Adaptive Sparse Grids for Solving Continuous Time Heterogeneous Agent Models

Steffen Ruttscheidt

Born 6th May 1994 in Bonn, Germany

12th March 2018

Master's Thesis Mathematics

Advisor: Prof. Dr. Jochen Garcke

Second Advisor: Prof. Dr. Michael Griebel

Institute for Numerical Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Acknowledgements

I would like to thank Prof. Dr. Garcke for offering this master's thesis under his supervision. The long discussions and advices were extremely helpful. Furthermore, I thank Prof. Moll for proposing several interesting model problems and for his great suggestions for improving this work. Additionally, I am highly thankful for the funding of the Department of Economics of Princeton University. Another important factor for the success of this thesis was SeHyoun Ahn from Princeton University (currently working at Norges Bank) who shared his wide knowledge of using sparse grid techniques on economic models. I would also like to thank Prof. Dr. Griebel for being the second assessor. Last but not least, I thank my family and my friends for their personal support.

Abstract

We present a finite difference method working on sparse grids to solve higher dimensional heterogeneous agent models. If one wants to solve the arising Hamilton-Jacobi-Bellman equation on a standard full grid, one faces the problem that the number of grid points grows exponentially with the number of dimensions. Discretizations on sparse grids only involve $\mathcal{O}(N(\log N)^{d-1})$ degrees of freedom in comparison to the $\mathcal{O}(N^d)$ degrees of freedom of conventional methods, where N denotes the number of grid points in one coordinate direction and d is the dimension of the problem. Whereas one can show convergence for the used finite difference method on full grids by using the theory introduced by Barles and Souganidis [BS90], we explain why one cannot simply use their results for sparse grids. Our numerical studies show that our method converges to the full grid solution for a two-dimensional model. Further, we analyze the convergence behavior for higher dimensional models, experiment with different sparse grid adaptivity types, and propose ILUC preconditioned BiCGSTAB for solving the arising linear system, reducing the required computation time by a large amount.

Contents

1 Introduction

A lot of advances in economic research in recent years are due to the formulation of models that do not admit closed form solutions. One is particularly interested in models of higher dimensionality, such as heterogeneous agent models which may have a large amount of agents that differ in some dimensions. These heterogeneities, such as productivity, can be modeled by stochastic processes. Further, there are models with a large number of state variables (e.g. New Keynesian models), asset pricing models may feature many different assets and multi-country models may have a large number of countries.

Thus, it is important to develop efficient numerical methods to approximate and compute the solution of higher dimensional problems.

1.1 Setting and solution method used in this work

1.1.1 Setting and main idea

In this work we will show how to solve continuous time heterogeneous agent models with multiple assets in higher dimensions. Mathematically, this comes down to solving stochastic optimal control problems by approximating the solution of the *Hamilton-Jacobi-Bellman* (HJB) equation. Notice that the approach is not limited to asset problems but can be used for continuous time heterogeneous agent models in general.

With standard discretizations, one faces the problem that one cannot introduce arbitrarily many variables due to the *curse of dimensionality*, a terminology coined by Bellman in [Bel61] that describes the exponential dependence of the overall computational effort on the number of dimensions. That is why there are several approaches to circumvent or to at least reduce this bottleneck. We propose to use *sparse grids* which are built on a tensor product basis, see [BG04] for an overview article. In particular, we use a finite difference method on sparse grids to solve the HJB equation arising from continuous time heterogeneous agent models. This approach is motivated by the following work.

1.1.2 Basis of our approach

In [Can99] a finite difference method is used to solve the HJB equation in the economic context. This approach was improved in [AHL$^+$17] to first handle borrowing constraints (explained in Section 2) by mathematically recasting them as state constraints. Second, they couple the HJB equation with the *Fokker Planck* (FP) equation (also known as *Kolmogorov Forward* (KF) equation) which allows not only to investigate the optimal individual decisions but also the aggregate evolution of the distribution of state variables. Problems with these coupled systems are called *mean field games*. Third, they underlie their finite difference scheme by using the notion of viscosity solutions and prove converge due to [BS90]. This computational method was adopted in [KMV16] to handle non-convexities and multiple assets.

In [Sch98] finite differences on sparse grids were introduced and several theoretical results regarding consistency, stability and convergence are proved. Further studies have been made in [Gri98], [GS99] and [Zum00]. Nevertheless, the theory remains limited mostly due to the difficult handling of specific basis transformations used in the sparse grid finite difference operators. Furthermore, the implementation is non-trivial and most sparse grid libraries do not feature these operators. Therefore, we introduce a finite difference method based on an idea by [Ahn18] which is based on interpolation and can be implemented more easily. Notice that interpolation based ideas are already presented in [Kos].

1.1.3 Our work and main contributions

By simply combining the ideas presented above as proposed in upcoming work [Ahn18], we are able to solve continuous time heterogeneous agent models in higher dimensions. We use exactly the same finite difference approach like the one proposed in [AHL$^+$17] but instead of implementing it on full grids we do it on sparse grids.

We generalize several notations for the presentation of the algorithmic approach in the sparse grid setting and explain the sparse grid finite difference operator construction proposed for [Ahn18]. Further, we show that we do not get convergence of the scheme by simply using the results of [BS90] as in the full grid case. This comes down to the non-monotonicity of sparse grid interpolation. However, we present some ideas how to get monotone interpolation. Even though we do not have a theoretical convergence result, we give numerical results that show that our sparse grid solution converges to the full grid solution for a two-dimensional model. We further implement higher dimensional models for our numerical experiments in which we analyze the convergence behavior for regular sparse grids and for adaptive sparse grids with different adaptivity approaches. Additionally, we do runtime comparisons for different sparse grid levels including different approaches for solving the arising linear system.

1.2 Context of this work: related problem settings and solution methods

One can distinguish economic models by sevaral aspects.

First, models can be set either in *discrete time* or in *continuous time* as we do. Notice that discrete time models are already used extensively, whereas continuous time models just recently got more attention. One of the reasons is that discrete time models require less mathematical knowledge, e.g. due to the lack of partial differential equations (PDEs) and notions of weak solutions. However, here are several computational advantages of continuous time models over discrete time ones as explained in [AHL$^+$17]. First, the borrowing constraint only shows up as boundary condition and thus the first order condition (explained in Section 2) holds everywhere in the interior with equality. Contrarily, in the discrete case the first order condition is an inequality due to the possibility of binding at the prior period. Second, by using the first order conditions one can, given the derivatives of the value function, explicitly compute

the control variables by hand. Optimal choices in the discrete case though, are only defined implicitly and thus often costly root-finding methods have to be used. Other explained advantages like the sparsity structure of the discretization matrix and the link to the KF equation unfortunately do not remain in the sparse grid case as explained in Section 4.3.6.

Second, there are *finite time* and *infinite time* models. In economics, infinite time models are often just used to simplify theoretical aspects. Due to discount factors or similar model parameters the results often do not differ much. For example, there are also stopping time problems that give an extra utility at a specific stopping time. We point out that, even though we just solve infinite time models in this work, the numerical approach can be used for other model types in the same fashion.

Third, not only the model types differ but also the solution types. Namely, there are *closed loop* and *open loop solutions*. Whereas the former is a function of the states that is often called *feedback control* or *policy function*, the latter is a function of time which is a trajectory through the state space. Economists are mostly interested in closed loop solutions and this is also the type of solution we are aiming for.

Last but not least, note that we do not solve coupled systems with the KF as it is done in [AHL+17]. The reason for that is that sparse grids do not preserve mass and the function value range. However, in work in progress [PF18], it is shown how to preserve the function value range. Similar approaches could be used for solving mean field games.

1.2.1 Other approaches to solve high dimensional continuous time model problems

Instead of finite differences one can solve the HJB equation with several other approaches. The most popular one is the Semi-Lagrangian method which is based on characteristics. We refer to [FF13] for an introduction. It was combined with sparse grids by [BGGK13] and further experimentally studied in [GK17]. Some numerical bounds on the approximation error under specific assumptions can be found in [War14], and [KW15] and a related open source library was implemented for [GLW16].

An efficient Semi-Lagrangian approach to solve the HJB equation in higher dimensions can be found in [CFF04]. In [CF15] a Semi-Lagrangian method is combined with a dynamic domain decomposition, and in [KK17] a pseudo-spectral collocation approximation of the PDE dynamics is used. For a general overview of stochastic optimal control in continuous time and other numerical methods, see [KD13].

1.2.2 Approaches used in economics to handle high dimensional discrete time model problems

We refer to [SJ13] for a broad overview of computational methods for solving high dimensional discrete time economic models. Let us briefly summarize the most important approaches and additionally reference some more recent works.

Conventional numerical methods to solve dynamic economic models do not allow feasible or accurate computations in higher dimensions. Stochastic simulation algorithms build on Monte Carlo integration and least square learning. Whereas the former does not achieve a high accuracy, the latter may become unstable. Further, projection methods build on tensor product constructions and are thus not feasible in high dimensions. Last but not least, perturbation methods that solve for a steady state by using Taylor expansions have uncertain accuracy.

To overcome the above described issues, the approaches were adapted to handle high dimensional problems. In [JMM09] a generalized stochastic simulation approach is proposed that replaces the Monte Carlo integration with a deterministic one, and the least squares learning with numerically stable regression methods. In [KK04] sparse grids are used to replace the expensive tensor product grids. For perturbation methods that are feasible in higher dimensions, see [JJ02] and [MMV13].

Sparse grids in combination with a fixed point iteration on the Euler equation are proposed in [JMMV14] to solve a multi-country model featuring up to twenty state variables. Combining it with a simulation to determine the high probability area and then using a principal components transformation allows it to focus the computation on the relevant domain. Parallel adaptive sparse grids were recently used in [BS17] to solve high dimensional stochastic dynamic models where functions are interpolated on a sparse grid either within time or value iterations. Further, in work in progress, [Sch18], dynamic portfolio choice models are solved with adaptive sparse grids.

For a general overview of stochastic optimal control in the discrete time case, we refer to [BS04] and the references therein.

1.3 Structure of this work

This work is structured as follows. The setup and motivation is given in Section 2. In Section 3 we explain sparse grids and two different versions of finite differences on sparse grids. In Section 4 we present the convergence theory developed by [BS90], recall the basics about viscosity solutions and explain the numerical approach for a simple one-dimensional model. An investigation of sparse grid interpolation is done in Section 5 to explain why we do not simply get convergence by means of Barles and Souganidis. We further present some approaches to overcome some of the arising issues regarding non-convergence. After that, in Section 6 we present a two-dimensional model and the resulting discretization. It is followed by a detailed presentation of the algorithm and its implementation in Section 7. The numerical results are given in Section 8. We conclude this work with an outlook in Section 9. References can be found in the following bibliography. The Appendix A contains information about the implemented higher dimensional models and the choice of parameters for the numerical experiments.

We aim to describe everything in a way that is useful and understandable for both mathematicians and economists.

2 Setup and Basics

In this work we aim to solve heterogeneous agent models. Even though traditionally heterogeneous agent models have mostly been set in discrete time, recently there is a lot of progress using continuous time formulations. E.g. several well known heterogeneous agent models (Bewley [Bew86], Huggett [Hug93] and Aiyagari [Aiy94] models) were recasted in continuous time by [AHL$^+$17] for detailed studies.

Note that agents in heterogeneous agent models are the same *ex ante* but face different idiosyncratic shocks and are thus different *ex post*. Contrarily, in representative agent models agents do not face these shocks and thus stay the same. Note that this is in stark contrast to the real world where one can observe a high degree of inequality in the population and thus a lot of heterogeneity. Therefore, the use of heterogeneous agent models is an important step to better match real world data. Further, note that the amount and quality of micro data has been constantly improving in the last years. Macro economists now can for example verify model solutions more easily and better adjust models by using this data. In [AHL$^+$17] it is further pointed out that one often wants to analyze welfare implications of particular policies or shocks.

In this work we restrict ourselves to models that feature uninsurable income shocks and multiple assets of different liquidity types with different returns. Note that agents can self-insure the arising income risk by savings in the different assets. Low-dimensional models of this type have been analyzed in detail in [KMV16]. These models allow insights regarding household consumption and saving behavior.
Even though we restrict ourselves to these type of models, we point out that the presented frameworks are basically applicable to any heterogeneous agent model.

As already mentioned in Section 1, in [AHL$^+$17] a coupled system of HJB equation and KF equation is solved. Whereas the HJB equation gives optimal choices of a single individual who takes prices as given, the KF equation characterizes the evolution of the distribution given the optimal choices of the individuals. Due to the mentioned difficulties of approximating distributions on sparse grids we focus on solving for the optimal individual choices in this work.

We begin this section by giving a general framework for optimal control problems, first for deterministic and then for stochastic settings. It is followed by a setup of a simple economic model with explanations of the model components. We explain how to extend this simple model to feature borrowing constraints, heterogeneity or multiple assets, also giving some explanations in the economic context. Finally, we give a short motivation for our numerical approach. Notice that we are mainly following [Mol16a] and [Mol16b]. For more mathematical descriptions and proofs, see [KD13].

2.1 General optimal control problems

2.1.1 Deterministic models

Most deterministic infinite time optimal control problems in continuous time can be written as

$$\max_{\{\alpha(t)\}_{t\geq 0}} \int_0^\infty e^{-\rho t} h(x(t), \alpha(t)) dt$$

such that the law of motion for state

$$\dot{x}(t) = f(x(t), \alpha(t)) \text{ and } \alpha(t) \in A$$

holds for $t \geq 0$ and $x(0) = x_0$ given using the notation $\dot{x}(t) = \frac{d}{dt}x(t)$.

Here $x \in X \subset \mathbb{R}^m$ denotes the *state vector*, $\alpha \in A \subset \mathbb{R}^n$ the *control vector* and $h : X \times A \to \mathbb{R}$ the *instantaneous return function*. Further, $\rho \geq 0$ denotes the *discount rate* which discounts future returns. Note that the state changes depending on the current state and action (control), following $f : X \times A \to \mathbb{R}^m$.

We can now introduce some important notions.

Definition 2.1.1. We define the *objective function* by

$$J(x, \alpha) = \int_0^\infty e^{-\rho t} h(x(t), \alpha(t)) dt.$$

The associated *value function* can then be written as

$$v(x) = \max_{\{\alpha(t)\}_{t\geq 0}} J(x, \alpha).$$

We define the *optimal control* as the $\hat{\alpha} \in A$ such that

$$v(x) = J(x, \hat{\alpha}).$$

Note that we aim to find the optimal controls for our model problems by solving for the value function which is the solution of the HJB equation to which we turn now.

HJB equation To solve optimal control problems, we use the *dynamic programming principle* (DPP) introduced by Bellman in the 1950s, see [Bel57]. It is based on the recursive structure of the problem and by using this principle one can show that the value function satisfies the HJB equation

$$\rho v(x) = \max_{\alpha \in A} h(x, \alpha) + D_x v(x) \cdot f(x, \alpha), \quad \forall x \in X.$$

To compute the optimal controls, one uses the *first order conditions* (FOCs) on the HJB equation, i.e. computing the derivatives with respect to the different controls and setting them to zero.

