Computational Economics
https://doi.org/10.1007/s10614-020-10045-x

A Guide on Solving Non-convex Consumption-Saving Models

Jeppe Druedahl[1]

Accepted: 4 September 2020

Abstract
Consumption-saving models with adjustment costs or discrete choices are typically hard to solve numerically due to the presence of non-convexities. This paper provides a number of tools to speed up the solution of such models. Firstly, I use that many consumption models have a nesting structure implying that the continuation value can be efficiently pre-computed and the consumption choice solved separately before the remaining choices. Secondly, I use that an endogenous grid method extended with an upper envelope step can be used to solve efficiently for the consumption choice. Thirdly, I use that the required pre-computations can be optimized by a novel loop reordering when interpolating the next-period value function. As an illustrative example, I solve a model with non-durable consumption and durable consumption subject to adjustment costs. Combining the provided tools, the model is solved almost 50 times faster than with standard value function iteration for a given level of accuracy. Software is provided in both Python and C++.

Keywords Endogenous grid method · Post-decision states · Stochastic dynamic programming · Continuous and discrete choices · Occasionally binding constraints

JEL Classification C6 · D91 · E21

1 Introduction

Multi-dimensional consumption-saving models with adjustment costs or discrete choices are typically hard to solve numerically due to the presence of non-convexities. Starting from value function iteration (VFI), which is simple and straightforward, but

✉ Jeppe Druedahl
jeppe.druedahl@econ.ku.dk

[1] CEBI, Department of Economics, University of Copenhagen, Øster Farimagsgade 5, Building 26, 1353 Copenhagen K, Denmark

Published online: 26 October 2020

inherently slow, this paper provides a guide on reducing computational time in such models using three layers of optimization.

In the first layer, I use that many consumption saving models have a nesting structure such that the continuation value can be efficiently pre-computed and the consumption choice solved separately before the remaining choices. I refer to this as the *nested value function iteration* (NVFI).

In the second layer, I use that the nested consumption problem under weak assumptions can be solved efficiently by using an extension of the endogenous grid method (EGM) originally developed by Carroll (2006) for one dimensional models.[1] I refer to this as the *nested endogenous grid method* (NEGM). This step relies on a one-dimensional version of the multi-dimensional upper envelope algorithm developed in Druedahl and Jørgensen (2017).

In the third layer of optimization, I use that the pre-computation of the continuation value needed in both NVFI and NEGM can be computed efficiently by introducing a novel loop reordering reducing the number of computations and improving memory access. I refer to the improved solution methods as NVFI+ and NEGM+.

To study the implications for speed and accuracy of the proposed optimizations, I use a buffer-stock consumption-saving model with non-durable and durable consumption and adjustment costs similar to Berger and Vavra (2015). I show that NEGM+ relative to VFI provides a speed-up factor of almost 50 for a given level of accuracy. The speed-up is reduced to a factor of 15 without the optimized interpolation approach. NVFI provides a speed-up factor of about 10.5, increasing to 13.5 with the optimized interpolation approach.

These results are furthermore obtained when using a non-robust version of VFI, where the underlying non-convex optimization problems are solved with a local solver without any multi-start, which therefore could end up in local maxima. I discuss how this speed-up can be expected to vary with both the model specification and implementation choices. Similar results are obtained for an extended benchmark model with two durable stocks adding both a state and a choice to the model.

Code libraries to implement the proposed solution algorithms are provided in both a C++ version, and a somewhat slower, but more accessible Python version.[2]

Related literature Existing extensions of the endogenous grid method does either not consider non-convexities (Barillas and Fernández-Villaverde 2007; Hintermaier and Koeniger 2010; White 2015; Ludwig and Schön 2018) or restrict attention to the one-dimensional case (Fella 2014; Iskhakov et al. 2017). This paper adds to the previous extensions of the endogenous grid method by simultaneously handling multi-dimensional models and non-convexities.

The only other paper, which provides a EGM algorithm to solve multi-dimensional models with non-convexities is Druedahl and Jørgensen (2017). Their G^{2}EGM algorithm, however, requires that the equation system of stacked discrepancies in the first order conditions and post-decision state equations have a unique solution. This can be cumbersome to determine for a given model, and small model changes can alter

[1] Jørgensen (2013) and Low and Meghir (2017) shows that EGM is orders of magnitude faster than VFI in one dimensional models.

[2] The code is available at github.com/NumEconCopenhagen/ConsumptionSavingNotebooks. The Python code is optimized with just-in-time compilation from the Numba package.

whether G^{2}EGM is applicable or not (see their discussion in Sect. 5.4). By comparison, the NEGM only uses the first order condition for consumption, and it is thus more generally applicable. As an example, the benchmark model used in this paper *cannot* be solved by G^{2}EGM. The modularity of NEGM is an additional benefit relative to G^{2}EGM. The algorithms for computing the upper envelope and speeding up the interpolation step presented in this paper can be re-used across different models with little (or no) modification. The code thus becomes more generic and easier to understand and debug. For models where both G^{2}EGM and NEGM can be applied, the former will most likely be faster, if the fast interpolation scheme proposed in this paper is used, because it completely avoids any VFI step. However, G^{2}EGM typically requires computing additional value function derivatives, and the multi-dimensional upper envelope is more costly than the one-dimensional upper envelope used in NEGM. I show that in the case of the model from Druedahl and Jørgensen (2017) the G^{2}EGM is only about 25% faster than NEGM in their baseline specification, but somewhat more accurate. In specifications with uncertainty where numerical integration is required the result can flip.

The idea of pre-computing the continuation value, which EGM fundamentally relies on, is well-known in the operations research and engineering literature. For *infinite* horizon models it is thus common to iterate on a Bellman equation in the post-decision value function; see in particular Van Roy et al. (1997), Powell (2011) and Bertsekas (2012).[3] The idea of using pre-computations to limit the costs of numerical integration was also proposed by Judd et al. (2017) in an algorithm where the value function is approximated by a parametric function. The idea of reformulating the model using its nesting structure is similar to the general idea of plan factorization discussed in Ma and Stachurski (2018).

Structure The paper is henceforth structured as follows. Section 2 presents the solution methods in light of a specific model. Section 3 presents speed and accuracy results comparing the proposed solution methods with VFI. Section 4 presents the general model class where the proposed solution methods can be used. Section 5 provides a comparison with G^{2}EGM. Section 6 concludes.

2 Solution Method

To fix ideas, consider a buffer-stock consumption-saving model with non-durable and durable consumption similar to Berger and Vavra (2015) and Harmenberg and Oberg (2017). The household's state variables are cash-on-hand m_t, the stock of the durable good n_t, and the persistent component of income p_t. Each period the household chooses consumption c_t and durable consumption d_t. Per period utility is CRRA over a Cobb–Douglas aggregate,

$$u(c_t, d_t) = \frac{(c_t^{\alpha}(d_t + \underline{d})^{1-\alpha})^{1-\rho}}{1-\rho}, \quad \alpha \in (0,1), \rho > 0, \tag{2.1}$$

[3] See Hull (2015) for some extensions and an application in economics.

where $\underline{d} \geq 0$ is a floor under durable consumption. Future utility is discounted with a factor $\beta > 0$. The household's income, y_t, follows an exogenous stochastic process with persistent and fully transitory shocks,

$$p_{t+1} = \psi_{t+1} p_t^{\lambda}, \quad \log \psi_{t+1} \sim \mathcal{N}(-0.5\sigma_{\psi}^2, \sigma_{\psi}^2), \lambda \in (0, 1] \tag{2.2}$$

$$y_{t+1} = \xi_{t+1} p_{t+1}, \quad \log \xi_{t+1} \sim \mathcal{N}(-0.5\sigma_{\xi}^2, \sigma_{\xi}^2). \tag{2.3}$$

If the household adjusts its stock of the durable good (i.e. $d_t \neq n_t$) it incurs a proportional adjustment cost $\tau \in (0, 1)$. It's cash-on-hand after selling the beginning-of-period stock of the durable good, x_t, is therefore

$$x_t = m_t + (1 - \tau) n_t. \tag{2.4}$$

The household's end-of-period assets, a_t, is therefore

$$a_t = \begin{cases} m_t - c_t & \text{if } d_t = n_t \\ x_t - c_t & \text{if } d_t \neq n_t \end{cases}. \tag{2.5}$$

Further, the household cannot borrow, $a_t \geq 0$, and the interest rate on savings is r. Define $R = 1 + r$. The next-period cash-on-hand therefore is

$$m_{t+1} = R a_t + y_{t+1}. \tag{2.6}$$

Finally, the durable stock depreciates with the rate of $\delta \in (0, 1)$, i.e.

$$n_{t+1} = (1 - \delta) d_t. \tag{2.7}$$

2.1 Bellman Equation

The household's value function can be written as a maximum over the value function when not adjusting, v_t^{keep}, and the value function when adjusting $v_t^{adj.}$, i.e.

$$\begin{aligned} v_t(p_t, n_t, m_t) &= \max\{v_t^{keep}(p_t, n_t, m_t), v_t^{adj.}(p_t, x_t)\} \\ &\text{s.t.} \\ x_t &= m_t + (1 - \tau) n_t, \end{aligned} \tag{2.8}$$

where x_t is cash-on-hand after selling the beginning-of-period stock of the durable good. The value function when keeping is

$$\begin{aligned} v_t^{keep}(p_t, n_t, m_t) &= \max_{c_t} u(c_t, n_t) + \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})] \\ &\text{s.t.} \end{aligned}$$

$$\begin{aligned} a_t &= m_t - c_t \\ m_{t+1} &= Ra_t + y_{t+1} \\ n_{t+1} &= (1-\delta)n_t \\ a_t &\geq 0. \end{aligned} \tag{2.9}$$

The value function when adjusting is

$$\begin{aligned} v_t^{adj.}(p_t, x_t) &= \max_{c_t, d_t} u(c_t, d_t) + \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})] \\ &\text{s.t.} \\ a_t &= x_t - c_t - d_t \\ m_{t+1} &= Ra_t + y_{t+1} \\ n_{t+1} &= (1-\delta)d_t \\ a_t &\geq 0. \end{aligned} \tag{2.10}$$

This problem can be solved straightforwardly with a *value function iteration* (VFI) algorithm.[4] The keeper problem in Eq. (2.9) is costly because the state space is multi-dimensional, and the adjuster problem in Eq. (2.10) is costly because the decision space is multi-dimensional. The main computational cost in both problems, however, is the computation of the continuation value for each new guess of the optimal choice(s). The continuation value, $\mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})]$, must be computed using some form of numerical integration, which will always require multiple interpolations of the next-period value function.

