OPEN ACCESS
OPUS AUGSBURG

Universität Augsburg
Universitätsbibliothek

Value Function Iteration as a Solution Method for the Ramsey Model

Alfred Maußner, Burkhard Heer

Angaben zur Veröffentlichung / Publication details:

Maußner, Alfred, and Burkhard Heer. 2011. "Value Function Iteration as a Solution Method for the Ramsey Model." *Jahrbücher für Nationalökonomie und Statistik* 231 (4): 494–515.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Jahrbücher für Nationalökonomie und Statistik

Journal of Economics and Statistics

Begründet von

Bruno Hildebrand

Fortgeführt von

Johannes Conrad, Ludwig Elster
Otto v. Zwiedineck-Südenhorst
Gerhard Albrecht, Friedrich Lütge
Erich Preiser, Knut Borchardt
Alfred E. Ott und Adolf Wagner

Herausgegeben von

Peter Winker, Wolfgang Franz
Werner Smolny, Peter Stahlecker
Adolf Wagner, Joachim Wagner

Band 231

Lucius & Lucius Stuttgart 2011

© Lucius & Lucius Verlagsgesellschaft mbH · Stuttgart · 2011
Alle Rechte vorbehalten
Satz: Mitterweger & Partner Kommunikationsgesellschaft mbH, Plankstadt
Druck und Bindung: Neumann Druck, Heidelberg
Printed in Germany

Jahrbücher f. Nationalökonomie u. Statistik (Lucius & Lucius, Stuttgart 2011) Bd. (Vol.) 231/4

Value Function Iteration as a Solution Method for the Ramsey Model

By Burkhard Heer, Bozen, and Alfred Maußner, Augsburg*

JEL C63; C68; E32

Value function iteration; policy function iteration; Howard's Algorithm; acceleration; cubic interpolation; stochastic Ramsey model; heterogeneous agents.

Received: 17.10.2008

Revision received: 16.09.2010

Accepted: 17.09.2010

Summary

Value function iteration is one of the standard tools for the solution of dynamic general equilibrium models if the dimension of the state space is one ore two. We consider three kinds of models: the deterministic and the stochastic growth model and a simple heterogenous agent model. Each model is solved with six different algorithms: (1) simple value function iteration as compared to (2) smart value function iteration neglects the special structure of the problem. (3) Full and (4) modified policy iteration are methods to speed up convergence. (5) linear and (6) cubic interpolation between the grid points are methods that enhance precision and reduce the size of the grid. We evaluate the algorithms with respect to speed and accuracy. Accuracy is defined as the maximum absolute value of the residual of the Euler equation that determines the household's savings. We demonstrate that the run time of all algorithms can be reduced substantially if the value function is initialized stepwise, starting on a coarse grid and increasing the number of grid points successively until the desired size is reached. We find that value function iteration with cubic spline interpolation between grid points dominates the other methods if a high level of accuracy is needed.

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 (2008). 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

* We would like to thank two anonymous referees for their helpful comments. All remaining errors are ours.

that computational time can be reduced significantly if an initial guess of the value function is computed on a coarse grid which is refined stepwise until the desired number of grid points, and thus, accuracy, is achieved. Depending on the method and the size of the final grid we are able to reduce run time in the deterministic Ramsey model between forteen and several ten thousand percent.

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 its computational speed, 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 methods 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 since they 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 threshold value. As a consequence, the optimal labor supply function displays a kink at this point and function approximation methods may behave poorly.[2] To study the sensitivity of our results in such an economy, we also apply the different methods to the computation of a simple heterogeneous agent model with a binding borrowing constraint.

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.[3] 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 demonstrate 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/Maußner 2009, 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

---

[1] Another discrete-space method that may be applied in these cases is the finite-element method. For this method, see McGrattan (1999). For an introduction to and a discussion of the different numerical solution methods see Judd (1998) or Heer and Maußner (2009).

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

[3] 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.

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.[4] 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, we present our findings: 1) Initializing the value function stepwise reduces run time considerably. In one case it reduces run time from five hours and 27 minutes to one hour and 25 minutes, thus, saving more than four hours. 2) Interpolation between grid points with cubic splines delivers highly accurate solutions with errors in the range between 1.4E-4 and 4.1E-7 for grids of sizes between 10 and 1000 points, respectively. 3) Except for linear interpolation on a grid with 10 points no other method is able to compute as precise a solution at the same short run time of less than six seconds. 4) Modified policy function iteration is superior to Howard's algorithm. In Section 4, we extend our analysis to the two-dimensional case. We consider two applications: the stochastic growth model and a simple heterogeneous agent model. In this extended framework simple value function iteration is no longer feasible as the computational time becomes prohibitive. Thus, in Section 5 we determine ranking of the methods based on the time they need to solve our three models for given levels of accuracy. We find that value function iteration with cubic spline interpolation is still the dominant algorithm in the case of high accuracy. In the deterministic and stochastic Ramsey model value function iteration with linear interpolation computes less accurate solutions faster than the former method, since it requires less floating point operations per iteration. Full policy iteration dominates modified policy iteration only in two out of six simulations. Section 6 concludes.