2.1.2 Stochastic models

The theory explained above for deterministic problems can be easily extended to stochastic settings which is covered by several textbooks such as [Sto08]. Thus, in this section we focus our presentation on different types of stochastic processes that are well suited to model economic models and just present the respective arising HJB equations.

Stochastic processes We consider two types of stochastic processes, namely jump type processes and diffusion processes, i.e. continuous-time Markov processes with almost surely continuous sample paths.

Let us begin with the latter and give the arising stochastic differential equation for a *diffusion process*,

$$\dot{x}(t) = \mu(x)dt + \sigma(x)dW(t) \tag{1}$$

with $W(t) \sim \mathcal{N}(0,t)$ normally distributed, where $\mu(\cdot)$ is called *drift* and $\sigma(\cdot)$ is called *diffusion*.
By choosing $\mu(x) = \theta(\bar{x} - x)$ with parameter θ where $\bar{x}$ denotes the mean of x and $\sigma(x) = \sigma$, we get a stationary process which is called *Ornstein-Uhlenstein process*.

Note that numerous other processes can be used, e.g. geometric Brownian motions or Feller square root processes. The simplest way of modeling uncertainty in continuous time is the use of a *two state Poisson process*, i.e.

$$x(t) \in \{x_1^P, x_2^P\} \text{ Poisson with intensities } \lambda_1, \lambda_2 \tag{2}$$

with $x_2^P > x_1^P$, i.e. the process jumps from state 1 to state 2 with intensity λ_1 and vice versa with intensity λ_2.

Stochastic models Consider the following general optimal control problem,

$$\max_{\{\alpha(t)\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} h(x(t), \alpha(t))dt \tag{3}$$

such that the law of motion for state

$$\dot{x}(t) = f(x(t), \alpha(t)) + \sigma(x(t))dW(t) \tag{4}$$

holds for $\alpha(t) \in A$, $t \geq 0$ and $x(0) = x_0$ given.

Note that one should distinguish between endogenous and exogenous state variables. Consider $x = (x_1, x_2)$ such that

$$\begin{aligned} \dot{x}_1 &= \tilde{f}(x_1, x_2, \alpha) \\ \dot{x}_2 &= \tilde{\mu}(x_2)dt + \tilde{\sigma}(x_2)dW \end{aligned} \tag{5}$$

where the *endegenous* state variable x_1 depends on the control and the other state, whereas *exegenous* state variable x_2 does not. For the above framework (3) - (4), we get

$$f(x,\alpha) = \begin{bmatrix} \tilde{f}(x_1, x_2, \alpha) \\ \tilde{\mu}(x_2) \end{bmatrix}, \quad \sigma(x) = \begin{bmatrix} 0 \\ \tilde{\sigma}(x_2) \end{bmatrix}. \tag{6}$$

Note that we can similarly set up models with Poisson type processes.

HJB in stochastic settings Using Itô's lemma, see e.g. [Pro05], one can show that the model (3) - (6) leads to the HJB equation

$$\begin{aligned} \rho v(x_1, x_2) = &\max_{\alpha \in A} h(x_1, x_2, \alpha) + v_{x_1}(x_1, x_2)\tilde{f}(x_1, x_2, \alpha) \\ &+ v_{x_2}(x_1, x_2)\tilde{\mu}(x_2) + \frac{1}{2} v_{x_2 x_2}(x_1, x_2)\tilde{\sigma}^2(x_2). \end{aligned} \tag{7}$$

If we take the Poisson process given by (2) for x_2 instead of the process defined in (5), we get the HJB equation

$$\begin{aligned} \rho v(x_1, x_i^P) = &\max_{\alpha \in A} h(x_1, x_i^P, \alpha) + v_{x_1}(x_1, x_i^P) f(x_1, x_i^P, \alpha) \\ &+ \lambda_i (v(x_1, x_j^P) - v(x_1, x_i^P)) \end{aligned} \tag{8}$$

with Poisson states $i, j = 1, 2,\ j \neq i$. Note that this can easily be extended to more than two Poisson states.

2.2 Optimal control problems in economics

The above framework can be used to model economic settings. We present a simple economic model and explain several possible extensions in the economic context. The arising HJB equations are stated at the end of the respective paragraphs. Note that we omit the initial state in the following and denote the time dependence of the states by a subscript t.

2.2.1 Simple economic model

Consider the following simple one-dimensional deterministic model. We want to maximize

$$\max_{\{c_t\}_{t \geq 0}} \int_0^\infty e^{-\rho t} u(c_t) dt \tag{9}$$

subject to

$$\dot{b}_t = w + r^b b_t - c_t. \tag{10}$$

Here c_t is consumption and b_t are liquid assets at time t respectively. Further, r^b denotes returns on b and w is wage.

Whereas *wage* and *consumption* are self-explanatory, we aim to explain the other components of the model. One can label an asset as liquid or illiquid

depending on the extend to which transaction costs are involved for buying or selling them. As it is done in [KMV16], we define *liquid assets* as deposits in financial institutions saving, checking, call and money market accounts, government bonds and corporate bonds net of revolving consumer credit. The *rate of returns* indicates at which rate the assets generate earnings. Note that for negative b this is a borrowing rate.

Notice that for the framework given in Section 2.1 we have the state $x(t) = b_t$ and the control $\alpha(t) = c_t$ at time t. Moreover, the state changes at time t are modeled by $f(x(t), \alpha(t)) = f(b_t, c_t) = w + r^b b_t - c_t$. Thus, at time t, for the liquid asset state b_t, we want to choose an optimal control c_t, i.e. how much we consume, to maximize (9). Note again that this choice directly reflects in the change of the state. A standard choice for the return function h is the CRRA utility function given by

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

parameter $\gamma > 0$. Note that u is strictly convex and strictly monotone increasing in c.

For the model (9) - (10), we get the HJB equation

$$\rho v(b) = \max_c u(c) + v'(b)(w + r^b b - c).$$

Using the first order conditions explained in Section 2.1, one gets

$$c = (u')^{-1}(v'(b))$$

and given the derivative (or later its approximation) one can simply compute the optimal control.

2.2.2 Extension: borrowing constraint

One can additionally introduce a *borrowing constraint* which is the maximum amount of money an agent can borrow (e.g. from banks, firms or governments). It can be modeled by

$$b_t \geq \underline{b},$$

i.e. the value of liquid assets b cannot go below $\underline{b}$. Notice that $\underline{b} = 0$ means that the agent is not allowed to borrow but just to save. For an explanation of specific types of borrowing constraints like the natural borrowing limit, we refer to [AHL$^+$17]. For an explanation how the borrowing constraint shows up in the HJB equation, see Section 4.3.

2.2.3 Extension: heterogeneity - stochastic settings

It is easily possible to extend the deterministic setting described above to a stochastic one, i.e. adding heterogeneity to the model. We explain it for a two-state Poisson process and general diffusion type stochastic processes.

Let us begin with the modified model for the latter. We want to solve

$$\max_{\{c_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{11}$$

subject to

$$\begin{aligned} \dot{b}_t &= w z_t + r^b b_t - c_t \\ \dot{z}_t &= \mu(z_t)dt + \sigma(z_t)dW_t \\ b_t &\geq \underline{b}. \end{aligned} \tag{12}$$

We assume that the exogenous productivity state z evolves stochastically over time on a bounded interval $[\underline{z}, \bar{z}]$ with $\underline{z} \geq 0$ such that the diffusion is reflected on the boundaries in dimension z, i.e.

$$\partial_z v(b, \underline{z}) = 0 \text{ and } \partial_z v(b, \bar{z}) = 0, \text{ for } b \in (\underline{b}, \infty).$$

Instead of simply getting wage w, one now gets uninsured income wz_t. Notice further that we now have to take the expected value in (11) since the model is no longer deterministic. Thus, *productivity*, which is a measure for the output per unit of input, is modeled such that it influences the households income. For the same model setup one could also interpret z as specific skill that influences the income. Note that all agents face different productivity shocks and thus this is an example of an economic model featuring heterogeneity.

As explained for (7), using Itô's formula we get the HJB equation

$$\begin{aligned} \rho v(b,z) = \max_c u(c) &+ \partial_b v(b,z)(wz + r^b b - c) \\ &+ \partial_z v(b,z)\mu(z) + \frac{1}{2}\partial_{zz} v(b,z)\sigma^2(z) \end{aligned} \tag{13}$$

for the modified model.

If we replace $\dot{z}_t = \mu(z_t)dt + \sigma(z_t)dW_t$ in (12) by

$$z_t \in \{z_1, z_2\} \text{ Poisson with intensities } \lambda_1, \lambda_2, \tag{14}$$

we have a two-state Poisson type process instead of continuous stationary diffusion process for productivity z.

As stated for the general model by (8), the HJB equation for model (11)-(12) modified by (14) is

$$\rho v(b,z) = \max_c u(c) + \partial_b v(b,z)(wz + r^b b - c) + \lambda_i (v(b, z_j) - v(b_1, z_i)) \tag{15}$$

with Poisson states $i, j = 1, 2,\ j \neq i$.

2.2.4 Extension: illiquid assets

Instead of just having liquid assets, we can extend the model to also feature illiquid assets. One can model this by asset holdings of a household evolving according to

$$\begin{aligned} \dot{b}_t &= wz_t r^b(b_t)b_t - d_t - \chi(d_t, a_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t \\ b_t &\geq b, \;\; a_t \geq 0 \end{aligned} \tag{16}$$

where a_t denotes illiquid assets, d_t is the deposit rate and $\chi(d_t, a_t)$ the transaction cost function for time t respectively. The other variables remain the same as above. We point out that the modified model features both an additional control, i.e. d, and an additional state, i.e. a.

Contrary to liquid assets b_t, illiquid assets a_t cannot be sold that easily without loosing value since (higher) transaction costs for selling and buying are involved. We follow [KMV16] by defining *illiquid assets* as real estate wealth net of mortgage debt, consumer durables net of non-revolving consumer credit, plus equity in the corporate and non-corporate business sectors. The *deposit rate* is the amount one transfers into the other account. If $d_t > 0$, one deposits into the illiquid account and if $d_t < 0$, one withdraws from the illiquid account. Households have to pay a *transaction cost* $\chi(d_t, a_t)$ for depositing or withdrawing from their illiquid account. In [KMV16] it is pointed out that in the equilibrium illiquid assets pay a higher return than liquid assets due to the transaction costs, i.e. $r^a > r^b$. Furthermore, note that short positions are not allowed.

2.2.5 Further extensions and other models

Obviously, there are a lot of other models that feature completely different components but the ones we described. However, there are also several variables that could be introduced to the above setting, e.g. a housing asset that pays a utility return instead of monetary return. Check model (51)-(52) in Appendix A which features such an asset and which we implemented for our studies. Moreover, one can introduce variables like age to the model, e.g. combined with a death rate. Note that one also may want to introduce different types of skill modeled by stochastic processes that evolve differently over time, see e.g. model (53)-(54) in Appendix A.

2.3 Motivation to solve higher dimensional models

Using standard approaches, one is only able to solve low-dimensional models with at most three state variables. Therefore, one has to either choose very simple settings or one has to make a lot of assumptions (e.g. about specific variables not being important). These are hard restrictions to formulate models which allow for interesting implications. That is why we present an approach to handle higher dimensional economic models. We propose to use sparse grids to which we turn now.

3 Sparse Grids

Sparse grids were introduced by the russian mathematician Smolyak [Smo63] and were further studied in [Zen91]. A detailed overview of sparse grids can be found in [BG04]. We mainly follow the presentations of [GG10] and [Pfl10].

Sparse grids are known under several different names like hyperbolic cross points, Smolyak method or boolean interpolation. They are based on a higher-dimensional multiscale basis which can be derived from a one-dimensional multiscale basis by using a tensor product construction.

Let N denote the number of grid points in one coordinate direction and let d be the dimensionality of the problem. Discretizations on sparse grids only involve $\mathcal{O}(N(\log N)^{d-1})$ degrees of freedom in comparison to the $\mathcal{O}(N^d)$ degrees of freedom of conventional methods. Thus, they do not underly the curse of dimensionality, i.e. the exponential dependence on dimension d, anymore and are thus well-suited for high-dimensional problems. Notice that there still is an exponential dependence on the logarithmic term and thus it is not possible to use them in arbitrarily high dimensions. To approximate solutions with bounded mixed derivatives, one achieves an accuracy of $\mathcal{O}(N^{-2}(\log N)^{d-1})$ with respect to the L_2 and L_∞-norm, if piecewise linear basis functions are used. Hence, they are applicable to higher dimensional settings without losing too much accuracy.

If one knows that the solution depends more heavily on some variables, one can use anisotropic sparse grids which have more grid points in individual dimensions. To approximate non-smooth solutions, there are sparse grid approaches with a posteriori adaptive refinement which are based on error indicators in the sparse grid points. Further, one can use higher-order polynomials, prewavlets, interpolets, wavelets and other basis functions instead of piecewise linear basis functions which lead to better accuracy in some settings.

There exist sparse grid methods for solving partial differential equations, numerical quadrature, data mining and several other numerical fields. That is why they are used in areas such as physics, machine learning or economics.

In the first part of this section, we describe the construction of sparse grids and recall some theoretical results. Then we extend the theory to non-zero boundaries and adaptive sparse grids. We conclude this section with a presentation of two approaches for the construction of finite difference operators working on sparse grids.

