Continuous vs. Discrete Time
Some Computational Insights

Rendahl, Pontus

Document Version
Final published version

Published in:
Journal of Economic Dynamics and Control

DOI:
10.1016/j.jedc.2022.104522

Publication date:
2022

Journal of Economic Dynamics & Control 144 (2022) 104522

Contents lists available at ScienceDirect

Journal of Economic Dynamics & Control

journal homepage: www.elsevier.com/locate/jedc

Continuous vs. discrete time: Some computational insights

Pontus Rendahl[1]

Department of Economics, Copenhagen Business School and CEPR, Porcelaenshaven 16, Frederiksberg 2000, Denmark

ARTICLE INFO

Article history:
Received 7 June 2022
Revised 5 September 2022
Accepted 6 September 2022
Available online 12 September 2022

Keywords:
Implicit method
Howard's improvement algorithm
Computation

ABSTRACT

Solving a workhorse incomplete markets model in continuous time is many times faster compared to its discrete time counterpart. This paper dissects the computational discrepancies and identifies the key bottlenecks. The implicit finite difference method – a commonly used tool in continuous time – accounts for a large share of the difference. This method is shown to be a special case of Howard's improvement algorithm, efficiently implemented by relying on sparse matrix operations. By representing the policy function with a transition matrix it is possible to formulate a similar procedure in discrete time, which effectively eliminates the differences in run-times entirely.

1. Introduction

Continuous time has resurfaced as a popular environment for economic models. A prominent reason is that those environments are particularly suited for an efficient numerical analysis. In the context of a workhorse heterogenous agents model, for instance, Achdou et al. (2022) show that a framework in continuous time strictly dominates that in discrete time, and can be many times faster to compute.[2] This paper identifies key components of these speed-gains, and shows how they can be efficiently implemented also in a discrete-time setting. By doing so, the discrepancy in computational time is eliminated.

The departure point of the paper is a standard heterogeneous agent model in discrete time. Continuous time is then defined as the limit when time intervals between periods approach zero. For the main analysis I confine attention to three canonical solution methods: value function iteration for discrete time, and two implementations of a finite difference method for continuous time; the *explicit* method and the *implicit* method see, for instance, (Candler, 2001).[3] While the implicit method has robust convergence properties, the explicit method can be less reliable in numerical implementations. This latter feature often necessitates very small updating steps – known as "time steps" – leading to slow convergence.[4]

In discrete time, value function iteration provides a natural benchmark as it has well-documented convergence properties. These properties carry over to any length of model's time intervals, which can be arbitrarily small as long as they remain

E-mail address: pontus.rendahl@gmail.com

[1] The author would like to thank Wouter den Haan, Jeppe Druedahl, Benjamin Moll and Karl Harmenberg for providing helpful comments and suggestions.

[2] The continuous-time framework in Achdou et al. (2022) generally ranges from being two to 30 times faster than the discrete time equivalent, but can be as much as 500 times faster depending on the metric.

[3] Value function iteration is sometimes also referred to as *the method of successive approximations*.

[4] These features are discussed in both Barles and Souganidis (1991) and Candler (2001). The explicit method does indeed converge *in theory*, but for numerical implementations the required time step may easily become prohibitively small.

https://doi.org/10.1016/j.jedc.2022.104522

strictly positive. However, at shorter time intervals the discount factor is close to one, and convergence is slow. When the interval length approaches zero, value function iteration converges to the explicit method. Thus, by heuristically linking the explicit method in continuous time to value function iteration in discrete time, this paper provides an insight as to why the explicit method generally has slow convergence.

However, while value function iteration is susceptible to a discount factor close to one – and, by extension, also the explicit method in continuous time – the same is not true for *Howard's improvement algorithm* (see e.g. Rust, 1996).[5] Thus, this improvement algorithm also appears promising for a continuous-time setting. Indeed, as the length of time intervals approach zero, the limit of Howards improvement algorithm coincides with the implicit method. Thus, the implicit method is the continuous time equivalent to Howards improvement algorithm in discrete time.[6] The "time step" in the implicit method corresponds to a smoothing, or dampening, parameter for the update in the improvement algorithm; setting this time step to infinity is then isomorphic to setting the dampening parameter in discrete time to zero.[7] This strong connection between the improvement algorithm in discrete time and the implicit method in continuous time, provides a heuristic argument as to why the implicit method requires few iterations for convergence.

Yet, while Howard's improvement algorithm in discrete time is often considered prohibitively slow, the implicit method is fast.[8] The reason is relatively straightforward: the continuous-time formulation of the Bellman equation – the Hamilton-Jacobi-Bellman (HJB) equation – does not contain future values of the value function, but only (the derivative of) current values. As a consequence, the update of the value function can be formulated as the solution to a system of linear equations. And as this system is sparse it can be solved efficiently. The same is not true in discrete time.

To obtain a similar result in discrete time, I use an idea from Young (2010) and represent the policy function with a transition matrix. Each row of this transition matrix corresponds to a grid point of the current state. And the *only* nonzero elements of each row are the two adjacent grid points surrounding the optimal choice at the indicated state. These elements contain weights such that multiplied by their associated grid points, and added together, recovers the optimal choice. This representation of the policy function permits an approximate formulation of the improvement algorithm written as a (sparse) system of linear equations. By exploiting this formulation, there is no remaining difference in run-time between the discrete and continuous time, and the model can be solved 30–40 times faster compared to value function iteration. Furthermore, I extend this idea to be applicable directly on the Euler equation, and similar speed gains obtains.[9]

Some of these results are, of course, partly driven by the specific model studied, and continuous-time frameworks may have other advantages that are not fully exploited in the current setting (e.g. Ahn et al., 2017 and Fernández-Villaverde et al., 2019). For instance, and as mentioned above, the HJB equation does not contain future values of the value function. Thus, in the presence of continuous stochastic processes – such as an Ornstein-Uhlenbeck process and its discrete time autoregressive representation – continuous time yields additional benefits, as it does not require the explicit computation of expected values.[10] This paper instead considers a stochastic process that follows a two-state Markov chain, which is straightforward to implement both in discrete and continuous time. Furthermore, continuous time may be advantageous in the presence of occasionally binding constraints, as these are dealt with using boundary conditions rather than inequalities of the optimality conditions. Yet, despite these features, the sparse structure of the implicit method is by many considered the most important practical implication of continuous time, and this paper shows how to implement a similar procedure in discrete time.[11]

2. Model

The model is populated by a unit measure of ex-ante identical households, a price-taking (representative) firm, and a government. Households can either be employed, e, or unemployed, u. The employment status of the households is govern by an exogenous Markov process, specified below. The capital market is incomplete, and households may only save in a one-period safe asset that pays a constant, given, return, r. That is, attention is confined to the steady state in which all aggregate variables – and thereby prices – are constant.[12] Aggregate (net) savings is made available as capital to rent for firms, which operate in a competitive rental market. Thus, the equilibrium rental rate/interest rate and wage rate is such that the labor- and capital markets clear. Time is denoted $t = 0, \Delta, 2\Delta, 3\Delta \ldots$, with $\Delta \in (0, 1]$. Following the usual convention,

[5] Howard's improvement algorithm is sometimes also referred to as *policy function iteration* or the *policy improvement algorithm*.

[6] While not focussing on discrete time, Phelan and Eslami (2022a) also note that there is a close relationship between Howard's improvement algorithm – in the context of the Markov Chain Approximation Method by Kushner and Dupuis (2014) – and the implicit method.

[7] Indeed, a salient feature of the implicit method is that the time step can often be set to infinity, which stands in marked contrast with the explicit method for which the time step must be (often prohibitively) small.

[8] For instance, Rust, 1996 suggests in a discretized setting to use the improvement algorithm only if the number of gridpoints falls short of 500 and the discount factor is close to one. Santos and Rust (2004) show that when using linear interpolation the improvement algorithm is significantly slower than value function iteration when the number of gridpoints reaches 1000. Achdou et al. (2022) reveal the speed of the implicit method.

[9] It is commonly perceived as advantageous in discrete time to operate directly on the Euler equation (see, for instance, section 1.3 in Rendahl, 2015).

[10] Continuous stochastic processes are commonly used to model aggregate shocks to the economy, which is beyond the scope of this paper. Nevertheless, Nuño and Moll (2018), Section 3, considers a framework similar to that of this paper, in which idiosyncratic risk follows an Ornstein-Uhlenbeck process. Their approach may yield some computational advantages that are not investigated here.

[11] It should also be noted that the model considered in this paper indeed includes a non-trivial first-order condition in the presence of an occasionally binding liquidity constraint. However, neither of these features are sufficiently complex to add substantial computational time.

[12] Section 2.3 provides a precise definition

the limit $\Delta \to 0$ is referred to as continuous time, while any $\Delta > 0$ is referred to as discrete time, with $\Delta = 1$ being the default. Finally, the government administrates unemployment insurance scheme, and runs a balanced budget.

For notational convenience, any variable in period t, such as x_t, will be simply denoted x. Similarly, any variable in period $t + \Delta$, such as $x_{t+\Delta}$, will be referred to as x_Δ.

2.1. Households

An employed household's optimization problem is given by the Bellman equation

$$\begin{aligned} v(a,e) &= \max_c \{\Delta u(c) + (1-\Delta\rho)[(1-\Delta\delta)v(a_\Delta,e) + \Delta\delta v(a_\Delta,u)]\}, \\ \text{s.t. } a_\Delta &= a + \Delta(ra + w(1-\tau) - c), \quad a_\Delta \geq 0, \end{aligned} \tag{1}$$

where a denotes (current) assets; c consumption; ρ the rate of time preference; δ the separation rate, r the interest rate; w the wage rate; and τ are taxes.[13] As previously mentioned, the employment status of the household is indicated by the letters e and u, representing employment and unemployment, respectively. The letter s will henceforth denote either e or u when the precise status of employment is unimportant; thus, $s \in \{e, u\}$. The flow utility of consumption is given by the function $u(c)$, and a_Δ denotes assets in the subsequent period. The support of the endogenous state-space is given by the interval $[0, \bar{a})$, and it is assumed that $a_\Delta < \bar{a}$ for all $a \in [0, \bar{a}]$.

