsilon > 0$. That is, the error from accepting $\mathbf{v}^1$ as solution instead of the true solution $\mathbf{v}^*$ cannot exceed $\epsilon(1-\beta)$. In our applications, we set $\epsilon = 0.01$.

Method 2: Value Function Iteration that Exploits the Monotonicity of the Policy Function and the Concavity of the Value Function We can improve upon the method 1 if we take advantage of the specific nature of the problem. First, the number of iterations can be reduced substantially if the initial value function is closer to its final solution. Using K^* from the continuous valued problem as our guess of the stationary solution, the stationary value function is defined by

$$v_i^* = u(f(K^*) - K^*) + \beta v_i^*, \quad \forall i = 1, 2, \dots, n,$$

and we can use $v_i^* = u(f(K^*) - K^*)/(1-\beta)$ as our initial guess.

Second, we can exploit the monotonicity of the policy function, that is:

$$K_i \geq K_j \Rightarrow K_i' = h(K_i) \geq K_j' = h(K_j).$$

As a consequence, once we find the optimal index j_1^* for K_1, we do not need to consider capital stocks smaller than $K_{j_1^*}$ in the search for j_2^* any longer. More generally, let j_i^* denote the index of the maximization problem in Step 2 for i. Then, for $i+1$ we evaluate $u(F(N, K_i) - K_j) + \beta v_j^0$ only for indices $j \in \{j_i^*, \dots n\}$.

Third, we can shorten the number of computations in the maximization Step 2, since the function

$$\phi(K') := u(f(K) - K') + \beta v(K') \tag{2}$$

is strictly concave.[9] A strictly concave function ϕ defined over a grid of n points either takes its maximum at one of the two boundary points or in the interior of the grid. In

[9]Since the value function, as well as the utility and the production function, are strictly concave.

the first case the function is decreasing (increasing) over the whole grid, if the maximum is the first (last) point of the grid. In the second case the function is first increasing and then decreasing. As a consequence, we can pick the mid-point of the grid, K_m, and the point next to it, K_{m+1}, and determine whether the maximum is to the left of K_m (if $\phi(K_m) > \phi(K_{m+1})$) or to the right of K_m (if $\phi(K_{m+1}) > \phi(K_m)$). Thus, in the next step we can reduce the search to a grid with about half the size of the original grid. Kremer (2001), pp. 165f, proves that search based on this principle needs at most $\log_2(n)$ steps to reduce the grid to a set of three points that contains the maximum. For instance, instead of 1000 function evaluations, binary search requires no more than 13! We describe this principle in more detail in the following algorithm:

Algorithm 2.1 (Binary Search)

Purpose: *Find the maximum of a strictly concave function $f(x)$ defined over a grid of n points $\mathscr{X} = \{x_1, ..., x_n\}$*

Steps:

Step 1: Initialize: Put $i_{min} = 1$ and $i_{max} = n$.

Step 2: Select two points: $i_l = floor((i_{min} + i_{max})/2)$ and $i_u = i_l + 1$, where $floor(i)$ denotes the largest integer less than or equal to $i \in \mathbb{R}$.

Step 3: If $f(x_{i_u}) > f(x_{i_l})$ set $i_{min} = i_l$. Otherwise put $i_{max} = i_u$.

Step 4: If $i_{max} - i_{min} = 2$, stop and choose the largest element among $f(x_{i_{min}})$, $f(x_{i_{min+1}})$, and $f(x_{i_{max}})$. Otherwise return to Step 2.

Finally, the closer the value function gets to its stationary solution, the less likely it is that the policy function changes with further iterations. So usually one can terminate the algorithm, if the policy function has remained unchanged for a number of consecutive iterations. Algorithm 2.2 summarizes our second method:

Algorithm 2.2 (Value Function Iteration in the Deterministic Growth Model)

Purpose: *Find an approximate policy function of the recursive problem*

Steps:

Step 1: Choose a grid

$$\mathscr{K} = [K_1, K_2, \dots, K_n], \; K_i < K_j, \; i < j = 1, 2, \dots n.$$

Step 2: Initialize the value function: $\forall i = 1, \dots, n$ *set*

$$v_i^0 = \frac{u(f(K^*) - K^*)}{1-\beta},$$

where K^* *denotes the stationary solution to the continuous-valued Ramsey problem.*

Step 3: Compute a new value function and the associated policy function, $\mathbf{v}^1$ *and* $\mathbf{h}^1$*, respectively: Put* $j_0^* \equiv 1$*. For* $i = 1, 2, \dots, n$*, and* j_{i-1}^* *use Algorithm 2.1 to find the index* j_i^* *that maximizes*

$$u(f(K_i) - K_j) + \beta v_j^0$$

in the set of indices $\{j_{i-1}^*, j_{i-1}^* + 1, \dots, n\}$*. Set* $h_i^1 = j_i^*$ *and* $v_i^1 = u(f(K_i) - K_{j_i^*}) + \beta v_{j_i^*}^0$.

Step 4: Check for convergence: If $\|\mathbf{v}^0 - \mathbf{v}^1\|_\infty < \epsilon(1-\beta)$, $\epsilon \in \mathbb{R}_{++}$ *(or if the policy function has remained unchanged for a number of consecutive iterations) stop, else replace* $\mathbf{v}^0$ *with* $\mathbf{v}^1$ *and* $\mathbf{h}^0$ *with* $\mathbf{h}^1$ *and return to step 3.*

Method 3: Policy Function Iteration Value function iteration is a slow procedure since it converges linearly at the rate β, that is, successive iterates obey

$$\|\mathbf{v}^{s+1} - \mathbf{v}^*\| \le \beta \|\mathbf{v}^s - \mathbf{v}^*\|,$$

for a given norm $\|x\|$. Howard's improvement algorithm or policy function iteration is a method to enhance convergence. Each time a policy function $\mathbf{h}^s$ is computed, we solve for the value function that would occur, if the policy were followed forever. This value function is then used in the next step to obtain a new policy function $\mathbf{h}^{s+1}$. As pointed out by Puterman and Brumelle (1979), this method is akin to Newton's method for locating the zero of a function so that quadratic convergence can be achieved under certain conditions.