3.1 Construction and extensions of sparse grids

3.1.1 One-dimensional basis functions

The so-called *hierarchical basis functions* can be used as a one-dimensional multilevel basis. They are based on the well known hat functions

$$\Phi(x) := \begin{cases} 1 - |x| & \text{if } x \in [-1, 1], \\ 0 & \text{otherwise.} \end{cases}$$

Let us define the set Ω_l of level $l \in \mathbb{N}$ as the set of equidistant grids on the unit interval $[0, 1]$ with mesh width 2^{-l}. This allows us to describe the grid points by

$$x_{l,i} := i \cdot h_l, \ 0 \leq i \leq 2^l,$$

with index $i \in \mathbb{N}$. By translation and dilation, we obtain a family of basis functions $\Phi_{l,i}(x)$ with support in $[x_{l,i} - h_l, x_{l,i} + h_l]$,

$$\Phi_{l,i}(x) := \Phi\left(\frac{x - i \cdot h_l}{h_l}\right),$$

called *nodal basis*. Notice that the location of the basis functions and grid points is indicated by the index i. We define the *space of piecewise linear nodal basis functions*

$$V_l := \text{span}\{\Phi_{l,i} : 1 \leq i \leq 2^l - 1\}$$

and the *space of hierarchical increment basis functions*

$$W_l := \text{span}\{\Phi_{l,i} : 1 \leq i \leq 2^l - 1, \ i \text{ odd}\}.$$

Notice that we have the following relationship between the two spaces. First, it holds

$$V_l = \bigoplus_{k \leq l} W_k$$

which is visualized in Figure 1, and second, any function $u \in V_l$ can be uniquely represented as

$$u(x) = \sum_{k=1}^{l} \sum_{i \in I_k} a_{k,i} \Phi_{k,i}(x) \tag{17}$$

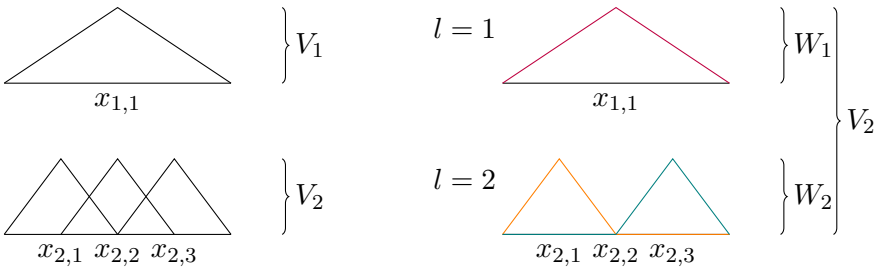


Figure 1: Visualization of the subspace splitting in one dimension for level $l = 2$: the nodal basis functions on the left and the hierarchical basis functions on the right, both for levels $l = 1$ and $l = 2$

where $a_{k,i} \in \mathbb{R}$ are hierarchical coefficient values, $\Phi_{k,i}$ are mutually disjoint hierarchical basis functions spanning W_k, and I_k is defined as

$$I_k = \{i \in \mathbb{N} : 1 \leq i \leq 2^l - 1,\ i \text{ odd}\}.$$

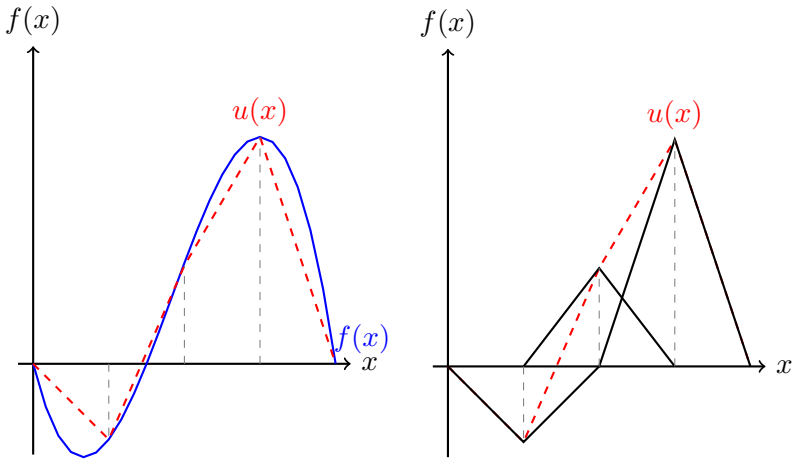


Figure 2: Interpolation using nodal basis functions: approximation $u(x)$ (red) of the function $f(x)$ (blue) by using nodal basis functions in the left plot, the used nodal basis functions (black) in the right plot where one can see that the coefficient values at the grid points simply are the respective function values

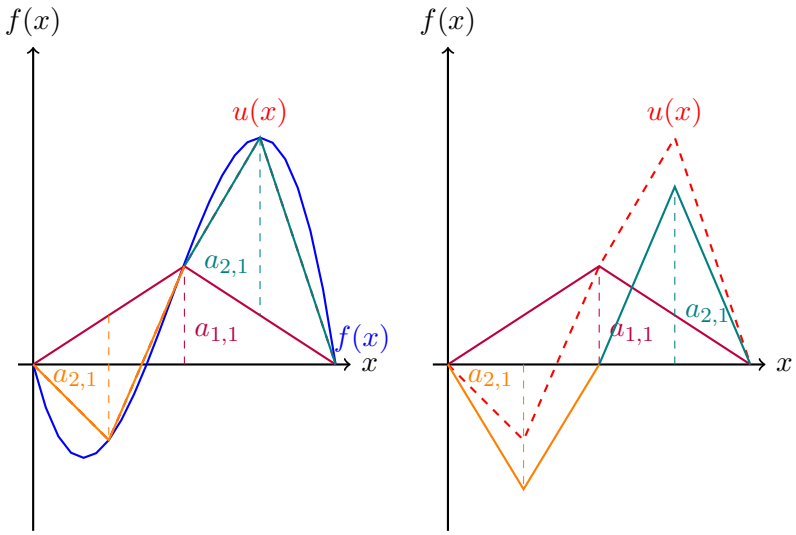


Figure 3: Interpolation using hierarchical basis functions: approximation $u(x)$ to $f(x)$ using hierarchical basis functions with coefficient values $a_{l,i}$ (dashed lines) in the left plot where one starts with the purple basis function for level $l = 1$ (purple) and adds the basis function for level $l = 2$ (orange, teal) on top of that to improve the approximation, in the right plot the used basis functions themselves are visualized, i.e. not as additions on the basis functions of the lower levels

See Figures 2 and 3 for a visualization of interpolation with nodal and hierarchical basis functions respectively.

3.1.2 Multi-dimensional basis functions

By a tensor product construction, we can obtain a multi-dimensional basis on the d-dimensional unit cube $\bar{\Omega} := [0,1]^d$ from the one-dimensional hierarchical basis. To indicate the level in a multivariate sense, let us introduce the multi-index $\mathbf{l} = (l_1, \ldots, l_d) \in \mathbb{N}^d$. Note that we are indicating the multi-index property by a bold formatting. We then consider the set of d-dimensional rectangular grids $\Omega_{\mathbf{l}}$ with mesh size

$$\mathbf{h_l} := (h_{l_1}, \ldots, h_{l_d}) := 2^{-\mathbf{l}}.$$

Thus, we obtain a grid with equidistant points with respect to the individual coordinate directions where the mesh size can vary for the different dimensions. Hence, for $\Omega_{\mathbf{l}}$, we have the grid points

$$x_{\mathbf{l},\mathbf{i}} := (x_{l_1,i_1}, \ldots, x_{l_d,i_d}),\ \mathbf{1} \leq \mathbf{i} \leq 2^{\mathbf{l}} - \mathbf{1}.$$

With each individual grid point $x_{\mathbf{l},\mathbf{i}}$, we associate a *piecewise d-linear nodal basis function*

$$\Phi_{\mathbf{l},\mathbf{i}} := \prod_{j=1}^{d} \Phi_{l_j,i_j}(x_j),$$

which is the product of the one-dimensional basis functions and has support of size $2h_{\mathbf{l}}$. Using these basis functions, we can define the *d-dimensional nodal function spaces*

$$V_{\mathbf{l}} := \operatorname{span}\{\Phi_{\mathbf{l},\mathbf{i}} : \mathbf{1} \leq \mathbf{i} \leq 2^{\mathbf{l}} - \mathbf{1}\}$$

which are zero on the boundary $\partial\Omega$ and consist of piecewise d-linear functions, and the *d-dimensional hierarchical increment spaces*

$$W_{\mathbf{l}} := \operatorname{span}\{\Phi_{\mathbf{l},\mathbf{i}} :\ \mathbf{i} \in \mathbb{N}^d :\ \mathbf{1} \leq \mathbf{i} \leq 2^{\mathbf{l}} - \mathbf{1},\ i_j \text{ odd } \forall\ \mathbf{1} \leq j \leq d\ \}.$$

Like in the one-dimensional setting, we have the following relationship between the two spaces. First, it holds

$$V_{\mathbf{l}} = \bigoplus_{\mathbf{k} \leq \mathbf{l}} W_{\mathbf{k}}$$

and second, any function $u \in V_{\mathbf{l}}$ can be uniquely represented as

$$u_{\mathbf{l}}(\mathbf{x}) := \sum_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} \sum_{\mathbf{i} \in \mathbf{I_k}} a_{\mathbf{k},\mathbf{i}} \Phi_{\mathbf{k},\mathbf{i}}(\mathbf{x}) \tag{18}$$

where $a_{\mathbf{k},\mathbf{i}}$ are hierarchical coefficient values, $\Phi_{\mathbf{k},\mathbf{i}}$ are mutually disjoint hierarchical multi-dimensional basis functions spanning $W_{\mathbf{k}}$, and $\mathbf{I_k}$ is defined as

$$\mathbf{I_k} := \{\mathbf{i} \in \mathbb{N}^d :\ \mathbf{1} \leq \mathbf{i} \leq 2^{\mathbf{k}} - \mathbf{1},\ i_j \text{ odd } \forall\ 1 \leq j \leq d\}.$$

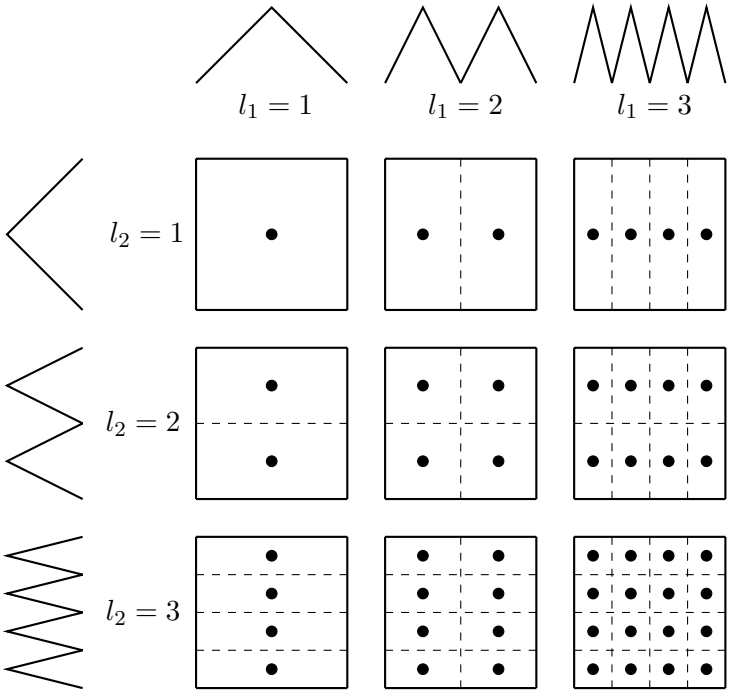


Figure 4: Basis functions $\Phi_{\mathbf{l},\mathbf{i}}$ of the hierarchical subspaces $W_{\mathbf{l}}$ of the space $V_{\mathbf{3}} = V_{3,3}$ with their respective disjoint supports

3.1.3 Regular sparse grids

To understand the theory for sparse grids and especially the convergence results, we should recall the Sobolev space with bounded mixed derivatives

$$H_2^{\text{mix}}(\bar{\Omega}) := \{u : \bar{\Omega} \to \mathbb{R} : D^{\boldsymbol{\alpha}} u \in L_2(\Omega),\ |\boldsymbol{\alpha}|_\infty \leq 2,\ u|_{\partial\Omega} = 0\}$$

with $D^{\boldsymbol{\alpha}} u := \frac{\partial^{|\boldsymbol{\alpha}|_1} u}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}}$.

Considering functions $u \in H_2^{\text{mix}}(\bar{\Omega})$, the hierarchical coefficients $a_{\mathbf{l},\mathbf{i}}$ decay as

$$|a_{\mathbf{l},\mathbf{i}}| = \mathcal{O}(2^{-2|\mathbf{l}|_1})$$

and the number of degrees of freedom of the subspaces $W_{\mathbf{l}}$ is given by

$$|W_{\mathbf{l}}| = \mathcal{O}(2^{|\mathbf{l}|_1}).$$

Instead of the *full grid spaces*

$$V_n := V_{(n,\ldots,n)} = \bigoplus_{|\mathbf{l}|_\infty \leq n} W_{\mathbf{l}}$$

the idea is now to use *sparse grid spaces* $\hat{V}_n$ of level n defined by

$$\hat{V}_n := \bigoplus_{|\mathbf{l}|_1 \leq n+d-1} W_{\mathbf{l}},$$

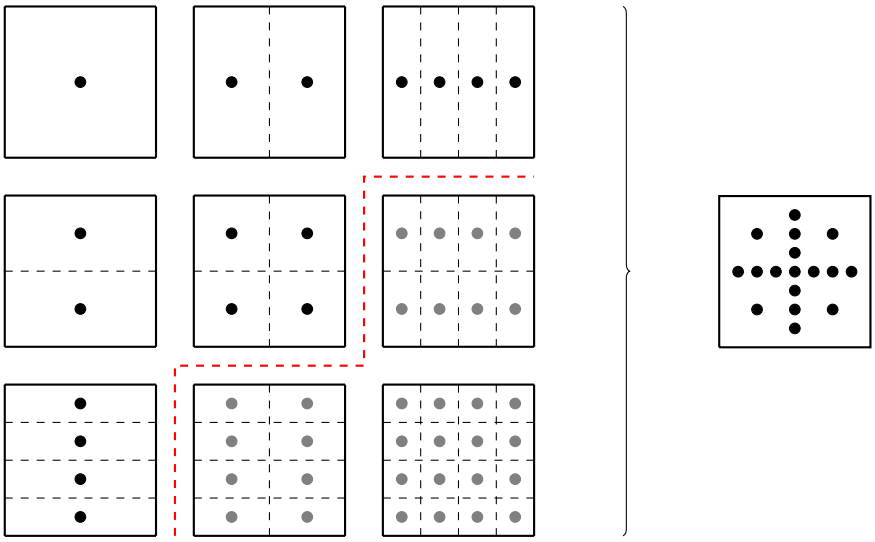


Figure 5: The triangular scheme instead of the rectangular one yields the sparse grid on the right instead of a full grid that would be the set of all points on the left (for the same level)

see Figure 5 for a visualization. This allows us to reduce the number of degrees of freedom. In particular, for sparse grids we have

$$|\hat{V}_n| = \sum_{i=0}^{n-1} 2^i \cdot \binom{d-1+i}{d-1} = \mathcal{O}(h_n^{-1} \cdot |\log_2 h_n|^{d-1})$$

degrees of freedom and thus it is of order $\mathcal{O}(2^n n^{d-1})$, whereas in the full grid case it is of order $\mathcal{O}(2^{nd})$. Despite the big difference in the number of used grid points it is possible to achieve a good accuracy by using sparse grids. For functions in $H_2^{\text{mix}}(\bar{\Omega})$, the approximation error on sparse grids measured in the L_p-norms is given by

$$||u - \hat{u}_n||_p = \mathcal{O}(h_n^2 \cdot n^{d-1}),$$

whereas for full grids it is

$$||u - \hat{u}_n||_p = \mathcal{O}(h_n^2).$$

We can thus overcome the curse of dimensionality to some extent.

3.1.4 Non-zero boundaries

In the previous subsection we have just considered functions that are zero on the domain's boundary $\partial\Omega$. If we want to allow non-zero boundary values, the usual approach is to introduce additional nodes on the boundary. This can be done by adding an extra level $l = 0$ in the construction of sparse grids. On this level we use two overlapping one-dimensional basis functions $\Phi_{0,0}$ and $\Phi_{0,1}$ with indices $i = 0$ and $i = 1$. Doing this we can obtain a modified set of subspaces

$\tilde{W}_{\mathbf{l}}$ by the same tensor product construction explained before. Notice that we then have the associated modified non-zero boundary sparse grid space defined by

$$V_n^{0B(1)} := \bigoplus_{|\mathbf{l}|_1 \leq n+d-1} \tilde{W}_{\mathbf{l}},$$

where level 0 boundary functions are included. With this approach, for the same value of n, we obtain the same interior nodes as before. Note further that in two dimensions there are already twice as many grid points on the boundary as there are on the main axis. Hence, especially in higher dimensions, almost all grid points lie on the boundary. An approach to mitigate this effect is to collate the basis functions of level 0 and level 1. This leads to the same number of grid points on the main axis as on the boundary. Thus, by not spending another level just for the boundary points one can reduce the portion of points that lie on the boundary. One still has to deal with many more grid points than in the case without boundary points. Therefore, the importance of accuracy on and near the boundary should be analyzed since one can for example also use extrapolation towards the boundary instead of introducing new points.