Correspondingly, an unemployed household's optimization problem is given by the Bellman equation

$$\begin{aligned} v(a,u) &= \max_c \{\Delta u(c) + (1-\Delta\rho)[\Delta\phi v(a_\Delta,e) + (1-\Delta\phi)v(a_\Delta,u)]\}, \\ \text{s.t. } a_\Delta &= a + \Delta(ra + \mu w(1-\tau) - c), \quad a_\Delta \geq 0, \end{aligned} \tag{2}$$

where the notation follows the taxonomy above, but with ϕ denoting the job-finding rate, and μ the replacement rate.

Rearranging Eqs. (1) and (2) gives

$$0 = \max_c \{u(c) + \frac{v(a_\Delta,e) - v(a,e)}{\Delta} - \rho v(a_\Delta,e) + \delta(1-\Delta\rho)(v(a_\Delta,u) - v(a_\Delta,e))\}, \tag{3}$$

and

$$0 = \max_c \{u(c) + \frac{v(a_\Delta,u) - v(a,u)}{\Delta} - \rho v(a_\Delta,u) + \phi(1-\Delta\rho)(v(a_\Delta,e) - v(a_\Delta,u))\}, \tag{4}$$

subject to the associated constraints from before.

Evaluating the limit $\Delta \to 0$ and rearranging yields the Bellman equations in continuous time; the Hamilton-Jacobi-Bellman (HJB) equations,

$$\rho v(a,e) = \max_c \{u(c) + v_a(a,e)(ra + w(1-\tau) - c) + \delta(v(a,u) - v(a,e))\}, \tag{5}$$

$$\rho v(a,u) = \max_c \{u(c) + v_a(a,u)(ra + \mu w(1-\tau) - c) + \phi(v(a,e) - v(a,u))\}, \tag{6}$$

where $v_a(a,s)$ denotes the derivative of $v(a,s)$ with respect to a.[14]

It ought to be noted that the above derivation is valid for $a \in (0, \bar{a})$, such that $a_\Delta > 0$.[15] The borrowing constraint, $a_\Delta \geq 0$, is then incorporated through the boundary condition $v_a(0,u) = u'(\mu w(1-\tau))$.[16] A similar approach may be applied for $a = \bar{a}$ if needed. Achdou et al. (2022) provides an excellent exposition of the underlying mathematics as well as the intuition.

Lastly, as an unemployed households in any given period finds a job for the subsequent period with probability ϕ, and any employed household keeps their job with probability $(1-\delta)$, the aggregate supply of labor, n^s, follows the law of motion

$$n^s_\Delta = (1-\delta)n^s + \phi(1-n^s),$$

[13] It should be noted that it is common to use the discount factor $e^{-\Delta\rho}$, instead of $(1-\Delta\rho)$. However, as the limiting behaviour of these formulations coincide as Δ approaches zero, and as the corresponding values for $\Delta = 1$ is a matter of calibration, the simpler notation will be used. Moreover, any probability of an event denoted $(1-\Delta x)$ is commonly written instead as $e^{-\Delta x}$, in which x is the Poisson arrival rate of the event. Again, as the limiting behavior coincides in continuous time, and the discrete time counterpart is a matter of calibration, the simpler formulation is preferred.

[14] While these emerge as a direct consequence of analyzing the limit $\Delta \to 0$, they may also be derived by acknowledging that a first order Taylor expansion of the value function is given by

$$\begin{aligned} v(a_\Delta,e) &\approx v(a,e) + v_a(a,e)(a_\Delta - a), \\ &= v(a,e) + v_a(a,e)(ra + w(1-\tau) - c) \end{aligned}$$

where $v_a(a,e)$ denotes the derivative of the value function with respect to a. A similar expression of course obtains for the unemployed.

[15] Provided that the optimal choice, a_Δ, is monotonically increasing in a and continuous in Δ, then for each $a > 0$ there exist a value for $\Delta > 0$ such that $a_\Delta > 0$.

[16] Here it is assumed that the borrowing constraint is binding at state $(a,s) = (0,u)$, and not otherwise – a condition that will be met in the numerical exercise.

Thus, the steady state level of labor is given by

$$n^s = \frac{\phi}{\phi + \delta}.$$

2.1.1. Distribution

The cumulative distribution function (cdf) of the economy, $G(a, s)$, evolves according to

$$G_\Delta(a, e) = (1 - \Delta\delta)G(a - \Delta z(a_{-\Delta,e}, e), e) + \Delta\phi G(a - \Delta z(a_{-\Delta,u}, u), u), \tag{7}$$

$$G_\Delta(a, u) = (1 - \Delta\phi)G(a - \Delta z(a_{-\Delta,u}, u), u) + \Delta\delta G(a - \Delta z(a_{-\Delta,e}, e), e). \tag{8}$$

with $a_{-\Delta,e}$ and $z(a_{-\Delta,e}, e)$ defined as

$$\begin{aligned} a &= a_{-\Delta,e} + \Delta(ra_{-\Delta,e} + w(1-\tau) - c(a_{-\Delta,e}, e)), \\ &= a_{-\Delta,e} + \Delta z(a_{-\Delta,e}, e), \end{aligned} \tag{9}$$

and where $c(a_{-\Delta,e}, s)$ and $z(a_{-\Delta,e}, s)$ denote the optimal consumption and saving choices associated with the problem in Eqs. (1) and (2).[17] An analogous expression obtains for the unemployed. A stationary distribution is such that $G_\Delta(a, s) = G(a, s)$, $\forall a$ and s.

Subtracting $G(a, s)$ from Eqs. (7) and (8), using the approximation $a_{-\Delta,s} \approx a - \Delta z(a, s)$ for a sufficiently small value of Δ, and taking the limit $\Delta \to 0$ gives the Kolmogorov Forward (KF) equations

$$\dot{G}(a, e) = -g(a, e)z(a, e) - \delta G(a, e) + \phi G(a, u), \tag{10}$$

$$\dot{G}(a, u) = -g(a, u)z(a, u) - \phi G(a, u) + \delta G(a, e), \tag{11}$$

where $\dot{G}(a, e)$ denotes the time-derivative of the cdf, and $g(a, s)$ is the probability density function (pdf). That is,

$$\dot{G}(a, s) = \frac{\partial G(a, s)}{\partial t}, \qquad \text{and} \qquad g(a, s) = \frac{\partial G(a, s)}{\partial a}.$$

A stationary distribution in continuous time is such that $\dot{G}(a, s) = 0$. From hereon, I will let $G(a, s)$ denote the *stationary* distribution (whether in discrete or continuous time).

2.2. Firms

Firms hire workers, n, and rent capital, k, in a competitive spot-market, taking prices as given. As the analysis is confined to the steady state – and firms are not exposed to any exogenous disturbances – these variables are constant. The optimization problem is given by

$$\max_{n,k} \{F(k, n) - k(r + \hat{\delta}) - nw\},$$

where $\hat{\delta}$ denotes the depreciation rate of capital.

The production function is of a Cobb-Douglas type – i.e. $F(k, n) = k^\alpha n^{1-\alpha}$, for $\alpha \in (0, 1)$ – such that the first order conditions are given by

$$r = \alpha\left(\frac{k}{n}\right)^{\alpha-1} - \hat{\delta}, \tag{12}$$

$$\begin{aligned} w &= (1-\alpha)\left(\frac{k}{n}\right)^{\alpha}, \\ &= (1-\alpha)\left(\frac{r+\hat{\delta}}{\alpha}\right)^{\frac{\alpha}{\alpha-1}}. \end{aligned} \tag{13}$$

[17] In words, $a_{-\Delta,e}$ can be interpreted as "the level of assets possessed by an employed household that renders a an optimal savings choice". Moreover, it should be mentioned that this definition of the law of motion hinges on the assumption that $a_\Delta(\cdot)$ is (weakly) increasing in a; a condition that, as we will see, is satisfied in the quantitative exercise.

2.3. Equilibrium

An stationary equilibrium is a constant interest rate, r, and a wage rate, w, such that

1. The stationary distribution, $G(a,s)$, satisfies the requirements in the preceding section.
2. The labor market clears; $n = n^s = \phi/(\phi+\delta)$.
3. The asset market clears; $k = \int_{a,s} a\, dG(a,s)$.
4. The equilibrium prices, $\{r, w\}$, satisfy the first order conditions in Eqs. (12) and (13).
5. Taxes, τ, balances the government's budget. That is

$$wn\tau + w\mu(1-n)\tau = w(1-n)\mu,$$

or simply

$$\tau = \frac{\mu(1-n)}{n+\mu(1-n)}.$$

Notice that it is superfluous to make any (direct) reference to the households' optimization problem, as the associated requirements are implicit within the definition of the stationary distribution.

3. Computational algorithms

To outline the computational algorithms it is instructive to use some cross-over notation which bridges the gap between mathematics and implementable code. To this end, I will let a lowercase bold letter, $\mathbf{x}_s$, denote an $N\times 1$ (column) vector, where the subscript s refers to, as previously, the labor market state; $s\in\{e,u\}$. The associated stacked $2N\times 1$ vector, $\mathbf{x}$ is then given by $\mathbf{x} = (\mathbf{x}_e', \mathbf{x}_u')'$. Any function $f(\mathbf{x}_s)$ refers to an $N\times 1$ vector such that $f_i = f(x_i)$, $i = 1,\dots,N$; and $f(\mathbf{x})$ follows a similar taxonomy. There is one exception to this notational convention; the vector $\mathbf{a}$ will refer to the $N\times 1$ vector of *gridpoints* for assets. That is, $\mathbf{a}$ will not carry a subscript as the grid of assets is invariant across employment states. Instead, $\vec{\mathbf{a}}$ will denote the stacked $2N\times 1$ vector $\vec{\mathbf{a}} = (\mathbf{a}', \mathbf{a}')'$, which is used sparingly.