2 Description of the 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. smart 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. Henceforth we will refer to this framework as Model 1.

[4] 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.

2.1 Model 1

We assume that a fictitious planer[5] equipped with initial capital K_0 chooses a sequence of future capital stocks $\{K_t\}_{t=1}^{\infty}$ 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 $N \equiv 1$ 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:

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

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. Our aim is to obtain an approximation $\hat{h}$ of this function, which is sufficiently close to the true h.

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.[6]

Our next decision concerns the number of points n. A fine grid $\mathcal{K} = \{K_1, K_2, \ldots K_n\}$, $K_i < K_{i+1}$, $i = 1, 2, \ldots, n$, 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.

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

---

[5] 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.

[6] 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.

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 $j \in 1, 2, ..., n$ 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

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

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.[7]

2.2 Methods

Method 1: Simple Value Function Iteration. The following steps describe an algorithm for the computation of $\mathbf{v}^*$ that is very simple to program. First, we initialize the value function. One potential candidate for the choice of the initial value function $\mathbf{v}^0$ is the zero vector. We will also discuss a more elaborate initialization below. In the next step, we find a new value and policy function by iterating over all next-period capital stocks $K' = K_j$ that imply strictly positive consumption. Let $n(i)$ denote the index of the largest grid point $K' = K_{n(i)}$ that implies non-negative consumption, i.e. if $f(K_i) < K_n$, $n(i)$ satisfies $K_{n(i)} < f(K_i) <= K_{n(i)+1}$ and $n(i) = n$ otherwise. For each $i = 1, .., n$:

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

Step 2: Find the index j^* such that
$$w_{j^*} \geq w_j \, \forall j = 1, ..., n(i).$$

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 $\|\mathbf{v}^1 - \mathbf{v}^*\| \leq \varepsilon(1-\beta)$ for each $\varepsilon > 0$. That is, the error from accepting $\mathbf{v}^1$ as solution instead of the true solution $\mathbf{v}^*$ cannot exceed $\varepsilon(1-\beta)$. In our applications, we set $\varepsilon = 0.01$.

[7] See, e.g., Theorem 12.1.1 of Judd (1998: 402).

Method 2: Smart Value Function Iteration. We can improve upon the method 1, if we take advantage of the specific nature of the problem.
First, 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 $j \in \{j_i^*, \ldots n(i+1)\}$.

Second, 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.[8] 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 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 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 $\mathcal{X} = \{x_1, \ldots, x_n\}$*

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 $i \in \mathrm{R}$.

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:

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

Algorithm 2.2 (Value Function Iteration in the Deterministic Growth Model)
Purpose: *Find an approximate policy function of the recursive problem* (1)
Steps:

Step 1: Choose a grid

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

Step 2: Initialize the value function: $\forall i = 1, ..., n$ *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* $i = 1, 2, ..., n$*, 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 $\{j_{i-1}^*, j_{i-1}^* + 1, ..., n(i)\}$*. 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 $\|\mathbf{v}^0 - \mathbf{v}^1\|_\infty < \varepsilon(1 - \beta)$*,* $\varepsilon \in \mathbb{R}_{++}$ *(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

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

for a given norm $\|\mathbf{v}\|$. 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

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

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 $\mathbf{u} = [u_1, u_2, ..., u_n]$, $u_i = u(f(K_i) - K_j))$, 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.[9]

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 the following steps:

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

The parameter k is set by the user. Its optimal value depends on the number of grid points n and will be discussed in the next section. 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.

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 $\mathcal{K} = \{K_1, K_2, \ldots, K_n\}$. 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.[10] 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 anywhere, 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

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

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

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) lies in the continuous interval $I_j = [K_{j_i^*-1}, K_{j_i^*+1}]$. 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 $K'(i) \in I_j$ in the vector $\mathbf{h}$ in position i. $\hat{\phi}(K'(i))$ 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 $K'(i)$ in $[K_1, K_2]$. We proceed analogously, if $j_i^* = n$.

Choice of the Initial Value Function. The time needed for any of our methods to converge depends critically on the initial value function. In Heer and Maußner (2009) we discuss several different choices for v^0. Here we advocate for the stepwise computation of an appropriate initial value function. On a small grid, methods 1 through 4 require only a few seconds to converge, irrespective of the choice of the initial value function. Given the solution on the small grid, we successively increase the size of the grid until the desired number of grid points is reached which delivers the required accuracy of the solution. In each step we use the previous solution and interpolate linearly between the grid points to obtain the new starting value for the value function. In the case of methods 5 and 6, we always use method 4 to find an initial value function iteratively, since these methods converge very slowly.

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 $F(N,K) = K^a$ 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.