Notice that we use boundary points in our implementation since we implement boundary constraints. Further, these often lead to high gradients close to the boundary.

3.1.5 Adaptive sparse grids

There are cases in which we want to use an adaptive sparse grid. For example, this is the case if the function does not fulfill the required smoothness conditions or the function is flat in most parts of the domain but has some steep regions in other parts. The idea is to add new points to the sparse grid if it is likely that they increase the obtained accuracy enough. This is called *adaptive refinement*. We need a criterion which tells us where to add new points. Most often one uses a local error estimation based on the hierarchical surplus (coefficient). If one finds such a point, one can add its child nodes (in the hierarchical structure). Notice further that one should ensure that all missing parent nodes of the added nodes are in the grid to be consistent with the hierarchical structure.

Note that one has to take care of the possibility that the adaption focuses on just one feature and ends up there with a high level without a good approximation in other locations. However, for high dimensional adaption we cannot just add a lot of children and thus one has to make sure the adaption is well thought out.

In case that additional knowledge about the problem is available, a criterion suited to the problem can be used. Often dimension adaptivity, which leads to anisotropic sparse grids, can be suitable. Here the idea is to have a higher sparse grid level in some dimensions than in others. This makes sense if these dimensions have more influence on the solution or if the function is less smooth in these dimensions than in the others.

We explain the algorithmic details and our refinement strategies in Section 7.2.

3.2 Finite difference schemes on sparse grids

We present two different approaches to implement finite differences on sparse grids. The first one, which was introduced in [Sch98], is based on dimensional splitting and just uses the sparse grid points. The second one, proposed in upcoming work [Ahn18], introduces new points and interpolates on these points. Note that we use the latter approach for our implementation and just present the former to give the reader an insight into the original approach and existing theoretical results for finite difference operators on sparse grids.

3.2.1 Finite difference schemes on sparse grids based on dimensional splitting

In this subsection we explain finite differences on sparse grids which were first introduced in [Sch98]. Notice that this reference contains consistency proofs and presents convergence results for low levels by investigating matrix properties and using specific computations in Matlab. For explanations in English, see [Gri98] and [GS99] and [Zum00]. Note that we follow [Sch98] and [Gri98] for our presentation but do not use this approach in our implementation.

To explain the construction of finite difference operators that are based on dimensional splitting combined with a nodal to hierarchical basis transformation and its respective back-transform, we now present some necessary notions.

Prerequisites Let us first look into some properties of the introduced hierarchical subspaces $W_{\mathbf{l}}$.

The computation of the hierarchical coefficients $a_{\mathbf{l},\mathbf{i}}$ introduced in Section 3.1.2, is done via

$$a_{\mathbf{l},\mathbf{i}} = \Big(\prod_{j=1}^{d} H_{x_{l_j,i_j},h_j} \Big) u(\mathbf{x}) =: H_{x_{\mathbf{l},\mathbf{i},\mathbf{h}}} u(\mathbf{x}) \tag{19}$$

with

$$H_{x_{\mathbf{l},\mathbf{i}},\mathbf{h}} : V_{\mathbf{l}} \to \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}}.$$

The operators $H_{x_{l_j,i_j},h_j} : V_{l_j} \to \bigoplus_{k_j=1}^{l_j} W_{k_j}$ in (19) in one dimension are computed via the stencil

$$H_{x_{l_j,i_j},h_j} = \Big[-\frac{1}{2} \quad 1 \quad \frac{1}{2} \Big]_{x_{l_j,i_j},h_j},$$

i.e.

$$H_{x_{l_j,i_j},h_j} \circ u(x_{l_j,i_j}) = u(x_{l_j,i_j}) - \frac{u(x_{l_j,i_j} - h_j) + u(x_{l_j,i_j} + h_j)}{2}$$

with $1 \leq i_j \leq 2^{l_j} - 1$ in both notations. For d dimensions, this is defined by

$$\begin{aligned} H_{x_{l_j,i_j},h_j} \circ u(x_{\mathbf{l},\mathbf{i}}) =& u(x_{l_1,i_1}, \ldots, x_{l_j,i_j}, \ldots, x_{l_d,i_d}) \\ & - \frac{u(x_{l_1,i_1}, \ldots, x_{l_j,i_j} - h_j, \ldots, x_{l_d,i_d})}{2} \\ & - \frac{u(x_{l_1,i_1}, \ldots, x_{l_j,i_j} + h_j, \ldots, x_{l_d,i_d})}{2}. \end{aligned}$$

Note that this operator yields the identity for boundary points.

We point out that $H_{x_{\mathbf{l},\mathbf{i}},\mathbf{h}}$ is a transformation from piecewise d-linear nodal basis to piecewise d-linear hierarchical basis. Let us turn to the definition of these operators.

Definition 3.2.1 (Hierarchization and Dehierarchization)**.** Let us introduce the operator $\mathbf{H}^j$ for $1 \leq j \leq d$ that denotes the application of a one-dimensional hierarchization with respect to dimension j on all grid points. Note that this operator is defined in d dimensions. Formally, it is denoted by

$$(\mathbf{H}^j \circ u)(x_{\mathbf{l},\mathbf{i}}) = H_{x_{l_j,i_j},h_j} \circ u(x_{\mathbf{l},\mathbf{i}})$$

with $H_{x_{l_j,i_j},h_j}$ being the vector with hierarchical coefficients in dimension j and nodal coefficients in all other dimensions, i.e.

$$\begin{aligned} \mathbf{H}^j : & \Big(\bigotimes_{s=1}^{j-1} X_{l_s} \Big) \otimes \Big(V_{l_j} \Big) \otimes \Big(\bigotimes_{s=j+1}^{d} X_{l_s} \Big) \to \\ & \Big(\bigotimes_{s=1}^{j-1} X_{l_s} \Big) \otimes \Big(\bigoplus_{t_j=1}^{l_j} W_{t_j} \Big) \otimes \Big(\bigotimes_{s=j+1}^{d} X_{l_s} \Big) \end{aligned}$$

with $X_{l_s} \in \{V_{l_s}, \bigoplus_{t_s=1}^{l_s} W_{t_s}\}$.

The d-dimensional *hierarchization operator* $\mathbf{H}$ that performs the basis transformation in all coordinate directions is defined as

$$\mathbf{H} = \prod_{j=1}^{d} \mathbf{H}^j : \bigotimes_{s=1}^{d} V_{l_s} \to \bigotimes_{s=1}^{d} \Big(\bigoplus_{t_s=1}^{l_s} W_{t_s} \Big).$$

We additionally introduce the operator

$$\mathbf{H}_k = \prod_{j=1,\ j \neq k}^{d} \mathbf{H}^j$$

subject to

$$\begin{aligned} \mathbf{H}_k : & \Big(\bigotimes_{s=1}^{k-1} V_{l_s} \Big) \otimes \Big(\bigoplus_{s_k=1}^{l_k} X_{s_k} \Big) \otimes \Big(\bigotimes_{s=k+1}^{d} V_{l_s} \Big) \to \\ & \Big(\bigotimes_{s=1}^{k-1} \Big(\bigoplus_{t_s=1}^{l_s} W_{t_s} \Big) \Big) \otimes \Big(\bigoplus_{s_k=1}^{l_k} X_{s_k} \Big) \otimes \Big(\bigotimes_{s=k+1}^{d} \Big(\bigoplus_{t_s=1}^{l_s} W_{t_s} \Big) \Big) \end{aligned}$$

which performs the hierarchization in all directions but dimension k.

We further introduce the respective inverse operators $\mathbf{E} = \mathbf{H}^{-1}$, $\mathbf{E}^j = (\mathbf{H}^j)^{-1}$ and $\mathbf{E}_k = (\mathbf{H}_k)^{-1}$ and call them *dehierarchization operators.*

Finite differences based on dimensional splitting We introduce finite difference operators which use the neighbor grid points of the respective grid points in the respective dimensions, i.e. the closest grid points in the respective dimensions. For the approximation of the first derivative on regular sparse grids, we use the backward difference stencil and the forward difference stencil given by

$$\frac{1}{2^{-l_{max_j}}}[-1 \quad 1 \quad 0]_{x_{l_j,i_j},l_{max_j}} \quad \text{and} \quad \frac{1}{2^{-l_{max_j}}}[0 \quad -1 \quad 1]_{x_{l_j,i_j},l_{max_j}} \tag{20}$$

respectively and for the approximation of the second derivative, we use the stencil given by

$$\frac{1}{2^{-2\cdot l_{max_j}}}[1 \quad -2 \quad 1]_{x_{l_j,i_j},l_{max_j}} \tag{21}$$

where $l_{max_j} = n+d-1-\sum_{\tilde{j}=1,\tilde{j}\neq j}^{d} l_{\tilde{j}}$. For each interior grid point, these stencils belong to an equidistant grid in the j-th coordinate direction with local mesh size $2^{-l_{max_j}}$. If adaptive refinement is used, the grid points in the different dimensions are no longer equidistant, i.e. the distance is no longer defined by l_{max_j} but the stencil is still chosen such that the closest neighbors in the respective dimensions are used. Then the entries are the coefficients known from non-uniform grids.

Finite difference operators on sparse grids are always constructed with the same scheme regardless of the approximated derivative. Additionally, every operator can be denoted with both a nodal and a hierarchical basis representation. Due to the similarity we just state it for the nodal notation and refer to [Sch98] for the hierarchical one.

The sparse grid finite difference operators are a composition of three partial operators,

- $\mathbf{H}_j$, the basis transformation from nodal to hierarchical basis in all dimensions but dimension j in which we aim to use the finite difference stencil,
- application of a finite difference stencil in dimension j with mesh size given as the local step size to next neighbor grid point in dimension j, e.g. (20) or (21) for a regular sparse grid,
- $\mathbf{E}_j$, the basis transformation from hierarchical to nodal basis in all dimensions but dimension j.

This procedure is visualized in Figure 6. Formally, we can denote the *sparse grid forward difference operator* $\mathbf{D}_j^{S,F}$ by

$$\mathbf{D}_j^{S,F} := \mathbf{E}_j \circ \mathbf{D}_j^F \circ \mathbf{H}_j,$$

the *sparse grid backward difference operator* $\mathbf{D}_j^{S,B}$ by

$$\mathbf{D}_j^{S,B} := \mathbf{E}_j \circ \mathbf{D}_j^B \circ \mathbf{H}_j$$

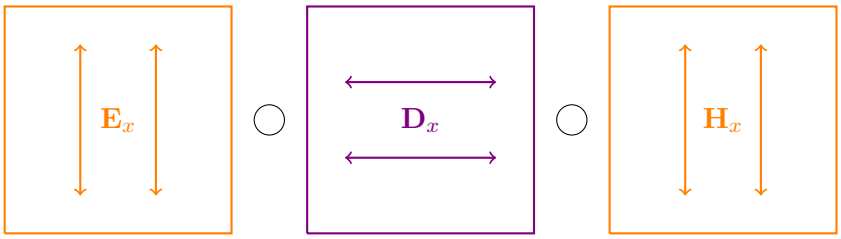


Figure 6: Sparse grid operator for the forward difference in x-dimension as a composition of three partial operators

and the *sparse grid second derivative operator* $\mathbf{D}_{jj}^S$ by

$$\mathbf{D}_{jj}^S := \mathbf{E}_j \circ \mathbf{D}_{jj} \circ \mathbf{H}_j$$

where $\mathbf{D}_j^F$, $\mathbf{D}_j^B$ and $\mathbf{D}_{jj}$ are defined by (20) and (21).

We refer to [Sch98] for an explanation of dimensional splitting for the approximation of d-dimensional operators such as the Laplacian and for the above noted theory regarding consistency for specific types of PDEs.

3.2.2 Finite difference schemes on sparse grids based on interpolation

We now present an approach proposed in a forthcoming paper [Ahn18]. It comes with an easy intuitive implementation. We work closely together with Ahn by testing the current implementation of these sparse grid finite difference operators.

The proposed sparse grid finite difference operators are based on interpolation. Instead of just using the function values on the sparse grid points, one interpolates on nodes that we will refer to as *ghost nodes*. This way we do not have to use specific basis transformations and one can simply take any sparse grid library, such as SG++ [Pfl10], to implement this approach.

To describe the approach, we first define the above noted ghost points. Then we present interpolation operators working on these points. Finally, we introduce the finite difference operators by using these interpolation operators. Note that we explain the handling of finite differences on the boundary at the end of this subsection.

To define ghost nodes, we start by defining the *ghost node step size*.

Definition 3.2.2 (Ghost node step size)**.** We define the *ghost node step size* h_{g_j} in dimension j, $1 \leq j \leq n$, for a grid point $x_{\mathbf{l},\mathbf{i}}$ by $h_{g_j} := 2^{-k_j}$ where k_j denotes the maximal level used in dimension j.

Note that this is half of the size of the smallest support of the basis functions in this dimension. This makes sense since for this step size the local behavior

of the approximation is still captured. For adaptive sparse grids, one could also take a bigger distance in some grid points but due to the linearity of the approximation in this part this does not change the result.

Note that for the different sparse grid operators, we need to interpolate on different points, i.e. for the forward difference we have to add the ghost node step size in the respective dimension and for the backward difference we have to subtract it in the respective dimension. We refer to them as *forward difference ghost nodes* and *backward difference ghost nodes*. Notice that for the second derivative finite difference, we can use the first derivative operators. Further, other difference operators are possible but we restrict ourselves to these operators for a simplified presentation. We explain how to use these finite difference operators on boundary points later.

Definition 3.2.3 (Ghost node)**.** For a grid point $x_{\mathbf{l},\mathbf{i}}$ in which we aim to compute the finite differences in dimension $j,\ 1 \leq j \leq d$, we define the corresponding *forward difference ghost node* by

$$g_{\mathbf{l},\mathbf{i}}^{F,j} := (x_{l_1,i_1}, \ldots, x_{l_j,i_j} + h_{g_j}, \ldots, x_{l_d,i_d})$$

and similarly for the backward difference we define the corresponding *backward difference ghost node* by

$$g_{\mathbf{l},\mathbf{i}}^{B,j} := (x_{l_1,i_1}, \ldots, x_{l_j,i_j} - h_{g_j}, \ldots, x_{l_d,i_d}).$$

We can now define the finite difference operators on sparse grids that are based on interpolation.