The value function that results from following a given policy $\mathbf{h}$ forever is defined by

$$v_i = u(f(K_i) - K_j) + \beta v_j, \quad i = 1, 2, \dots, n.$$

This is a system of n linear equations in the unknown elements v_i. We shall write this system in matrix-vector notation. Towards this purpose we define the vector $\mathbf{u} = [u_1, u_2, \dots, u_n]$, $u_i = u(f(K_i) - K_j))$, where, as before, j is the index of the optimal next-period capital stock K_j given the current capital stock K_i. Furthermore, we introduce a matrix Q with zeros everywhere except for its row i and column j elements, which equal one. The above equations may then be written as

$$\mathbf{v} = \mathbf{u} + \beta Q \mathbf{v}, \tag{3}$$

with solution $\mathbf{v} = [I - \beta Q]^{-1}\mathbf{u}$.

Policy function iterations may either be started with a given value function or a given policy function. In the first case, we compute the initial policy function by performing Step 3 of Algorithm 2.2 once. The difference occurs at the end of Step 3, where we set $\mathbf{v}^1 = [I - \beta Q^1]\mathbf{v}^0$. Q^1 is the matrix obtained from the policy function $\mathbf{h}^1$ as explained above.

If n is large, Q is a sizeable object and one may encounter a memory limit on the personal computer. For instance, if the grid contains 10,000 points Q has 10^8 elements. Stored as double precision this matrix requires 0.8 gigabyte of memory. Fortunately, Q is a sparse matrix and many linear algebra routines are able to handle this data type.[10]

Method 4: Modified Policy Iteration If it is not possible to implement the solution of the large linear system or if it becomes too time consuming to solve this system, there is an alternative to full policy iteration. Modified policy iteration with k steps computes the value function $\mathbf{v}^1$ at the end of Step 3 of Algorithm 2.2 in these steps:

$$\begin{aligned} \mathbf{w}^1 &= \mathbf{v}^0, \\ \mathbf{w}^{l+1} &= \mathbf{u} + \beta Q^1 \mathbf{w}^l, \quad l = 1, \dots, k, \\ \mathbf{v}^1 &= \mathbf{w}^{k+1}. \end{aligned} \tag{4}$$

As proved by Puterman and Shin (1978) this algorithm achieves linear convergence at rate β^{k+1} (as opposed to β for value function iteration) close to the optimal value of the current-period utility function.

[10]For instance, using the Gauss sparse matrix procedures allows to store Q in an $n \times 3$ matrix which occupies just 240 kilobyte of memory.

Methods 5 and 6: Interpolation Between Grid-Points Applying methods 1-4, we confine the evaluation of the next-period value $v(K')$ to the grid points $\mathscr{K} = \{K_1, K_2, \dots, K_n\}$. In methods 5 and 6, we also evaluate $v(K')$ off grid points using interpolation techniques. We will consider two kinds of function approximation: linear interpolation (method 5) and cubic spline interpolation (method 6). The two interpolation schemes assume that a function $y = f(x)$ is tabulated for discrete pairs (x_i, y_i). Linear interpolation computes $\hat{y} \simeq f(x)$ for $x \in [x_i, x_{i+1}]$ by drawing a straight line between the points (x_i, y_i) and (x_{i+1}, y_{i+1}). The cubic spline determines a function $\hat{f}_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ that connects neighboring points and where the first and the second derivatives agree at the nodes.[11] The first method provides a smooth function between grid points that is continuous (but not differentiable) at the nodes (K_i, v_i). The second method determines a smooth (continuously differentiable) function over the complete set of points (K_i, v_i). Since the current-period utility function is smooth anyway, we are able to approximate the rhs of the Bellman equation (2) by a continuous function $\hat{\phi}(K)$:

$$\hat{\phi}(K) := u(f(K_i) - K_j) + \hat{v}(K_j), \tag{5}$$

where $\hat{v}$ is determined by interpolation, either linearly or cubically.

In the interval $[K_{j-1}, K_{j+1}]$ the maximum of $\hat{\phi}$ is located either at the end-points or in the interior. For this reason, we need a method that is able to deal with both boundary and interior solutions of a one-dimensional optimization problem. In order to locate the maximum, we use Golden Section Search.

Accordingly, for methods 5 and 6, we need to modify Step 3 of Algorithm 2.2 in the following way: we determine j_i^* as before and then refine the solution. First, assume that j_i^* is the index neither of the first nor of the last grid-point so that the optimum of (2) is bracketed by $I_j = [K_{j_i^*-1}, K_{j_i^*+1}]$. Instead of storing the index j_i^*, we now locate the maximum of (5) in I_j with the aid of Golden Section Search and store the maximizer $\tilde{K}_{j_i^*} \in I_j$ in the vector $\mathbf{h}$ in position i. $\hat{\phi}(\tilde{K}_j)$ is stored in v_i. If $j_i^* = 1$, we evaluate (5) at a point close to K_1. If this returns a smaller value than the one at K_1, we know that the maximizer is equal to K_1. Otherwise, we locate $\tilde{K}_{j_i^*}$ in $[K_1, K_2]$. We proceed analogously, if $j_i^* = n$.