Measure of Accuracy. We evaluate their performance with respect to computation time and accuracy as measured by the error e in the Euler equation:[11]

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

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 (2008).

Calibration. The parameters of the model are set equal to $a = 0.27, \beta = 0.994, \eta = 2.0$, and $\delta = 0.011$. Our parameters are chosen in accordance with empirical observations from the West German economy during 1975-1989. $1 - a$ equals the average wage share in gross domestic product at factor prices assuming that self-employed earn the average wage of employees. β is chosen so that the annual real rate of return is approximately equal to 6 %. The quarterly depreciation rate δ equals 1.1 % in West Germany. Finally, as the empirical evidence from microeconometric studies for the intertemporal elasticity of substitution $1/\eta$ varies between values of 1/4 and 1 we follow Heer and Maußner (2009).

[11] The use of this standardized Euler equation error has been suggested by Judd (1992) and Judd and Gaspar (1997).

They choose $\eta = 2.0$ in order to have a favorable match between the empirical variances of aggregate variables and those implied by their RBC model.

Computation. We used a workstation with a quad core 2.66 gigahertz processor and 12 gigabyte of RAM.[12] 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 $k = 35$. The Euler equation residuals are computed for 20,000 equally spaced points in the interval $[0.75K^*, 1.25K^*]$. 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 model 1

In Table 1 we present our results with respect to run time, and Table 2 provides information on the accuracy of the solutions in terms of the maximum absolute value of the 20,000 Euler equation residuals. For each method the respective third line gives run time and precision relative to method 6 for $n = 10$. These numbers should be independent of the machine and the operating system.[13]

The first row of Table 1 demonstrates that 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 two and a half hours to converge. For this size of the grid, iterative initialization saves more than one hour of run time. For the same n, Algorithm 2.2 needs just two and a half minutes, and modified policy iteration (method 4) requires less than four seconds.

The rows labeled 3 and 4 in Table 1 convey a second finding. Policy iteration requires more time than modified policy iteration for the number of grid points considered in our experiments.[14] The time needed to solve the large linear system (3) slows down the algorithm. For a sizable grid of $n = 10,000$ points, method 4 is more than twice as fast as method 3. Intuitively, one would think that full policy iteration is slower than modified policy iteration at larger grids since it requires more time to solve the respective sparse linear system. However, and more importantly, there is a second reason. Without iterative initialization method 3 requires much more time to converge than method 4. Thus, a second reason for the relative worse performance of method 3 as compared to method 4 is slow convergence if the initial solution is far from the final value function.

The key parameter determining the speed of method 4 is the number of iterations with a given policy function k. Small values of k reduce the computational time spent on step (4), but the algorithm needs more iterations to converge. If k is small relative to n and if the initial value function is far from the final solution, the algorithm may not even converge in a reasonable number of steps. For instance, if n=10,000 and k <=25, the algo-

[12] The source code is available in the Gauss program Model_1.g. All computer codes can be found in the electronic data archive of the Journal of Statistics and Econometrics under 'http://www.jbnst.de/'.

[13] We had to recognize that the run time is not absolutely stable between different runs of the program on the same machine. We attribute this finding to background work of the operating system.

[14] The speed of method 3 also depends on the linear sparse matrix solver. The Gauss command Sparse Solve implements the sparse LU factorization of Demmel et al. (1999).

Table 1 Run Time for Model 1

Method	n = 10	n = 250	n = 1,000	n = 5,000	n = 10,000
1		00:00:22:85	00:04:27:82	02:38:26:14	
		00:00:22:89	00:06:41:43	03:51:28:19	
		4.04	47.37	1681.31	
2		00:00:03:99	00:00:18:03	00:02:26:55	00:03:29:26
		00:00:04:10	00:00:20:49	00:02:50:90	00:06:33:19
		0.71	3.19	25.92	37.01
3		00:00:01:39	00:00:01:72	00:00:04:80	00:00:08:28
		00:00:01:40	00:00:09:26	00:03:46:59	00:14:02:81
		0.25	0.30	0.85	1.46
4		00:00:01:01	00:00:01:61	00:00:03:12	00:00:04:12
		00:00:01:05	00:00:03:75	00:00:37:60	00:04:57:14
		0.18	0.28	0.55	0.73
5	00:00:04:64	00:00:25:100	00:02:12:27	00:25:18:53	01:23:39:35
	00:00:03:71	00:01:25:91	00:07:57:28	01:36:05:56	05:17:36:59
	0.82	4.60	23.39	268.58	887.75
6	00:00:05:65	00:00:27:82	00:02:23:43	00:26:22:40	01:25:26:00
	00:00:04:22	00:01:42:06	00:08:55:45	01:40:17:16	05:27:10:40
	1.00	4.92	25.37	279.87	906.62

Notes: The method numbers are explained in the main text. Run time is given in hours:minutes:seconds:hundreth of seconds. The empty entries refer to simulations which we did not run. For each method, the first (second) line presents the run times with (without) iterative initialization of the value function. The third line shows the relation between the run time in the respective first line relative to the boxed run time.

rithm had not converged after 1,000 iterations. Table 3 displays the relation between run time and the number of steps k in modified policy iteration. In the range k=25 and k=35 the differences in run time are small and the minimizer (the boxed values) is also in this range. Our choice of k=35 in our simulations, thus, is well justified.