Definition 3.2.4 (Interpolation based sparse grid finite difference operator)**.** Let us denote the interpolation operator on the sparse grid by $\mathbf{I}_s : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}}$. We define the interpolation operator for the by h_{g_j} shifted sparse grid, i.e. the ghost node grid, for the forward difference in dimension $j,\ 1 \leq j \leq d$ by $\mathbf{I}_{h_{g_j}}^F : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}}$ and for the backward difference in dimension $j,\ 1 \leq j \leq d$ by $\mathbf{I}_{h_{g_j}}^B : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}}$. For a given grid and the desired difference operator, we can simply compute all respective ghost nodes and interpolate on these. We define the *sparse grid forward difference operator* $\tilde{\mathbf{D}}_j^{S,F}$ by

$$\tilde{\mathbf{D}}_j^{S,F} := \mathbf{I}_{h_{g_j}}^F - \mathbf{I}_s : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}}$$

and the *sparse grid backward difference operator* $\tilde{\mathbf{D}}_j^{S,B}$ by

$$\tilde{\mathbf{D}}_j^{S,B} := \mathbf{I}_s - \mathbf{I}_{h_{g_j}}^B : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}}.$$

We point out that one does not have to use interpolation for the boundary points in the respective dimension (see the right picture in Figure 7 with red points on top of sparse grid points) since the function values for these grid

Figure 7: Visualization of the ghost points for the forward difference in x-dimension: ghost node (red) that is used for the sparse grid forward finite differences in x-dimension in the green grid point in the left graphic, and on the right all forward difference ghost nodes that are used for the sparse grid are drawn in red

points are already known. For higher levels though, these points only make a small portion of the overall needed ghost points and thus one does not gain a lot by excluding these points from the interpolation operation.

Notice that the interpolation operators work on hierarchical values. Note further that we can similarly define other finite difference operators by interpolating on the required points. Let us turn to the boundary handling. Since the forward difference is not defined on the upper boundary, we use the backward difference and thus also the backward difference ghost nodes here. Similarly, we use the forward difference on the lower boundary since the backward difference is not defined here.

For the second derivative difference operator, we can also use the above approach by interpolating to the respective points. If we need both the first and the second derivative, there are two approaches to avoid recomputations. First, one can use the interpolated values that one used for the first derivative finite differences also for the second derivative finite differences. Second, one can use the computed first derivative operators to compute the second derivative one by using the following relationship between the first and the second derivative operators on sparse grids given by

$$\tilde{\mathbf{D}}_{jj}^{S} = \tilde{\mathbf{D}}_{j}^{S,B} \circ \tilde{\mathbf{D}}_{j}^{S,F} = \tilde{\mathbf{D}}_{j}^{S,F} \circ \tilde{\mathbf{D}}_{j}^{S,B}$$

which is a well known identity for the full grid operators is also pointed out in [Sch98]. Notice that it is extremely inefficient to do the interpolation for both the first and the second derivative operator respectively and one thus should avoid to compute both on their own.

3.2.3 Example of finite difference operator construction

Note that the operators are linear and can thus be represented by matrices. We show the construction for the forward difference in y direction in a two-dimensional sparse grid with sparse grid level $l = 1$. Notice that the version presented in Section 3.2.1 is working on nodal basis functions, whereas we presented the interpolation based version in Section 3.2.2 as operators working on hierarchical basis coefficients. We thus apply the nodal to hierarchical

basis transformation H as first operation in the interpolation based version to compare the arising discretization matrices of both sparse grid finite difference approaches.

Consider sparse grid points $\mathbf{G}$ and basis transformation $\mathbf{H}$ given by

$$\mathbf{G} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0.50 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0.50 & 1 \\ 0 & 0.50 \\ 1 & 0.50 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & -0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & -0.5 & 1 & 0 & 0 \\ -0.5 & 0 & 0 & -0.5 & 0 & 0 & 1 & 0 \\ 0 & -0.5 & 0 & 0 & -0.5 & 0 & 0 & 1 \end{bmatrix},$$

where the first column of $\mathbf{G}$ shows the x-coordinates and the second column the y-coordinates of the points.

Example of finite difference operators based on basis tranformations We need the basis transformation in all dimensions but the one in which we apply the difference operator, i.e. in all dimensions but dimension y. We further need the finite difference operator $\mathbf{D}_y$ which works on the sparse grid points. These are given by

$$\mathbf{E}_y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{D}_y = \begin{bmatrix} -2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2 \end{bmatrix}$$

and

$$\mathbf{H}_y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & -0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & -0.5 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Notice first that $\mathbf{E}_y = \mathbf{H}_y^{-1}$. For $\mathbf{D}_y$, you can see that for example for the first row we take the grid points at index 1 and 7. This is the case since for the forward difference we take the point itself, i.e. $(0, 0)$, and the next grid point in y-dimension, i.e. $(0, 0.5)$. The matrix entries themselves arise from the distance of the used points.

Example of finite difference operators based on basis tranformations For the version based on interpolation, we get

$$\tilde{\mathbf{D}}_y^{S,F} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 2 \\ -0.5 & -0.5 & -1 & 0.5 & 0.5 & 1 & 1 & 1 \\ -1 & 0 & 0 & 1 & 0 & 0 & -2 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & -2 \\ -0.5 & -0.5 & -1 & 0.5 & 0.5 & 1 & -1 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 & -2 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & -2 \end{bmatrix}$$

which can be computed by subtracting the interpolation operator for the sparse grid and from the interpolation operator for the ghost node grid.

Comparison of the two approaches for an example We can now easily compare the arising operators by a simple computation. Note that for both approaches this yields the same finite difference operator, i.e.

$$\mathbf{D}_y^{S,F} := \mathbf{E}_y \cdot \mathbf{D}_y \cdot \mathbf{H}_y = \tilde{\mathbf{D}}_y^{S,F} \cdot \mathbf{H} = \begin{bmatrix} -2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 2 \\ -0.5 & -0.5 & -2 & -0.5 & -0.5 & 2 & 2 & 2 \\ 0 & 0 & 0 & 2 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2 \\ 0.5 & 0.5 & -2 & 0.5 & 0.5 & 2 & -2 & -2 \\ 0 & 0 & 0 & 2 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2 \end{bmatrix}.$$

We point out that the only two rows in which more than two entries are non-zero are the ones for the grid points with $x = 0.5$. The forward difference operation in y-dimension in all other grid points only requires actual grid points. To compute the finite differences in the points with $x = 0.5$ with the interpolation based version, one interpolates on $(0.5, 0.5)$. Note that all basis functions are used to interpolate on this point.

We further checked that for levels $l = 2, 3$ both approaches yield the same matrix. The theoretical relationship between the two approaches will be explained in upcoming work [Ahn18].

4 Viscosity Solutions and General Convergence Theory

In the beginning of section, we briefly recall the basics of viscosity solutions and the convergence theory developed by [BS90]. We then present the numerical scheme used by [AHL$^+$17] for a simple model on full grids.

4.1 Viscosity solutions

It is known that for PDEs the existence of solutions is not trivial. Note that the weak formulation known from linear PDEs cannot be used for nonlinear PDEs like the HJB equation. Thus, one defines a new type of weak solutions, called *viscosity solutions*, that is testing the solution in a specific way. In the past 30 years there has been a lot of research regarding these solutions and thus there exist several results regarding existence, uniqueness and regularity of solutions. Crandall and Lions introduced the notion of viscosity solutions for first order equations in [CL83]. They generalized the theory to second order PDEs in [LS88]. Also see [CILS92] and [Cra97] which are some of the classical papers. A more recent introduction can be found in [Bar13] or [CP14] and for a simple nontechnical introduction with economic context, see [Mol16c]. In this subsection we mainly follow [Pha09] and [Yu08] which are themselves heavily based on [CILS92].

4.1.1 Important definitions and the notion of viscosity solutions

There are a lot of different types of fully nonlinear PDEs, e.g. Hamilton-Jacobi equations like the eikonal equation, Hamilton-Jacobi-Bellman equations or the Monge-Ampère equation.

Let us present the theory required to understand the kind of solution we have for the above mentioned PDEs. We begin by defining several function properties.

Definition 4.1.1. Consider a function $F : \Omega \times \mathbb{R} \times \mathbb{R}^n \times \mathcal{S}_n \to \mathbb{R}$, where $\mathcal{S}_n$ denotes the space of symmetric $n \times n$ matrices, with arguments x, r, p, X, i.e. $F(x, r, p, X)$. We call F *degenerate elliptic* if it is non-increasing in its matrix argument, i.e. for all $x \in \Omega, r \in \mathbb{R}, p \in \mathbb{R}^n, M, \hat{M} \in \mathcal{S}_n$ it holds

$$M \leq \hat{M} \implies F(x, r, p, \hat{M}) \leq F(x, r, p, M),$$

which means that $\hat{M} - M$ is positive semi-definite. If it is additionally non-decreasing in r, i.e. for all $x \in \Omega, s, r \in \mathbb{R}, p \in \mathbb{R}^n, M, \hat{M} \in \mathcal{S}_n$ it holds

$$s \leq r, M \leq \hat{M} \implies F(x, s, p, \hat{M}) \leq F(x, r, p, M),$$

we call F *proper*.

We should also recall the following important definitions.

Definition 4.1.2. For a locally bounded function $v : \Omega \to \mathbb{R}$, we define the *upper-semicontinuous envelope* v^* and the *lower-semicontinuous envelope* w_* on $\bar{\Omega}$ by

$$v^*(x) := \limsup_{\tilde{x} \to x} v(x),$$
$$v_*(x) := \liminf_{\tilde{x} \to x} v(x).$$

Recall that w is called *upper-semicontinuous (usc)* if

$$v = v^*$$

and similarly, w is called *lower-semicontinuous (lsc)* if

$$v = v_*.$$

We call v *continuous* if it is both lower-semicontinuous and upper-semicontinuous, i.e.

$$v = v^* = v_*$$

Using the above definitions, we can introduce the notion of viscosity solutions. We denote the space of twice continuously differentiable functions by $\mathcal{C}^2$.

Definition 4.1.3. Let Ω be an open domain and let $w : \Omega \to \mathbb{R}$ be locally bounded. Further, let F be a proper fully nonlinear equation with

$$F(x, v(x), Dv(x), D^2v(x)) = 0. \tag{22}$$

Then v is a *viscosity subsolution* of (22) on Ω if it is upper-semicontinuous and for every $\varphi \in \mathcal{C}^2(\Omega)$ and $\hat{x} \in \Omega$ which is a local maximum of $v - \varphi$, it holds

$$F(\hat{x}, v(\hat{x}), D\varphi(\hat{x}), D^2\varphi(\hat{x})) \leq 0,$$

and it is a *viscosity supersolution* of (22) on Ω if it is lower-semicontinuous for every $\varphi \in \mathcal{C}^2(\Omega)$ and $\hat{x} \in \Omega$ which is a local minimum of $v - \varphi$, it holds

$$F(\hat{x}, v(\hat{x}), D\varphi(\hat{x}), D^2\varphi(\hat{x})) \geq 0.$$

We call it a *viscosity solution* on Ω if it is both a viscosity subsolution and a viscosity supersolution of (22).

Notice that there also exists a notion of discontinuous viscosity solutions using the envelopes.

Since only the derivatives $D\varphi$ and $D^2\varphi$ and not the function φ itself is involved in the definition, adding constants to φ does not change these properties. Therefore, the notion of the local maximum of $v - \varphi$ can be interpreted as the graph of φ touching v from above and similarly for the notion of the local minimum, see Figure 8.

For examples which show the utility of this notion, see [Yu08] or some of the references given in the beginning of this subsection.

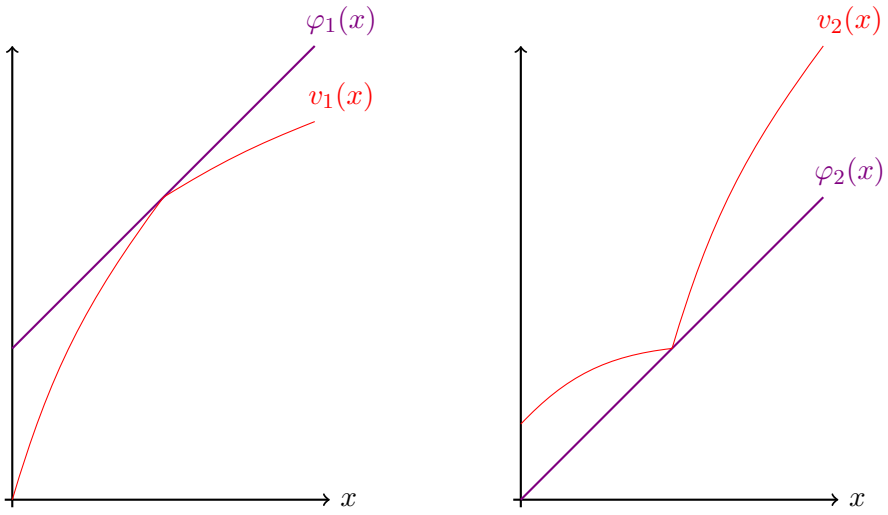


Figure 8: Subsolution φ_1 for a function v_1 on the left and supersolution φ_2 for a function v_2 on right

4.1.2 Uniqueness, existence and regularity of viscosity solutions

The theory for uniqueness, existence and stability is non-trivial, especially if we have boundary inequalities in the viscosity sense. Due to space constraints we do not cover these topics in detail. Instead, we just state some important results, present some ideas and reference the respective literature.

Uniqueness – comparison principles To get a uniqueness result for viscosity solutions, a *comparison principle* which is also called *maximum principle* can be used.

Definition 4.1.4 (Property: Strong comparison principle)**.** If $v_1 \in B(\bar{\Omega})$ is an upper semicontinuous solution of (22) and $v_2 \in B(\bar{\Omega})$ is a lower semicontinuous solution of (22), then $v_1 \leq v_2$ on Ω.

The strong comparison principle allows us to compare a subsolution and a supersolution on the entire domain by just comparing it on the boundary of the domain. This directly implies the uniqueness of the viscosity solution of (22) by a condition on the boundary given by $v_1^* = v_{1*} = g$ on $\partial\Omega$. The reason for this is that by the strong comparison principle we have $v_1^* \leq v_{2*}$ and $v_2^* \leq v_{1*}$ and by definition we also have $v_{1*} \leq v_1^*$ and $v_{2*} \leq v_2^*$. Thus, it holds

$$v_{1*} = v_1^* = v_{2*} = v_2^*$$

and we obtain the uniqueness of a viscosity solution $v_1 = v_2$ on Ω.

The comparison principle can be easily proved by contradiction using first and second order conditions if v_1 and v_2 are smooth. If they are non-smooth though, this is not possible anymore. A key trick, called *doubling the variables*, can be used to still prove the result in this case. For a complete proof in the state constrained case, we refer to Theorem 2.2 of [Son86].

Existence Like in the linear PDE case one is additionally interested in the existence of the solution in the defined weak sense. There is a method called *Perron's method* that allows us to prove existence using the comparison principle. In the Dirichlet setting the proof idea is to show that if a subsolution w is not a solution, then one can modify it to obtain another subsolution $\tilde{w}$ such that $\tilde{w} > w$ in a small neighborhood. For details see Section 9 of [Cra97] or see [CILS92] for the similar statements and the respective proofs.

Regularity For regularity results, we refer to [CC].