2.2 Nesting

In order to speed up the value function iteration algorithm it is beneficial to use some nesting structure present in the model.

Firstly, note that knowing the so-called *post-decision states*, i.e. the persistent component of income p_t, the stock of the durable good d_t and end-of-period assets a_t, is *sufficient* for computing the continuation value. In particular, knowing consumption c_t is irrelevant for computing the continuation value when already conditioning on the post-decision states. This leads me to specify the *post-decision value function*

$$w_t(p_t, d_t, a_t) = \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})], \tag{2.11}$$

and re-write the keeper problem as

$$\begin{aligned} v_t^{keep}(p_t, n_t, m_t) &= \max_{c_t} u(c_t, n_t) + w_t(p_t, d_t, a_t) \\ &\text{s.t.} \\ a_t &= m_t - c_t \geq 0 \\ d_t &= n_t. \end{aligned} \tag{2.12}$$

[4] An alternative would be time iteration, see Rendahl (2015).

Secondly, note that we are always allowed to view the adjuster problem as sequential. We can therefore imagine that the household first chooses how much to buy of the durable good and then afterwards chooses consumption. The consumption choice is then exactly the same as for a keeper starting the period with the chosen stock of the durable good for some level of cash-on-hand. We can thus re-write the adjuster problem as[5]

$$\begin{aligned} v_t^{adj.}(p_t, x_t) &= \max_{d_t} v_t^{keep}(p_t, d_t, m_t) \\ &\text{s.t.} \\ m_t &= x_t - d_t \\ d_t &\in [0, x_t]. \end{aligned} \tag{2.13}$$

Taken together this double nesting structure allows us to specify the following *nested value function iteration* (NVFI) with the following three steps in each time period:

Nested Value Function Iteration (NVFI)

Step 1 Compute the post-decision value function w_t in Eq. (2.11) on a grid over the post-decision states p_t, d_t, and a_t

Step 2 Solve the keeper problem in Eq. (2.12) on a grid over the pre-decision states p_t, n_t, and m_t using interpolation of the post-decision value function w_t computed in step 1

Step 3 Solve the adjuster problem in Eq. (2.13) using interpolation of the keeper value function found in step 2

The nested value function iteration algorithm provides a speed up relative to the standard value function iteration algorithm for two reasons. Firstly, it replaces multiple interpolations of the next-period value function during numerical integration of the continuation value with a single interpolation of the post-decision value function when solving the keeper problem, plus a pre-computation step to construct the post-decision value function itself. Even allowing the post-decision grid to be relatively dense, to limit the loss in precision from the additional interpolation layer, this implies a lot fewer interpolations.[6]

Secondly, it reduces the dimensionality of the decision problem for the adjuster, and replaces evaluating the utility function and multiple evaluations of the next-period value function, with a single interpolation of the value function for the keeper.

[5] An alternative is to instead interpolate the keepers consumption function and then calculate the implied value of choice by evaluating the utility function and the post-decision value function. This could in principle improve precision, but the effect seems minor in practice.

[6] Assume that K evaluations of the value-of-choice is needed when solving the keeper problem for each node in the state space, and Q is the number of integration nodes needed to calculate the expectation in the continuation value. In NVFI we then need $K + \vartheta Q$ interpolations for each node in the state grid, where ϑ is the ratio of nodes in the post-decision grid relative to the state grid. In VFI we need KQ interpolations for each node. We have $KQ > K + \vartheta Q \Leftrightarrow \vartheta < K(1 - Q^{-1})$, which for reasonable Q roughly implies that fewer interpolations is needed whenever $\vartheta < K$.

2.3 EGM with an Upper Envelope

The bottleneck in the NVFI proposed above can be shown (see Sect. 3) to be the solution of the keeper problem due to the multi-dimensional state space. Since it is a pure consumption problem, it can however be solved by using an endogenous grid method (EGM) instead of a numerical solver for each node in the state grid.

The general idea in EGM, originally developed in Carroll (2006) for one-dimensional models without non-convexities, is to fix the end-of-period asset state and then use the Euler-equation and the budget constraint to infer respectively the consumption choice and the level of cash-on-hand. For non-convex multi-dimensional models several complications arise. Firstly, the non-convexity of the decision problem due to the adjustment cost implies that the value function is not globally concave and that the first order conditions are not *sufficient*. By a standard variational argument the Euler-equation is, however, still *necessary*. Secondly, the multi-dimensionality of the model implies that interpolation to a regular grid is required because interpolation of irregular grids are very costly in multiple dimensions.[7] A parsimonious solution to both problems is to use a one dimensional version of the multi-dimensional upper envelope algorithm developed in Druedahl and Jørgensen (2017).[8]

In the present model, the Euler-equation is given by

$$u_c(c_t, d_t) = \alpha c_t^{\alpha(1-\rho)-1}(d_t + \underline{d})^{(1-\alpha)(1-\rho)} = q_t(p_t, d_t, a_t), \tag{2.14}$$

where $q_t(p_t, d_t, a_t)$ is the post-decision marginal value of cash,[9]

$$\begin{aligned} q_t(p_t, d_t, a_t) &\equiv \beta R \mathbb{E}_t[u_c(c_{t+1}, d_{t+1})] \\ &= \beta R \mathbb{E}_t[\alpha c_{t+1}^{\alpha(1-\rho)-1}(d_{t+1} + \underline{d})^{(1-\alpha)(1-\rho)}]. \end{aligned} \tag{2.15}$$

By solving the Euler-equation for c_t, this lets us define the following functions from post-decision states to consumption and then cash-on-hand,

$$c_t = z(p_t, a_t, d_t) = \left(\frac{q_t(p_t, d_t, a_t)}{\alpha(d_t + \underline{d})^{(1-\alpha)(1-\rho)}}\right)^{\frac{1}{\alpha(1-\rho)-1}} \tag{2.16}$$

$$m_t = a_t + c_t. \tag{2.17}$$

Whenever the Euler-equation is necessary all points on the consumption function can be generated from Eqs. (2.16)–(2.17) starting from some values of the post-decision states. When the Euler-equation is not also sufficient, some of the points created will, however, not be a point on the consumption function. Such non-optimal points

[7] See the detailed discussion in Ludwig and Schön (2018) regarding triangulazation and Delaunay interpolation.

[8] The first upper envelope algorithms were developed by Fella (2014) and Iskhakov et al. (2017) for one dimensional models, and were thus not designed to deliver regular grids in multi-dimensional models.

[9] Due to the envelope theorem, we have that $v_{t,m}(p_t, n_t, m_t) = u_c(c_t, d_t)$, and therefore we could instead have used the definition $q_t(p_t, d_t, a_t) \equiv w_{t,a}(p_t, d_t, q_t) = \beta R \mathbb{E}_t[v_{t+1,m}(p_{t+1}, n_{t+1}, m_{t+1})]$.

must therefore be disregarded. Algorithm 1 solves this task and returns the optimal consumption choices on an exogenously chosen grid of cash-on-hand. The inputs are exogenously chosen values of the persistent component of income p and the stock of the durable good d, and exogenous vectors of end-of-period assets a and beginning-of-period cash-on-hand m.

Lines 1–2 initialize the optimal value v at negative infinity. Lines 3–6 apply Eqs. (2.11) and (2.16)–(2.17) for all points in the input end-of-period asset vector. Lines 7–10 use that the borrowing constraint is binding for all cash-on-hand levels lower than the cash-on-hand level implied by assuming no saving (i.e. $a^1 = 0$). Line 11 starts a loop over neighboring points in the endogenously created cash-on-hand grid, while the loop in line 12 is over each point in the exogenously chosen cash-on-hand grid. If a point in the exogenous grid is between two neighboring points in the endogenous grid (line 13) the implied linear interpolated value of consumption (line 14) and value-of-choice (line 15) is calculated. If the value-of-choice is the best yet found (line 16) the optimal consumption and value entries are updated (lines 17–18). For dense enough grids, the endogenously created points with non-optimal value of consumption are thus disregarded because they imply lower values-of-choice.[10]

With the upper envelope algorithm in hand the *nested endogenous grid method* (NEGM) is given by the following three steps:

Nested endogenous grid method (NEGM)

Step 1 Compute the post-decision functions w_t and q_t in Eqs. (2.11) and (2.16) on a grid over the post-decision states p_t, d_t and a_t

Step 2 Solve the keeper problem in Eq. (2.12) on a grid over the pre-decision states p_t, n_t, and m_t, where the combined EGM and upper envelope in Algorithm 1 is applied for each combination of p_t and n_t

Step 3 Solve the adjuster problem in Eq. (2.13) using interpolation of the keeper value function found in step 2

NEGM preserves all the benefits of NVFI and speeds up the solution of the keeper problem by using EGM. Note that the upper envelope part of Algorithm 1 (lines 7–18) is a cheap operation as it involves only simple logical and algebraic operations. Furthermore, Algorithm 1 is completely general in the sense that it does not depend on any specific structure of the model considered here, and can easily be extended to incorporate e.g. additional post-decision states.