In Table 2, we present our results with respect to the precision of the algorithms. In the case of methods 1 through 4 the Euler equation residuals are almost proportional to the size of the grid. They are of the order of magnitude 0.04 for n=250 and decrease to about 0.001 for n=10,000. It, thus, requires a sizable grid to obtain an accurate solution. Note also that the precision of methods 1-4 is of the same order of magnitude for a given size n of the grid (see Table 2).[15]

Adding interpolation between grid-points to Step 3 of Algorithm 2.2 increases the accuracy of the solution considerably. Even for a grid of n=10 we obtain Euler equation residuals that are almost ten times smaller than those obtained from methods 1 through 4 for n=10,000.

As can be seen from Tables 1 and 2, high precision of 4.E-5 can be obtained from method 5 for $n = 5000$. The algorithm needs about 25 minutes to compute this solution, and iterative initialization saves over one hour of run time. Method 6 obtains an even higher precision of 4.E-7 in about two and a half minutes, and, thus, is the method of choice in

[15] Methods 1 and 2 compute the same solution per construction. These solutions need not be identical to the solutions obtained from either method 3 or method 4, since these methods have different rates of convergence, as explained above.

Table 2 Euler Equation Residuals for Model 1

Method	n = 10	n = 250	n = 1,000	n = 5,000	n = 10,000
1		4.31E-2	9.89E-3	1.95E-3	
		4.31E-2	9.89E-3	1.93E-3	
		300.28	68.81	13.40	
2		4.31E-2	9.89E-3	1.93E-3	1.07E-3
		4.31E-2	9.89E-3	1.93E-3	1.08E-3
		300.28	68.81	13.40	7.48
3		4.31E-2	9.88E-3	2.11E-3	1.30E-3
		4.31E-2	1.09E-2	4.28E-3	3.29E-3
		300.28	68.79	14.68	9.07
4		4.31E-2	1.01E-2	2.26E-3	1.99E-3
		4.31E-2	1.14E-2	5.68E-3	3.34E-3
		300.28	70.37	15.71	13.87
5	1.54E-4	6.61E-4	2.40E-4	4.12E-5	2.56E-5
	1.54E-4	6.62E-4	2.40E-4	4.13E-5	2.56E-5
	1.07	4.60	1.67	0.29	0.18
6	1.44E-4	2.66E-5	4.40E-7	4.14E-7	4.30E-7
	1.44E-4	2.65E-5	4.04E-7	4.44E-7	4.55E-7
	1.00	0.19	0.00	0.00	0.00

Notes: The method numbers are explained in the main text. Euler equation residuals are the maximum absolute value of 20,000 residuals computed on an equally spaced grid of 20,000 points over the interval $[0.75K^*, 1.25K^*]$. For each method, the first (second) line presents the maximum Euler equation residual with (without) iterative initialization of the value function. The third line shows the relation between the run time in the respective first line relative to the boxed residual.

Model 1. Note, also, that we are not able to increase the precision of this method if we increase n beyond 5000 points.[16]

In the next section we extend our algorithms to a stochastic framework. We introduce two further models: the stochastic growth model and a simple heterogenous agent model. Though the algorithms to compute the value function in both models are essentially the same, there is one important difference: in the stochastic growth model we can confine the grid for the capital stock to a small interval around the stationary solution of the deterministic growth model considered in this section. In a heterogenous agent environment, however, the admissible assets are in a much larger interval and the value function is more curved at the lower endpoint of this interval.

4 Stochastic-economy extensions of the basic ramsey model

4.1 Model 2: The stochastic growth 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.

[16] To obtain the very small errors of orders 1.E-5 or even 1.E-7 in the case of the iterative initialization of the value function, we had to adjust the stopping criterium downwards for methods 5 and 6. Otherwise, the algorithms stopped to early.

Table 3 Run Time and k

k	n = 250	n = 500	n = 1,000	n = 5,000	n = 10,000
5	01:31	02:63	06:03	10:23	11:37
10	01:29	02:07	02:81	04:17	04:100
15	01:19	01:43	01:100	04:04	04:91
20	00:95	01:18	01:69	03:48	04:33
25	00:82	01:06	01:43	03:14	04:04
30	00:84	01:09	01:44	02:91	03:81
35	00:84	01:10	01:41	02:72	03:88
40	00:86	01:12	01:42	02:94	03:85

Notes: Run time is given in seconds:hundreth of seconds on a quad core 2.66 gigahertz processor. The boxed values indicate the value of k that minimizes runtime.

