A Guide On Solving Non-Convex Consumption-Saving Models*

Jeppe Druedahl†

February 10, 2019

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 efficiently solve 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 about 45 times faster than with standard value function iteration for a given level of accuracy. Software is provided in both Python and C++. (JEL: C6; D91; E21)

Keywords: Endogenous grid method, post-decision states, stochastic dynamic programming, continuous and discrete choices; occasionally binding constraints.

---

*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.

†CEBI, Department of Economics, University of Copenhagen, Øster Farimagsgade 5, Building 26, DK-1353 Copenhagen K, Denmark. E-mail: jeppe.druedahl@econ.ku.dk. Website: http://econ.ku.dk/druedahl.

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 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 about 45 for a given level of accuracy. The speed-up is reduced to a factor 15 for NEGM without the optimized interpolation approach. NVFI provides a speed-up factor of about 10.5 increasing to 13.5 with optimized interpolation.

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

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

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.

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, 2016) or restrict attention to the one-dimensional case (Fella, 2014; Iskhakov, Jørgensen, Rust and Schjerning, 2017).

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 is not required by NEGM, where only the first order condition for consumption is used. As an example, the benchmark model used in this paper *cannot* be solved by G^{2}EGM.

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 is it thus common to iterate on a Bellman equation in the post-decision value function; see in particular Van Roy, Bertsekas, Lee and Tsitsiklis (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, Maliar and Maliar (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 Starchurski (2018).

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

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

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 discuss the class of consumption-saving model where the proposed solution methods can be applied and 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,

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

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,

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

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

Further, the household cannot borrow, and the interest rate on savings is r. If the household adjusts its stock of the durable good (i.e. $d_t \neq n_t$) it incurs a proportional adjustment cost $\tau \in (0,1)$. The durable stock depreciates with a rate of $\delta \in (0,1)$, i.e.

$$n_{t+1} = (1-\delta) d_t. \tag{2.4}$$

2.1 Belleman 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.

$$\begin{aligned} v_t(p_t, n_t, m_t) &= \max\{v_t^{keep}(p_t, n_t, m_t), v_t^{adj.}(p_t, x_t)\} \\ &\text{s.t.} \\ x_t &= m_t + (1-\tau)n_t, \end{aligned} \tag{2.5}$$

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

$$\begin{aligned} v_t^{keep}(p_t, n_t, m_t) &= \max_{c_t} u(c_t, n_t) + \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})] \\ &\text{s.t.} \\ a_t &= m_t - c_t \\ d_t &= n_t \\ m_{t+1} &= (1+r)a_t + y_{t+1} \\ a_t &\geq 0. \end{aligned} \tag{2.6}$$

The value function when adjusting is

$$\begin{aligned} v_t^{adj.}(p_t, x_t) &= \max_{c_t, d_t} u(c_t, d_t) + \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})] \\ &\text{s.t.} \\ a_t &= m_t - c_t \\ m_{t+1} &= (1+r)a_t + y_{t+1}. \\ a_t &\geq 0. \end{aligned} \tag{2.7}$$

This problem can be solved straightforwardly with a *value function iteration* (VFI) algorithm.[4] The keeper problem in eq. (2.6) is costly because the state space is multi-dimensional, and the adjuster problem in eq. (2.7) 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.

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

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*

$$w_t(p_t, d_t, a_t) = \beta \mathbb{E}_t[v_{t+1}(p_{t+1}, n_{t+1}, m_{t+1})], \tag{2.8}$$

and re-write the keeper problem as

$$\begin{aligned} v_t^{keep}(p_t, n_t, m_t) &= \max_{c_t} u(c_t, n_t) + w_t(p_t, d_t, a_t) \\ &\text{s.t.} \\ a_t &= m_t - c_t \geq 0 \\ d_t &= n_t. \end{aligned} \tag{2.9}$$

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]

$$\begin{aligned} v_t^{adj.}(p_t, x_t) &= \max_{d_t} v_t^{keep}(p_t, d_t, m_t) \\ &\text{s.t.} \\ m_t &= x_t - d_t. \end{aligned} \tag{2.10}$$

[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.

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. (2.8) on a grid over the post-decision states p_t, d_t, and a_t.

Step 2: Solve the keeper problem in eq. (2.9) 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. (2.10) 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 relative 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.