2.4 Fast Interpolation

The main bottleneck in NEGM can be shown (see Sect. 3) to be the computation of the post-decision functions $w_t(p_t, d_t, a_t)$ and $q_t(p_t, d_t, a_t)$ in step 1. Again, however, there is some structure in the problem, which can be used for speed up.

To see this, first note that multi-linear interpolation for a single point can be computed as described in Algorithm 2. This algorithm works by first finding the grid

[10] See Druedahl and Jørgensen (2017) for additional details.

Algorithm 1: EGM and Upper Envelope

```
input:
        persistent component of income: $p$
        stock of durable good: $d$
        cash-on-hand grid vector: $\mathcal{G}_m = \{m_j\}_{j=1}^{\#_m}, m_1 = 0$
        end-of-period asset grid vector: $\mathcal{G}_a = \{a^i\}_{i=1}^{\#_a}, a^1 = 0$
for $j \in \{1, 2, \ldots, \#_m\}$ do
    $v_j = -\infty$
for $i \in \{1, 2, \ldots, \#_a\}$ do
    $w^i = w(p, d, a^i)$
    $c^i = z(p, d, a^i)$
    $m^i = a^i + c^i$
for $j \in \{1, 2, \ldots, \#_m\}$ do
    if $m_j \leq m^1$ then
        $c_j = m_j$
        $v_j = u(c_j, d) + w^1$
for $i \in \{1, 2, \ldots, \#_a - 1\}$ do
    for $j \in \{1, 2, \ldots, \#_m\}$ do
        if $m_j \in [m^i, m^{i+1}]$ then
            $c^i_j = c^i + \frac{c^{i+1}-c^i}{m^{i+1}-m^i}(m_j - m^i)$
            $v^i_j = u(c^i_j, d) + \left[w^i + \frac{w^{i+1}-w^i}{a^{i+1}-a^i}((m_j - c^i_j) - a^i)\right]$
            if $v^i_j > v^j$ then
                $v_j = v^i_j$
                $c_j = c^i_j$
return $v_1, v_2, \ldots, v_{\#_m}, c_1, c_2, \ldots, c_{\#_m}$
```

positions in each dimension for the point to be interpolated (lines 1–3); e.g. by binary search. Secondly, the interpolated value is calculated as the weighted sum of the function values at the corners of the hypercube surrounding the point to be interpolated (lines 4–12).

The standard approach to calculate the w_t and q_t functions is to first fix the post-decision states (p_t, n_t, a_t) and then use multi-linear interpolation of the next-period value function $v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})$ for all nodes in a grid of the shocks (ψ_{t+1}, ξ_{t+1}) and then take the appropriately weighted sum of the interpolated values. This is formalized in Algorithm 3.

We know, however, that we for each combination of p_t, n_t, ψ_{t+1} and ξ_{t+1} need to interpolate the next-period value function for a vector of a_t values. This implies that we for given p_{t+1} and n_{t+1} need to interpolate the next-period value function for a vector of m_{t+1} values. When the vector for a_t is chosen as monotonically increasing, the implied vector for m_{t+1} will also be monotonically increasing for given ψ_{t+1} and ξ_{t+1}. This implies that the interpolation can be done with Algorithm 4. This is beneficial for three reasons: Firstly the search in the p and n dimensions (lines 1–2) is done once instead of for each m. Secondly, the search for each a is faster because of the ordering (lines 3–9). Thirdly, the memory access when looking up the value of the next-period value function (line 19) is often made at neighboring indices due to the ordering, which implies fewer cache misses.

Algorithm 5 shows (see lines 12–13) how to use Algorithm 4 when computing the w and q functions using a reordering of the loops such that the loop over a is inside the loops over the shocks ψ and ξ. Algorithm 5 returns exactly the same results as Algorithm 3, and is only slightly more complicated to code. Note also that the vectorized interpolation approach in Algorithm 4 does not depend on any specific structure of the model considered here, and can be straightforwardly extended to higher dimensional problems.[11]

Algorithm 2: Linear Interpolation (interp)

```
input:
    grid vectors: $\mathcal{G}_p = \{p\}_{j_p=1}^{\#_p}, \mathcal{G}_n = \{n\}_{j_n=1}^{\#_n}, \mathcal{G}_m = \{m\}_{j_m=1}^{\#_m}$
    array of known values at tensor product of grid vectors: $v[:,:,:]$
    point to interpolate for: $(p, n, m)$
Find $j_p$ such that $p \in [p_{j_p}, p_{j_p+1})$
Find $j_n$ such that $n \in [n_{j_n}, n_{j_n+1})$
Find $j_m$ such that $m \in [m_{j_m}, m_{j_m+1})$
Initialize $\hat{v} = 0$
Calculate $\Omega = (p_{j_p+1} - p_{j_p})(n_{j_n+1} - n_{j_n})(m_{j_m+1} - m_{j_m})$
for $k_p \in \{0, 1\}$ do
    if $k_p = 0$ then $\omega_p = p_{j_p+1} - p$ else $\omega_p = p - p_{j_p}$
    for $k_n \in \{0, 1\}$ do
        if $k_n = 0$ then $\omega_n = n_{j_n+1} - n$ else $\omega_n = n - n_{j_n}$
        for $k_m \in \{0, 1\}$ do
            if $k_m = 0$ then $\omega_m = m_{j_m+1} - m$ else $\omega_m = m - m_{j_m}$
            $\hat{v} \mathrel{+}= \frac{\omega_m \omega_n \omega_p}{\Omega} V[j_p + k_p, j_n + k_n, j_m + k_m]$
return $\hat{v}$
```

2.5 Implementation Details

This sub-section explains the implementations choices made in the underlying code.

Parametrization The chosen parameters are $\beta = 0.965$, $\rho = 2$, $\alpha = 0.9$ $\underline{d} = 10^{-2}$, $R = 1.03$, $\tau = 0.10$, $\delta = 0.15$, $\sigma_\psi = \sigma_\xi = 0.1$, and $\lambda = 1$.

Grids The grid for p is constructed by the recursion:

$$p^1 = \underline{p}$$

[11] A slight further speed-up is possible by using that the interpolations in lines 12–13 in Algorithm 5 is for the same vector of next-period states. Lines 1–9 of Algorithm 4 can therefore be skipped when calling the interpolation algorithm for the second time in line 13 of Algorithm 5.

Algorithm 3: Post-decision functions: Standard approach

```
input:
    grid vectors: $\mathcal{G}_p = \{p\}_{j_p=1}^{\#_p}, \mathcal{G}_n = \{n\}_{j_n=1}^{\#_n}, \mathcal{G}_m = \{m\}_{j_m=1}^{\#_m}, \mathcal{G}_a = \{a\}_{j_a=1}^{\#_a}$
    shock vectors: $\mathcal{G}_\psi = \{\psi\}_{j_\psi=1}^{\#_\psi}, \mathcal{G}_\xi = \{\xi\}_{j_\xi=1}^{\#_\xi}$
    shock weight vectors: $\mathcal{G}_{\psi^w} = \{\psi^w\}_{j_\psi=1}^{\#_\psi}, \mathcal{G}_\xi = \{\xi^w\}_{j_\xi=1}^{\#_\xi}$
    next-period value function: $v_+[:,:,:]$
    next-period marginal utility of consumption: $u_{c,+}[:,:,:]$

Initialize $w[:,:,:] = 0$
Initialize $q[:,:,:] = 0$
for $j_p \in \{1, 2, \dots, \#_p\}$ do
    for $j_n \in \{1, 2, \dots, \#_n\}$ do
        for $j_a \in \{1, 2, \dots, \#_a\}$ do
            for $j_\psi \in \{1, 2, \dots, \#_\psi\}$ do
                for $j_\xi \in \{1, 2, \dots, \#_\xi\}$ do
                    $p_+ = p_{j_p} \psi_{j_\psi}$
                    $n_+ = (1-\delta) n_{j_n}$
                    $y_+ = p_+ \xi_{j_\xi}$
                    $m_+ = R a_{j_a} + y_+$
                    $\hat{v}_+ = interp(\mathcal{G}_p, \mathcal{G}_n, \mathcal{G}_m, v_+, (p_+, n_+, m_+))$
                    $\hat{u}_{c,+} = interp(\mathcal{G}_p, \mathcal{G}_n, \mathcal{G}_m, u_{c,+}, (p_+, n_+, m_+))$
                    $w[j_p, j_n, j_a] \mathrel{+}= \beta \psi^w_{j_\psi} \xi^w_{j_\xi} \hat{v}_+$
                    $q[j_p, j_n, j_a] \mathrel{+}= \beta R \psi^w_{j_\psi} \xi^w_{j_\xi} \hat{u}_{c,+}$
return $w, q$
```

$$p^i = p^{i-1} + \frac{\overline{p} - p^{i-1}}{(\#_p - (i-1))^{1.1}}, \; i = 2, 3, \dots, \#_p,$$

with $\underline{p} = 10^{-4}$, $\overline{p} = 3$ and $\#_p = 150$. The other grids are constructed similarly with $\underline{n} = 0$, $\overline{n} = 3$, $\#_n = 150$, $\underline{m} = 0$, $\overline{m} = 10$, $\#_m = 300$, $\underline{x} = 0$, $\overline{x} = 13$, $\#_x = 300$, $\underline{a} = 0$, $\overline{a} = 11$, and $\#_a = 300$.[12] Extrapolation outside of the grids for p and n are not allowed.

Interpolation Define the following negative inverse transformations of the value functions and marginal utilities of consumption:

$$\tilde{v}_t^{keep}(p_t, n_t, m_t) \equiv -\frac{1}{v_t^{keep}(p_t, n_t, m_t)}$$

$$\tilde{v}_t^{adj.}(p_t, x_t) \equiv -\frac{1}{v_t^{adj.}(p_t, x_t)}$$

[12] The value for $\overline{x} = \overline{m} + \overline{n}$ ensures that the grid is wide enough for comparing the adjust and keep cases. The value for $\overline{a} < \overline{x}$ is chosen somewhat lower because the household will always choose to consume some it's resources.