In summary, we use the six different algorithms to compute the approximate solution of the infinite-horizon Ramsey model with $u(C) = [C^{1-\eta} - 1]/(1-\eta)$ and $F(N, K) = K^\alpha$

[11]In particular, we use secant Hermite splines where the first derivative at the endpoints is set equal to the slope of the secant.

providing us with the solutions $C_t = \hat{h}^C(K_t)$ and $K_{t+1} = \hat{h}^K(K_t)$ for consumption and the capital stock, respectively. We evaluate their performance with respect to computation time and accuracy as measured by the error e in the Euler equation:

$$u'((1+e)C_t) = \beta u'(C_{t+1}) f'(K_{t+1}), \tag{6}$$

with $C_t = \hat{h}^C(K_t)$, $K_{t+1} = \hat{h}^K(K_t)$ and $C_{t+1} = \hat{h}^C(K_{t+1})$. The Euler residual e provides a unit-free measure of the percentage error in the first-order equation of the household and is a standard measure of accuracy in similar studies like Aruoba et al. (2006) or Heer and Maußner (2008a).

We used a notebook with a dual core 2 gigahertz processor.[12] The parameters of the model are set equal to $\alpha = 0.27$, $\beta = 0.994$, $\eta = 2.0$, and $\delta = 0.011$. The value and the policy functions are computed on a grid of n points over the interval $[0.75K^*, 1.25K^*]$. We stopped iterations if the maximum absolute difference between successive approximations of the value function became smaller than $0.01(1-\beta)$ or if the policy function remained unchanged in 30 consecutive iterations (this latter criterium is only applicable for methods 1 through 4). Modified policy iterations use $k = 30$. The Euler equation residuals are computed for 200 equally spaced points in the smaller interval $[0.8K^*, 1.2K^*]$. Linear – and in the case of method 6 – cubic interpolation was used to compute the policy function between the elements of the vector $\mathbf{h}$.

3 Evaluating the Algorithms in the Deterministic Infinite-Horizon Ramsey Model

Table 1 presents the maximum absolute value of the 200 Euler equation residuals in the computation of the deterministic Ramsey model. As can be seen from the first row of this table, computation time becomes prohibitive for simple value function iteration if n is getting large. Even on a grid of 5,000 points the algorithm requires more than 7 hours to converge. For the same n, Algorithm 2.2 needs just 4 minutes, and modified policy iteration (method 4) 1 minute and 18 seconds. The rows labeled 3 and 4 in the upper panel of Table 1 convey a second finding. Policy iteration requires more time than modified policy iteration if n is reasonably large. In our example, this occurs somewhere between $n = 250$ and $n = 500$. The time needed to solve the large linear system (3) considerably slows down the algorithm. For a sizable grid of $n = 10,000$ points, method 4 is about five times faster than method 3.

[12]The source code is available in the Gauss program *Ramsey2d.g* and can be downloaded from Alfred Maußner's homepage 'http://www.wiwi.uni-augsburg.de/vwl/maussner/'.

Table 1

Run Time

Method	n=250	$n = 500$	$n = 1,000$	$n = 5,000$	$n = 10,000$
1	0:00:43:06	0:03:04.44	0:12:39:51	7:16:36:28	
2	0:00:05:63	0:00:12:91	0:00:28.94	0:04:00:67	0:09:16:91
3	0:00:02:08	0:00:05:02	0:00:14:22	0:06:18:61	0:22:11:48
4	0:00:02:31	0:00:04:47	0:00:08:31	0:01:18:53	0:04:39:17
5	0:01:05:97	0:02:34:89	0:06:36:89	1:25:07:61	7:43:13:78
6	0:01:15:92	0:02:27:94	0:04:48:80	0:22:41:84	0:44:14:28

Euler Equation Residuals

Method	$n = 250$	$n = 500$	$n = 1,000$	$n = 5,000$	$n = 10,000$
1	4.009E-2	2.061E-2	9.843E-3	1.835E-3	
2	4.009E-2	2.061E-2	9.843E-3	1.835E-3	8.542E-4
3	4.026E-2	2.061E-2	9.363E-3	2.562E-3	8.722E-4
4	4.026E-2	2.061E-2	8.822E-3	3.281E-3	8.542E-4
5	5.814E-4	4.605E-4	2.339E-4	4.093E-5	2.013E-5
6	3.200E-7	3.500E-7	3.200E-7	3.800E-7	3.600E-7

Notes: The method numbers are explained in the main text. Run time is given in hours:minutes:seconds:hundreth of seconds on a dual core 2 gigahertz processor. The empty entry pertains to a simulation which we interrupted after 8 hours of computation time. Euler equation residuals are computed as maximum absolute value of 200 residuals computed on an equally spaced grid of 200 points over the interval $[0.8K^*, 1.2K^*]$.

It should come as no surprise that adding interpolation between grid-points to Step 3 of Algorithm 2.2 increases computation time. After all, we must determine the line connecting two points of the grid and must locate the maximizer of (5) via a search routine. Method 5 requires almost eight hours to converge, if n equals 10,000. It is, however, surprising, that cubic interpolation, which requires additional computations as compared to linear interpolation, is nevertheless quite faster for large grids. In the case of $n = 10,000$ the algorithm converged after about three quarters of an hour. It seems that the smoother cubic function – though more expensive to compute – allows a quicker determination of $\tilde{K}_{j_i^*}$.

In the case of methods 1 through 4 the Euler equation residuals decrease from about 4.E-2 to about 9.E-4, if n increases from 250 to 10,000. It, thus, requires a sizable grid to obtain an accurate solution. Linear interpolation (method 5) achieves residuals of size 6.E-4 already with $n = 250$. In the case of $n = 10,000$ (i.e., with 40 times more points), the Euler residual shrinks by a factor of 20 at the cost of many hours of patience before we could discover this result. Cubic interpolation achieves very high accuracy at $n = 250$ that cannot be increased by making the grid finer. The high degree of accuracy that can be achieved with this method even for a small number of grid-points is further illustrated in Figure 1.