The 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:

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

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 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

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

where $\mathrm{E}[\cdot|Z]$ 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 $\mathrm{E}[\cdot|Z]$. 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), $K_i \in \mathscr{K} = \{K_1, K_2, ..., K_n\}$, $Z_j \in \mathscr{Z} = \{Z_1, Z_2, ..., Z_m\}$.

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

$$v_{ij} = \max_{K_r \in \mathscr{D}_{ij}} u(Z_j, K_i, K_r) + \beta \sum_{l=1}^{m} p_{jl} v_{rl}, \quad i = 1,2,...,n,\ j = 1,2,...,m, \tag{8}$$

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:

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

As explained in Heer and Maußner (2009), the best way to tackle this case is to use Tauchen's algorithm that provides a Markov chain approximation of the process (9) (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 $\mathscr{Z} = \{Z_1, Z_2, ..., Z_m\}$ 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 Basic Algorithm. The problem that we, thus, have to solve, is to determine V iteratively from

$$v_{ij}^{s+1} = \max_{K_r \in \mathscr{D}_{ij}} \quad u(Z_j, K_i, K_r) + \beta \sum_{l=1}^{m} p_{jl} v_{rl}^{s}, \quad i = 1,2,...,n,\ j = 1,2,...,m. \tag{10}$$

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 $K_r' \in \mathscr{K}$ 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* $\mathscr{Z} = \{Z_1, Z_2, ..., Z_m\}$ *and transition matrix* P.

Steps:

Step 1: Choose a grid $\mathscr{K} = \{K_1, K_2, ..., K_n\}$, $K_i < K_j$, $i < j = 1,2, ...n$, *and initialize* V^0.

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

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

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

$$w_r = u(Z_j, K_i, K_r) + \beta \sum_{l=1}^{m} p_{jl} v_{rl}^0$$

in the set of indices $r \in \{r_{i-1j}^*, r_{i-1j}^* + 1, ..., n(i)\}$. *Set* $r_{ij}^* = r^*$. *If interpolation is not desired, set* $h_{ij}^1 = r^*$ *and* $v_{ij}^1 = w_{r^*}$, *else proceed as follows: ii) (optional) If* $r^* = 1$ *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_{ij}^1 and $\hat{\phi}(\tilde{K})$ in v_{ij}^1. Proceed analogously if $r = n$. If r^* equals neither 1 nor n, find the maximizer $\tilde{K}$ of $\hat{\phi}$ in the interval $[K_{r^*-1}, K_{r^*+1}]$ and put $h_{ij}^1 = \tilde{K}$ and $v_{ij}^1 = \hat{\phi}(\tilde{K})$.*

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

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

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

Step 3: Check for convergence: if

$$\max_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}} |v_{ij}^1 - v_{ij}^0| \leq \varepsilon(1-\beta), \quad \varepsilon \in \mathbb{R}_{++}$$

(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 k and $\mathbf{V}^0$ 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^*).

In the previous section we have shown that an iterative initialization of the value function reduces run time considerably. Thus, we also use this strategy in the stochastic context.

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):

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

In matrix notation, this may be written as

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

vec V (*vec* U) is the nm column vector obtained from vertically stacking the rows of V (U). The $nm \times nm$ matrix $Q = (q_{rc})$ is obtained from H and P. It has elements $q_{rc} = p_{jl}$,

$r = (j-1)n + i$, $c = (l-1)n + h_{ij}$ and zeros elsewhere. Even for a grid $\mathscr{Z}$ with only a few elements m, Q is much larger than its respective counterpart in equation (3), and one must use sparse matrix method to solve equation (13).

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.

Measuring Accuracy. 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

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

We use Gauss-Hermite quadrature to evaluate the integral on the right-hand side of the previous equation. Again, C, C' and K' are computed by the policy functions. For example, the police function for consumption is given by

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

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 $[0.75K^*, 1.25K^*] \times [0.95, 1.05]$.[17]

Calibration. The parameters α, β, δ, and η are set as in the model of Section 3. For the parameters of the technology process, $\rho = 0.90$ and $\sigma = 0.0072$, we use the estimates obtained by Heer and Maussner (2009) who fit an AR(1) process to the time series for German productivity. 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

[17] To save on computational time we did not use the same number of points, i. e., 20,000, that we used in Section 3. We found that the difference between the maximum Euler residual on a grid over $[0.75K^*, 1.25K^*]$ with 200 and a grid with 20,000 points, respectively, is small.

policy iterations use $k = 35$. The stochastic process for the productivity shock (9) is approximated by a Markov chain with $m = 9$ points.[18]

4.2 Model 3: A simple heterogeneous agent model

As a third model we will consider a very simple model of an exchange economy that is based upon Huggett (1993). In our model, agents are heterogeneous with regard to their wealth a and their employment status $e \in \{e^h, e^l\}$. Employment is stochastic. Households are either employed, $e = e^h$, or unemployed, $e = e^l$, in which cases they receive the income $y(e^h)$ and $y(e^l)$, respectively, with $y(e^h) > y(e^l)$. In addition, they face a borrowing constraint $a_t >= \bar{a}$ in each period t.