Algorithm 4: Linear interpolation for monotone vector (interpvec)

```
input:
    grid vectors: $\mathcal{G}_p = \{p_{j_p}\}_{j_p=1}^{\#_p}, \mathcal{G}_n = \{n_{j_n}\}_{j_n=1}^{\#_n}, \mathcal{G}_m = \{m_{j_m}\}_{j_m=1}^{\#_m}$
    array of known values at tensor product of grid vectors: $v[:, :, :]$
    first two dimensions in points to interpolate for: $p, n$
    last dimension in points to interpolate for:
    $m_1, m_2, \dots, m_\#$ where $m_1 < m_2 < \cdots < m_\#$
Find $j_p$ such that $p \in [p_{j_p}, p_{j_p+1})$
Find $j_n$ such that $n \in [n_{j_n}, n_{j_n+1})$
for $i \in \{1, 2, \dots, \#\}$ do
    if $i = 1$ then
        Find $j_m^1$ such that $m_1 \in [m_{j_m^1}, m_{j_m^1+1})$
    else
        $j_m^i = j_m^{i-1}$
        while $m_i \geq m_{j_m^i+1}$ do
            $j_m^i \mathrel{+}= 1$
Initialize $\hat{v}_i = 0$ for $i \in \{1, 2, \dots, \#\}$
for $k_p \in \{0, 1\}$ do
    if $k_p = 0$ then $\omega_p = p_{j_p+1} - p$ else $\omega_p = p - p_{j_p}$
    for $k_n \in \{0, 1\}$ do
        if $k_n = 0$ then $\omega_n = n_{j_n+1} - n$ else $\omega_n = n - n_{j_n}$
        for $i \in \{1, 2, \dots, \#\}$ do
            Calculate $\Omega = (p_{j_p+1} - p_{j_p})(n_{j_n+1} - n_{j_n})(m_{j_m^i+1} - m_{j_m^i})$
            for $k_m \in \{0, 1\}$ do
                if $k_m = 0$ then $\omega_m = m^i_{j_m+1} - m$ else $\omega_m = m - m_{j_m^i}$
                $\hat{v}_i \mathrel{+}= \frac{\omega_m \omega_n \omega_p}{\Omega} V[j_p + k_p, j_n + k_n, j_m^i + k_m]$
return $\hat{v}_1, \hat{v}_2 \dots, \hat{v}_\#$
```

$$\begin{aligned}
\tilde{u}_{c,t}^{keep}(p_t, n_t, m_t) &\equiv \frac{1}{u_c(c_t^{keep}(p_t, n_t, m_t), n_t)} \\
\tilde{u}_{c,t}^{adj.}(p_t, x_t) &\equiv \frac{1}{u_c(c_t^{adj.}(p_t, x_t), d_t^{adj.}(p_t, x_t))}.
\end{aligned}$$

These transformed functions are always positive and increasing, and satisfy

$$\begin{aligned}
\lim_{m_t \to 0} \tilde{v}_t^{keep}(p_t, n_t, m_t) &= 0 \\
\lim_{x_t \to 0} \tilde{v}_t^{adj.}(p_t, x_t) &= 0 \\
\lim_{m_t \to 0} \tilde{u}_{c,t}^{keep}(p_t, n_t, m_t) &= 0 \\
\lim_{x_t \to 0} \tilde{u}_{c,t}^{adj.}(p_t, x_t) &= 0,
\end{aligned}$$

Algorithm 5: Post-decision functions: Reordered loops

```
input:
    grid vectors: $\mathcal{G}_p = \{p\}_{j_p=1}^{\#_p}, \mathcal{G}_n = \{n\}_{j_n=1}^{\#_n}, \mathcal{G}_m = \{m\}_{j_m=1}^{\#_m}, \mathcal{G}_a = \{a\}_{j_a=1}^{\#_a}$
    shock vectors: $\mathcal{G}_\psi = \{\psi\}_{j_\psi=1}^{\#_\psi}, \mathcal{G}_\xi = \{\xi\}_{j_\xi=1}^{\#_\xi}$
    shock weight vectors: $\mathcal{G}_{\psi^w} = \{\psi^w\}_{j_\psi=1}^{\#_\psi}, \mathcal{G}_\xi = \{\xi^w\}_{j_\xi=1}^{\#_\xi}$
    next-period value function: $v_+[:,:,:]$
    next-period marginal utility of consumption: $u_{c,+}[:,:,:]$

Initialize $w[:,:,:] = 0$
Initialize $q[:,:,:] = 0$
for $j_p \in \{1, 2, \dots, \#_p\}$ do
    for $j_n \in \{1, 2, \dots, \#_n\}$ do
        for $j_\psi \in \{1, 2, \dots, \#_\psi\}$ do
            for $j_\xi \in \{1, 2, \dots, \#_\xi\}$ do
                $p_+ = p_{j_p} \psi_{j_\psi}$
                $n_+ = (1 - \delta) n_{j_n}$
                $y_+ = p_+ \xi_{j_\xi}$
                for $j_a \in \{1, 2, \dots, \#_a\}$ do
                    $m_+^{j_a} = R a_{j_a} + y_+$
                $\hat{v}_+ = interpvec(\mathcal{G}_p, \mathcal{G}_n, \mathcal{G}_m, v_+, (p_+, n_+, m_+^1, m_+^2, \dots, m_+^{\#_a})$
                $\hat{u}_{c,+} = interpvec(\mathcal{G}_p, \mathcal{G}_n, \mathcal{G}_m, u_{c,+}, (p_+, n_+, m_+^1, m_+^2, \dots, m_+^{\#_a})$
                for $j_a \in \{1, 2, \dots, \#_a\}$ do
                    $w[j_p, j_n, j_a] \mathrel{+}= \psi_{j_\psi}^w \xi_{j_\xi}^w \hat{v}_+^{j_a}$
                    $q[j_p, j_n, j_a] \mathrel{+}= \psi_{j_\psi}^w \xi_{j_\xi}^w \hat{u}_{c,+}^{j_a}$
return $w, q$
```

which is beneficial when interpolating because no infinities are involved.

To improve on precision the post-decision value function are calculated as

$$
\begin{aligned}
w_t(p_t, d_t, a_t) &= \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})] \\
&= \beta \mathbb{E}_t \left[\begin{cases} -\frac{1}{\tilde{v}_{t+1}^{keep}} & \text{if } \tilde{v}_{t+1}^{keep} \geq \tilde{v}_{t+1}^{adj} \\ -\frac{1}{\tilde{v}_{t+1}^{adj}} & \text{if } \tilde{v}_{t+1}^{keep} < \tilde{v}_{t+1}^{adj} \end{cases} \right],
\end{aligned}
$$

where $\tilde{v}_{t+1}^{keep}$ and $\tilde{v}_{t+1}^{adj.}$ are interpolated separately. Similarly, the post-decision marginal value of cash function is calculated as

$$
\begin{aligned}
q_t(p_t, d_t, a_t) &= \beta R \mathbb{E}_t[u_c(c_t(p_t, n_t, m_t), d_t(p_t, n_t, m_t))] \\
&= \beta R \mathbb{E}_t \left[\begin{cases} -\frac{1}{\tilde{u}_{c,t+1}^{keep}} & \text{if } \tilde{v}_{t+1}^{keep} \geq \tilde{v}_{t+1}^{adj} \\ -\frac{1}{\tilde{u}_{c,t+1}^{adj}} & \text{if } \tilde{v}_{t+1}^{keep} < \tilde{v}_{t+1}^{adj} \end{cases} \right],
\end{aligned}
$$

where $\tilde{u}_{c,t}^{keep}$ and $\tilde{u}_{c,t}^{adj.}$ are interpolated separately.

The required changes to lines 12–15 of Algorithm 3 and lines 12–16 of Algorithm 5 are straightforward.

All numerical integration is done with Gauss–Hermite quadrature using 5 nodes for both the persistent and the transitory shocks.

Simulation The initial values in the simulation are drawn as:

$$\begin{aligned}\log p_0 &\sim \mathcal{N}(\log(1), 0.2)\\ \log d_0 &\sim \mathcal{N}(\log(0.8), 0.2)\\ \log a_0 &\sim \mathcal{N}(\log(0.2), 0.1).\end{aligned}$$

3 Speed and Accuracy

To assess the speed and accuracy of the proposed solution methods, I compare them with a standard value function iteration. I solve the model backwards for $T = 50$ periods (iterations) using the implementation settings described in Sect. 2.5.

Following Judd (1992) and Santos (2000), I measure accuracy as the $\log_{10}$ of the relative absolute Euler error across households optimally making an interior consumption choice. Specifically, I use a simulation sample of $N = 100,000$ and calculate

$$\overline{\mathcal{E}} \equiv \frac{\sum_{t=1}^{T}\sum_{i=1}^{N}\mathcal{E}_{it}\mathbf{1}_{a_{it}>\epsilon}}{\sum_{t=1}^{T}\sum_{i=1}^{N}\mathbf{1}_{a_{it}>\epsilon}}, \tag{3.1}$$

where

$$\begin{aligned}\mathcal{E}_{it} &\equiv \log_{10}(|\Delta_{it}/c_{it}|)\\ \Delta_{it} &\equiv c_t - \left(\beta R\mathbb{E}_t[c_{t+1}^{-\rho}]\right)^{-\frac{1}{\rho}},\end{aligned}$$

and $\epsilon = 0.02$. A value of $\overline{\mathcal{E}}$ of -2 and -4 are interpreted as average approximation errors of respectively 1 and 0.01% of consumption.