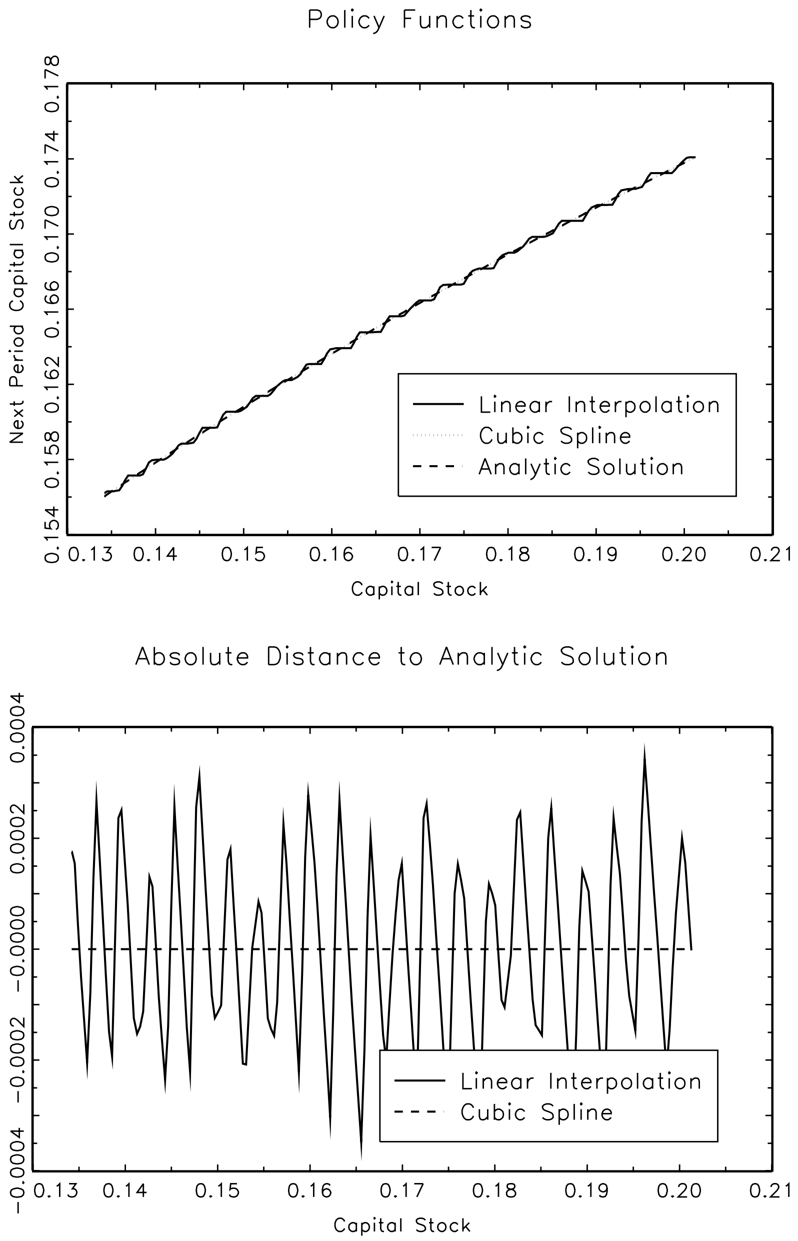

Figure 1: Policy Functions of the Next-Period Capital Stock of the Infinite-Horizon Ramsey Model

The upper panel of this figure plots the analytic policy function of the model, which is given by $K' = \alpha\beta K^{\alpha}$ in the case of $\eta = \delta = 1$ together with two approximate solutions. Both use a grid of $n = 100$ points over $[0.75K^*, 1.25K^*]$. The solution obtained from linear interpolation between the grid-points wriggles around the true solution, whereas the solution based on cubic interpolation is visually not distinguishable from the latter. Although even the first approximate solution is close to the true one (the maximum absolute value of the distance to the true solution is less than 4.E-4) the second approximation is so close that the distance to the analytic solution is almost zero (see the lower panel of Figure 1).

In summary, the cubic interpolation between grid-points outperforms the other five methods. It needs only slightly more than a minute (see Table 1) to compute a highly accurate approximate solution of the deterministic growth model (see the column $n = 250$ in Table 1).

4 Adapting and Evaluating the Algorithms for the Stochastic Infinite-Horizon Ramsey Model

In this section, we extend our analysis from a one-dimensional to a two-dimensional value function problem. We, therefore, introduce a productivity shock in the deterministic infinite-horizon model. Production Y_t in period t is now given by:

$$Y_t = Z_t f(K_t).$$

The stochastic productivity Z_t is assumed to follow a stationary stochastic process. The central planner maximizes the expected discounted life-time utility:

$$U_0 = E_0 \sum_{t=0}^{\infty} \beta^t u(C_t), \quad \beta \in (0,1),$$

subject to the resource constraint

$$Z_t f(K_t) + (1-\delta)K_t \geq C_t + K_{t+1},$$

and non-negativity constraints on consumption C_t and the capital stock K_{t+1}. Expectations E_0 are taken conditional on the information available at time $t = 0$.

We can also reformulate the problem in a recursive representation. As the problem is independent of time, we, again, drop the time index. The solution of the problem is a value function $v(K, Z)$ that solves the Bellman equation

$$v(K,Z) = \max_{K' \in \mathscr{D}_{K,Z}} \quad u(Z,K,K') + \beta E\left[v(K',Z')|Z\right] \tag{7}$$

where $E[\cdot|Z]$ is the mathematical expectations operator conditional on the realization of Z at the time the decision on K' is to be made, $u(Z, K, K') = u(Zf(K)+(1-\delta)K-K')$, and $\mathscr{D}_{K,Z} := \{K' : 0 \leq K' \leq Zf(K) + (1-\delta)K\}$.

Approximations of $E[\cdot|Z]$ As in the previous section, we replace the original problem by a discrete valued problem and approximate the value function by an $n\times m$ matrix $V = (v_{ij})$, whose row i and column j argument gives the value of the optimal policy, if the current state of the system is the pair (K_i, Z_j), $K_i \in \mathscr{K} = \{K_1, K_2, \dots, K_n\}$, $Z_j \in \mathscr{Z} = \{Z_1, Z_2, \dots, Z_m\}$.