Finally, an uppercase bold letter, $\mathbf{X}_s$, denotes an $N\times N$ matrix, with the associated "stacked" $2N\times 2N$ matrix analogue, $\mathbf{X}$, defined as

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_u \end{pmatrix},$$

where $\mathbf{0}$ is an $N\times N$ matrix of zeros. Lastly, the derivative of a function, $f(\cdot)$, is denoted $\partial f(\cdot)$. Such that, for instance, $v_a(a,s)$, is denoted $\partial v(a,s)$. In similarity, $\partial v(\mathbf{a},s)$ refers to the $N\times 1$ vector $(\partial v(a_1,s), \partial v(a_2,s),\dots)'$.

I will proceed in this section by first describing the benchmark procedure for discrete time; *value function iteration*. Next, I will consider two iterative procedures in continuous time – the *explicit* and the *implicit method* – and describe (and briefly document) the superiority of the latter. Lastly, I will show how to translate the implicit method used in continuous time into a discrete-time setting. Section 5 will then present the results.

3.1. Discrete time

Using the above notation, the problem outlined in Eqs. (1) and (2) is given by

$$\begin{aligned} v(\mathbf{a},s) &= \max_{\mathbf{c}_s}\Big\{u(\mathbf{c}_s) + (1-\rho)\Big[(1-p_s)v\big(\mathbf{a}',s\big) + p_s v\big(\mathbf{a}',\not{s}\big)\Big]\Big\}, \\ \text{s.t.} \quad & \mathbf{a}' = \mathbf{a} + (\mathbf{y}_s - \mathbf{c}_s), \quad \mathbf{a}' \geq 0, \end{aligned} \tag{14}$$

with $p_e = \delta$ and $p_u = (1-\phi)$. Similarly, $\mathbf{y}_e = r\mathbf{a} + w(1-\tau)$ and $\mathbf{y}_u = r\mathbf{a} + \mu w(1-\tau)$. Following convention, the variable $\mathbf{a}'$ corresponds to $\mathbf{a}_\Delta$, evaluated at $\Delta = 1$. The crossed-over letter, $\not{s}$, is interpreted as "*not-s*", such that if $s = e$, then $\not{s} = u$, and vice versa.

Value function iteration follows the procedure

$$\begin{aligned} v_{n+1}(\mathbf{a},s) &= \max_{\mathbf{c}_s}\Big\{u(\mathbf{c}_s) + (1-\rho)\Big[(1-p_s)v_n\big(\mathbf{a}',s\big) + p_s v_n\big(\mathbf{a}',\not{s}\big)\Big]\Big\}, \\ \text{s.t.} \quad & \mathbf{a}' = \mathbf{a} + (\mathbf{y}_s - \mathbf{c}_s), \quad \mathbf{a}' \geq 0, \end{aligned} \tag{15}$$

until $\|v_{n+1} - v_n\| < \varepsilon$, for some suitably chosen ε and v_0, and where $\|\cdot\|$ refers to the chosen *norm*.[18] The associated policy functions that solve the optimization problem in Eq. (15) are denoted $a_n'(\mathbf{a},s)$ and $c_n(\mathbf{a},s)$.

[18] In theoretical expositions the most frequently used norm is given by $\sup|v_{n+1} - v_n|$. However, in any numerical implementation the corresponding norm is $\max|v_{n+1} - v_n|$, which will be used from hereon.

An important aspect of value function iteration is the *contraction property*.[19] That is, for a given value of w and r, the procedure outlined in Eq. (15) is such that $\|v_{n+1} - v\| < \|v_n - v\|$, and $v_n \to v$ (Stokey et al., 1989). Morover, it follows that, theoretically, $a'_n \to a'$ and $c_n \to c$. That is, each additional iteration is an improvement over the previous, and each iteration brings the outcome, v_{n+1}, closer to the fixed point, v. For this reason, this method is also sometimes referred to as *the method of successive approximations*, as each v_n is an, ever improving, approximation of v.

As a preview of the results presented in Section 5, this procedure converges after 840 iteration and takes 1.6 seconds.[20] This may be considered a useful benchmark.

3.2. Continuous time

In continuous time, the HJB equations outlined in Eqs. (5) and (6) are given by

$$\rho v(\mathbf{a}, s) = \max_{\mathbf{c}_s}\{u(\mathbf{c}_s) + \partial v(\mathbf{a}, s)(\mathbf{y}_s - \mathbf{c}_s) + p_s(v(\mathbf{a}, \not{s}) - v(\mathbf{a}, s))\}. \tag{16}$$

To solve this problem I will follow some of the recent literature and confine attention to the *explicit* and the *implicit method* (see, for instance, Candler, 2001, for an early treatment).

The explicit method is given by the iterative scheme

$$\frac{v_{n+1}(\mathbf{a}, s) - v_n(\mathbf{a}, s)}{\Gamma} + \rho v_n(\mathbf{a}, s) = \max_{\mathbf{c}_s}\{u(\mathbf{c}_s) + \partial v_n(\mathbf{a}, s)(\mathbf{y}_s - \mathbf{c}_s) + p_s(v_n(\mathbf{a}, \not{s}) - v_n(\mathbf{a}, s))\}. \tag{17}$$

In order to obtain convergence, Γ must be set to a small value, often close to zero. Indeed, when setting Γ as high as it is permissible for convergence, which for the problem at hand necessitates $\Gamma \leq 0.0003$, the above procedure reaches convergence after 10,423 iterations and takes about 2.4 s. Thus, while each iteration is fast, many are needed as convergence requires a very small value of Γ, which leads to small improvement steps.

The implicit method is instead given by the iterative scheme

$$\frac{v_{n+1}(\mathbf{a}, s) - v_n(\mathbf{a}, s)}{\Gamma} + \rho v_{n+1}(\mathbf{a}, s) = u(\mathbf{c}_{s,n}) + \partial v_{n+1}(\mathbf{a}, s)(\mathbf{y}_s - \mathbf{c}_s) + p_s(v_{n+1}(\mathbf{a}, \not{s}) - v_{n+1}(\mathbf{a}, s)). \tag{18}$$

with

$$\mathbf{c}_{s,n} = \arg\max_{\mathbf{c}_s}\{u(\mathbf{c}_s) + \partial v_n(\mathbf{a}, s)(\mathbf{y}_s - \mathbf{c}_s) + p_s(v_n(\mathbf{a}, \not{s}) - v_n(\mathbf{a}, s))\}, \tag{19}$$

satisfying the first order condition

$$u'(\mathbf{c}_{s,n}) = \partial v_n(\mathbf{a}, s). \tag{20}$$

The implicit method is *significantly* less sensitive to the choice of Γ. Indeed, setting Γ to infinity, convergence is obtained after 0.07 s using 5 iterations. That is, the implicit method is, for this example, about 30 times faster than the explicit method, and about 20 times faster than value function iteration in discrete time.[21] The remaining parts of this section relates both the explicit and implicit method to procedures in discrete time in order to clarify some of these numerical discrepancies.

3.2.1. The explicit method and value function iteration

Value function iteration in Δ-units of time is given by (cf. Eq. (15))

$$\begin{aligned} &v_{n+1}(\mathbf{a}, s) = \max_{\mathbf{c}_s}\{\Delta u(\mathbf{c}_s) + (1 - \Delta\rho)[(1 - \Delta p_s)v_n(\mathbf{a}_\Delta, s) + \Delta p_s v_n(\mathbf{a}_\Delta, \not{s})]\}, \\ &\text{s.t.} \qquad \mathbf{a}_\Delta = \mathbf{a} + \Delta(\mathbf{y}_s - \mathbf{c}_s), \end{aligned} \tag{21}$$

where, for simplicity, the borrowing constraint is (temporarily) ignored. It should be emphasized that this is a contraction mapping for *any* value of $\Delta > 0$. That is, $\|v_{n+1} - v\| < \|v_n - v\|$, and $v_n \to v$. However, the speed at which convergence occurs depends on the value of Δ according to $\|v_{n+1} - v\| \leq (1 - \Delta\rho) \times \|v_n - v\|$. Thus, a smaller value of Δ leads to slower convergence.

Using the same algebraic manipulations as in Eqs. (1)–(4) gives

$$\rho v_n(\mathbf{a}, s) = \max_{\mathbf{c}_s}\{u(\mathbf{c}_s) + \frac{v_n(\mathbf{a}_\Delta, s) - v_{n+1}(\mathbf{a}, s)}{\Delta} + (1 - \Delta\rho)p_s(v_n(\mathbf{a}_\Delta, \not{s}) - v_n(\mathbf{a}_\Delta, s)\}. \tag{22}$$

[19] That is, the procedure in (15) is a *contraction mapping*.

[20] Computational time is not particularly interesting per se. After all, computers improve year after year, and what is considered "fast" in one decade, is often considered "slow" the next. Yet, the computational time of one algorithm *relative* to another, may be more interesting as this relative speed measure is less likely to change with improving technology. Moreover, the actual *number of iterations* is unlikely to change as well. Hence, a strong focus of this paper is on the *relative* speed, and the *number* of iterations.

[21] While these numbers refers to the modal run-time of several repetitions, any number below 0.1 s should not be treated as entirely precise. However, it is safe to conclude that the implicit method is vastly superior to both the explicit method and to value function iteration.

or

$$\rho v_n(\mathbf{a},s) + \tfrac{v_{n+1}(\mathbf{a},s)-v_n(\mathbf{a},s)}{\Delta} = \max_{\mathbf{c}_s}\{u(\mathbf{c}_s) + \tfrac{v_n(\mathbf{a}_\Delta,s)-v_n(\mathbf{a},s)}{\Delta} + (1-\Delta\rho)p_s(v_n(\mathbf{a}_\Delta,\not{s}) - v_n(\mathbf{a}_\Delta,s)\}. \tag{23}$$

By taking the limit $\Delta \to 0$ of the right-hand side and renaming the Δ on the left-hand side as Γ, we recover the formulation for the explicit method in Eq. (17).

This clarifies why Γ must take on a small value for convergence; for any $\Delta > 0$ the value Γ cannot deviate too much from Δ in order for the contraction mapping theorem to hold.[22] Thus, it appears reasonable that as $\Delta \to 0$, Γ must also be very close to zero.[23] However, a small value of Γ also leads to very slow convergence as the discount factor approaches unity.