[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 Section 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

$$u_c(c_t, d_t) = \alpha c_t^{\alpha(1-\rho)-1} d_t^{(1-\alpha)(1-\rho)} = q(p_t, d_t, a_t) \tag{2.11}$$

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

$$\begin{aligned} q_t(p_t, d_t, a_t) &= \beta R \mathbb{E}_t[u_c(c_{t+1}, d_{t+1})] \\ &= \beta R \mathbb{E}_t[\alpha c_{t+1}^{\alpha(1-\rho)-1} d_{t+1}^{(1-\alpha)(1-\rho)}] \end{aligned} \tag{2.12}$$

By solving the Euler-equaiton for c_t, this lets us define the following functions

[7] See the detailed discussion in Ludwig and Schön (2016) regarding triangulazation and Delaunay interpolation.

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

from post-decision states to consumption and then cash-on-hand,

$$c_t = z(a_t, d_t, p_t) = \left(\frac{q(a_t, d_t, p_t)}{\alpha d_t^{(1-\alpha)(1-\rho)}} \right)^{\alpha(1-\rho)-1} \tag{2.13}$$

$$m_t = a_t + c_t \tag{2.14}$$

Whenever the Euler-equation is necessary all points on the consumption function can be generated from eq. (2.13)-(2.14) 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 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.

Line 1-2 initialize the optimal value v at negative infinity. Line 3-6 apply eq. (2.8) and eq. (2.13)-(2.14) for all points in the input end-of-period asset vector. Line 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 (line 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.[9]

[9] 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**
  $w^i = w(a^i, d, p)$
  $c^i = z(a^i, d, p)$
  $m^i = a^i + c^i$
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**
      $c_j^i = c^i + \frac{c^{i+1} - c^i}{m^{i+1} - m^i}(m_j - m^i)$
      $v_j^i = u(c_j^i, d) + \left[w^i + \frac{w^{i+1} - w^i}{a^{i+1} - a^i}((m_j - c_j^i) - a^i)\right]$
      if $v_j^i > v^j$ **then**
        $v_j = v_j^i$
        $c_j = c_j^i$
return $v_1, v_2, \ldots, v_{\#_m}, c_1, c_2, \ldots, c_{\#_m}$

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 eq. (2.8) and eq. (2.13) on a grid over the post-decision states p_t, d_t and a_t.

Step 2: Solve the keeper problem in eq. (2.9) 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. (2.10) 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 (line 7-18) is a cheap operation as it involves only simple logical and 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 Section 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 positions in each dimension for the point to be interpolated (line 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 (line 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 (line 1-2) is done once instead of for each m. Secondly, the search for each a is faster because of the ordering (line 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 line 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.[10]

[10] A slight further speed-up is possible by using that the interpolations in line 12-13 in Algorithm 5 is for the same vector of next-period states. Line 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 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** $\omega_p = p - p_{j_m}$
    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** $\omega_m = m - m_{j_m}$
            $\hat{v}$ += $\frac{\omega_m \omega_n \omega_p}{\Omega} V[j_p + k_p, j_n + k_n, j_m + k_m]$
return $\hat{v}$

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

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 poitns to interpolate for:
$m_1, m_2, \ldots, m_{\#}$ 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 $i \in \{1, 2, \ldots, \#\}$ **do**
    if $i = 1$ **then**
        Find j_m^1 such that $m_1 \in [m_{j_m}, m_{j_m+1})$
    else
        $j_m^i = j_m^{i-1}$
        while $m_i \geq m_{j_m^i} + 1$ **do**
            j_m^i += 1
Initialize $\hat{v}_i = 0$ for $i \in \{1, 2, \ldots, \#\}$
for $k_p \in \{0, 1\}$ **do**
    if $k_p = 0$ **then** $\omega_p = p_{j_p+1} - p$ **else** $\omega_p = p - p_{j_m}$
    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 $i \in \{1, 2, \ldots, \#\}$ **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** $\omega_m = m_{j_m+1} - m$ **else** $\omega_m = m - m_{j_m^i}$
                $\hat{v}_i$ += $\frac{\omega_m \omega_n \omega_p}{\Omega} V[j_p + k_p, j_n + k_n, j_m^i + k_m]$
return $\hat{v}_1, \hat{v}_2 \ldots, \hat{v}_{\#}$