The further procedure depends on the model's assumptions with respect to Z. There are models that assume that Z is governed by a Markov chain with realizations given by the set $\mathscr{Z}$ and transition probabilities given by a matrix $P = (p_{jl})$, whose row j and column l element is the probability of moving from Z_j to state Z_l. Given $\mathscr{Z}$ and the matrix P, the Bellman equation of the discrete valued problem is

$$\begin{aligned} v_{ij} = \max_{K_k \in \mathscr{D}_{ij}} \quad & u(Z_j, K_i, K_k) + \beta \sum_{l=1}^{m} p_{jl} v_{kl}, \\ & i = 1, 2, \dots, n,\ j = 1, 2, \dots, m, \end{aligned} \tag{8}$$

where we use $\mathscr{D}_{ij}$ as a shorthand for the set $\mathscr{D}_{K_i,Z_j}$. As in the previous section, we can use iterations over this equation to determine the matrix V.

Suppose, as it is often the case in the modelling of business cycle fluctuations, that $\ln Z$ follows an AR(1)-process:

$$\ln Z' = \varrho \ln Z + \sigma\epsilon', \quad \varrho \in [0,1),\ \epsilon' \sim N(0,1). \tag{9}$$

The first approach to tackle this case is to use Tauchen's algorithm that provides a Markov chain approximation of the continuous valued AR(1)-process (see Tauchen, 1986). To use this algorithm, one must provide the size of the interval $I_Z = [Z_1, Z_m]$ and the number of grid-points m. The algorithm determines the grid $\mathscr{Z} = \{Z_1, Z_2, \dots, Z_m\}$ and the matrix of transition probabilities $P = (p_{jl})$ so that the discrete-valued Bellman equation (8) still applies. The boundaries of $\mathscr{Z}$ must be chosen so that Z remains in the interval I_Z. The usual procedure is to set $Z_m - Z_1$ equal to a multiple of the unconditional standard deviation of the process (9), which equals[13]

$$\sigma_Z = \sqrt{\frac{\sigma^2}{1-\varrho^2}}.$$

[13]See, e.g., Hamilton (1994), pp. 53-56 for a derivation of this formula.

One can use simulations of this process to find out if it leaves a given interval. Usually, an interval of size equal to $9\sigma_Z$ or $10\sigma_Z$ is large enough. Tauchen (1986) provides evidence that even 9 grid-points are sufficient for a reasonably good approximation of (9).

The second approach to approximate the conditional expectation on the rhs of the Bellman equation (7) rests on the analytic expression for $E(\cdot|Z)$. In the case of the process (9) this equals

$$E\left[v(K',Z')|Z\right] = \int_{-\infty}^{\infty} v\left(K', e^{\varrho \ln Z + \sigma\epsilon'}\right) \frac{e^{\frac{-(\epsilon')^2}{2}}}{\sqrt{2\pi}}\, d\epsilon'.$$

If the value function is tabulated in the matrix $V = (v_{ij})$, we can interpolate between the row-elements of V to obtain an integrable function of Z, which allows us to employ numeric integration techniques to compute $E[\cdot|Z]$. For the normal distribution, Gauss-Hermite quadrature is a suitable method since the weight function is given by $w(x) = e^{-x^2}$. In Heer and Maußner (2008a), however, we point to a serious drawback of this approach. Gauss-Hermite quadrature requires a much larger interval for Z than it will be necessary for simulations of the model. I_Z must contain the integration nodes $\pm\sqrt{2}\sigma x$, where x denotes the largest node used by the respective Gauss-Hermite formula. For instance, $x \simeq 1.65$ in the four-nodes formula that we usually employ to compute a conditional expectation. In particular, we have to ascertain that $\varrho \ln Z_m + \sqrt{2}\sigma x \leq \ln Z_m$ and $\varrho \ln Z_1 - \sqrt{2}\sigma x \geq \ln Z_1$. For given ϱ, σ, and x these equations can be solved for the lower and the upper bound Z_1 and Z_m, respectively. For our parameter values this delivers $|\ln Z_m - \ln Z_1| \simeq 21\sigma_Z$. Thus, instead of using an interval of size $10\sigma_Z$, one must use an interval of size $21\sigma_Z$. Yet, as explained below, the boundaries of $\mathscr{K}$ will usually depend on the boundaries of $\mathscr{Z}$. For a given number of grid-points n, a larger interval $I_K = [K_1, K_n]$ implies a less accurate solution that may outweigh the increase of precision provided by the continuous-valued integrand. With respect to the stochastic infinite-horizon Ramsey model we indeed find that the Markov chain approximation allows a much faster computation of the value function for a given degree of accuracy.[14] For this reason, we will consider this approach only.

[14]See Heer and Maußner (2008a).

The Basic Algorithm The problem that we, thus, have to solve, is to determine V iteratively from

$$\begin{aligned} v_{ij}^{s+1} &= \max_{K_k \in \mathscr{D}_{ij}} \quad u(Z_j, K_i, K_k) + \beta \sum_{l=1}^{m} p_{jl} v_{kl}^{s}, \\ &i = 1, 2, \dots, n,\; j = 1, 2, \dots, m. \end{aligned} \tag{10}$$

This process will also deliver the policy function $H = (h_{ij})$. In our basic algorithm, this matrix stores the index k_{ij}^* of the optimal next-period state variable $K_k' \in \mathscr{K}$ in its ith row and jth column element. The pair of indices (i, j) denotes the current state of the system, that is, (K_i, Z_j). We assume that the value function v of our original problem is concave in K and that the policy function h is monotone in K so that we can continue to use all of the methods encountered in Section 2. As we have seen in Section 3, a reasonable fast algorithm should at least exploit the concavity of v and the monotonicity of h. Our basic algorithm, thus, consists of steps 1, 2.1, and 2.2i of Algorithm 4.1. We first discuss the choice of $\mathscr{K}$ and V^0 in Step 1 before we turn to methods that accelerate convergence and increase precision in Step 2.