4.2 Convergence of general numerical approximation schemes for fully non-linear second order PDEs

In this section we discuss the convergence of numerical approximation schemes to the (unique) viscosity solution of fully non-linear second order PDEs and thus also for the HJB equation. Barles and Souganidis proved that schemes that are monotone, consistent and stable are convergent. Hence, the idea is to construct numerical schemes that fulfill these conditions. Note that we closely follow the classical paper [BS90] for the following presentation of the theory.

Let us first give the framework. We want to approximate (22) with a numerical scheme denoted by S, i.e.

$$S(\rho, x, v^\rho(x), v^\rho) = 0, \tag{23}$$

where $S : \mathbb{R}^+ \times \Omega \times \mathbb{R} \times B(\bar{\Omega}) \to \mathbb{R}$ is locally bounded and $B(\bar{\Omega})$ is the space of bounded functions defined on $\bar{\Omega}$. Further, ρ denotes the discretization parameter, e.g. the mesh size in full grid finite difference schemes. For every $\rho > 0$, we denote the solution of the approximation scheme (23) by v^ρ.

Let us now prove that schemes that are monotone, stable and consistent converge to (22) as long as (22) admits a comparison principle. To do this, we first give the required definitions.

Definition 4.2.1. We call S *monotone* if for $\rho \in \mathbb{R}^+,\ x \in \bar{\Omega},\ r \in \mathbb{R}$ and $v_1,\ v_2 \in B(\bar{\Omega})$ with $v_1 \geq v_2$, we have

$$S(\rho, x, r, v_1) \leq S(\rho, x, r, v_2).$$

Definition 4.2.2. We call S *stable* if for all $\rho > 0$, there exists a solution $v^\rho \in B(\bar{\Omega})$ of (23) such that

$$\sup_{\rho > 0} ||v^\rho||_\infty \leq C$$

where C is a constant independent of ρ.

Definition 4.2.3. We call S *consistent* if for all $x_0 \in B(\bar{\Omega})$ and $\varphi \in C_b^2(\bar{\Omega})$ it holds

$$\limsup_{\rho \to 0, x \to 0, h \to 0} \frac{S(\rho, x, \varphi(x) + h, \varphi + h)}{\rho} \leq F^*(x_0, \varphi(x_0), D\varphi(x_0), D^2\varphi(x_0)),$$

and

$$\limsup_{\rho\to 0, x\to 0, h\to 0} \frac{S(\rho, x, \varphi(x)+h, \varphi+h)}{\rho} \geq F_*(x_0, \varphi(x_0), D\varphi(x_0), D^2\varphi(x_0)),$$

We now turn to the main result proven by Barles and Souganidis.

Theorem 4.2.4. *Suppose that the above defined strong comparison principle holds for* (22)*. If S is a monotone, stable and consistent scheme, then the solution of v^ρ of* (23) *converges locally uniformly to the unique viscosity solution v of* (22) *for $\rho \to 0$.*

Proof. Let us define $\bar{v}, \underline{v} \in B(\bar{\Omega})$ by

$$\bar{v}(x_0) = \limsup_{x\to x_0, \rho\to 0} v^\rho(x) \text{ and } \underline{v}(x_0) = \liminf_{x\to x_0, \rho\to 0} v^\rho(x).$$

We claim that $\bar{v}$ and $\underline{v}$ are respectively subsolutions and supersolutions of (22). Then, according to strong comparison principle, it holds $\bar{v} \leq \underline{v}$ on $\bar{\Omega}$. By the definitions of $\bar{v}$ and $\underline{v}$, we have $\bar{v} \leq \underline{v}$ and thus the equality $v = \bar{v} = \underline{v}$. By the strong comparison principle, we get that v is the unique continuous solution of (22). This gives us the locally uniform convergence of v^ρ to v.

We still need to prove our claim, i.e. that $\bar{v}$ and $\underline{v}$ are respectively subsolutions and supersolutions of (22). Let us just show the proof for $\bar{v}$ since it is analogous for the other case. Hence, let x_0 be a local maximum of $\bar{v} - \varphi$ on $\bar{\Omega}$ for some $\varphi \in \mathcal{C}^2(\bar{\Omega})$. Without loss of generality, we assume that φ is bounded, x_0 is a strict local maximum, $\bar{v}(x_0) = \varphi(x_0)$ and additionally $\varphi \geq 2\sup_\rho ||v^\rho||_\infty$ outside the ball $B_r(x_0)$, where $r > 0$ is such that

$$\bar{v}(x) - \varphi(x) \leq 0 = \bar{u}(x_0) - \varphi(x_0) \text{ in } B(x_0, r).$$

Then there exist sequences $(\rho_n)_n \subset \mathbb{R}^+$ and $(x_n)_n \subset \bar{\Omega}$ such that

$$x_n \text{ is a global maximum point of } v^{\rho_n}(\cdot) - \varphi(\cdot) \tag{24}$$

and

$$\rho_n \to 0,\ x_n \to x_0,\ v^{\rho_n}(x_n) \to \bar{v}(x_0)$$

for $n \to \infty$. Define a new sequence $h_n := v^{\rho_n}(x_n) - \varphi(x_n)$. Notice that we get

$$h_n \to 0 \text{ and } v^{\rho_n}(x_n) \leq \varphi(x_n) + h_n.$$

Due to (24), the definition of v^{ρ_n} and the monotonicity property of S we get

$$S(\rho_n, x_n, \varphi(x_n) + h_n, \varphi + h_n) \leq 0. \tag{25}$$

By passing to the limit in (25) and using the consistency property of S, we obtain

$$\begin{aligned} 0 &\geq \liminf_n \frac{S(\rho, x_n, \varphi(x_n) + h_n, \varphi + h_n)}{\rho_n} \\ &\geq \liminf_{x\to x_0, \rho\to 0, h\to 0} \frac{S(\rho, x, \varphi(x) + h, \varphi + h)}{\rho} \\ &\geq F_*(x_0, \varphi(x_0), D\varphi(x_0), D^2\varphi(x_0))). \end{aligned}$$

This gives us the desired result since $\bar{v}(x_0) = \varphi(x_0)$.

□

Notice that the monotonicity assumption plays the role of the ellipticity property in guaranteeing a comparison principle type property for the scheme.

4.3 Finite difference approach on full grids for a simple model

Let us follow the very recent work of [AHL+17] and give a simple example model to explain how to construct a consistent, stable and monotone finite difference scheme on full grids for solving the HJB equation arising from economic models. You can find a more mathematical description of finite difference schemes for solving the HJB equation that is not targeted to economic models in [ABIL13].

Notice that in our approach we follow exactly the same scheme but instead of using a full grid with standard finite differences we use a sparse grid with the sparse grid finite difference operators introduced in Section 3.2.2. At the end of this subsection, we generalize the matrix notation so that it can easily be extended to the sparse grid setting.

4.3.1 Model

Let us consider the easy one-dimensional deterministic model already presented and explained in Section 2.2. We want to solve

$$\max_{\{c_t\}_{t\geq 0}} \int_0^\infty e^{-\rho t} u(c_t) dt \tag{26}$$

subject to

$$\dot{b}_t = w + r^b b_t - c_t \tag{27}$$

$$b_t \geq \underline{b} \tag{28}$$

where c_t and b_t denote the consumption and the liquid asset at time t respectively and r^b denotes the returns on b. We additionally have wage w and face a borrowing constraint $b_t \geq \underline{b}$. Notice that c_t is our control and b_t reflects the state at time t. Thus, given the state, we want to choose an optimal control and this choice directly reflects in the change of the state.

We get the HJB equation

$$\rho v(b) = \max_c u(c) + v'(b)(w + r^b b - c).$$

4.3.2 Discretization

The finite difference approximation, using J discrete points, is

$$\rho v(b_j) = u(c_j) + v'(b_j)(w + r^b b_j - c_j), \quad c_j = (u')^{-1}(v'(b_j)),\ j = 1, \dots, J$$

where $v(b_j)' = v_j'$ is either the forward or the backward difference approximation. Note that the computation for c arises from the first order condition with respect to c.

In the following, whenever we state an equation for j, this holds true for $j = 1, \ldots, J$. This is also the case for other states added later on.

In the formulation of Barles-Souganidis we get

$$0 = F(b, v(b), v'(b), v''(b)) \tag{29}$$

with $F(b, v(b), v'(b), v''(b)) = \rho v(b) - \max_c u(c) - v'(b)(w + r^b b - c))$. A corresponding finite difference scheme is of the form

$$0 = S(\Delta b, b, v_j, (v_{j-1}, v_{j+1})) \tag{30}$$

with $S(\Delta b, b, v_j, (v_{j-1}, v_{j+1})) = \rho v_j - u(c_j) - v'(b_j)(w + r^b b_j - c_j))$.

Recall that we have a convergent scheme if we fulfill the three conditions monotonicity, consistency and stability. Notice that for finite difference schemes on full grids there is an equivalent local formulation that is just based on the respective neighbors of the points. Hence, for our example model, we want our finite difference scheme to be

- monotone, i.e. non-increasing in both v_{j-1} and v_{j+1},
- consistent, i.e. for every smooth function v with bounded derivatives, it holds
$$S(\Delta b, b_j, v_j, (v_{j-1}, v_{j+1})) \to F(v(b), v'(b), v''(b))$$
for $\Delta b \to 0$, $b_j \to b$ and
- stable, i.e. for all $\Delta b > 0$, it has a solution v_j, $j = 1, \ldots, J$ which is uniformly bounded independently of Δb.

The main issue of constructing such a scheme is the monotonicity condition. We use an upwind scheme that gives us a rule for the choice of the finite difference: we use the forward difference when the drift of the state variable (here: savings $s_j = w + r^b b_j - c_j$) is positive and the backward difference if it is negative. To formalize this let us, for a function v, denote the forward difference by v^F and the backward difference by v^B. For the drift the superscripts indicate which finite difference operation is used on the value function. Let us define

$$s_j^F = w + r^b b_j - c_j^F \text{ and } s_j^B = w + r^b b_j - c_j^B$$

with $c_j^F = (u')^{-1}(v_j^F)$ and $c_j^B = (u')^{-1}(v_j^B)$.
Notice that since v is concave, it holds $v_j^F \leq v_j^B$ and thus directly $s_j^F \leq s_j^B$. If $s_j^F \leq 0 \leq s_j^B$, we set $s_j = 0$ which leads to $v'(b_j) = u'(w + r^b b_j)$ by simple algebra, i.e. we are in the steady state. Note that we can thus approximate the derivative v_j' by

$$v_j' = v_j^F \mathbb{1}_{\{s_j^F > 0\}} + v_j^B \mathbb{1}_{\{s_j^B < 0\}} + \bar{v}_j \mathbb{1}_{\{s_j^F \leq 0 \leq s_j^B\}}$$

with $\bar{v}_j = u'(w + r^b b_j)$. This construction obviously yields monotonicity but there is also an intuition for this: if the continuation value at v_{j-1} or v_{j+1} is

higher we are at least as well off.

Denoting $\max\{x, 0\}$ as x^+ and $\min\{x, 0\}$ as x^- for any $x \in \mathbb{R}$ we end up with

$$\rho v_j = u(c_j) + \frac{v_{j+1} - v_j}{\Delta b}(s_j^F)^+ + \frac{v_j - v_{j-1}}{\Delta b}(s_j^B)^-.$$

We should mention that there is a circular element to the above equation in the sense that v_j' is also used to compute c_j. Due to the well known envelope condition this does not change the monotonicity, see [AHL$^+$17]. Furthermore, it is possible to construct other monotone schemes but this one is perfectly suited to implement borrowing constraints which we turn to now.

4.3.3 Numerical approach for handling the borrowing constraint

At the lower end of the state space, i.e. at b_1, we aim to impose the borrowing constraint $b_t \geq \underline{b}$. We have two main ingredients:

- the first order condition still holds at the boundary: $u'(c(\underline{b})) = v'(\underline{b})$
- to respect the constraint we need: $s(\underline{b}) = w + r^b\underline{b} - c(\underline{b}) \geq 0$.

Since u is strictly monotonically increasing and concave we get

$$v'(\underline{b}) \geq u'(w + r\underline{b})$$

by a simple combination of the above points. We can ensure that the borrowing constraint is never violated by setting

$$v_1^B \equiv \frac{v_1 - v_0}{\Delta b} = u'(w + rb_1).$$

Hence, the boundary condition is only imposed if $s_1 < 0$ and thus only for the backward difference.
Let us turn to the upper end of the state space, i.e. b_J. One should make sure that the backward difference is used at the upper bound. If b_J is large enough, savings are always negative and thus $s_J^+ = 0$. Therefore, the forward difference is never used at the upper bound so that no boundary condition has to be imposed. In practice, it can be appropriate to use an artificial state constraint $a \leq a_{max}$ and treat it like the borrowing constraint, just for the upper bound.
We further use the concept of soft borrowing constraints in order to avoid spikes that are counter-factual to empirical observations. We refer to [AHL$^+$17] for a description.

4.3.4 Numerical approach for overcoming the non-linearity

The HJB equation is highly nonlinear due to the maximum operator. This is why we use an iterative scheme to solve this equation, see e.g. *policy iteration* explained in [FF13]. Its general idea is to linearize the HJB equation by omitting the maximum operator and using an iteration instead of searching for the maximum.

In our setting, we end up with the following explicit method for the discretized HJB equation

$$\frac{v_j^{n+1} - v_j^n}{\Delta} + \rho v_j^n = u(c_j^n) + \frac{v_{j+1}^n - v_j^n}{\Delta b}(s_j^{F,n})^+ + \frac{v_j^n - v_{j-1}^n}{\Delta b}(s_j^{B,n})^- \tag{31}$$

where n denotes the iteration step.

Let us describe the algorithmic approach by presenting Algorithm 1. Notice that all computations are done for all grid points even though we omit this in the presentation. We have a grid on which we compute the policy functions for every grid point in every iteration. We start with an initial guess for the policy functions by simply setting these to zero. For this initial guess, we can directly compute a value function approximation, see row 2. In row 5, we can use the value function approximation at every point to compute the respective optimal control. With the new optimal control, we can then compute an improved approximation to the value function in row 6. Doing this, we stop when the value function approximation of consecutive iterations does not differ much, see row 7.

Algorithm 1 Iteration to solve the HJB equation

Data: convergence parameter ε
Result: solution v_j of HJB equation

1: **Initialization:**
2: Set initial guess for value function v_j^0 ▷ staying "put": by setting policy functions equal to zero one gets an initial guess that is easily computed and behaves well
3: **Iterative part:**
4: **for** $n = 0, 1, \ldots$ **do**
5: Compute optimal policy c_j^n ▷ use finite differences of v_j^n
6: Solve (31) for v_j^{n+1} ▷ linearized HJB equation
7: **if** $max_j|v_j^n - v_j^{n+1}| < \varepsilon$ **then** ▷ stop if iterates are close
8: $v_j \leftarrow v_j^{n+1}$
9: STOP
10: **end if**
11: **end for**

Note that we have fixed grid points and always optimize the controls on these points. This is in contrast to value iteration where one iterates on the value function and determines the optimal state.