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 $j_p \in \{1, 2, \ldots, \#_p\}$ **do**
  for $j_n \in \{1, 2, \ldots, \#_n\}$ **do**
    for $j_\psi \in \{1, 2, \ldots, \#_\psi\}$ **do**
      for $j_\xi \in \{1, 2, \ldots, \#_\xi\}$ **do**
        $p_+ = p_{j_p} \psi_{j_\psi}$
        $n_+ = (1 - \delta) n_{j_n}$
        $y_+ = p_+ \xi_{j_\xi}$
        for $j_a \in \{1, 2, \ldots, \#_a\}$ **do**
          $m_+^{j_a} = R a_{j_a} + y_+$
        $\hat{v}_+ = interpvec(\mathcal{G}_p, \mathcal{G}_n, \mathcal{G}_m, v_+, (p_+, n_+, m_+^1, m_+^2, \ldots, m_+^{\#_a})$
        $\hat{u}_{c,+} = interpvec(\mathcal{G}_p, \mathcal{G}_n, \mathcal{G}_m, u_{c,+}, (p_+, n_+, m_+^1, m_+^2, \ldots, m_+^{\#_a})$
        for $j_a \in \{1, 2, \ldots, \#_a\}$ **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

3 Speed and accuracy

To asses 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 using 150 grid points for p_t and n_t and 300 grid points for m_t, a_t and x_t. All numerical integration is done with Gauss-Hermite quadrature using 8 nodes for both the persistent and the transitory shocks.

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

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

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 percent 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 more than 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 precompute. 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 roughly half of the initial gain is lost by more time spend in computing the post-decision functions, where the post-decision marginal value of cash must now also be calculated. Compared to NVFI and VFI, NEGM, however, provides a slight improvement in the level of accuracy. Turning to

NVFI+ and NEGM+ we see that the post-decision functions are computed three times faster, which especially benefits NEGM, where this step is the bottleneck. In the end, NEGM+ is about 45 times faster than VFI with a minor improvement in 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 very model and implementation specific. 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).

The complexity of the adjuster can also be important. In a model with a more complex adjuster problem in terms of more state 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.

Appendix Table A1 presents speed results for the Python implementations of NVFI+ and NEGM+ optimized with just-in-time compitation using the Numba package. Solution time only increases with a factor of about 1.5 in the Python

Table 1: Speed and Accuracy

	VFI	NVFI	NEGM	NVFI+	NEGM+
			relative Euler errors		
All (average)	−4.245	−4.173	−4.268	−4.173	−4.268
5th percentile	−5.811	−5.691	−5.908	−5.691	−5.908
95th percentile	−2.689	−2.708	−2.715	−2.708	−2.715
Adjusters (average)	−4.232	−4.352	−4.502	−4.352	−4.502
Keepers (average)	−4.247	−4.140	−4.225	−4.140	−4.225
			timings (in minutes, best of 5)		
Total	63.54	5.96	4.12	4.71	1.45
Post-decision functions	0.00	1.75	3.53	0.49	0.82
Keeper problem	62.26	4.19	0.57	4.20	0.61
Adjuster problem	1.28	0.02	0.02	0.02	0.02
Speed-up relative to VFI		10.66	15.42	13.49	43.85
			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

Notes: The model is solved backwards for $T = 50$ periods using 150 grids point for p_t and n_t and 300 grids points for m_t, a_t and x_t. All numerical integration is done with Gauss-Hermite quadrature using 8 nodes for both the persistent and the transitory shocks. The simulation outcomes are computed from a sample of 100,000 households, and the Euler errors are calculated as in eq. (3.1). Further details on the implementation is provided in Appendix A. For VFI, the optimization problems are solved by the *method of moving asymptotes* from Svanberg (2002), implemented in `NLopt` by Johnson (2014). For NVFI and NEGM, the optimization problems are solved by a simple *golden section search*. These optimizers were found to deliver the best performance. 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 5 runs.

implementations.[11]

4 Discussion and conclusions

The solution methods proposed in this paper can be used for many other models than the one considered here.