Algorithm 4.1 (Value Function Iteration 2)

Purpose: *Find an approximate policy function of the recursive problem* (7) *given a Markov chain with elements* $\mathscr{Z} = \{Z_1, Z_2, \dots, Z_m\}$ *and transition matrix* P.

Steps:

Step 1: Choose a grid

$$\mathscr{K} = \{K_1, K_2, \dots, K_n\}, \; K_i < K_j, \; i < j = 1, 2, \dots n,$$

and initialize V^0.

Step 2: Compute a new value function V^1 *and an associated policy function* H^1*: For each* $j = 1, 2, \dots, m$ *repeat these steps:*

Step 2.1: Initialize: $k_{0j}^* = 1$.

Step 2.2: i) For each $i = 1, 2, \dots, n$ *and* k_{i-1j}^* *use Algorithm 2.1 to find the index* k^* *that maximizes*

$$w_k = u(Z_j, K_i, K_k) + \beta \sum_{l=1}^{m} p_{jl} v_{kl}^0$$

in the set of indices $k \in \{k^*_{i-1j}, k^*_{i-1j}+1, \dots, n\}$. Set $k^*_{ij} = k^*$. If interpolation is not desired, set $h^1_{ij} = k^*$ and $v^1_{ij} = w_{k^*}$, else proceed as follows: ii) (optional) If $k^* = 1$ evaluate the function $\hat{\phi}$ defined by equation (14) at a point close to K_1. If this returns a smaller value than at K_1, set $\tilde{K} = K_1$, else use Golden Section Search to find the maximizer $\tilde{K}$ of $\hat{\phi}$ in the interval $[K_1, K_2]$. Store $\tilde{K}$ in h^1_{ij} and $\hat{\phi}(\tilde{K})$ in v^1_{ij}. Proceed analogously if $k^* = n$. If k^* equals neither 1 nor n, find the maximizer $\tilde{K}$ of $\hat{\phi}$ in the interval $[K_{k^*-1}, K_{k^*+1}]$ and put $h^1_{ij} = \tilde{K}$ and $v^1_{ij} = \hat{\phi}(\tilde{K})$.

Step 2.3: (optional, if Step 2.2.i was taken) Set $\mathbf{w}^1 = \operatorname{vec} V^1$, and for $l = 1, 2, \dots, k$ iterate over

$$\mathbf{w}^{l+1} = \operatorname{vec} U + \beta Q^1 \mathbf{w}^l,$$

and replace $\mathbf{V}^1$ by the respective elements of $\mathbf{w}^{k+1}$.

Step 3: Check for convergence: if

$$\max_{\substack{i=1,\dots n\\ j=1,\dots m}} |v^1_{ij} - v^0_{ij}| \le \epsilon(1-\beta), \quad \epsilon \in \mathbb{R}_{++}$$

(or if the policy function has remained unchanged for a number of consecutive iterations) stop, else replace V^0 with V^1 and H^0 with H^1 and return to Step 2.

Choice of $\mathscr{K}$ and V^0 This choice is a bit more delicate than the respective step of Algorithm 2.2. In the deterministic growth model considered in the previous sections the optimal sequence of capital stocks is either increasing or decreasing, depending on the given initial capital stock K_0. This makes the choice of $\mathscr{K}$ easy. In a stochastic model, the future path of K depends on the expected path of Z, and we do not know in advance whether for any given pair (K_i, Z_j) the optimal policy is to either increase or decrease K. For this reason, our policy to choose $\mathscr{K}$ is "guess and verify". We will start with a small interval. If the policy function hits the boundaries of this interval, that is, if $h_{ij} = 1$ or $h_{ij} = n$ for any pair of indices, we will enlarge $\mathscr{K}$. In the case of the stochastic growth model an educated guess is the following: If the current shock is Z_j and we assume that $Z = Z_j$ forever, the sequence of capital stocks will approach K^*_j determined from

$$1 = \beta(1 - \delta + Z_j f'(K^*_j)). \tag{11}$$

Approximate lower and upper bounds are, thus, given by K_1^* and K_m^*, respectively. Since, the stationary solution of the discrete-valued problem will not be equal to the solution of the continuous-valued problem, K_1 (K_n) should be chosen as a fraction (a multiple) of K_1^* (K_m^*).

As the computation time also depends on the initial V^0, using the zero matrix is usually not the best choice, but it may be difficult to find a better starting value. For instance, in the stochastic growth model we may try $v_{ij}^0 = u(Z_j f(K_i) - \delta K_i)$, that is, the utility obtained from a policy that maintains the current capital stock for one period. Or, we may compute V^0 from the m different stationary solutions that result if Z equals Z_j forever:

$$v_{ij}^0 = u(Z_j f(K_j^*) - \delta K_j^*) + \beta \sum_{l=1}^{m} p_{jl} v_{il}^0,$$

where K_j^* solves (11). This is a system of linear equations in the nm unknowns v_{ij}^0 with solution

$$\begin{aligned} V^0 &= (I - \beta P')^{-1} U, \\ U &= (u_{ij}),\ u_{ij} = u(Z_j f(K_j^*) - \delta K_j^*), \quad \forall i, j. \end{aligned}$$

A third choice is $v_{ij}^0 = u(f(K^*) - \delta K^*)/(1-\beta)$, that is, the value obtained from the stationary solution of the deterministic growth model.

As we will show below, there is, however, an even better strategy: i) start with a coarse grid on the interval $[K_1, K_n]$; ii) use the basic algorithm to compute the value function V^* on this grid; iii) make the grid finer by using more points n. iv) interpolate column-wise between neighboring points of the old grid and the respective points of V^* to obtain an estimate of the initial value function on the finer grid. Since on a coarse grid the algorithm will converge quickly, the choice of V^0 in step i) is not really important and $V^0 = 0$ may be used.