There are two main interpretations for (31). The first one is the use of the Newton method for solving the system of non-linear equations (30). The

other one is that the iterative scheme is equivalent to solving the HJB equation backward in time. Hence, (31) basically sets $v(b,T)$ as initial guess and solves

$$\rho v(b,t) = \max_c u(c) + \partial_k v(b,t)(w + r^b b - c) + \partial_t v(b,t)$$

backward in time, i.e. $v(b) = \lim_{t\to-\infty} v(b,t)$.

Note that (31) denotes an explicit scheme, i.e. it is possible to compute v^{n+1} given v^n by a simple computation. Contrarily, the implicit version is given by

$$\frac{v_j^{n+1} - v_j^n}{\Delta} + \rho v_j^n = u(c_j^n) + \frac{v_{j+1}^{n+1} - v_j^{n+1}}{\Delta b}(s_j^{F,n})^+ + \frac{v_j^{n+1} - v_{j-1}^{n+1}}{\Delta b}(s_j^{B,n})^- \quad (32)$$

where the value functions on the right hand side are of step $n+1$. Thus, one has to solve a linear system to solve (32) for every grid point. We note that this system is not fully implicit since the consumption c of step n is used in the computation (also for the drifts $s_j^{F,n}$ and $s_j^{B,n}$). Using a Newton method, one could also solve the fully implicit method.

Explicit schemes are easier to understand but they are only stable if they satisfy the so-called *CFL condition* which gives an upper bound on the step size. Contrarily, implicit schemes are unconditionally stable. We restrict ourselves to implicit schemes, such as (32), in our implementations and the following presentation since for explicit schemes one has to use an extremely small time step size and thus a lot of iterations are required. For more details regarding implicit and explicit approaches, see for example [PR55].

4.3.5 Numerical approach for stochastic settings

For the heterogeneous agent model (11) - (12) featuring a diffusion process, we add another grid dimension for the productivity state z using $k = 1, \dots, K$. Note that this leads to a full grid of $J \times K$ points. We discretize the HJB equation (13) which arises for this model by

$$\begin{aligned}\frac{v_{j,k}^{n+1} - v_{j,k}^n}{\Delta} + \rho v_{j,k}^n =& u(c_{j,k}^n) + \frac{v_{j+1,k}^{n+1} - v_{j,k}^{n+1}}{\Delta b}(s_{j,k}^{F,n})^+ + \frac{v_{j,k}^{n+1} - v_{j-1,k}^{n+1}}{\Delta b}(s_{j,k}^{B,n})^- \\ &+ \frac{v_{j,k+1}^{n+1} - v_{j,k}^{n+1}}{\Delta z}(\mu_k)^+ + \frac{v_{j,k}^{n+1} - v_{j,k-1}^{n+1}}{\Delta z}(\mu_k)^- \\ &+ \frac{\sigma_k^2}{2}\frac{v_{j,k+1}^{n+1} - 2v_{j,k}^{n+1} + v_{j,k-1}^{n+1}}{(\Delta z)^2}.\end{aligned} \quad (33)$$

Note that we can easily implement reflecting boundary conditions by using

$$\partial_z v_{j,1} = \frac{v_{j,1} - v_{j,0}}{\Delta z} = 0 \text{ and } \partial_z v_{j,1} = \frac{v_{j,K} - v_{j,K+1}}{\Delta z} = 0$$

which implies $v_{j,0} = v_{j,1}$ and $v_{j,K+1} = v_{j,K}$ respectively.

For the heterogeneous agent model (11) - (12) featuring a Poisson process instead of a diffusion one, we add another grid dimension using $k = 1, \dots, K$

where k refers to the respective Poisson state and K is the total number of Poisson states. We discretize the HJB equation (15) which arises for this model with a two-state Poisson process by

$$\begin{aligned}\frac{v_{j,k}^{n+1} - v_{j,k}^{n}}{\Delta} + \rho v_{j,k}^{n} =& u(c_{j,k}^{n}) + \frac{v_{j+1,k}^{n+1} - v_{j,k}^{n+1}}{\Delta b}(s_{j,k}^{F,n})^{+} + \frac{v_{j,k}^{n+1} - v_{j-1,k}^{n+1}}{\Delta b}(s_{j,k}^{B,n})^{-} \\ &+ \lambda_k(v_{j,-k}^{n+1} - v_{j,k}^{n+1})\end{aligned} \tag{34}$$

where $-k$ denotes the other Poisson state respectively. Note that Poisson states cannot be discretized by sparse grids since they are already discrete.

4.3.6 Matrix notation

After linearizing and discretizing the HJB equation, we can easily formulate the resulting equations as a linear system. For the discretizations described in (34) and (33), this can be done by stacking the value function values into a vector $\mathbf{v}$ of length $m = J \cdot K$. Note that we use the same ordering for all other functions that depend on the grid points. Further, note that we indicate vectors, i.e. one-dimensional arrays by bold formatting and lower-case letters, whereas we indicate matrices by bold formatting and upper-case letters. By reordering the discretized HJB equation by its subscripts, we can then easily setup the respective matrices to formulate the discretized HJB equation by

$$\frac{1}{\Delta}(\mathbf{v}^{n+1} - \mathbf{v}^{n}) + \rho \mathbf{v}^{n+1} = \mathbf{u}^{n} + (\mathbf{A}^{n} + \mathbf{\Lambda})\mathbf{v}^{n+1} \tag{35}$$

where $\mathbf{A} \in \mathbb{R}^{m \times m}$ is the non-stochastic *drift matrix* and $\mathbf{\Lambda} \in \mathbb{R}^{m \times m}$ is the *intensity matrix* which models the stochastic process for productivity z. By simple algebra, we get

$$\underbrace{((\frac{1}{\Delta} + \rho)\mathbf{I} - \mathbf{A}^{n} - \mathbf{\Lambda})}_{\mathbf{B}}\,\mathbf{v}^{n+1} = \underbrace{\mathbf{u}^{n} + \frac{1}{\Delta}\mathbf{v}^{n}}_{\mathbf{b}^{n} = \mathbf{b}(\mathbf{v}^{n})} \tag{36}$$

with identity matrix $\mathbf{I} \in \mathbb{R}^{m \times m}$, i.e. we want to solve the linear system given by

$$\mathbf{B}\mathbf{v}^{n+1} = \mathbf{b}^{n} \tag{37}$$

with $\mathbf{B} \in \mathbb{R}^{m \times m}$ and $\mathbf{b}^{n} \in \mathbb{R}^{m}$. Note that $\mathbf{\Lambda}$ does not depend on n and thus can be precomputed.

To define $\mathbf{A}^{n}$ and $\mathbf{\Lambda}$ more formally, one can denote the construction via finite difference operators and specific scalar matrix-row multiplications. Let us show this for equation (33). We denote the row-wise vector matrix scalar multiplication, i.e. scalar multiplication of vector entry i, $1 \leq i \leq m$ with matrix row i by $\star$. To denote (33) using matrix notation (35), we have

$$\mathbf{A}^{n} = (\mathbf{s}^{F,n})^{+} \star \mathbf{D}_{b}^{F} + (\mathbf{s}^{B,n})^{-} \star \mathbf{D}_{b}^{B} \tag{38}$$

and

$$\mathbf{\Lambda}^n = (\boldsymbol{\mu})^+ \star \mathbf{D}_z^F + (\boldsymbol{\mu})^- \star \mathbf{D}_z^B + \frac{1}{2}\boldsymbol{\sigma}^2 \star \mathbf{D}_{zz} \tag{39}$$

where the standard operations should be understood entry-wise. Note that e.g. $((\mathbf{s}^{F,n})^+ \star \mathbf{D}_b^F)\mathbf{v}^n$ is nothing else but $(\mathbf{D}_b^F \mathbf{v}^n)\tilde{\star}((\mathbf{s}^{F,n})^+)$ where $\tilde{\star}$ denotes the entry-wise vector multiplication. The difference matrices $\mathbf{D}$ with sub- and superscripts indicating the taken derivative, are build using the standard full grid finite difference stencils, see any standard textbook such as [Lan13] for a description. For the sparse grid setting on the other hand, we use the respective derivative operators defined in Section 3.2.2.

5 (Non-)Convergence of Sparse Grid Finite Difference Schemes for solving the HJB equation

As we have explained in the last section, there are clear requirements that we need to fulfill to obtain a convergent approximation scheme by means of Barles and Souganidis [BS90], i.e. we want a stable, consistent and monotone scheme.

Thus, for our sparse grid finite differences there are be several steps involved to prove convergence. First, one aims for a local monotonicity property similar to the full grid setting. Then, second, it comes down to the monotonicity of the sparse grid interpolation, in particular at the points used for the sparse grid finite differences. Additionally, one needs to prove stability and consistency for the numerical scheme.

Whereas it is trivial to show that we need monotone interpolation, we face the problem that one cannot prove that interpolation on sparse grids is monotone in general, as it is pointed out in [Hem00].

Obviously, in just one dimension it is monotone since it is just a specific type of linear interpolation. This does not help when it comes to solving higher dimensional problems and we thus need monotone interpolation in multi-dimensional settings. Notice though that for our approach we only use one-dimensional monotonicity and we thus just need monotonicity with respect to the different dimensions, in particular with respect to the used ghost points.

We could easily construct examples that yield non-monotone interpolation, e.g. in two dimensions by considering functions that have steep gradients with respect to both dimensions. However, our hope was that we can achieve monotonicity if we restrict ourselves to concave monotonically increasing functions like the value functions arising from our models.

In accordance with the above motivation we aim to investigate monotonicity of sparse grid interpolation in the following, in particular with a restriction to concave increasing functions and the used ghost nodes. At the end of this section, we present some approaches to overcome the described non-monotonicities.

5.1 (Non-)monotonicity of interpolation on sparse grids

Let us begin by giving our definition of monotone interpolation.

Definition 5.1.1 (Monotone interpolation in one dimension)**.** Let $x_1, \ldots, x_n$ be data points with $x_1 < \ldots < x_n$. For a monotone function f, it holds that $f(x_1) \leq \ldots \leq f(x_n)$ with strict inequalities if it is strictly monotone. The interpolation f_I is *monotone*, if for every two points $\tilde{x}_1 < \tilde{x}_2,\ \tilde{x}_1, \tilde{x}_2 \in [x_1, x_n]$ it holds

$$f_I(\tilde{x}_1) \leq f_I(\tilde{x}_2)$$

with strict inequality for *strictly monotone interpolation*.

Note that we are interested in higher dimensions and aim for one-dimensional monotone interpolation for the restriction to the different dimensions respectively (and in particular with respect to the ghost points).

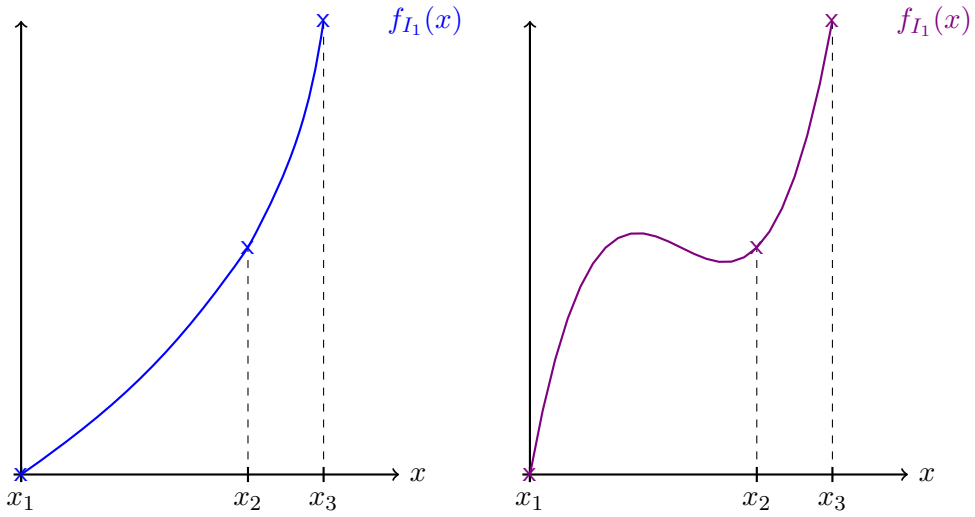


Figure 9: Monotone interpolation on the left and non-monotone interpolation on the right

In the following examples we add a high constant in the function examples to not exclude boundary points when we say "all" hierarchical coefficients. This is not necessary and we could also restrict ourselves to the inner points. Further, whenever we say "coefficients", we mean the hierarchical coefficients in the sparse grid function representation introduced in (18). Let us begin our investigation by motivating the restriction to concave monotonically increasing functions.

5.1.1 Strictly concave monotonically increasing functions with positive coefficients

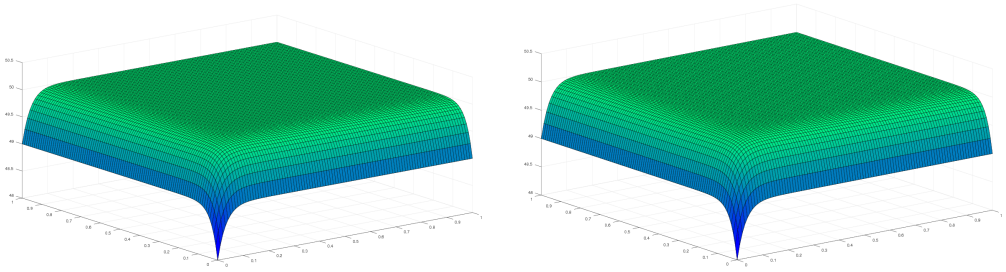
Notice that for the strictly concave monotonically increasing function

$$f_1(x, y) = -1000(1 - x)^2 - 1000(1 - y)^2 + 10000,$$

we end up with just positive coefficients and we also get a monotone interpolation. A more interesting case arises for the function

$$f_2(x, y) = -(1 - x)^{30} - (1 - y)^{30} + 50,$$

which has extremely steep gradients close to $(0, 0)$ and is flat in the other parts of the domain. This function is also strictly concave and monotonically increasing. Notice further that all hierarchical coefficients are positive. In Figure 10 one can see that the interpolation looks monotone. Explicitly checking the monotonicity shows that the approximation is monotone up to machine accuracy and thus the above example shows the motivation for the restriction to concave monotonically increasing functions to have monotone interpolation.