A pure consumption problem like the keeper problem can always be solved separately and relied upon when solving for the remaining choices. In some models it might, however, be necessary to expand the pure consumption problem with additional states making the proposed solution methods less attractive. 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 also on current labor supply if consumption and leisure are non-separable. Such expansions of the state space can, however, be avoided whenever the derivative of the utility function wrt. consumption only depend on consumption and post-decision states.

The pre-computation of the post-decision value function w_t (and the post-decision marginal value of cash q_t) is also always possible, and it is beneficial whenever the dimensionality of the post-decision state space is not larger than the dimensionality of the pre-decision state space.

The use of the endogenous grid method, however, requires that the Euler-equation is necessary, which can normally be ensured by a standard variation argument.[12] Additionally, the derivative of the utility function wrt. consumption should be invertible with respect to consumption, which ensures that the z_t function in eq. (2.13) exists.

The use of the efficient interpolation Algorithm **??** finally directly requires that next-period cash-on-hand is a monotone function of end-of-period assets and that end-of-period assets does not affect other next-period variables.

In sum, the proposed solution methods seems to be applicable for a broad class of

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

[12] In cases where next-period cash-on-hand is not everywhere differentiable in end-of-period assets a separate Euler-equation will be necessary for each differentiable segment.

consumption-saving models, and were shown to be much faster than a standard value function iteration approach for a given level of accuracy, while still being relatively parsimonious and simple to use in practice. 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.

References

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

Berger, D. and J. Vavra (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. and T. H. Jørgensen (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. and D. Z. Valencia (2018): "A Practial Guide to Parallization in Economics," Discussion paper.

Harmenberg, K. and E. Oberg (2017): "Consumption Dynamics under Time-varying Unemployment Risk," Discussion paper.

Hintermaier, T. and W. Koeniger (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., T. H. Jørgensen, J. Rust and B. Schjerning (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., L. Maliar and S. Maliar (2017): "How to Solve Dynamic Stochastic Models Computing Expectations Just Once," *Quantitative Economics*, 8(3).

Low, H. and C. Meghir (2017): "The Use of Structural Models in Econometrics," *Journal of Economic Perspectives*, 31(2), 33–58.

Ludwig, A. and M. Schön (2016): "Endogenous Grids in Higher Dimensions: Delaunay Interpolation and Hybrid Methods," *Computational Economics*, pp. 1–30.

Ma, Q. and J. Starchurski (2018): "Dynamic Programming Deconstructed," Discussion paper.

Powell, W. B. (2011): *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons.

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*, pp. 555–573.

Van Roy, B., D. P. Bertsekas, Y. Lee and J. N. Tsitsiklis (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.

A Implementation details

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

Grids The grid for p is constructed by the recursion:

$$\begin{aligned} p^1 &= \underline{p} \\ p^i &= p^{i-1} + \frac{\overline{p} - p^{i-1}}{(\#_p - (i-1))^{1.1}},\ i = 2, 3, \ldots, \#_p, \end{aligned}$$

with $\underline{p} = 10^{-4}$, $\overline{p} = 3$ and $\#_p = 150$. The other grids are constructed similarly with $\underline{n} = 0$, $\overline{n} = 3$, $\#_n = 150$, $\underline{m} = 0$, $\overline{m} = 10$, $\#_m = 300$, $\underline{x} = 0$, $\overline{x} = 13$, $\#_x = 300$, $\underline{a} = 0$, $\overline{a} = 14$, and $\#_a = 300$. 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:

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

These transformed functions are always positive and increasing, and satisfy

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

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.

Simulation The initial values in the simulation is drawn as:

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

B Tables and Figures

Table A1: Speed and Accuracy in Python

	NVFI+		NEGM+	
	C++	Python	C++	Python
	relative euler errors			
All (average)	−4.173	−4.173	−4.268	−4.268
5th percentile	−5.691	−5.691	−5.908	−5.908
95th percentile	−2.708	−2.708	−2.715	−2.715
Adjusters (average)	−4.352	−4.352	−4.502	−4.502
Keepers (average)	−4.140	−4.140	−4.225	−4.225
	timings (in minutes, best of 5)			
Total	4.71	7.12	1.45	2.22
Post-decision functions	0.49	0.97	0.82	1.59
Keeper problem	4.20	6.13	0.61	0.60
Adjuster problem	0.02	0.03	0.02	0.03
	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

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