Another way of seeing how convergence is affected by Γ is, with a slight abuse of notation, to define the term

$$\frac{\partial v_n(\mathbf{a},s)}{\partial t} = \frac{v_{n+1}(\mathbf{a},s) - v_n(\mathbf{a},s)}{\Delta},$$

which can be solved for using Eq. (23). The value function, v_{n+1} is then obtained according to

$$v_{n+1}(\mathbf{a},s) = v_n(\mathbf{a},s) + \Gamma\frac{\partial v_n(\mathbf{a},s)}{\partial t}$$

or simply

$$v_{n+1}(\mathbf{a},s) = \frac{\Gamma}{\Delta}v_{n+1}(\mathbf{a},s) + (1-\frac{\Gamma}{\Delta})v_n(\mathbf{a},s),$$

which only holds with mathematical precision if $\Gamma = \Delta$.[24] That is, if $\Gamma \leq \Delta$ this algorithm leads to dampening, and convergence follows from the contraction mapping theorem (Stokey et al., 1989). However, $\Gamma > \Delta$ instead leads to extrapolation and convergence is no longer guaranteed. Moreover, the smaller Γ is the slower is convergence. In continuous time we have $\Delta \to 0$, which implies that for any (small) $\Gamma > 0$ convergence will be both slow and problematic.[25]

3.2.2. The implicit method and Howard's improvement algorithm

An alternative approach to standard value function iteration is known as *Howard's improvement algorithm* (e.g. Puterman and Brumelle, 1979).[26] This algorithm is particularly useful when the discount factor, $(1-\Delta\rho)$, is close to one. The main idea is to solve the optimization problem in Eq. (21) *once*, and use the associated policy function, $\mathbf{c}_{s,n}$, *repeatedly* to "improve" on the value function many times. And then repeat this process. More specifically, find $\mathbf{c}_{s,n}$ as

$$\begin{aligned} &\mathbf{c}_{s,n} = \arg\max_{\mathbf{c}_s}\{\Delta u(\mathbf{c}_s) + (1-\Delta\rho)[(1-\Delta p_s)v_n(\mathbf{a}_\Delta,s) + \Delta p_s v_n(\mathbf{a}_\Delta,\not{s})]\},\\ \text{s.t.}\quad &\mathbf{a}_\Delta = \mathbf{a} + \Delta(\mathbf{y}_s - \mathbf{c}_s), \end{aligned}$$

and then apply this repeatedly as

$$\begin{aligned} v_{n+1}^{h+1}(\mathbf{a},s) &= \Delta u(\mathbf{c}_{s,n}) + (1-\Delta\rho)\left[(1-\Delta p_s)v_{n+1}^h(\mathbf{a}_{\Delta,n},s) + \Delta p_s v_{n+1}^h(\mathbf{a}_{\Delta,n},\not{s})\right],\\ \mathbf{a}_{\Delta,n} &= \mathbf{a} + \Delta(\mathbf{y}_s - \mathbf{c}_{s,n}), \qquad h = 0,1,\ldots \end{aligned} \tag{24}$$

with $v_{n+1}^0 = v_n$. In some applications this iteration continues until h reaches some finite number, H, such that $v_{n+1} = v_{n+1}^H$. In others the procedure repeats until convergence, such that $v_{n+1} = \lim_{h\to\infty} v_{n+1}^h$. For reasons that will become apparent, focus will be on this latter approach.

Thus, under the assumption that the procedure has been repeated until convergence, Eq. (24) can be written as

$$\begin{aligned} v_{n+1}(\mathbf{a},s) &= \Delta u(\mathbf{c}_{s,n}) + (1-\Delta\rho)[(1-\Delta p_s)v_{n+1}(\mathbf{a}_{\Delta,n},s) + \Delta p_s v_{n+1}(\mathbf{a}_{\Delta,n},\not{s})],\\ \mathbf{a}_{\Delta,n} &= \mathbf{a} + \Delta(\mathbf{y}_s - \mathbf{c}_{s,n}). \end{aligned}$$

Rearranging this expression yields the analogue to Eq. (22) but under the improvement algorithm

$$\rho v_{n+1}(\mathbf{a},s) = u(\mathbf{c}_{s,n}) + \frac{v_{n+1}(\mathbf{a}_{\Delta,n},s) - v_{n+1}(\mathbf{a},s)}{\Delta} + (1-\Delta\rho)p_s(v_{n+1}(\mathbf{a}_{\Delta,n},\not{s}) - v_{n+1}(\mathbf{a}_{\Delta,n},s)). \tag{25}$$

with the limit

$$\rho v_{n+1}(\mathbf{a},s) = u(\mathbf{c}_{s,n}) + \partial v_{n+1}(\mathbf{a},s)(\mathbf{y}_s - \mathbf{c}_{s,n}) + p_s(v_{n+1}(\mathbf{a},\not{s}) - v_{n+1}(\mathbf{a},s)). \tag{26}$$

As in the previous section, the value function may be updated as a linear combination of the solution to Eq. (26) and the previous value, v_n. Specifically, if – again with a slight abuse of notation – $\hat{v}_{n+1}$ represent the value of the left-hand side of Eq. (26), then the updated value function, v_{n+1}, is given by

$$v_{n+1}(\mathbf{a},s) = \eta\hat{v}_{n+1}(\mathbf{a},s) + (1-\eta)v_n(\mathbf{a},s), \quad \eta \in (0,1]. \tag{27}$$

[22] For any $\Gamma > \Delta$ it is no longer possible to show that the mapping in (22) satisfies *discounting* (Blackwell, 1965).

[23] See Barles and Souganidis (1991) for precise conditions.

[24] From a programming perspective, Γ can differ from Δ, as the v_{n+1} on the left-hand side is then seen as an "update" of the v_{n+1} on the right-hand side.

[25] It is slow as a small value of Γ is akin to a discount factor close to unity, and it's problematic as $\Gamma > \Delta$ implies extrapolation.

[26] Sometimes also referred to as *policy function iteration*.

Inserting Eq. (26) into Eq. (27) gives

$$\frac{v_{n+1}(\mathbf{a},s)-v_n(\mathbf{a},s)}{\Gamma}+\rho v_{n+1}(\mathbf{a},s)=u(\mathbf{c}_{s,n})+\partial v_{n+1}(\mathbf{a},s)(\mathbf{y}_s-\mathbf{c}_s)+p_s(v_{n+1}(\mathbf{a},\not{s})-v_{n+1}(\mathbf{a},s)), \tag{28}$$

with $\Gamma=\rho\eta/(1-\eta)$, which is identical to the implicit method in Eq. (18).

While Howard's improvement algorithm is not a contraction mapping, it does have excellent convergence properties under some mild regulatory conditions.[27]

3.3. Numerical implementation

While the methods outlined above can be implemented numerically in several ways, this section describes a coherent procedure that makes the most out of the above insights. To this end, I will describe a finite difference method for continuous time (see, for instance, Achdou et al., 2022), and subsequently suggest a discrete time analogue.

3.3.1. Continuous time

Let $\mathbf{D}$ denote a $2N\times 2N$ *finite differencing matrix* $\mathbf{D}$ such that $\mathbf{D}\times f(\mathbf{x})\approx f'(\mathbf{x})$. There are a few ways to accomplish this, but to fix ideas it is instructive to consider the matrix

$$\mathbf{D}=\begin{pmatrix} -\frac{1}{da} & \frac{1}{da} & 0 & \dots & \dots & \dots & 0\\ -\frac{1}{2\cdot da} & 0 & \frac{1}{2\cdot da} & 0 & \dots & \ddots & \vdots\\ 0 & -\frac{1}{2\cdot da} & 0 & \frac{1}{2\cdot da} & 0 & \dots & \vdots\\ 0 & 0 & -\frac{1}{2\cdot da} & 0 & \frac{1}{2\cdot da} & 0 & \vdots\\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & \dots & 0 & -\frac{1}{da} & \frac{1}{da} \end{pmatrix},$$

where da is referring to the (constant) step-size between grid points. Thus, multiplying $\mathbf{D}$ with a vector of function values $f(\mathbf{x})$ gives a vector of approximated derivatives, $f'(\mathbf{x})$. The first element of this vector of derivatives is approximated using forward differencing, and the last using backward differencing. All other derivatives are approximated using central differencing.[28]

Moreover, define $\mathbf{P}$ as the $2N\times 2N$ matrix

$$\mathbf{P}=\begin{pmatrix} -\mathbf{P}_e & \mathbf{P}_e\\ \mathbf{P}_u & -\mathbf{P}_u \end{pmatrix},$$

where $\mathbf{P}_s$ is an $N\times N$ matrix with p_s on the diagonals and zero elsewhere. Moreover, let $\mathbf{S}_n$ denote a $2N\times 2N$ matrix containing the $2N\times 1$ vector $(\mathbf{y}-\mathbf{c}_n)$ on the diagonals (and zeros elsewhere). That is,

$$\mathbf{S}_n=\begin{pmatrix} \text{diag}(\mathbf{y}_e-\mathbf{c}_{n,e}) & \mathbf{0}\\ \mathbf{0} & \text{diag}(\mathbf{y}_u-\mathbf{c}_{n,u}) \end{pmatrix}.$$

Lastly, let $\mathbf{v}_n=(v_n(\mathbf{a},e)',(v_n(\mathbf{a},u)')'$. Then Eq. (18) can be compactly written as

$$\frac{1}{\Gamma}\mathbf{v}_{n+1}+\rho\mathbf{v}_{n+1}-\frac{1}{\Gamma}\mathbf{v}_n=u(\mathbf{c}_n)+\mathbf{S}_n\mathbf{D}\mathbf{v}_{n+1}+\mathbf{P}\mathbf{v}_{n+1},$$

or[29]

$$\mathbf{v}_{n+1}=(\mathbf{I}(1/\Gamma+\rho)-\mathbf{S}_n\mathbf{D}-\mathbf{P})^{-1}[u(\mathbf{c}_n)+\mathbf{v}_n/\Gamma]. \tag{29}$$

Eq. (29), alongside with the reasoning in Section 3.2.2, provides some insights to why the implicit method is both fast and robust. First, as discussed in Section 3.2.2, the implicit method closely follows the logic of Howard's improvement algorithm, cast in continuous time. Thus there are reasons to believe that it may inherit some of the associated numerical properties. Second, Eq. (29) reveals that the associated computational burden is no higher than solving a (potentially large) linear system of equations at each iteration. While solving such a system could be costly, this is not necessarily the case. In particular, due to its proximity to Howard's improvement algorithm – which generally requires fewer iterations to convergence than value function iteration – the system only needs to be solved a small number of times. Moreover, the $2N\times 2N$

[27] In particular, Howard's improvement algorithm converges quadratically or superlinearly, which is a significant improvement on value function iteration which converges linearly (see e.g. Puterman and Brumelle, 1979 and Santos and Rust, 2004).