In order to compute the solution for such a model, we need to compute the value function over a much larger interval than, for example, in the deterministic Ramsey model. Moreover, the value function is characterized by much more curvature at and in the proximity of the borrowing constraint. In our model, unemployed agents with minimum wealth $a = \bar{a}$ would like to accumulate higher debt but cannot do so. As a consequence, the computation of the policy function is prone to higher inaccuracy and/or lower speed.

The Model. The model follows Huggett (1993) with some simplifications. In our economy, households hold a single asset a that cannot be stored but can be deposited at a central credit authority at an interest rate r_t. The central credit authority administers the credit balances of all households without any transaction cost and without any fees. Furthermore, it sets an interest rate r and imposes a credit constraint $\bar{a} \leq a$. The budget constraint of the household in period t is given by

$$c_t + a_{t+1} = (1 + r_t)a_t + y(e_t), \tag{15}$$

where c_t, again, denotes consumption in period t.

The employment process follows a first-order Markov process with transition probability $\pi(e'|e) = Prob(e_{t+1} = e'|e_t = e) > 0$ for $e', e \in \{e^h, e^l\}$. The agent maximizes expected discounted utility:

$$\mathrm{E}_0\left[\sum_{t=0}^{\infty} \beta^t u(c_t)\right], \tag{16}$$

where $\beta < 1$ denotes the discount factor and instantaneous utility $u(\cdot)$ is a CES function of consumption:

$$u(c) = \frac{c^{1-\eta}}{1-\eta}. \tag{17}$$

As in the previous two sections, $1/\eta$ denotes the intertemporal elasticity of substitution. We further assume that insurance markets are missing and that households cannot pool their assets and income. As a consequence, households build up different levels of wealth a depending on their employment histories. For such an economy, it is possible to define a stationary equilibrium in which the distribution of assets is invariant and the interest rate

[18] The source code is available in the Gauss program Model_2.g from Alfred Maußner's homepage. Also available is a Fortran version of Algorithm 4.1 from the home page of our book Heer and Maußner (2009).

r is constant and set such that the credit market clears. The net credit balance of all households is equal to zero and the central credit authority has zero profits.[19]

Calibration. The income of the employed and unemployed are set equal to $y(e^h) = 1$ and $y(e^l) = 0.1$, respectively. One model period corresponds to 8.5 weeks so that 6 periods are equal to one year. The transition probabilities are calibrated such that the average duration of the low income shock (unemployment) is two model periods and the standard deviation of annual earnings is 20 %:

$$\pi(e'|e) = \begin{pmatrix} 0.925 & 0.075 \\ 0.500 & 0.500 \end{pmatrix}.$$

The equilibrium unemployment rate is 13 %.[20] Huggett (1993) sets the discount factor equal to $\beta = 0.99322$ implying an annual discount rate of 0.96. To allow for a comparison of our results to those in previous sections, we set the risk aversion coefficient η equal to 2. For the credit limit $\bar{a}$, we consider a value $\bar{a} = -2$. A credit limit of -5.3 corresponds to one year's average income. Finally, we choose an interest rate $r = -2.357\%$ which is found to be the interest rate in the stationary equilibrium of our economy.

Computation. We compute the policy functions of the households over the interval $a \in [-2, 3]$ for the employment types $e \in \{e^h, e^l\}$. The upper limit of the interval is non-binding such that in the stationary equilibrium households only hold wealth levels below it. Besides, we employ the same computational techniques as in the previous subsection except that we now consider the individual employment status rather than the aggregate productivity level as the exogenous stochastic state variable. The Euler equation residual e is computed from:

$$((1+e)c)^{-\eta} = \mathrm{E}\{\beta(c')^{-\eta}(1+r)\},$$

where c' denotes next-period consumption. Different from our procedure in the models of the previous two sections, however, we do not compute the Euler equation residuals over the complete interval $[-2, 3]$, but only for those agents that are not credit-constrained.[21, 22]

[19] The equilibrium concept and the uniqueness and existence of the solution is described in more detail in Huggett (1993). The computation of the invariant distribution is explained in Chapter 7.3.1 in Heer and Maußner (2009).

[20] Heer and Maußner (2009) explain in Chapter 12.2 how to compute the ergodic distribution.

[21] The source code is available in the Gauss program Model_3.g from Alfred Maußner's homepage.

[22] In the case of the credit constrained agent, the Euler equation for the savings decision is $u'(c_t) = \lambda_t + \beta \mathrm{E}_t u'(c_{t+1})(1+r)$, where λ_t is the Lagrange multiplier of the constraint $a_{t+1} \geq \bar{a}$. We can determine c_t as well as $u'(c_{t+1})$ from the approximate policy function of the agent. However, we have to use the optimality condition to back out λ_t so that the Euler equation residual is zero per construction.