Acceleration In Section 3, we discovered that policy function iteration is a method to accelerate convergence. This method assumes that a given policy H^1 is maintained forever. In the context of the Bellman equation (8) this provides a linear system of equation in the nm unknowns v_{ij} (for the moment, we suppress the superscript of V):

$$\begin{aligned} v_{ij} &= u_{ij} + \beta \sum_{l=1}^{m} p_{jl} v_{h_{ij} l}, \\ u_{ij} &:= u(Z_j, K_i, K_{h_{ij}}),\ i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, m. \end{aligned} \tag{12}$$

In matrix notation, this may be written as

$$\operatorname{vec} V = \operatorname{vec} U + \beta Q \operatorname{vec} V, \quad U = (u_{ij}). \tag{13}$$

$\operatorname{vec} V$ ($\operatorname{vec} U$) is the nm column vector obtained from vertically stacking the rows of V (U). The $nm \times nm$ matrix Q is obtained from H and P: Its row $r = (i-1)m + j$ elements in columns $c_1 = (h_{ij} - 1)m + 1$ through $c_m = (h_{ij} - 1)m + m$ equal the row j elements of P. All other elements of Q are zero. Even for a grid $\mathscr{Z}$ with only a few elements m, Q is much larger than its respective counterpart in equation (3). In the previous section we have seen that for $n > 500$ (and, in the notation of this section $m = 1$), modified policy iteration is faster than full policy iteration. For this reason, we only will implement modified policy iteration into our algorithm. This is done in Step 2.3 of Algorithm 4.1

Interpolation We know from the results obtained in Section 3 that interpolation between the points of $\mathscr{K}$ is one way to increase the precision of the solution. Within the current framework the objective is to obtain a continuous function $\hat{\phi}(K)$ that approximates the rhs of (7) given the tabulated value function in the matrix V and the grid $\mathscr{K}$. We achieve this by defining

$$\hat{\phi}(K) = u(Z_j, K_i, K) + \beta \sum_{l=1}^{m} p_{jl} \hat{v}_l(K). \tag{14}$$

The function $\hat{v}_l(K)$ is obtained from interpolation between two neighboring points K_i and K_{i+1} from $\mathscr{K}$ and the respective points v_{il} and v_{i+1l} from the matrix V. Thus, each time the function $\hat{\phi}(K)$ is called by the maximization routine, m interpolation steps must be performed. For this reason, interpolation in the context of a stochastic model is much more time consuming than in the case of a deterministic model. Our algorithm allows for either linear or cubic interpolation in the optional Step 2.2.ii.

Evaluation The four methods 2, 4, 5, and 6 for the value function iteration are applied to the computation of the stochastic infinite-horizon Ramsey model. We use the functions $u(C) = [C^{1-\eta} - 1]/(1-\eta)$ and $f(K) = K^\alpha$ and measure the accuracy of the solution by the residuals of the Euler equation

$$((1+e)C)^{-\eta} = E\left\{ \left[\beta (C')^{-\eta} \left(1 - \delta + \alpha (e^{\varrho \ln Z + \sigma \epsilon'})(K')^{\alpha - 1}\right)\right] \Big| Z \right\}.$$

Again, C, C' and K' are computed by the policy functions. For example, the police function for consumption is given by

$$\hat{h}^C(K, Z) = Z K^\alpha + (1-\delta)K - \hat{h}^K(K, Z).$$

The policy function for the next-period capital stock $\hat{h}^K$ is obtained from bilinear interpolation between the elements of the matrix H. The residuals are computed over a grid of 200^2 points over the interval $[0.8K^*, 1.2K^*] \times [0.95, 1.05]$. Table 2 displays the maximum absolute value of the 200^2 residuals. We used a notebook with a dual core 2 gigahertz processor.[15] The parameters of the model are set equal to $\alpha = 0.27$, $\beta = 0.994$, $\eta = 2.0$, $\delta = 0.011$, $\varrho = 0.90$, and $\sigma = 0.0072$. The value and the policy function are computed on a grid of $n \times m$ points. The size of the interval $I_Z = [Z_1, Z_m]$ equals 11 times the unconditional standard deviation of the AR(1)-process in equation (9). We stopped iterations, if the maximum absolute difference between successive approximations of the value function became smaller than $0.01(1-\beta)$ or if the policy function remained unchanged in 50 consecutive iterations (this latter criterium is only applicable for methods 2 and 4.) Modified policy iterations use $k = 30$.

Table 2

Method	n	m	Run Time		Euler Equation Residual
			i	ii	
2	250	9	0:00:22:06		7.407E-2
4	250	9	0:00:22:94		7.407E-2
5	250	9	2:13:37:84	0:13:31:16	1.272E-3
6	250	9	2:04:01:67	0:21:01:69	1.877E-4
6	500	9	5:12:58:44	0:23:17:52	1.876E-4
6	250	15		1:04:39:22	4.930E-6
2	10,000	9	2:33:26:16	0:20:10:94	1.933E-3
4	10,000	9	1:06:48:58	0:03:52:42	1.933E-3
4	10,000	31	1:06:49:52	0:13:40:80	1.931E-3
4	100,000	15		0:17:59:56	2.089E-4
4	500,000	15		3:43:03:81	4.387E-5

Notes: The method numbers are explained in the main text. Run time is given in hours:minutes:seconds:hundreth of seconds on a dual core 2 gigahertz processor. The column labeled i gives the run time where the initial value function was set equal to $u(f(K^*) - \delta K^*)/(1-\beta)$, column ii presents computation time from a sequential approach: we start with a coarse grid of $n = 250$ and increase the number of grid points in a few steps to the desired value of n given in the second column. Except in the first step – where we use the same initial V^0 as in the third column – each step uses the value function obtained in the previous step to initialize V^0. Euler equation residuals are computed as maximum absolute value of 200^2 residuals computed on an equally spaced grid over the interval $[0.8K^*, 1.2K^*] \times [0.95, 1.05]$. Empty entries indicate simulations, which we have not performed for obvious reasons.