[28] Appendix 1.1.1 provides more information regarding the differencing matrix for the continuous-time case. See also Achdou et al. (2022), pp. 71-72 and their associated Supplementary Appendix.

[29] Notice that the original formulation of the term $\mathbf{S}_n\mathbf{D}\mathbf{v}_{n+1}$ is actually $\mathbf{D}[\mathbf{v}_{n+1}\circ(\mathbf{y}-\mathbf{c}_n)]$, where $\circ$ denotes that element-by-element multiplication – or Hadamard multiplication – of the vectors $\mathbf{v}_{n+1}$ and $(\mathbf{y}-\mathbf{c}_n)$. However, by exploiting the diagonal nature of $\mathbf{S}_n$, this rearrangement is possible, and it allows us to reach Eq. (29).

matrix $(\mathbf{I}(1/\Gamma+\rho)-\mathbf{S}_n\mathbf{D}-\mathbf{P})$ is very *sparse*, as it primarily contains non-zero elements on its diagonal axis. Exploiting this sparse structure is known to lead to substantial computational advantages (see Achdou et al., 2022), and is further discussed in Section 5.2.

3.3.2. Discrete time I

The discrete-time counterpart to the implicit method is given by Howard's Improvement Algorithm with a dampening parameter

$$\begin{aligned}&\frac{v_{n+1}(\mathbf{a},s)-v_n(\mathbf{a},s)}{\Gamma}+v_{n+1}(\mathbf{a},s)=u(\mathbf{c}_{s,n})+(1-\rho)\Big[(1-p_s)v_{n+1}\big(\mathbf{a}'_n,s\big)+p_s v_{n+1}\big(\mathbf{a}'_n,\not{s}\big)\Big],\\&\mathbf{a}'_n=\mathbf{a}+(\mathbf{y}_s-\mathbf{c}_{s,n}).\end{aligned}\tag{30}$$

In the continuous-time case, the use of a finite differencing matrix, $\mathbf{D}$, allows us to rewrite the problem as the solution to a linear system of equations. To obtain a similar result in discrete time consider a *transition matrix*, $\mathbf{T}^n_{\mathbf{s}}$, such that if $a'_n(a_i,s)\in[a_j,a_{j+1})$ then $\mathbf{T}^n_{\mathbf{s},ij}=1-\gamma_i^n$ and $\mathbf{T}^n_{\mathbf{s},ij+1}=\gamma_i^n$ with

$$\gamma_i^n=\frac{a'_n(a_i,s)-a_j}{a_{j+1}-a_j}.$$

That is, $\mathbf{T}^n_{\mathbf{s}}$ expresses the vector $\mathbf{a}'_n$ as, $\mathbf{a}'_n=\mathbf{T}^n_{\mathbf{s}}\mathbf{a}$, where each $a'_n(a_i,s)$ is described as a linear combination of its two adjacent points on the grid a_j and a_{j+1}. It should be notice that for *any piecewise linear* function, $f(\mathbf{a})$, we have $f(\mathbf{a}'_n)=\mathbf{T}^n_{\mathbf{s}}f(\mathbf{a})$.

Thus, using the approximation $v_{n+1}(\mathbf{a}'_n,s)\approx\mathbf{T}^n_{\mathbf{s}}v_{n+1}(\mathbf{a},s)$, together with the matrix

$$\mathbf{T}^n=\begin{pmatrix}(1-p_e)\mathbf{T}^n_{\mathbf{e}} & p_e\mathbf{T}^n_{\mathbf{e}}\\ p_u\mathbf{T}^n_{\mathbf{u}} & (1-p_u)\mathbf{T}^n_{\mathbf{u}}\end{pmatrix},\tag{31}$$

the problem in Eq. (30) can be rewritten as

$$\frac{1}{\Gamma}\mathbf{v}_{n+1}+\mathbf{v}_{n+1}-\frac{1}{\Gamma}\mathbf{v}_n=u(\mathbf{c}_n)+(1-\rho)\mathbf{T}^n\mathbf{v}_{n+1},$$

or simply

$$\mathbf{v}_{n+1}=(\mathbf{I}(1/\Gamma+1)-(1-\rho)\mathbf{T}^n)^{-1}[u(\mathbf{c}_n)+\mathbf{v}_n/\Gamma].\tag{32}$$

Eq. (32) carries similar properties as those of Eq. (29). In particular, conditional on $\mathbf{c}_n$ and $\mathbf{v}_n$, the solution amounts to solving a large, sparse, linear system of equations. Thus, apart from finding the values $\mathbf{c}_n$, there are reasons to believe that these two problems share similar computational properties.[30]

3.3.3. Discrete time II

The first-order condition to Eq. (30) is given by

$$u'(\mathbf{c}_{s,n})+(1-\rho)[(1-p_s)\partial v_{n+1}(\mathbf{a}'_n,s)+p_s\partial v_{n+1}(\mathbf{a}'_n,\not{s})]=0,$$

where the derivatives $\partial v_{n+1}(\mathbf{a}'_n,s)$ are taken with respect to $\mathbf{a}'_n$. This equation pins down the optimal choice of consumption/savings as a function of the *derivative* of the value function. Thus, for the purpose of finding the optimal choice, the *level* of the value function is superfluous. As a consequence, a popular alternative approach is to operate directly on the first-order condition in order to obtain the derivative of the value function.[31] This section provides a simple extension to the idea in the previous section in order to obtain a method that focuses only on the first-order condition, and repeatedly updates derivative of the value function.

[30] Although the key matrix in Eq. (32) is sparse, it is still not immune to complications arising due to size; inverting a large sparse matrix can become burdensome. However, it should be noted that – assuming $\Gamma\to\infty$ – Eq. (32) can be rewritten as

$$\mathbf{v}_{n+1}=\left[\lim_{H\to\infty}\sum_{i=0}^{H}[(1-\rho)\mathbf{T}^n]^i\right]\times u(\mathbf{c}_n)+\lim_{H\to\infty}[(1-\rho)\mathbf{T}^n]^H\mathbf{v}_n,$$

which can be approximated as

$$\mathbf{v}_{n+1}\approx\left[\sum_{i=0}^{H}[(1-\rho)\mathbf{T}^n]^i\right]\times u(\mathbf{c}_n)+[(1-\rho)\mathbf{T}^n]^H\mathbf{v}_n,$$

for some finite H. Proceeding with this approximation is commonly known as *modified policy iteration*, and can help to alleviating the burden of solving a large linear system of equations. Phelan and Eslami (2022a) show that this may be an advantageous procedure for large scale problems in the context of the Markov Chain Approximation method (Kushner and Dupuis, 2014). Moreover, Phelan and Eslami (2022b) extend this idea further to incorporate discrete time problems. Thus, there are reasons to believe that this idea may further improve on the algorithm proposed in this paper when the dimensionality is large.

[31] A common approach is to simply iterate on the Euler equation, as the Euler equation both determines optimality, *and* provides the derivative of the value function via the envelope theorem.

Differentiating Eq. (30) with respect to **a** gives

$$\frac{\partial v_{n+1}(\mathbf{a},s)-\partial v_n(\mathbf{a},s)}{\Gamma}+\partial v_{n+1}(\mathbf{a},s)=\partial u(\mathbf{c}_{s,n})+(1-\rho)\Big[(1-p_s)\partial v_{n+1}\big(\mathbf{a}'_n,s\big)+p_s\partial v_{n+1}\big(\mathbf{a}'_n,\not{s}\big)\Big], \tag{33}$$

where each derivative – $\partial u(\mathbf{c}_{s,n})$ and $\partial v_{n+1}(\mathbf{a}'_n,s)$, and so on – are with respect to **a**.

Using the chain rule

$$\partial v_{n+1}(\mathbf{a}'_n,s)=\frac{\partial v_{n+1}(\mathbf{a}'_n,s)}{\partial \mathbf{a}}=\frac{\partial v_{n+1}(\mathbf{a}'_n,s)}{\partial \mathbf{a}'}\frac{\partial \mathbf{a}'}{\partial \mathbf{a}}=\frac{\partial v_{n+1}(\mathbf{a}'_n,s)}{\partial \mathbf{a}'}\times(1+r-\partial \mathbf{c}_{s,n}),$$

together with the approximation

$$\frac{\partial v_{n+1}(\mathbf{a}'_n,s)}{\partial \mathbf{a}'}\approx \mathbf{T}_{\mathbf{s}}^{n}\partial v_{n+1}(\mathbf{a},s),$$

allows Eq. (33) to be rewritten as

$$\frac{1}{\Gamma}\mathbf{dv}_{n+1}+\mathbf{dv}_{n+1}-\frac{1}{\Gamma}\mathbf{dv}_n=\mathbf{D}u(\mathbf{c}_n)+(1-\rho)(1+r-\mathbf{dC}_n)\mathbf{T}^n\mathbf{dv}_{n+1}, \tag{34}$$

where $\mathbf{dC}_n$ is a $2N\times 2N$ matrix containing the $2N\times 1$ vector $\mathbf{D}\times\mathbf{c}_n$ on the diagonals (and zeros elsewhere). Solving this expression yields

$$\mathbf{dv}_{n+1}=[\mathbf{I}(1/\Gamma+1)-(1-\rho)(1+r-\mathbf{dC}_n)\mathbf{T}^n]^{-1}[\mathbf{D}u(\mathbf{c}_n)+\mathbf{dv}_n/\Gamma]. \tag{35}$$

Thus, Eq. (35) provides an alternative approach to solving the problem without ever updating the value function itself.