5 Evaluation

In this section we consider the relative performance of methods 2 through 6 with regard to the solution of our three models. We do not consider method 1, since we know from Section 3 that this is definitely the worst method to use. All solutions were computed on a workstation with quad core, 2.66 gigahertz CPU, and 12 gigabyte of RAM. For each method we first determined the size of the grid that provides a given accuracy in terms of the maximum absolute value of the Euler equation residuals. We know from Table 2 that methods 2-4 require large grids to achieve an accuracy in the order of magnitude of 1.E-3. Therefore we limit attention to two sets of solutions. Table 4 ranks the methods according to the run time, if the desired accuracy is about 1.E-3. Table 5 rests on maximum Euler equations residuals around 1.E-2.

Smart value function iteration (method 2) performs worst in all our simulations. The trade-off, which determines the ranking of the remaining methods, is between high precision on small grids but many floating point operations (flops) per iteration on the one hand and low precision on small grids but few flops on the other. Methods 5 and 6 must compute the connecting line between neighboring grid points, whereby cubic spline interpolation is computationally more intensive than linear interpolation. In addition, both methods have to use an optimization routine to locate the maximizer of the right hand side of the Bellman equation between neighboring grid points. Methods 2 through 4 lack these steps. Therefore, it is not surprising that method 6 ranks first, when a small grid achieves the required accuracy. This happens in Models 2 and 3 for 1.E-3. For Model 1 both methods need only 3 points to achieve an accuracy that is even higher than the required precision of E-2 and E-3, respectively. Since both methods need only fractions of a second to compute the solution, it happened in some runs (10 out of 100) that method 5 (which needs less flops than method 6 per iteration) was slower than method 6. We attribute this imprecision in time measurement to the environment (hard- and software) in

Table 4 Relative Performance of Models 1-3: 1.E-03

Rank	Model		
	1	2	3
1	Method 5 $n=3$ 00:00:00:37 [1.00]	Method 6 $n=7$ 00:00:27:53 [1.00]	Method 6 $n=1030$ 00:00:15:50 [1.00]
2	Method 6 $n=3$ 00:00:00:0039 [1.04]	Method 3 $n=21000$ 00:02:53:72 [6.31]	Method 4 $n=13100$ 00:00:24:15 [1.56]
3	Method 4 $n=10745$ 00:00:12:28 [33.19]	Method 5 $n=125$ 00:05:06:92 [11.15]	Method 5 $n=4400$ 00:00:29:35 [1.89]
4	Method 3 $n=10745$ 00:00:24:40 [65.95]	Method 4 $n=21000$ 00:08:24:23 [18.31]	Method 3 $n=13100$ 00:01:17:31 [4.99]
5	Method 2 $n=10745$ 00:03:30:21 [568.23]	Method 2 $n=21000$ 01:01:11:81 [133.37]	Method 2 $n=13100$ 00:05:53:69 [22.82]

Notes: The method numbers are explained in the text. n is the number of grid points that delivers a maximum Euler equation residual of about 1.E-03. Euler equation residuals are computed as explained in text. Run time is given in hours:minutes:seconds:hundreth of seconds on a quad core 2.66 gigahertz processor. Numbers in brackets are run time relative to the run time of the first ranked method.

Table 5 Relative Performance of Models 1-3: 1.E-02

Rank	Model 1	Model 2	Model 3
1	Method 5 $n=3$ 00:00:37 [1.00]	Method 5 $n=4$ 00:11:63 [1.00]	Method 3 $n=1400$ 00:04:48]1.00]
2	Method 6 $n=3$ 00:00:39 [1.04]	Method 6 $n=3$ 00:20:91 [1.80]	Method 4 $n=1400$ 00:06:81 [1.52]
3	Method 4 $n=988$ 00:01:63 [4.42]	Method 4 $n=2000$ 00:57:96 [4.98]	Method 6 $n=305$ 00:07:83 [1.75]
4	Method 3 $n=985$ 00:01:74 [4.69]	Method 3 $n=2000$ 01:14:24 [6.38]	Method 5 $n=945$ 00:10:15 [2.27]
5	Method 2 $n=985$ 00:18:44 [49.84]	Method 2 $n=2000$ 10:13:75 [52.77]	Method 2 $n=1400$ 00:22:65 [5.06]

Notes: The method numbers are explained in the text. n is the number of grid points that delivers a maximum Euler equation residual of about 1.E-02. Euler equation residuals are computed as explained in text. Run time is given in minutes:seconds:hundreth of seconds on a quad core 2.66 gigahertz processor. Numbers in brackets are run time relative to the run time of the first ranked method.

which we run our models. To account for it, we report averages of runtime from 100 consecutive solutions of both models.