[15]The source code is available in the Gauss program *Ramsey3d.g* from Alfred Maußner's homepage. Also available is a Fortran version of Algorithm 4.1.

On a coarse grid for the capital stock, $n = 250$, the first four rows in Table 2 confirm our intuition. Interpolation increases computation time drastically, from about 25 seconds (for methods 2 and 4) to over 2 hours but provides reasonably accurate solutions. In the case of method 5 (method 6) the Euler equation residual is about 50 times (400 times) smaller than that obtained from methods 2 and 4.

The run times given in column ii highlight the importance of a good initial guess for the value function. The results presented there were obtained in the following way. We used method 4 to compute the value function on a grid of $n = 250$ points (given the choice of m as indicated in the table). For this initial step we use $v_{ij} = u(f(K^*) - \delta K^*)/(1 - \beta)$ as our guess of V. In successive steps we made the grid finer until the number of points given in column 2 was reached. Each step used the previous value function, employed linear interpolation to compute the additional points in the columns of V, and took the result as initial guess of the value function. The computation time in column ii is the cumulative sum over all steps. In the case of method 5 this procedure reduced computation time by about 2 hours! The entries for method 2 and 4 and $n = 10,000$ in column i confirm our findings from the deterministic Ramsey model that modified policy iteration is an adequate way to reduce computation time (by almost one and half an hour). Since it is faster close to the true solution, it clearly outperforms method 2 in successive iterations (see the entries for $n = 10,000$ and $n = 9$ in column i and ii): it is about 5 times faster as compared to 2.3 times in the simulations without a good initial value function.

The entries for method 6 document that increased precision does not result from additional points in the grid for the capital stock but in the grid for the productivity shock. In the case $n = 250$ and $m = 15$ the Euler equation residual of about 5.E-6 indicates a very accurate solution. However, even with good starting values, it takes about an hour to compute this solution.

There are two adequate ways to compute a less precise but still sufficiently accurate solution with Euler equation residual of about 2.E-4: either with method 6 on a coarse grid, $n = 250$ and $m = 9$ or with method 4 on a much finer grid, $n = 100,000$ and $m = 15$. Both methods require about 20 minutes to compute the policy function. Thus, different from our findings in the previous section, cubic interpolation is not unambiguously the most favorable method.

However, if high precision is needed, cubic interpolation on a coarse grid is quite faster than method 4. As the last row of Table 2 shows, even on a fine grid of $n = 500,000$ points the Euler equation residual is still about 10 times larger than that from method

6 for $n = 250$ and $m = 15$. Yet, whereas method 6 requires about an hour to compute the policy function, method 4 needs almost four hours.

5 Conclusion

We find that modified policy function iteration dominates Howard's algorithm in terms of speed for high accuracy and in higher dimensional state space. Modified policy function iteration is also shown to be an ideal candidate for the provision of an initial guess for the value function, which may speed up the computation by the factor 10. Value function iteration with cubic spline interpolation dominates the other algorithms in terms of speed in both the deterministic and stochastic infinite-horizon model if high accuracy is needed. This might be the case, for example, if the researcher would like to study the mean return on equity in the Ramsey model as pointed out by Christiano and Fisher (2002). We, therefore, carefully advocate the use of value function iteration with cubic spline interpolation where the initial value is found in a tatônnement process over a coarser grid with the help of modified policy function iteration. If our results generalize to more complex models with three- or four-dimensional state space is an open question which requires more experience in a variety of alternative models.

References

Aruoba, S.B., J. Fernández-Villaverde, and J.F. Rubrio-Ramírez. 2006. Comparing Solution Methods for Dynamic Equilibrium Economies. *Journal of Economic Dynamics and Control*. Vol. 30. pp. 2477-2508.

Christiano, L.J., and J.D.M. Fisher. 2000. Algorithms for Solving Dynamic Models with Occasionally Binding Constraints. *Journal of Economic Dynamics and Control*. Vol. 24. pp. 1179-1232.

Erosa, A., and G. Ventura. 2002. On inflation as a regressive consumption tax. *Journal of Monetary Economics*. Vol. 49. pp. 761-95.

Hamilton, J. 1994. *Time Series Analysis*, Princeton, N.J.: Princeton University Press.

Heer, B., 2007, On the modeling of the income distribution business cycle dynamics. *CESifo working paper* No. 1945.

Heer, B., and A. Maußner. 2008a. Computation of Business Cycle Models: A Comparison of Numerical Methods. *Macroeconomic Dynamics*. forthcoming.

Heer, B., and A. Maußner. 2008b. *Dynamic General Equilibrium Modelling: Computational Methods and Applications*. 2nd edition. Berlin: Springer. forthcoming.

Judd, K.L. 1998. *Numerical Methods in Economics*, Cambridge, MA.: MIT Press.

Kremer, J. Arbeitslosigkeit, Lohndifferenzierung und wirtschaftliche Entwicklung. 2001. Köln: Josef Eul Verlag.

McGrattan, E.R. 1999. Application of Weighted Residual Methods to Dynamic Economic Models, in: R. Marimon and A. Scott (Eds.), *Computational Methods for the Study of Dynamic Economies*. Oxford and New York: Oxford University Press. pp. 114-142.

Puterman, Martin L. and Moon Chirl Shin. 1978. Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*. Vol. 24. pp. 1127-1237.

Puterman, Martin L. and Shelby Brumelle. 1979. On the Convergence of Policy Iteration in Stationary Dynamic Programming. *Mathematics of Operations Research*. Vol. 4. pp. 1979, 60-69.

Tauchen, G. 1986. Finite State Markov-Chain Approximations to Univariate and Vector Autoregressions. *Economics Letters*. Vol. 20. pp. 177-181.

Taylor, J.B., and H. Uhlig. 1990. Solving Nonlinear Stochastic Growth Models: A Comparison of Alternative Solution Methods. *Journal of Business and Economic Statistics*. Vol. 8. 1-17.