The results are shown in Table 1 for the C++ implementation of the proposed solution methods. Starting with VFI in the first column, we see that the model is solved in about an hour, and that most of the time is spend on the keeper problem. Continuing to NVFI in column two, we see that the model is now solved about 10 times faster with only a small reduction in accuracy due to the additional layers of interpolation. The keeper problem is solved about 15 times faster than in VFI because of the use of the post-decision value function, which it takes less than two minutes to compute. The adjuster problem is also solved much faster due to both the use of the post-decision value function and the reduction in the dimensionality of the decision space. Turning to NEGM in column three, we see a further reduction in the time it takes to solve the keeper problem, but some of the initial gain is lost by more time spend in computing the post-decision functions, where the post-decision marginal

Table 1 Speed and accuracy

	VFI	NVFI	NEGM	NVFI+	NEGM+
Relative Euler errors					
All (average)	– 4.670	– 4.613	– 4.709	– 4.613	– 4.709
5th percentile	– 5.730	– 5.620	– 5.581	– 5.620	– 5.581
95th percentile	– 3.694	– 3.704	– 3.775	– 3.704	– 3.775
Adjusters (average)	– 4.558	– 4.737	– 4.888	– 4.737	– 4.888
Keepers (average)	– 4.691	– 4.590	– 4.676	– 4.590	– 4.676
Timings (in min, best of 5)					
Total	63.05	5.96	4.07	4.68	1.32
Post– decision functions	0.00	1.76	3.51	0.47	0.77
Keeper problem	61.79	4.19	0.54	4.19	0.54
Adjuster problem	1.26	0.01	0.01	0.01	0.01
Speed– up relative to VFI		10.58	15.51	13.47	47.60
Simulation outcomes					
Expected discounted utility	– 32.213	– 32.213	– 32.213	– 32.213	– 32.213
Adjuster share ($d_t \neq n_t$)	0.172	0.172	0.172	0.172	0.172
Average consumption (c_t)	0.979	0.979	0.979	0.979	0.979
Variance of consumption (c_t)	0.256	0.256	0.256	0.256	0.256
Average durable stock (d_t)	0.562	0.562	0.562	0.562	0.562
Variance of durable stock (d_t)	0.112	0.112	0.112	0.112	0.112

The model is solved backwards for $T = 50$ periods (iterations) using the implementation settings described in Sect. 2.5. The simulation outcomes are computed from a sample of 100,000 households, and the Euler errors are calculated as in Eq. (3.1). The optimization problems are solved by a simple *golden section search* in both NVFI and NEGM. The optimization problems in VFI is solved by the *method of moving asymptotes* from Svanberg (2002), implemented in `NLopt` by Johnson (2014), which was found to be the fastest solver. The code was run using 8 threads on a Windows 10 computer with two *Intel(R) Xeon(R) Gold 6154 3.00 GHz CPUs* (18 cores, 36 logical processes each) and 192 GB of RAM. The code was compiled with the free *Microsoft Visual Studio 2017 C++ compiler* with full optimization (*Ox* flag) and parallelization with *OpenMP*. The timing reported is best out of 3 runs

value of cash must now also be calculated. Compared to NVFI and VFI, NEGM also provides an improvement in the level of accuracy. Turning to NVFI+ and NEGM+ we see that the post-decision functions are computed more than three times faster, which especially benefits NEGM, where this step is the bottleneck. In the end, NEGM+ is almost 50 times faster than VFI for the same accuracy. Across all solution methods selected simulation outcomes (bottom part of Table 1), including expected discounted utility, is indistinguishable from each other.

The speed-up factors presented above is naturally dependent on the chosen model and its implementation. We have for instance completely side-stepped the issue of using multi-start, when solving the keeper and adjuster problems. The non-concavity of the value function implies that some form of multi-start is required to limit the risk that the numerical solver ends up in a local maximum. Introducing multi-starts will in particular increase solution time for VFI, where all the time is spend on solving the keeper and adjuster problems. Solution time will also increase substantially for

NVFI, where most of the time is spend on the keeper problem. For NEGM, however, the solution time will almost not increase because the keeper problem is solved with EGM and the time spend on the adjuster problem is negligible. Adding multi-starts would therefore increase the speed-up substantially.

Another important margin is the number of quadrature nodes required for numerical integration. The computational costs of VFI is almost proportional to the number of quadrature nodes. In NVFI and NEGM, the cost of computing the post-decision functions is also proportional to the number of quadrature nodes, but the cost of solving the keeper and adjuster problems are independent of the number of quadrature nodes. With more (less) quadrature nodes the speed-ups relative to VFI will therefore increase (decrease). Likewise the chosen density of the post-decision grid is important. For a more dense grid the relative speed-up of NVFI and NEGM decrease, but their accuracy increase.

The complexity of the adjuster problem can also be important. In a model with a more complex adjuster problem in terms of more states and/or choices, there still will be a benefit of going from VFI to NVFI(+). If solving the adjuster problem becomes the bottleneck in NVFI(+) the improvement of going to NEGM(+) will, however, be low in relative terms.

Table 2 presents speed results for the Python implementations of NVFI+ and NEGM+ optimized with just-in-time computation using the Numba package. Solution time only increases with a factor of about 1.5 in the Python implementations of NVFI+ and NEGM+.[13]

3.1 Extended Model

In this sub-section, I consider an extended version of the benchmark model, where there are two different durable stocks (d_1^1, d_t^2) with different depreciation rates (δ_1, δ_2), different adjustment cost factors (τ_1, τ_2), and potentially differential impact on utility,

$$u(c_t, d_t^1, d_t^2) = \frac{(c_t^{\alpha}((d_t^1 + \underline{d}_1)^{\gamma}(d_t^2 + \underline{d}_2)^{1-\gamma})^{1-\alpha})^{1-\rho}}{1-\rho}, \quad \alpha, \gamma \in (0,1), \rho > 0.$$

This adds a state variable to the keeper problem, and imply that there are three different adjuster problems depending on whether one or both of the durable stocks are adjusted. The implied Bellman equations and further implementation details are provided in "Appendix".

Table 3 shows speed and accuracy results for VFI, NVFI+ and NEGM+. The speed-up factor provided by NVFI+ increases to around 20, while the speed-up factor for NEGM+ mildly decreases to 44. The accuracy NEGM+ of is now slightly worse than VFI.

[13] Comparing various approaches to parallelization on CPUs Fernandez-Villaverde and Valencia (2018) found C++ with either OpenMP or MPI to be the fastest.

Table 2 Speed and Accuracy in Python

	NVFI+		NEGM+	
	C++	Python	C++	Python
Relative euler errors				
All (average)	– 4.613	– 4.613	– 4.709	– 4.709
5th percentile	– 5.620	– 5.620	– 5.581	– 5.581
95th percentile	– 3.704	– 3.704	– 3.775	– 3.775
Adjusters (average)	– 4.737	– 4.737	– 4.888	– 4.888
Keepers (average)	– 4.590	– 4.590	– 4.676	– 4.676
Timings (in min, best of 5)				
Total	4.68	7.07	1.32	2.29
Post-decision functions	0.47	0.92	0.77	1.60
Keeper problem	4.19	6.12	0.54	0.67
Adjuster problem	0.01	0.02	0.01	0.02
Simulation outcomes				
Expected discounted utility	– 32.213	– 32.213	– 32.213	– 32.213
Adjuster share ($d_t \neq n_t$)	0.172	0.172	0.172	0.172
Average consumption (c_t)	0.979	0.979	0.979	0.979
Variance of consumption (c_t)	0.256	0.256	0.256	0.256
Average durable stock (d_t)	0.562	0.562	0.562	0.562
Variance of durable stock (d_t)	0.112	0.112	0.112	0.112

See Table 1. The Python code is optimized with just-in-time compilation from the Numba package

4 When Can NEGM be Used?

In this section, NEGM is presented in more general terms. The assumptions made below are sufficient for using NEGM, though not always necessary.

4.1 Model Class

Let m_t denote beginning-of-period *cash-on-hand* (i.e. liquid resources), c_t denote *consumption*, and a_t denote end-of-period *liquid assets*. Further let $n_t = (n_t^1, \ldots, n_t^{\#_N})$ be a vector of the $\#_N$ other states than m_t, and $d_t = (d_t^1, \ldots, d_t^{\#_D}) \in \mathcal{D}(M_t, N_t)$ a vector of the $\#_D$ other choices than c_t. Also let $b_t = (b_t^1, \ldots, b_t^{\#_B})$ be a vector of the remaining *post-decision states* besides a_t. Assume that b_t for some function $\mathcal{B}$ is given by

$$B_t = \mathcal{B}(n_t, d_t). \tag{4.1}$$

Table 3 Speed and accuracy with two durable stocks

	VFI	NVFI+	NEGM+
Relative euler errors			
All (average)	– 3.848	– 3.746	– 3.582
5th percentile	– 4.906	– 4.864	– 4.735
95th percentile	– 2.853	– 2.047	– 2.365
Adjusters (average)	– 3.732	– 3.832	– 3.822
Keepers (average)	– 3.895	– 3.711	– 3.479
Timings (in min)			
Total	265.41	13.71	6.00
Post-decision functions	0.00	3.08	4.91
Keeper problem	234.73	10.41	0.86
Adjuster problem	30.68	0.22	0.22
Speed-up relative to VFI		19.36	44.25
Simulation outcomes			
Expected discounted utility	– 34.288	– 34.289	– 34.295
Adjuster share ($d_t^1 \neq n_t^1 \vee d_t^2 \neq n_t^2$)	0.303	0.304	0.310
Average consumption (c_t)	0.981	0.981	0.978
Variance of consumption (c_t)	0.256	0.256	0.254
Average durable stock I (d_t^1)	0.380	0.379	0.377
Variance of durable stock I (d_t^1)	0.051	0.050	0.050
Average durable stock II (d_t^2)	0.217	0.216	0.214
Variance of durable stock II (d_t^2)	0.018	0.018	0.018