It should, however, be noted that Eq. (35) involves the vectors $\mathbf{dC}_n$ and $\mathbf{D}u(\mathbf{c})$ which contain the derivative of the policy function. As the policy function is unlikely to be differentiable – in particular in the presence of occasionally binding constraints – this may impede convergence. However, if one obtains a fixed point to Eq. (35), the resulting solution will nevertheless be accurate. To see this, notice that a fixed point implies that $\mathbf{dv}_{n+1}=\mathbf{dv}_n=\mathbf{dv}$, and the optimal policy function satisfies the first-order condition $u'(\mathbf{c})=(1-\rho)\mathbf{Tdv}$. Using these relations, Eq. (34) can be rewritten as

$$\begin{aligned}\mathbf{dv}&=\mathbf{dC}u'(\mathbf{c})+(1-\rho)(1+r-\mathbf{dC})\mathbf{Tdv}\\&=\mathbf{dC}\big[u'(\mathbf{c})-(1-\rho)\mathbf{Tdv}\big]+(1-\rho)(1+r)\mathbf{Tdv}\\&=(1-\rho)(1+r)\mathbf{Tdv}\\&=(1+r)u'(\mathbf{c}).\end{aligned}$$

Thus, a fixed point, **dv**, of Eq. (35) satisfies the envelope theorem for the value function, and any inaccuracies resulting from the presence of the derivatives of the policy function disappears once convergence is reached.[32]

4. Calibration

The calibration of the model is standard, and conducted to be relevant for a large share of incomplete markets models observed in the literature. One period in the model is one quarter. The discount factor, ρ, is therefore set to 0.0074, which roughly corresponds to an annual real interest rate of 3%. The parameter α in the production function is set to 1/3, giving rise to a capital share of income of the same magnitude. Following Shimer (2005) the separation rate, δ, is equal to 0.1, which, together with a job finding rate, ϕ, renders a steady-state unemployment rate of 5 percent. Unemployment benefits are set equal 40 percent of labor income, implying a value for μ of 0.4 (e.g. Kaplan et al., 2020). And taxes are set to balance the government's budget. That is

$$\tau=\frac{\mu(1-n)}{n+\mu(1-n)},$$

where n is the steady state employment rate.

Lastly, the utility function is of the CRRA type

$$u(c)=\frac{c^{1-\gamma}-1}{1-\gamma},$$

with γ equal to 3. A coefficient of risk aversion of 3 is in the upper range of most estimates (Chetty, 2006), but introduces quite strong nonlinearities in the model which increases the computational challenge. The calibrated parameters are summarized in Table 1.

The results are present using three different sizes of the grid for assets: $N=500$; $N=1,000$; and $N=1,500$. The first, $N=500$, is generally considered a quite fine grid and is commonly considered a natural benchmark for incomplete market models (see, for instance, Achdou et al., 2022 or Auclert et al., 2021). The remaining two grids are chosen to illustrate how the algorithms perform at larger computational scales.

[32] It should be kept in mind, however, that the above reasoning hinges on the idea that the fixed point is unique. For a given interest rate and wage this is – theoretically – indeed the case, and follows from the fixed point theorem. Computationally, however, this is not necessarily true. Nevertheless, the theoretical result may provide good guidance to what is reasonable to expect from a computational perspective.

Table 1
Calibrated parameters.

Parameter	Interpretation	Value	Source/steady state target
γ	Coefficient of risk aversion	3	Upper range of convention
$1-\rho$	Discount factor	$1.03^{-\frac{1}{4}}$	Annual real interest rate of 3%
δ	Separation rate	0.1	Shimer (2005)
ϕ	Job finding rate	0.9	Unemployment rate of 5%
α	Capital share of output	0.33	NIPA
μ	Replacement rate	0.4	Kaplan et al. (2020)

Notes. One period in the model corresponds to one quarter.

Table 2
Numerical efficacy.

	Benchmark.	Discrete Time		
		(a)	(b)	(c)
$N = 500$				
$Time_{tot.}$	0.54	32.5	0.51	0.45
$Time_{hh}$	0.02	1.35	0.02	0.02
$Iterations_{in}$	6	797	7	6
$Iterations_{out}$	22	24	25	24
$N = 1000$				
$Time_{tot.}$	1.64	41.7	0.93	0.94
$Time_{hh}$	0.07	1.60	0.04	0.04
$Iterations_{in}$	5	840	6	6
$Iterations_{out}$	24	26	26	26
$N = 1500$				
$Time_{tot.}$	3.07	43.8	1.25	1.05
$Time_{hh}$	0.13	1.99	0.05	0.04
$Iterations_{in}$	6	868	7	6
$Iterations_{out}$	24	22	27	25
Implicit method	Yes	No	Yes	Yes

Notes: Column (a) refers value function iteration in accordance with Eq. (15); (b) the discrete time counterpart to the implicit method using Eq. (32); and, (c), the related procedure that only updates the derivative of the value function in accordance with Eq. (35). $Time_{tot.}$ refers to the total run-time; $Time_{h}$ to the average run-time of the households' problem; $Iterations_{in}$ to the average number of iterations required so solve the households' problem given prices; and $Iterations_{out}$ refers to the number of iterations required to find the equilibrium prices. Time is measured in seconds. The computations are described in the main text.

5. Numerical results

5.1. Main results

The first column of Table 2 reports the results for the continuous-time framework using the implicit method; that is, using Eq. (29). This is considered the benchmark solution to which the other procedures are compared. The computational costs of solving the model are depicted using four different metrics: the total computational time, which includes finding the equilibrium interest rate; the time (on average) devoted to solving the household's problem *for a given interest rate*; the number of iterations required to obtain convergence of the household's problem; and, lastly, the number of iterations required to find the equilibrium prices.[33] Moreover, these four metrics are reported for each of the three different grid sizes.

The remaining three columns report the analogous results, but solving the model using: (a) value function iteration in accordance with Eq. (15); (b) the discrete time counterpart to the implicit method using Eq. (32); and, (c), the related procedure that only updates the derivative of the value function in accordance with Eq. (35). All computational times are reported in seconds.

Three results of this exercise stand out. *First*, value function iteration – column (a) – is by far the slowest procedure. The reason is that the household's problem requires 10–15 *times* as many iterations as any other method. It should be noticed, however, that each iteration is relatively fast; for instance, for $N = 1,000$, value function iteration takes on average 0.05 seconds per iteration, while the benchmark solution takes about 0.3 seconds per iteration.[34] Thus, a key bottleneck is the *number* of iterations, and not the *time per iteration*. *Second*, while value function iteration is relatively slow, it does not suffer as much when the problem is scaled up. The computational time increases by about 35 percent moving from 500 grid

[33] To find the equilibrium, the household's problem is solved multiple times; once for each candidate interest rate. Hence, the reported time for the household's problem is the average across all these candidates.

[34] This occurs as the benchmark necessitates solving a large system of linear equations for each iteration.

Table 3
Numerical efficacy: the role of sparsity.

	Benchmark.	Discrete Time		
		(a)	(b)	(c)
$N = 500$				
$Time_{tot.}$	2.82	32.5	2.66	2.44
$Time_{hh}$	0.13	1.35	0.11	0.10
$N = 1000$				
$Time_{tot.}$	17.6	41.7	13.4	13.9
$Time_{hh}$	0.73	1.60	0.52	0.53
$N = 1500$				
$Time_{tot.}$	46.0	43.8	32.7	28.8
$Time_{hh}$	1.92	1.99	1.21	1.15
Implicit method	Yes	No	Yes	Yes

Notes: All details follow those of Table 2.

point to 1500 grid points. For the baseline the equivalent number is 470%.[35] The reason is that all methods *except* for value function iteration requires a solution to a $2N \times 2N$ linear system of equations for each iteration. That is, the dimensionality of value function iteration grows linearly in N, while the dimensionality of the other procedures grows quadratically. *Lastly*, it should be noted that the discrete-time procedures in columns (b) and (c) are faster than the benchmark.

Thus, applying the implicit method in either discrete or continuous time gives rise to substantial computational gains. These gains can be attributed to the low number of iterations required to obtain a fixed point. Moreover, the discrete-time implementation of the explicit method is more efficient in terms of computational time compared to its continuous time counterpart. The subsequent section explores why these results emerge.

5.2. Sparsity

Table 3 shows the same outcomes as in Table 2, but without exploiting sparsity. As the number of iterations are unaffected by this alteration they are not reported.

For the case of $N = 500$ the computational times increase by roughly a factor of five. This applies both to the benchmark implicit method as well as the discrete time counterparts in columns (b) and (c). In contrast, regular value function iteration, presented in column (a), is unaffected.

A similar pattern unfolds as N increases. The computational burden rises dramatically for the benchmark as well as the discrete time counterparts, but the results for value function iteration is not affected. With $N = 1,000$, the computational cost for the benchmark and the methods in columns (b) and (c) rise to about ten times those that emerged when sparsity was exploited. And with $N = 1,500$, the cost is increased by a factor of around 15. At the same time, the computational burden for value function iteration does not change at all; indeed, as $N = 1,500$, value function iteration even outperforms the benchmark, which is not the case when using sparse computations.[36]

The reason behind these results can be found in Eq. (29) for the benchmark, and in Eqs. (32), and (35) for columns (b) and (c), respectively. Inspecting Eq. (29), for instance, reveals that the first part of the equation, i.e.

$$(\mathbf{I}(1/\Gamma + \rho) - \mathbf{S}_n\mathbf{D} - \mathbf{P}),$$

is an $2N \times 2N$ *sparse* matrix. That is, it contains mainly zeros. Thus, when the computational procedure takes advantage of this property, Eq. (29) can be computed extremely efficiently, and the computational speed is not particularly vulnerable to an increase in the scale. In contrast, when the problem is *dense*, and not sparse, the linear systems in Eqs. (29), (32), and (35), grow quadratically in N, and quickly become prohibitively large. Value function iteration, on the other hand, relies exclusively on $2N \times 1$ *vectors*, which instead grow linearly in N. As a consequence, value function iteration is less affected by the scale of the problem at hand, but also less amendable to sparse computations.