Table 5 shows, that, for low accuracy, linear interpolation is faster than cubic interpolation for Models 1 and 2. Both methods need only 3 or 4 grid points in this case. In the case of Model 3, even method 6 requires a relatively large grid of n=305 points to yield a precision below 1.E-2. For a smaller grid the policy function of the unemployed agent near the borrowing constraint is not sufficiently accurate. However, with n=305 points the method cannot compete in speed with methods 3 and 4, which rank first and second, respectively. Since method 5 requires over three times more grid points than method 6, it ranks fifth, though it requires less flops than method 6 per iteration. Note, that there are only two simulations in which full policy iteration is faster than modified policy iteration.

In summary, cubic spline interpolation dominates the other algorithms in case of high accuracy, while for low accuracy linear interpolation is faster. Nevertheless, methods without interpolation are viable alternatives: They rank second in both stochastic models with highly accurate solutions, and they rank top in the case of Model 3 when less precise solutions are aimed at. Remember, however, that these results depend upon our strategy to obtain a good initial guess of the value function, and that we use modified policy iteration (method 4) to obtain this guess in the case of methods 5 and 6.

6 Conclusion

We consider six different variants of value function iteration to solve three kinds of dynamic general equilibrium models. Our first method, simple value function iteration, serves an illustrative purpose only and cannot compete in speed with any other method. For this reason, we restrict attention to methods 2 through 6. We rank these methods

based on the time they need to compute a solution of a given accuracy. We measure accuracy as the maximum absolute value of the error in the model's respective Euler equation. We are able to reduce the run time of all methods considerably via an iterative initialization of the value function. Based on this strategy method 6, smart value function iteration with cubic spline interpolation, is the fastet method to solve all three models with a relatively high accuracy of about 1.E-3. Methods 5 (smart value function iteration with linear interpolation) computes solutions with larger error of about 1.E-2 in less time than method 6 in the case of the deterministic and the stochastic Ramsey model. Methods 3 and 4 (full policy iteration and modified policy iteration, respectively) solve our heterogenous agent model faster than both methods that rely upon interpolation of the value function, if the required precision is about 1.E-2. In addition, we find that except in two out of six simulations modified policy iteration is faster than full policy iteration.

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, J. F. Rubrio-Ramírez (2006), Comparing Solution Methods for Dynamic Equilibrium Economies. Journal of Economic Dynamics and Control 30: 2477-2508.

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

Demmel, J. W., S. C. Eisenstat, J. R. Gilbert, X. S. Li, J. W. H. Liu (1999), A Supernodal Approach to Sparse Partial Pivoting. SIAM Journal on Matrix Analysis and Applications 20: 720-755.

Erosa, A., G. Ventura (2002), On inflation as a Regressive Consumption Tax. Journal of Monetary Economics 49: 761-95.

Heer, B. (2007), On the Modeling of the Income Distribution Business Cycle Dynamics. CESifo Working Paper No. 1945.

Heer, B., A. Maußner (2008), Computation of Business Cycle Models: A Comparison of Numerical Methods. Macroeconomic Dynamics 12: 641-663.

Heer, B., A. Maußner (2009), Dynamic General Equilibrium Modeling: Computational Methods and Applications. 2nd Edition. Berlin.

Huggett, M. (1993), The Risk-Free Rate in Heterogenous-Agent Incomplete-Insurance Economies. Journal of Economic Dynamics and Control 17:953-69.

Judd, K. L. (1992), Projection Methods for Aggregate Growth Models. Journal of Economic Theory 58: 410-52.

Judd, K. L. (1998), Numerical Methods in Economics. Cambridge, MA.

Judd, K., J. Gaspar (1997), Solving Large-Scale Rational-Expectations Models. Macroeconomic Dynamics 1:45-75.

Kremer, J. (2001), Arbeitslosigkeit, Lohndifferenzierung und wirtschaftliche Entwicklung. Köln.

McGrattan, E. R. (1999), Application of Weighted Residual Methods to Dynamic Economic Models. Pp. 114-142 in: R. Marimon, A. Scott (eds.), Computational Methods for the Study of Dynamic Economies. Oxford/New York.

Puterman, M. L., M. C. Shin (1978), Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. Management Science 24: 1127-1237.

Puterman, M. L., S. Brumelle (1979), On the Convergence of Policy Iteration in Stationary Dynamic Programming. Mathematics of Operations Research 4: 60-69.

Tauchen, G. (1986), Finite State Markov-Chain Approximations to Univariate and Vector Autoregressions. Economics Letters 20: 177-181.

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

Prof. Dr. Burkhard Heer, Free University of Bolzano-Bozen, School of Economics and Management, Piazza Universitá, 39100 Bolzano-Bozen, Italy, and CESifo.
Burkhard.Heer@unibz.it

Prof. Dr. Alfred Maußner, University of Augsburg, Department of Economics, Universitätsstraße 16, 86159 Augsburg, Germany.
alfred.maussner@wiwi.uni-augsburg.de