The model is solved backwards for $T = 50$ periods (iterations) using the implementation settings described in "Appendix". The simulation outcomes are computed from a sample of 100,000 households, and the Euler errors are calculated as in Eq. (3.1). The optimization problems are solved by a simple *golden section search* in both NVFI and NEGM. The optimization problems in VFI is solved by the *method of moving asymptotes* from Svanberg (2002), implemented in `NLopt` by Johnson (2014), which was found to be the fastest solver. The code was run using 8 threads on a Windows 10 computer with two *Intel(R) Xeon(R) Gold 6154 3.00 GHz CPUs* (18 cores, 36 logical processes each) and 192 GB of RAM. The code was compiled with the free *Microsoft Visual Studio 2017 C++ compiler* with full optimization (*Ox* flag) and parallelization with *OpenMP*

For a vector of stochastic shocks, ψ_{t+1}, let the transition functions for m_t and n_t be denoted $\mathcal{T}_m$ and $\mathcal{T}_n$. Assume, for simplicity, that $\mathcal{T}_m$ is continuous, differentiable and strictly monotone wrt. a_t.[14] Also assume that $\mathcal{T}_n$ is independent of a_t.[15]

The fundamental restrictive assumption so far, is that consumption, c_t, only affects the future through its effect on first end-of-period assets, a_t, and then future cash-on-

[14] If $\mathcal{T}_m$ is not differentiable everywhere in a_t there will be multiple Euler-equations. If there are k kinks where $\mathcal{T}_m$ is not differentiable in a_t, there will be $k + 1$ intervals where $\mathcal{T}_m$ is differentiable in a_t. Consequently there will be $k + 1$ mutually exclusive Euler-equations. Interior optimal consumption choices not at kinks must still satisfy one of these, and the proposed NEGM can therefore still be applied.

[15] If $\mathcal{T}_n$ is continuous and differentiable wrt. to a_t it will still be possible to derive an equation like the standard Euler-equation. The fast vectorized interpolation scheme can, however, not be straightforwardly used as changes in a_t then affect both m_{t+1} and n_{t+1}.

hand, m_{t+1}. This is a mild assumption satisfied in most models, but e.g. not models with habit formation in consumption.

Further assume that the n_t states and d_t choices only affect end-of-period assets *additively*, i.e. that we for some function $\mathcal{L}$ can write

$$l_t = m_t + \mathcal{L}(n_t, d_t) \tag{4.2}$$

$$a_t = l_t - c_t, \tag{4.3}$$

where l_t are liquid assets just before consumption. This is also a rather weak assumption, and fulfilled in most consumption-saving models in the literature. Likewise assume that the minimum level of end-of-period liquid assets is only a function of the other post-decision states, i.e. $a_t \geq \underline{a}(b_t)$. This implies that the maximum level of consumption is given by $\overline{c}(l_t, b_t) \equiv l_t - \underline{a}(b_t)$.

Finally, assume that the effect of n_t and d_t on utility is *additively separable* such that the period utility function for functions u and h has the form

$$u(c_t, b_t) + h(n_t, d_t), \tag{4.4}$$

where it is required that u_c and u_{cc} exists, $u_c > 0$ and $u_{cc} < 0$. The assumption of additive separability for n_t and d_t can be loosed somewhat, but at a potential cost as discussed below. This might e.g. be interesting in models with labor supply.

In sum, the Bellman equation for the considered model class can be written as

$$\begin{aligned}
V_t(m_t, n_t) &= \max_{c_t, d_t} u(c_t, b_t) + h(n_t, d_t) + \beta \mathbb{E}_t \left[v_{t+1}(m_{t+1}, n_{t+1}) \right] \\
&\text{s.t.} \\
l_t &= m_t + \mathcal{L}(n_t, d_t) \\
a_t &= l_t - c_t \\
b_t &= \mathcal{B}(n_t, d_t) \\
m_{t+1} &= \mathcal{T}_M(a_t, b_t, \psi_{t+1}) \\
n_{t+1} &= \mathcal{T}_N(b_t, \psi_{t+1}) \\
d_t &\in \mathcal{D}(m_t, n_t) \\
c_t &\in [0, \overline{c}(l_t, b_t)],
\end{aligned} \tag{4.5}$$

where β is the discount factor. Denote the optimal choice functions by $c_t^{\star}(m_t, n_t)$ and $d_t^{\star}(m_t, n_t)$, and the implied optimal post-decision functions by $a_t^{\star}(m_t, n_t)$ and $b_t^{\star}(m_t, n_t)$.

Using a variational argument, it can be shown that optimal interior consumption choices, $c_t \in (0, \overline{c}(\bullet))$, must satisfy an Euler-equation

$$u_c(c_t, b_t) = q_t(a_t, b_t), \tag{4.6}$$

where

$$q_t(a_t, b_t) = \beta \mathbb{E}_t[\mathcal{T}_{m,a,t+1} u_{c,t+1}].$$

4.2 NEGM

In order to reformulate the problem in Eq. (4.5), it is beneficial to introduce the *post-decision value function* given by

$$w_t(a_t, b_t) \equiv \beta \mathbb{E}_t \left[v_{t+1}(\mathcal{T}_M(a_t, b_t, \psi_{t+1}), \mathcal{T}_N(b_t, \psi_{t+1})) \right]. \tag{4.7}$$

Next, define the following pure consumption problem

$$\begin{aligned} \mathcal{V}_t(l_t, b_t) &= \max_{c_t} u(c_t, b_t) + w_t(a_t, b_t) \\ &\text{s.t.} \\ a_t &= l_t - c_t \\ c_t &\in [0, \overline{c}(l_t, b_t)], \end{aligned} \tag{4.8}$$

If the separability in Eq. (4.4) was not assumed, then d_t should for example also be a state in the pure consumption problem. In labor supply models with human capital accumulation, for example, the pure consumption problem might depend not just of end-of-period human capital, but also on current labor supply if consumption and leisure are non-separable.

Solving the pure consumption problem is just like solving the keeper problem in the benchmark model. The $z(a_t, b_t)$ function is found as the solution for c_t of Eq. (4.6). And then l_t is found by $l_t = a_t + c_t$ as implied by Eq. (4.3). The upper envelope is applied just like in the benchmark model.

Based on the pure consumption problem, the model in Eq. (4.5) can be reformulated as,

$$\begin{aligned} v_t(m_t, n_t) &= \max_{d_t} \mathcal{V}_t(l_t, b_t) + h(n_t, d_t) \\ &\text{s.t.} \\ b_t &= \mathcal{B}(n_t, d_t) \\ l_t &= m_t + \mathcal{L}(n_t, d_t) \\ d_t &\in \mathcal{D}(m_t, n_t). \end{aligned} \tag{4.9}$$

This reformulation is greatly beneficial for two reasons. Firstly, it reduces the dimensionality of the optimization problem (only max over d_t, not also c_t). Secondly, it replaces multiple interpolations of the next period value function required to calculate the continuation value, $\mathbb{E}_t[V_{t+1}(\bullet)]$, with interpolation of just $\mathcal{V}_t(l_t, b_t)$. As in the benchmark model, the suggestion is to solve this model with standard value function iteration.

5 Comparison with G^{2}EGM

The benchmark model used in this paper cannot be solved by the G^{2}EGM proposed in Druedahl and Jørgensen (2017). The basic idea in G^{2}EGM is to fix the post-decision

decision states (in the benchmark model this is p_t, d_t and a_t) and then invert a system of first order conditions and post-decision state equations to find choices and pre-decision states. In the benchmark model, focusing on the adjuster problem, the relevant equations can be written as

$$\begin{aligned} u_c(c_t, d_t) &= w_{t,a}(p_t, d_t, a_t) \\ u_d(c_t, d_t) &= w_{t,a}(p_t, d_t, a_t) - w_{t,d}(p_t, d_t, a_t) \\ x_t &= a_t + c_t + d_t. \end{aligned}$$

The problem is that this equation system is not invertible in the sense required by G^{2}EGM. Consider fixing the post-decision states p_t, d_t and a_t. The first equation can then be used to find the consumption choice, c_t, and the third equation can be used to find the cash-on-hand state, x_t. But when p_t, d_t, a_t and c_t are all determined, both the LHS and the RHS of the second equation is fixed. The problem is that nothing insures it is satisfied. Therefore G^{2}EGM cannot be applied for the benchmark model.

Instead, I present a version of the model originally used Druedahl and Jørgensen (2017) and solve it with both G^{2}EGM and NEGM.

5.1 Model

Households are either retired or working. Working households can retire. Retired households can not begin to work again. In retirement, households solve a standard consumption-savings problem. The resources available for consumption in period t is denoted m_t, such that the post-decision assets a_t is given by

$$a_t = m_t - c_t.$$

Next period resources are given by

$$m_{t+1} = R_a a_t + \underline{y},$$

where R_a is the return factor, and $\underline{y}$ is a (deterministic) retirement income. We assume that households are not allowed to borrow, $a_t \geq 0$.

Working households solve a more general problem, and are allowed to save in both liquid assets (a_t) and illiquid pension assets (b_t). Denoting the pension fund deposits by d_t, we assume that the post-decision (or end-of-period) assets levels are given by

$$\begin{aligned} a_t &= m_t - c_t - d_t \\ b_t &= n_t + d_t + g(d_t), \end{aligned}$$

where $g(d_t)$ is a pension deposit function potentially allowing for an extra incentive to accumulate illiquid pension funds due to, for example, tax deductions of pension contributions. Deposits are required to be non-negative, i.e. $d_t \geq 0$, and the assumption of no borrowing, $a_t \geq 0$, is also maintained for the working households.

The resources available for consumption and pension savings in the next period are

$$\begin{aligned} m_{t+1} &= R_a a_t + \eta_{t+1},\ \log \eta_{t+1} \sim \mathcal{N}(-0.5\sigma_\eta^2, \sigma_\eta^2) \\ n_{t+1} &= R_b b_t \end{aligned}$$

where η_t is stochastic labor income, and we assume a higher return on pension assets than on liquid assets, i.e. $R_b \geq R_a$.