5.2.1. A brief digression

Santos and Rust (2004) investigate the convergence properties of *policy function iteration* – which is another name for Howard's improvement algorithm – when the value function is piecewise linear. They prove under which conditions convergence obtains, as well as the associated convergence rates. As a byproduct of their study, they propose a procedure that is virtually identical to that of Eq. (32). To some extent, this is not very surprising, as Section 3.3.2 makes it clear that piecewise linearity is sufficient to guarantee that the approximation proposed in this paper is exact. But from another perspective this is indeed surprising; the departure point of this paper is the computational advantages of continuous-time methods, and how these may be exported to a discrete-time setting. Yet the resulting algorithm is almost identical.

[35] The two other discrete-time procedures, reported in columns (b) and (c), are less susceptible to this adverse effect, with an increase by 145 and 133 percent, respectively.

[36] It should be noted that the discrete-time versions in columns (b) and (c) remain faster still.

Nevertheless, the procedure studied by Santos and Rust (2004) has not gained much traction in the literature. While it is difficult to exactly determine why, a reasonable answer appears to relate to the fact that their numerical results are very disappointing; already when using a grid with 1000 nodes their algorithm is twice as time-consuming as regular value function iteration. With 3000 grid points this number increases quickly to *forty*, leading Santos and Rust (2004, p. 2114) to conclude that "our results provide a rather pessimistic perspective on the usefulness of policy iteration for solving large-scale problems". A key reason for this conclusion is that Santos and Rust (2004) did not make use of sparse computations.

Lastly, the idea of letting a transition matrix represent the policy function in a discrete-time setting dates back to Young (2010). However, in Young's (2010) framework, the purpose is not to obtain the value- or policy functions, but to propose an efficient method to compute the evolution of the cross-sectional distributions in Eqs. (7) and (8). This paper shows that the same transition matrix that can be used to solve the household's optimization problem, may also be used to compute the cross-sectional distributions (see Appendix 1). This dual use of the transition matrix echos yet again some insights in continuous times; by solving the HJB equation, one obtains the matrix used for solving the KF equation "for free" (see Achdou et al., 2022 p. 4, and p. 29.).

6. Concluding remarks

The ideas developed in this paper departs from the observation that continuous-time models are often more efficient to solve compared to their discrete time counterparts. In an attempt to understand this discrepancy, it is shown that the implicit finite difference method in continuous time is a natural extension of Howard's improvement algorithm in discrete time, but efficiently implemented using sparse matrix operations. Using this insight, it is possible to obtain a similar problem-formulation in discrete time – i.e. one that relies heavily on sparse linear algebra – by making use of a transition matrix representation of the policy function. As a result, the difference in computational run-times between the continuous- and discrete-time problem is eliminated entirely, and the solution may be found up to 30 times faster compared to value function iteration.

Nevertheless, the results in this paper do not imply that continuous time is no more efficient than discrete time. On the contrary, continuous time may be advantageous for several other reasons. For instance, the first order conditions associated with continuous-time problems are static rather than dynamic, which often avoids the need for numerical root-finding algorithms to obtain the optimum. Moreover, occasionally binding constraints – such as collateral constraints – may be easier to implement, as they amount to boundary conditions in the Hamilton-Jacobi-Bellman equation, instead of inequalities in the first order conditions. And more general processes for income shocks – such as an Ornstein-Uhlenbeck process or its discrete time autoregressive representation – can be significantly easier to compute in continuous time (e.g. Nuño and Moll (2018)). Nevertheless, this paper has shown that the advantages brought forth by the implicit finite difference method and its sparse implementation – arguably one of the continuous-time features with most practical relevance – can be well translated, and exploited, also in discrete time.

Finally, there ought to be ways of improving on the proposed method. In particular, a key step is to reformulate Howard's improvement algorithm as a sparse system of linear equations. But in large-scale problems this may generate a bottleneck on its own. Hence, a formulation making use of *modified policy iteration* may prove beneficial (cf. Phelan and Eslami, 2022a and footnote 29). In addition, the proposed method necessitates a maximization step that entails solving several nonlinear equations each iteration. And even though the method explored in this paper requires few iterations, procedure that improves on this aspect will still be useful. This applies in particular when using modified policy iteration, which tends to require more iterations than Howard's improvement algorithm. This issue is left to future research.

Numerical details

This section outlines the numerical details. I will first discuss the aspects that are common to both continuous- and discrete time, and then discuss the remaining issues separately.

The grid for asset holdings, $\mathbf{a}$, is equidistant between 0 and 400. The lower bound of 0 is naturally given by the borrowing constraint. The upper limit is set sufficiently high that it does not influence the equilibrium prices. To solve for the equilibrium interest rate, r, I set $r_h = \rho$ and $r_l = 0$ and set

$$r = \frac{r_h + r_l}{2},$$

and r_h and r_l will subsequently be updated using the bisection method (see below). The equilibrium wage rate is then set to

$$w = (1-\alpha)\left(\frac{r+\hat{\delta}}{\alpha}\right)^{\frac{\alpha}{\alpha-1}}.$$

The initial policy functions, $\mathbf{c}_0 = [\mathbf{c}_e^0, \mathbf{c}_u^0]'$, are given by

$$\mathbf{c}_e^0 = r\mathbf{a} + w(1-\tau), \quad \mathbf{c}_u^0 = r\mathbf{a} + \mu w(1-\tau).$$

This implies that $a_{t+1} = a_t$ in discrete time, and $\dot{a}_t = 0$ in continuous time, for all $a \in \mathbf{a}$. As a consequence, the associated transition matrices in discrete time are simply identity matrices, and the savings matrix in continuous time is zero. Thus, the (associated) initial guesses for the value functions are

$$\mathbf{v}_0 = (\mathbf{I}\rho - \mathbf{P})^{-1}u(\mathbf{c}_0)$$
$$\mathbf{v}_0 = (\mathbf{I} - (1-\rho)\mathbf{P})^{-1}u(\mathbf{c}_0),$$

for continuous and discrete time, respectively. These expressions satisfy Eqs. (29) and (32) using the implications from the initial guesses of the policy functions above, and with $\Gamma = \infty$. For time iteration the initial value for the derivative of the value function is analogously set to

$$\mathbf{dv}_0 = (\mathbf{I} - (1-\rho)\mathbf{P})^{-1}u'(\mathbf{c}_0)r.$$

This gives all algorithms equal starting points.

A1. Continuous time

A1.1. Upwind scheme

The derivative matrix, $\mathbf{D}$, in Eq. (29) is constructed such that all approximated derivatives satisfy an upwind finite difference scheme (e.g. Candler, 2001). More precisely, for any grid point, a_i, such that the corresponding entry of $\mathbf{S}_n$ is *positive* I use forward differencing. That is, if $\mathbf{d}_i$ denotes the ith row of $\mathbf{D}$ then

$$\mathbf{d}_i\mathbf{x} = \frac{x_{i+1} - x_i}{da},$$

with $da = a_{i+1} - a_i$, and where $\mathbf{x}$ is an arbitrary $2N \times 1$ vector. Similarly, for any grid point, a_j, such that the corresponding entry of $\mathbf{S}_n$ is *negative* I use backward differencing. That is,

$$\mathbf{d}_j\mathbf{x} = \frac{x_j - x_{j-1}}{da}.$$

Thus, to the same extent as savings may flip sign at a given grid point and across iterations, the derivative matrix will change too. While this procedure is a little bit tedious, it is also indispensable in order to have a robust solution method (see Appendix 0.2).

A1.2. Equilibrium distribution

Differentiating the Kolmogorov Forward equations in (10) and (11) gives

$$\dot{g}(a,e) = \frac{\partial}{\partial a}[g(a,e)z(a,e)] - \delta g(a,e) + \phi g(a,u),$$
$$\dot{g}(a,u) = \frac{\partial}{\partial a}[g(a,u)z(a,u)] - \phi g(a,u) + \delta g(a,e),$$

where $g(\cdot)$ is the pdf associated with the cdf $G(\cdot)$. This can be approximated as

$$\dot{\mathbf{g}} = \mathbf{B}'\mathbf{g},$$

with $\mathbf{B} = \mathbf{SD} + \mathbf{P}$ from Eq. (29), and where the *prime* indicates the transpose. A key insight to obtain this approximation is that the transpose of $\mathbf{D}$ is – abstracting from some nuances – akin to replacing forward differencing with the negative of backward differencing, and vice verse. The stationary distribution is then found through the eigenvector problem

$$\mathbf{0} = \mathbf{B}'\mathbf{g}.$$

An efficient approach to solving this problem is to consider the alternative

$$\mathbf{b} = \hat{\mathbf{B}}\mathbf{v},$$

where $\mathbf{b}$ is has a value of 1 on its first element and zero elsewhere, and $\hat{\mathbf{B}}$ is equal to $\mathbf{B}'$ everywhere except for the first row which is replaced by a $1 \times 2N$ vector with a value of 1 on its first element and zero elsewhere. Then the vector $\mathbf{v}$ is indeed an eigenvector $\mathbf{g}$, but normalized such that the first element is equal to one. The vector $\mathbf{g}$ is subsequently (re-) normalized to sum to one. The supply of capital is then given by $k^s = \mathbf{g}'\vec{\mathbf{a}}$, where $\vec{\mathbf{a}} = (\mathbf{a}', \mathbf{a}')'$.

Lastly, I calculate the interest rate, r^s required for firms' to demand all of k^s. That is,

$$r^s = \alpha(k^s)^{\alpha-1} - \hat{\delta}.$$

If $r^s > r$, I set $r_l = r$, and $r_h = r$ otherwise.

A2. Discrete time

In discrete time, the derivative matrix **D** in Eq. (35) is such that central differences are used. That is, for each $i \in \{2, 3, \ldots, N-1\}$, if $\mathbf{d}_i$ denotes the ith row of **D** then

$$\mathbf{d}_i\mathbf{x} = \frac{x_{i+1} - x_{i-1}}{2da}.$$

At the endpoints, $i = 1$ and $i = N$, I use forward and backward differencing, respectively.

The first order conditions are solved using a standard Newton's method, which works well for this problem. The initial guess is always $a' = a$ for all $a \in \mathbf{a}$.