Denoting the discrete choice of retirement by $z_t = 0$ and the discrete choice of working by $z_t = 1$, the Bellman equation of the model can be formulated as

$$\begin{aligned} V_t(z_{t-1}, m_t, n_t) &= \max_{z_t \in \mathcal{Z}_t(z_{t-1})} \begin{cases} v_t(0, m_t + n_t) & \text{if } z_t = 0 \\ v_t(1, m_t, n_t) & \text{if } z_t = 1 \end{cases} \\ &\text{s.t.} \\ \mathcal{Z}_t(z_{t-1}) &= \begin{cases} \{0, 1\} & \text{if } z_{t-1} = 1 \\ 0 & \text{if } z_{t-1} = 0 \end{cases} \end{aligned} \tag{5.1}$$

The discrete-choice-specific value function for the working households is

$$\begin{aligned} v_t(1, m_t, n_t) &= \max_{c_t, d_t} u(c_t) - \alpha + \beta \mathbb{E}_t \left[V_{t+1}(1, m_{t+1}, n_{t+1}) \right] \\ &\text{s.t.} \\ a_t &= m_t - c_t - d_t \\ b_t &= n_t + d_t + g(d_t) \\ m_{t+1} &= R_a a_t + \eta_{t+1} \\ n_{t+1} &= R_b b_t \\ c_t &\geq 0 \\ d_t &\geq 0 \\ c_t + d_t &\in [0, m_t], \end{aligned} \tag{5.2}$$

where $u(c_t)$ denotes per-period utility flow from consuming, c_t, and α is the disutility of labor. The discrete-choice-specific value function for the retiring (or retired) households is

$$\begin{aligned} v_t(0, x_t) &= \max_{c_t} u(c_t) + \beta V_{t+1}(0, m_{t+1}, 0) \\ &\text{s.t.} \\ a_t &= x_t - c_t \\ m_{t+1} &= R_a a_t + \underline{y} \\ c_t &\in [0, x_t]. \end{aligned} \tag{5.3}$$

We assume the following functional forms

$$u(c_t, z_t) = \frac{c_t^{1-\rho}}{1-\rho} \tag{5.4}$$

$$g(d_t) = \chi \log(1 + d_t). \tag{5.5}$$

5.2 Solution

The problem of the retired households can be solved with standard EGM, and it is independent of the problem of the working households because retirement is an absorbing state. Further details are provided in Druedahl and Jørgensen (2017).

In G^{2}EGM the problem of the working households is solved by firstly computing the post-decision value function and it's derivatives, and then solving the full problem with EGM using first order conditions for both the consumption choice and the pension deposit choice. Details are again provided in Druedahl and Jørgensen (2017).

For NEGM the post-decision value is again computed first, but now only the derivative wrt. to end-of-period assets is needed. The problem of the working households is, however, solved in two separate steps. First the pure consumption problem is solved. Then the optimal deposit choice is determined.

Defining the post decision value function

$$w_t(a_t, b_t) = \beta \mathbb{E}_t[V_{t+1}(1, m_{t+1}, n_{t+1})],$$

the pure consumption problem is

$$\begin{aligned}
\mathcal{V}_t(1, l_t, b_t) &= \max_{c_t} u(c_t) - \alpha + w_t(a_t, b_t) \\
&\text{s.t.} \\
a_t &= l_t - c_t \\
m_{t+1} &= R_a a_t + \eta_{t+1} \\
n_{t+1} &= R_b b_t \\
c_t &\in [0, l_t].
\end{aligned} \tag{5.6}$$

This can be solved with EGM as explained in the previous section. The optimal deposit choice is then determined in an outer problem by

$$\begin{aligned}
v_t(1, m_t, n_t) &= \max_{d_t} \mathcal{V}_t(1, l_t, b_t) \\
&\text{s.t.} \\
l_t &= m_t - d_t \\
b_t &= n_t + d_t + g(d_t) \\
d_t &\in [0, m_t].
\end{aligned} \tag{5.7}$$

Table 4 Comparing G^{2}EGM and NEGM

	$\sigma_\eta^2 = 0.0$		$\sigma_\eta^2 = 0.1$	
	G^{2}EGM	NEGM	G^{2}EGM	NEGM
Relative Euler errors				
All (average)	– 6.233	– 5.367	– 5.493	– 5.208
5th percentile	– 7.361	– 7.205	– 6.768	– 6.588
95th percentile	– 4.266	– 3.347	– 3.964	– 3.708
Timings (in min, best of 5)				
Total	0.81	1.08	1.22	1.36
Post– decision functions	0.03	0.03	0.44	0.32
EGM– step	0.78	0.33	0.78	0.31
VFI– step		0.72		0.73

In the specification with income risk, $\sigma_\eta^2 = 0.1$, 16 quadrature nodes is used. Otherwise the exact same parameters and grids as in Druedahl and Jørgensen (2017) were used. The optimization problem in NEGM is solved by a simple *golden section search*. The code was run using 1 thread on a Windows 10 computer with two *Intel(R) Xeon(R) Gold 6154 3.00 GHz CPUs* (18 cores, 36 logical processes each) and 192 GB of RAM. The Python code is optimized with just-in-time compilation from the Numba package. A serial implementation is used because the two-dimensional upper envelope of G^{2}EGM is not straightforward to parallellize in Python when there is no exogenous states. Only the time spend solving the working household problem is included

5.3 Speed and Accuracy

Table 4 shows speed and accuracy results for G^{2}EGM and NEGM using Python implementations.[16] I consider one specification without income risk, and one specification with income risk and 16 quadrature nodes. Otherwise, I use the exact same parameters and grid sizes as in Druedahl and Jørgensen (2017) with the number of nodes in the grid for m_t at $\#_m = 600$, and calculate Euler errors as they do.

For the pure consumption problem in NEGM, I use the post-decision grid for b_t and for l_t I use the same grid as for m_t. The outer VFI step in NEGM is solved with a golden section search algorithm extended with explicit checks for whether a constrained choice is superior. For both G^{2}EGM and NEGM, the fast interpolation approach developed in this paper is used.

In the specification *without* income risk G^{2}EGM provides a speed-up of about 25% relative to NEGM, and the accuracy of G^{2}EGM is superior. In the specification *with* income risk the speed-up is only about 10%, and the accuracy of G^{2}EGM is only slightly better. The result changes because the post-decision functions becomes more costly to compute in the presence of income shocks due to numerical integration. NEGM is furthermore less demanding in this regarding because fewer derivatives are used.

If there were additional shocks, or more quadrature nodes were used, the results would likely turn in favor of NEGM. On the other hand, it could be argued that the VFI

[16] A serial implementation is used because the two dimensional upper envelope of G^{2}EGM is not straightforward to parallellize in Python when there is no exogenous states.

step in NEGM should use a multi-start algorithm to limit the risk of convergence to local maxima. Models with a more costly outer problem, where VFI is used in NEGM, would also benefit G^{2}EGM. Adjusting the relative size of grids also affect the results.

In sum, however, these results show that even in models where both G^{2}EGM and NEGM can be used, NEGM might be an OK choice purely in terms of speed. An examination of the code base furthermore reveal that NEGM is much easier to implement than G^{2}EGM, and as explained previously is applicable to a wider set of models.

6 Conclusions

We have seen that the solution methods proposed in this paper, the Nested Value Function Iteration (NVFI) and the Nested Endogenous Grid Method (NEGM), provide substantial speed-up compared to standard value function iteration (VFI). Furthermore, we have seen that the methods are applicable to a large class of models, and straightforward to implement as they rely on generic algorithms for interpolation and for finding the upper envelope across candidate solutions to the Euler-equation.

Faster solution methods makes it possible to solve and estimate richer consumption-saving models than previously, and thus makes it computationally feasible to perform policy analysis based on more realistic models.

Acknowledgements I thank Anders Munk-Nielsen, Bertel Schjerning, Alessandro Martinello, Giulio Fella, Matthew White and especially Thomas Høgholm Jørgensen for fruitful discussions and suggestions. The usual disclaimer applies. An earlier version of this paper has been circulated under the title "A fast nested endogenous grid method for solving general consumption-saving models". Financial support from the Danish Council for Independent Research in Social Sciences (Grant No. 5052-00086) is gratefully acknowledged.

Appendix: Model in Sect. 3.1

Bellman Equation

Defining the vector of state variables by $s_t = (p_t, n_t^1, n_t^2, m_t)$. The recursive Bellman equation for the extended benchmark model with two durable stocks can be written

$$\begin{aligned}
v_t(s_t) &= \max \left\{ \begin{array}{c} v_t^{keep}(p_t, n_t^1, n_t^2, m_t) \\ v_t^{adj.}(p_t, x_t) \\ v_t^{adj.,1}(p_t, n_t^2, x_t^1) \\ v_t^{adj.,2}(p_t, n_t^2, x_t^2) \end{array} \right\} \\
&\text{s.t.} \\
x_t &= m_t + (1-\tau_1)n_t^1 + (1-\tau_2)n_t^2 \\
x_t^1 &= m_t + (1-\tau_1)n_t^1, \\
x_t^2 &= m_t + (1-\tau_2)n_t^2,
\end{aligned} \tag{6.1}$$

where

$$
\begin{aligned}
v_t^{keep}(p_t, n_t^1, n_t^2, m_t) &= \max_{c_t} u(c_t, n_t^1, n_t^2) + \beta \mathbb{E}_t[v_{t+1}(s_{t+1})] \\
&\text{s.t.} \\
a_t &= m_t - c_t \\
m_{t+1} &= (1+r)a_t + y_{t+1} \\
n_{t+1}^1 &= (1-\delta_1)n_t^1 \\
n_{t+1}^2 &= (1-\delta_2)n_t^2 \\
a_t &\geq 0,
\end{aligned} \tag{6.2}
$$

and

$$
\begin{aligned}
v_t^{adj.}(p_t, x_t) &= \max_{c_t, d_t^1, d_t^2} u(c_t, d_t^1, n_t^2) + \beta \mathbb{E}_t[v_{t+1}(s_{t+1})] \\
&\text{s.t.} \\
a_t &= x_t - c_t - d_t^1 - d_t^2 \\
m_{t+1} &= (1+r)a_t + y_{t+1} \\
n_{t+1}^1 &= (1-\delta_1)d_t^1 \\
n_{t+1}^2 &= (1-\delta_2)d_t^2 \\
a_t &\geq 0,
\end{aligned} \tag{6.3}
$$

and

$$
\begin{aligned}
v_t^{adj.,1}(p_t, n_1^2, x_t^1) &= \max_{c_t, d_t^1} u(c_t, d_t^1, n_t^2) + \beta \mathbb{E}_t[v_{t+1}(s_{t+1})] \\
&\text{s.t.} \\
a_t &= x_t^1 - c_t - d_t^1 \\
m_{t+1} &= (1+r)a_t + y_{t+1} \\
n_{t+1}^1 &= (1-\delta_1)d_t^1 \\
n_{t+1}^2 &= (1-\delta_2)n_t^2 \\
a_t &\geq 0,
\end{aligned} \tag{6.4}
$$

and

$$
\begin{aligned}
v_t^{adj.,2}(p_t, n_1^1, x_t^2) &= \max_{c_t, d_t^2} u(c_t, n_t^1, d_t^2) + \beta \mathbb{E}_t[v_{t+1}(s_{t+1})] \\
&\text{s.t.} \\
a_t &= x_t^2 - c_t - d_t^2 \\
m_{t+1} &= (1+r)a_t + y_{t+1} \\
n_{t+1}^1 &= (1-\delta_1)n_t^1 \\
n_{t+1}^2 &= (1-\delta_2)d_t^2 \\
a_t &\geq 0.
\end{aligned} \tag{6.5}
$$

Nesting

Defining the post-decision value function

$$w_t(p_t, d_t^1, d_t^2, a_t) = \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}^1, n_{t+1}^1, m_{t+1})],$$

the keeper and adjuster value functions can be re-formulated as

$$\begin{aligned} v_t^{keep}(p_t, n_t^1, n_t^2, m_t) &= \max_{c_t} u(c_t, n_t^1, n_t^2) + w_t(p_t, d_t^1, d_t^2, a_t) \\ &\text{s.t.} \\ a_t &= m_t - c_t \\ m_{t+1} &= (1+r)a_t + y_{t+1} \\ n_{t+1}^1 &= (1-\delta_1)n_t^1 \\ n_{t+1}^2 &= (1-\delta_1)n_t^2 \\ a_t &\geq 0, \end{aligned} \tag{6.6}$$

and

$$\begin{aligned} v_t^{adj.}(p_t, x_t) &= \max_{d_t^1, d_t^2} v_t^{keep}(p_t, d_t^1, d_t^2, m_t) \\ &\text{s.t.} \\ m_t &= x_t - d_t^1 - d_t^2 \\ d_t^1 + d_t^2 &\in [0, x_t], \end{aligned} \tag{6.7}$$

and

$$\begin{aligned} v_t^{adj.,1}(p_t, n_1^2, x_t^1) &= \max_{d_t^1,} v_t^{keep}(p_t, d_t^1, n_t^2, m_t) \\ &\text{s.t.} \\ m_t &= x_t^1 - m_t - d_t^1 \\ d_t^1 &\in [0, x_t^1], \end{aligned} \tag{6.8}$$

and

$$\begin{aligned} v_t^{adj.,2}(p_t, n_1^1, x_t^2) &= \max_{d_t^2} v_t^{keep}(p_t, n_t^1, d_t^2, m_t) \\ &\text{s.t.} \\ m_t &= x_t^2 - d_t^2 \\ d_t^2 &\in [0, x_t^2]. \end{aligned} \tag{6.9}$$

EGM

Define the post-decision marginal value of cash by

$$q(p_t, d_t, a_t) = \beta R \mathbb{E}_t \left[\alpha c_{t+1}^{\alpha(1-\rho)-1} ((d_{t+1}^1 + \underline{d}^1)^\gamma (d_{t+1}^2 + \underline{d}^2)^{1-\gamma})^{(1-\alpha)(1-\rho)} \right]. \tag{6.10}$$

The Euler-equation for c_t is then given by

$$u_c(c_t, d_t^1, d_t^2) = q(p_t, d_t^1, d_t^2, a_t). \tag{6.11}$$

This implies that EGM can be performed as follows

$$c_t = z(a_t, d_t^1, d_t^2, p_t) \tag{6.12}$$

$$\begin{aligned} &= \left(\frac{q_t(p_t, d_t, a_t)}{\alpha((d_t^1 + \underline{d}_1)^\gamma (d_t^2 + \underline{d}_2)^{1-\gamma})^{(1-\alpha)(1-\rho)}} \right)^{\frac{1}{\alpha(1-\rho)-1}} \\ m_t &= a_t + c_t. \end{aligned} \tag{6.13}$$

The upper envelope algorithm is unchanged.

Implementation

Parametrization The chosen parameters are $\beta = 0.965$, $\rho = 2$, $\alpha = 0.9$ $\underline{d}_1 = \underline{d}_2 = 10^{-2}$, $R = 1.03$, $\tau_1 = 0.08$, $\tau_1 = 0.12$, $\delta_1 = 0.10$, $\delta_2 = 0.20$, $\gamma = 0.5$, $\sigma_\psi = \sigma_\xi = 0.1$, and $\lambda = 1$.

Grids Constructed as in the benchmark model, but with the following changes $\#_p = 50$, $\overline{n} = 2$, $\#_n = 50$, $\overline{m} = 10$, $\#_m = 100$, $\overline{x} = 12$, $\#_x = 100$, $\#_a = 100$.

Interpolation Done as in the benchmark model.

Simulation The initial values in the simulation are drawn as:

$$\begin{aligned} \log p_0 &\sim \mathcal{N}(\log(1), 0.2) \\ \log d_0^1 &\sim \mathcal{N}(\log(0.4), 0.2) \\ \log d_0^1 &\sim \mathcal{N}(\log(0.4), 0.2) \\ \log a_0 &\sim \mathcal{N}(\log(0.2), 0.1). \end{aligned}$$

References

Barillas, F., & Fernández-Villaverde, J. (2007). A generalization of the endogenous grid method. *Journal of Economic Dynamics and Control*, *31*(8), 2698–2712.

Berger, D., & Vavra, J. (2015). Consumption dynamics during recessions. *Econometrica*, *83*(1), 101–154.

Bertsekas, D. P. (2012) *Dynamic programming and optimal control: Approximate dynamic programming*. Athena Scientific

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

Druedahl, J., & Jørgensen, T. H. (2017). A general endogenous grid method for multi-dimensional models with non-convexities and constraints. *Journal of Economic Dynamics and Control*, *74*, 87–107.

Fella, G. (2014). A generalized endogenous grid method for non-smooth and non-concave problems. *Review of Economic Dynamics*, *17*(2), 329–344.

Fernandez-Villaverde, J., & Valencia, D. Z. (2018). *A practical guide to parallization in economics*. Technical report.

Harmenberg, K., & Oberg, E. (2017). *Consumption dynamics under time-varying unemployment risk*. Working Paper.

Hintermaier, T., & Koeniger, W. (2010). The method of endogenous gridpoints with occasionally binding constraints among endogenous variables. *Journal of Economic Dynamics and Control*, *34*(10), 2074–2088.

Hull, I. (2015). Approximate dynamic programming with post-decision states as a solution method for dynamic economic models. *Journal of Economic Dynamics and Control*, *55*, 57–70.

Iskhakov, F., Jørgensen, T. H., Rust, J., & Schjerning, B. (2017). The endogenous grid method for discrete-continuous dynamic choice models with (or without) taste shocks. *Quantitative Economics*, *8*(2), 317–365.

Johnson, S. G. (2014). The NLopt nonlinear-optimization package

Jørgensen, T. H. (2013). Structural estimation of continuous choice models: Evaluating the EGM and MPEC. *Economics Letters*, *119*(3), 287–290.

Judd, K. L. (1992). Projection methods for solving aggregate growth models. *Journal of Economic Theory*, *58*(2), 410–452.

Judd, K. L., Maliar, L., & Maliar, S. (2017). How to solve dynamic stochastic models computing expectations just once. *Quantitative Economics*, *8*(3), 851–893.

Low, H., & Meghir, C. (2017). The use of structural models in econometrics. *Journal of Economic Perspectives*, *31*(2), 33–58.

Ludwig, A., & Schön, M. (2018). Endogenous grids in higher dimensions: Delaunay interpolation and hybrid methods. *Computational Economics*, *51*, 1–30.

Ma, Q., & Stachurski, J. (2018). *Dynamic programming deconstructed*. Technical report.

Powell, W. B. (2011). *Approximate dynamic programming: Solving the curses of dimensionality*. New York: Wiley.

Rendahl, P. (2015). Inequality constraints and Euler equation-based solution methods. *The Economic Journal*, *125*(585), 1110–1135.

Santos, M. S. (2000). Accuracy of numerical solutions using the Euler equation residuals. *Econometrica*, *68*(6), 1377–1402.

Svanberg, K. (2002). A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, *12*(2), 555–573.

Van Roy, B., Bertsekas, D.P., Lee, Y., & Tsitsiklis, J.N. (1997). A neuro-dynamic programming approach to retailer inventory management. In *Decision and control, 1997., Proceedings of the 36th IEEE conference on* (Vol. 4, pp. 4052–4057). IEEE.

White, M. N. (2015). The method of endogenous gridpoints in theory and practice. *Journal of Economic Dynamics and Control*, *60*, 26–41.