The transition matrix, **T**, in Eq. (31) can be interpreted in two ways: as a piecewise linear approximation of evaluating the value function in the optimization problem; or as if individuals indeed stochastically transition between grid points. Using this latter interpretation – i.e. assuming that agents only exist on the grid points, and never in-between – the evolution of the probability distribution function of the economy is given by

$$\mathbf{g}_{t+1} = \mathbf{T}'\mathbf{g}_t.$$

and the stationary distribution is given by the eigenvector problem

$$0 = (\mathbf{T}' - \mathbf{I})\mathbf{g}.$$

From here I proceed exactly as in the continuous time case to obtain the supply of capital and update the interest rate bracket for the bisection method.

Equilibrium distributions and policy functions

Fig. B.1 depicts the equilibrium distribution for discrete (top left), and continuous time (top right), alongside the associated policy functions (bottom left and right, respectively) for $N = 1500$. The black line shows the property for the employed, while the grey for the unemployed. There are some visible differences between discrete and continuous time, although these are not very pronounced; indeed the equilibrium level of capital differs by about 0.05%.

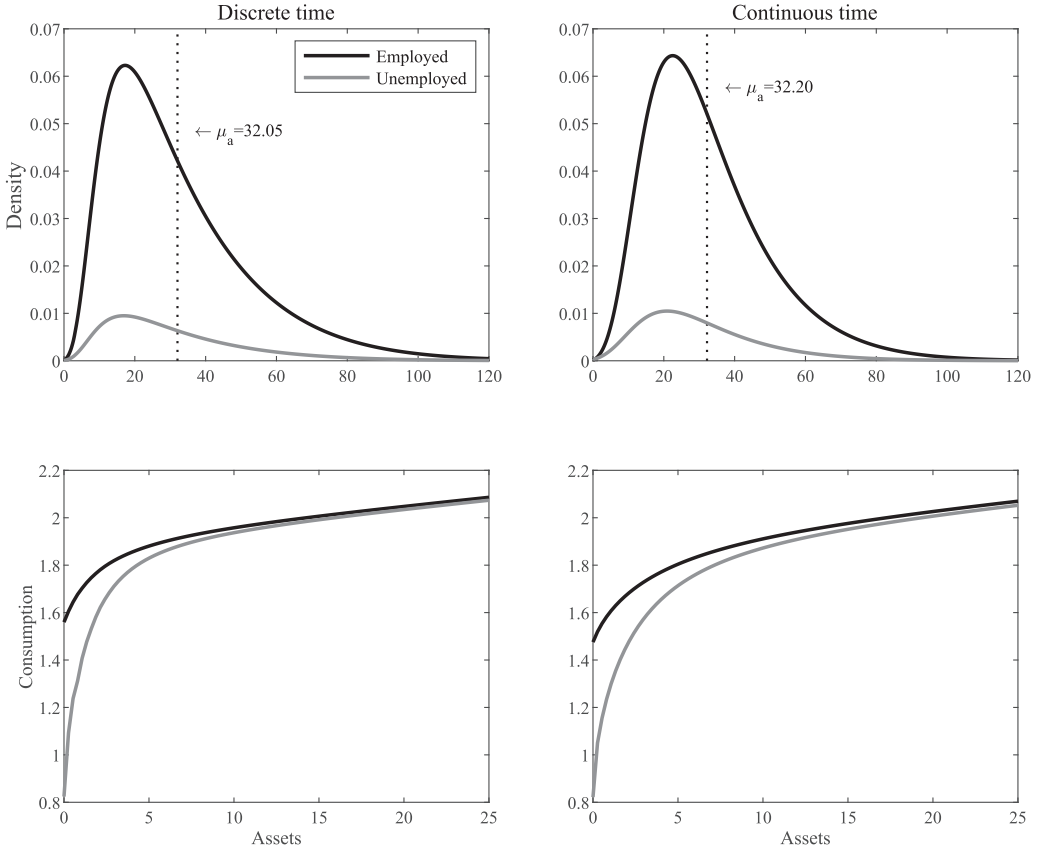

Fig. B.1. Equilibrium distributions (top) and policy functions (bottom) for discrete (left) and continous (right) time. $N = 1500$.

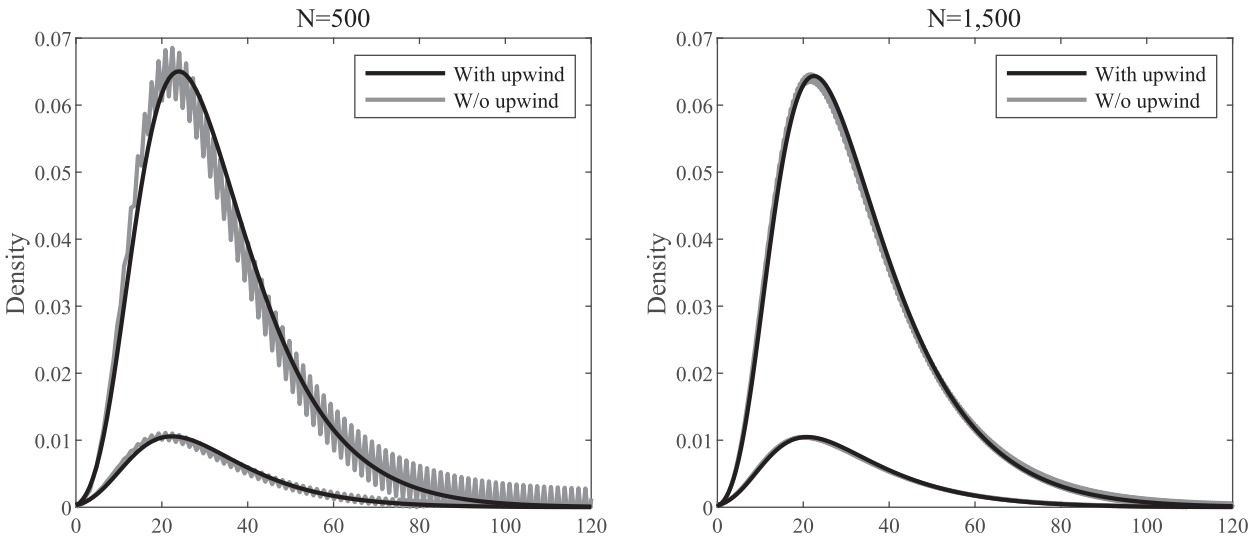

Fig. B.2. Equilibrium distributions with and without the upwind scheme for continous time. $N = 500$ to the left and $N = 1500$ to the right.

The black solid line in Fig. B.2 shows the equilibrium distribution in continuous time for $N = 500$ and $N = 1500$. The grey solid line shows the same outcomes, but when the upwind scheme is replaced by central differencing. As can be seen, this leads to quite pronounced "wiggles" in the distributions for $N = 500$, which get smaller as N increases, although they are still present. Thus, using the upwind scheme is always preferred to central differences.

References

Achdou, Y., Han, J., Lasry, J.-M., Lions, P.-L., Moll, B., 2022. Income and wealth distribution in macroeconomics: a continuous-time approach. Rev. Econ. Stud. 89 (1), 45–86.
Ahn, S., Kaplan, G., Moll, B., Winberry, T., Wolf, C., 2017. When inequality matters for macro and macro matters for inequality. In: Proceedings of the NBER Macroeconomics Annual, Vol. 32, pp. 1–75.
Auclert, A., Bence Bardóczy, M.R., Straub, L., 2021. Using the sequence-space jacobian to solve and estimate heterogeneous-agent models. Econometrica 89 (5), 2375–2408.
Barles, G., Souganidis, P., 1991. Convergence of approximation schemes for fully nonlinear second order equations. Asymptot. Anal. 4 (3), 271–283.
Blackwell, D., 1965. Discounted dynamic programming. Ann. Math. Stat. 36 (1), 226–235.
Candler, G., 2001. Finite–Difference Methods for Continuous–Time Dynamic Programming. In: Marimon, R., Scott, A. (Eds.), Computational Methods for the Study of Dynamic Economies. Oxford University Press.
Chetty, R., 2006. A new method of estimating risk aversion. Am. Econ. Rev. 96 (5), 1821–2834.
Kaplan, G., Violante, G.L., Mitman, K., 2020. The housing boom and bust: model meets evidence. J. Polit. Econ. 128 (9), 3285–3345.
Kushner, H.J., Dupuis, P.G., 2014. Numerical Methods for Stochastic Control Problems in Continuous Time, Vol. 24, 2 Springer New York, NY.
Nuño, G., Moll, B., 2018. Social optima in economies with heterogenous agents. Rev. Econ. Dyn. 28, 150–180.
Phelan, T., Eslami, K., 2022. Applications of markov chain approximation methods to optimal control problems in economics. J. Econ. Dyn. Control. Forthcoming
Fernández-Villaverde, J., Hurtado, S., Nuño, G., 2019. Financial frictions and the wealth distribution. NBER Working Paper 26302.
Phelan, T., Eslami, K., 2022b. The art of temporal approximation: an investigation into numerical solutions to discrete & continuous-time problems in economics. Manuscript. Toronto Metropolitan University.
Puterman, M.L., Brumelle, S.L., 1979. On the convergence of policy iteration in stationary dynamic programming. Math. Oper. Res. 4 (1), 60–69.
Rendahl, P., 2015. Inequality constraints and euler equation-based solution methods. Econ. J. 125 (585), 1110–1135.
Rust, J., 1996. Numerical dynamic programming in economics. In: Amman, H.M., Kendrick, D.A., Rust, J. (Eds.), Handbook of Computational Economics, 1. Elsevier, pp. 3–827.
Santos, M.S., Rust, J., 2004. Convergence properties of policy iteration. SIAM J. Control Optim. 42 (6), 2094–2115.
Shimer, R., 2005. The cyclical behavior of equilibrium unemployment and vacancies. Am. Econ. Rev. 95 (1), 25–49.
Stokey, N.L., Lucas, R.E., Prescott, E.C., 1989. Recursive Methods in Economic Dynamics. Harvard University Press.
Young, E.R., 2010. Solving the incomplete markets model with aggregate uncertainty using the Krusell–Smith algorithm and non-stochastic simulations. J. Econ. Dyn. Control 34 (1), 36–41.