(a) Function plot of f_2 (b) Interpolation plot of f_2

Figure 10: Plots of the original function f_2 on the left and its sparse grid interpolation of level $l = 7$ on the right

5.1.2 Concave monotonically increasing functions with negative coefficients

One important thing you can notice in the above examples is that we only have positive hierarchical coefficients. It is possible though to find concave monotonically increasing functions for which negative coefficients arise. Let us give an example with a low sparse grid level: we show the interpolation behavior for the function

$$f_3(x, y) = \frac{-1}{1 + 2x + 3y} + 50.$$

x-coordinate of point	y-coordinate of point	hierarchical coefficient
0	0	49.0000
1	0	49.7500
0	1	49.6667
1	0.5	49.8333
0	0.5	0.1667
1	0.25	0.0083
0	0.25	0.0833
1	0.75	0.0028
0	0.75	0.0167
1	0	0.0015
0.5	0	0.2250
0.5	1	0.0278
0.5	0.5	−0.0621
0.25	0	0.1286
0.75	0	0.0173
0.25	1	0.0111
0.75	1	0.0040

Table 1: Interpolation of f_3 on a sparse grid of level $l = 2$ — hierarchical coefficients at the respective grid points

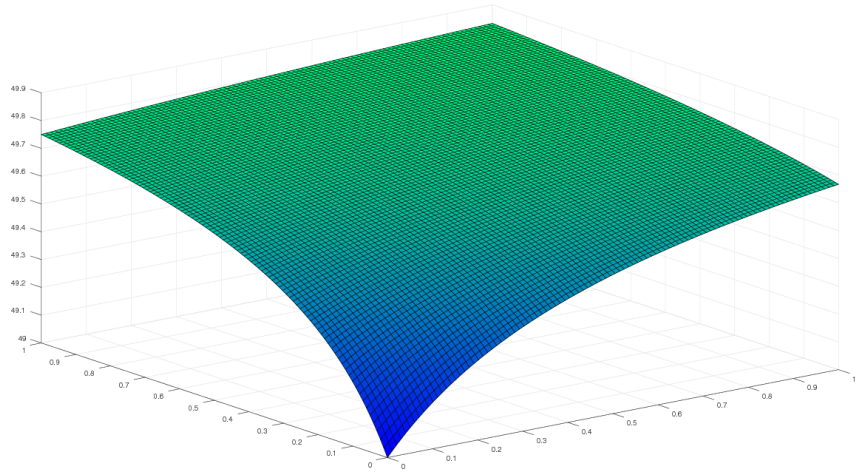


Figure 11: Plot of the sparse grid interpolation of f_3 for level $l = 7$

To show that the function is really concave, we compute the Hessian

$$\begin{pmatrix} -8/(2x+3y+1)^3 & -12/(2x+3y+1)^3 \\ -12/(2x+3y+1)^3 & -18/(2x+3y+1)^3 \end{pmatrix}.$$

It has the eigenvalues

$$\lambda_1 = \frac{-26}{(2x+3y+1)^3} \text{ and } \lambda_2 = 0.$$

Since we are in $[0,1]^2$, the Hessian is negative semidefinite and thus the function is concave.

In Table 1, a computation of the hierarchical coefficient for two-dimensional sparse grid is shown. We point out that the hierarchical coefficient at $(0.5/0.5)$ is negative. Notice that the interpolation which is given in Figure 11 is monotone. This can change for different function parameters, as we show in the following subsection.

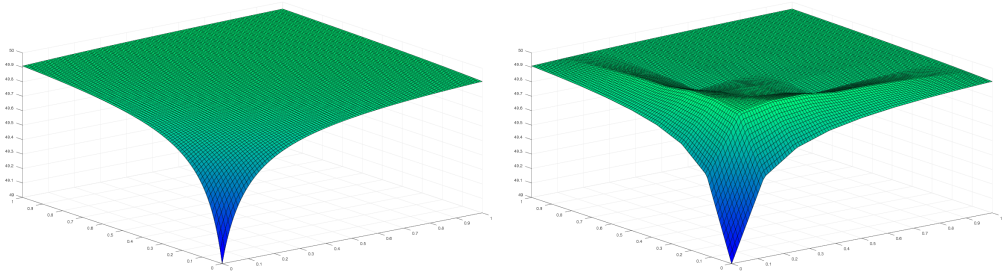
5.1.3 Non-monotone sparse grid interpolation for concave monotonically increasing functions

We now come to the main result of this subsection, which is the fact that interpolation for concave monotonically increasing functions is not monotone in general. Let us give a counter example. Notice that we use sparse grid level $l = 3$ in the following but similar counter examples can be used for other levels.

Let us analyze the interpolation for the above function but with other parameters, i.e.

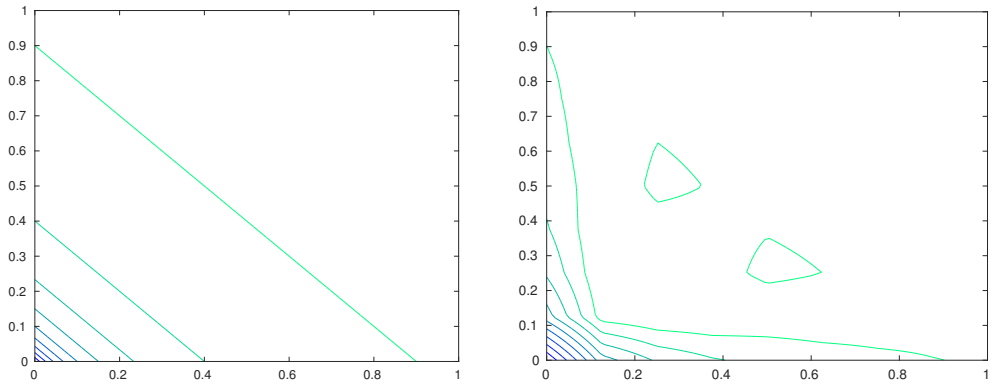
$$f_4(x,y) = \frac{-1}{1+10x+10y} + 50.$$

This function is obviously also concave but let us look at the plots of f_4 and its sparse grid interpolation. Unfortunately, we see in Figure 12 and Figure 13



(a) Function plot of f_4 (b) Interpolation plot of f_4

Figure 12: Plots of the original function f_4 on the left and its sparse grid interpolation of level $l = 3$ on the right



(a) Contour plot of function f_4 (b) Contour plot of the interpolation of f_4

Figure 13: Contour plots of the function f_4 on the left and its sparse grid interpolation of level $l = 3$ on the right

that the interpolation is not monotone. Increasing the factors in front of x and y also increases this effect. Note that the function is not strictly concave. You can also see this by looking at the contour plot of the function, whereas you see the non-monotonicity of the interpolation in the interpolation contour plot.

Note further that the above interpolation is in particular not monotone with respect to the points used for our sparse grid finite differences. In Figure 14 one can see that the value shown in red is higher than the one in green. Thus, even if we restrict ourselves to the ghost points, we do not have the monotonicity we hoped for.

The reason that we chose the above function is that it is similar to functions that arise as value function for some models. You may come to the conclusion that we just have monotone interpolation for *strictly* concave functions. There are several examples for strictly concave functions though which also yield non-monotone sparse grid interpolation, e.g. $f_5(x, y) = \frac{-1}{1+(x+0.01)^{0.2}+(y+0.01)^{0.2}} + 50$. The interested reader can check the concavity by using the leading principal minors criteria. Further, we point out that one cannot simply set the negative

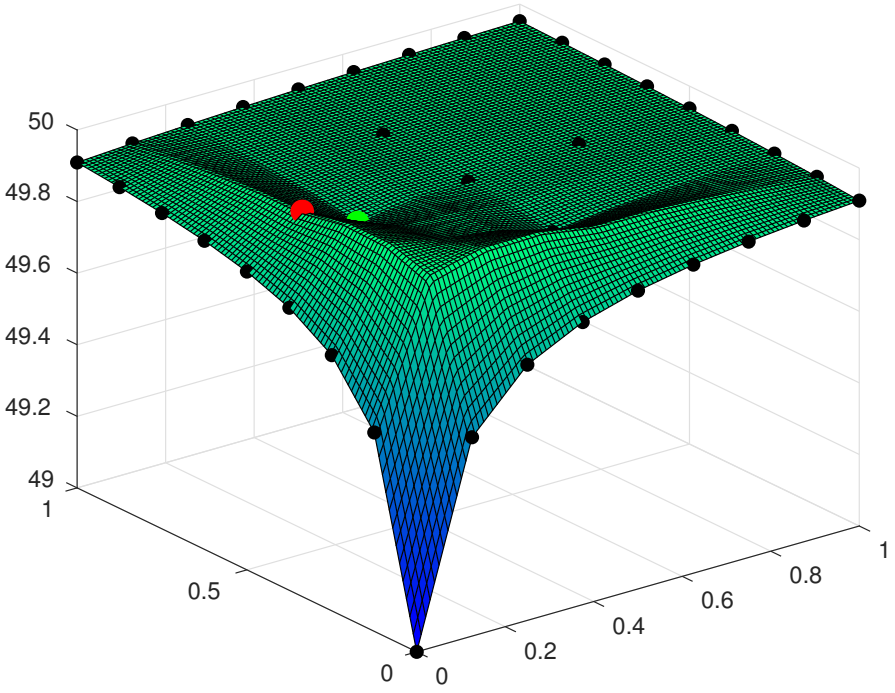


Figure 14: For the backward difference at (0.5/0.25), one uses the values drawn as the green and the red point

coefficients to zero to get a monotone approximation. Additionally, notice that the finite difference version presented in Section 3.2.1, which is based on hierarchical to nodal basis transformations also does not lead to monotonicity in general. Recall that we checked that both finite difference operator constructions lead to the same finite difference operator matrix for levels $l = 1, 2, 3$. We can use the same counter examples and it is therefore also not possible to get a general result respective monotonicity for this version. Thus, we propose other approaches to get monotone interpolation, to which we turn now.

5.1.4 Overcoming the non-monotonicity of sparse grid interpolation

In the following paragraphs we discuss some approaches to get monotone interpolation on sparse grids.

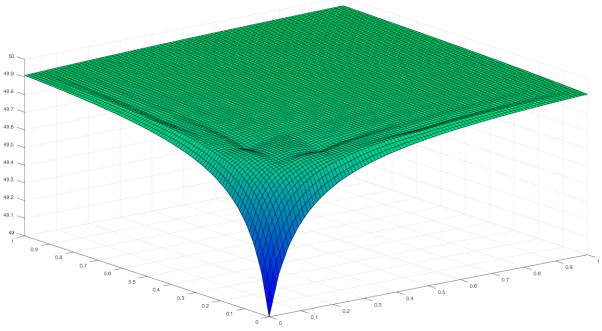
If one wants to prove convergence for the case of monotone interpolation and one uses a upwind scheme like the one explained in Section 4.3, one also has to ensure that the interpolation is concave on top of being monotone. Recall that one there uses that the forward difference is smaller than the backward difference for the value function since it is concave and monotonically increasing. This is given for full grids, but on sparse grids there can be points in which this is not the case even if the interpolation is monotone. Thus, one should either use an upwind scheme that does not rely on concavity or one should ensure that the used interpolations are concave in the above explained sense.

Note that one can easily check if the computed finite differences of the value function yield the theoretically required properties by checking if the finite dif-

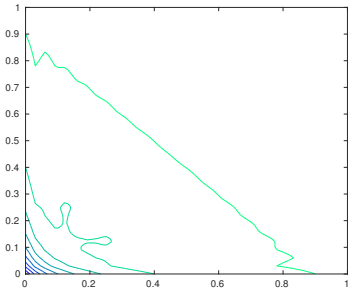
ferences are non-negative for the required monotonicity and by checking if the forward difference is smaller than the backward difference for the required concavity.

Let us present our ideas to overcome the non-monotonicity.

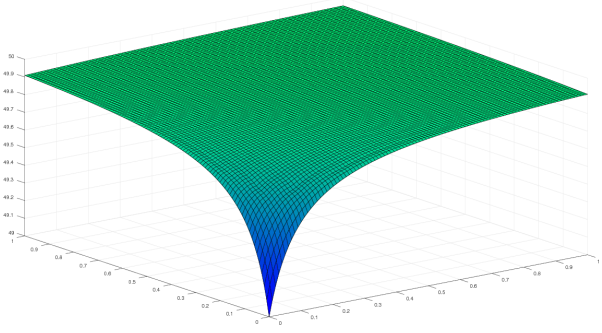
The most trivial way is to go to a higher sparse grid level which is visualized in Figure 15 where the interpolation of f_4 is presented for sparse grid



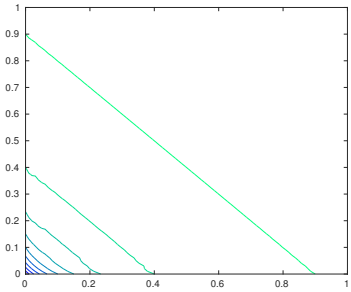
Interpolation plot of f_4 for level $l = 5$



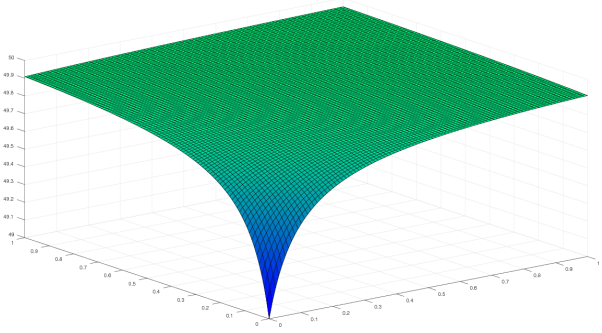
Contour plot for the interpolation of f_4 for level $l = 5$



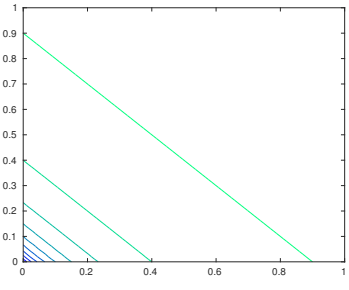
Interpolation plot of f_4 for level $l = 7$



Contour plot for the interpolation of f_4 for level $l = 7$



Interpolation plot of f_4 for level $l = 9$



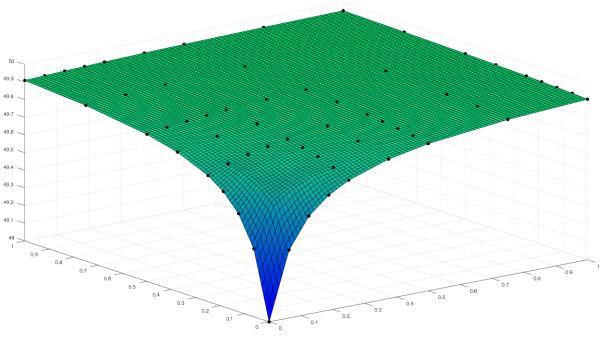
Contour plot for the interpolation of f_4 for level $l = 9$

Figure 15: Plots of the sparse grid interpolation of f_4 for different levels on the left and the respective contour plots on the right

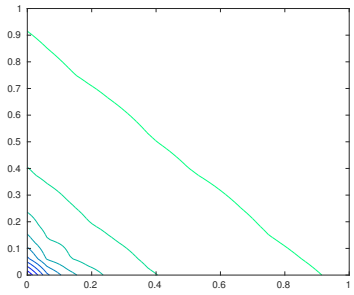
levels $l = 5, 7, 9$ instead of $l = 3$. The approach is simple but we cannot go to arbitrarily high levels in higher dimensions. Thus, we present some approaches that are based on identifying the areas where non-monotonicities arise and insert points only in this area.

We present two approaches to introduce new points to the sparse grid to achieve monotone sparse grid interpolation. The first one is to simply refine the grid like in the standard adaptive sparse grids approach yielding some higher level basis functions in these areas. Alternatively, one could go to a "full" grid in the critical area, we call this *partial full grid*. By that we mean that after setting up the sparse grid, one determines the maximal one-dimensional level used in this area and then simply uses the full grid level rule instead of the triangular sparse grid one there. The advantage of the second approach is that no interpolation is required on the partial full grid points and thus no non-monotonicity can arise there. Let us investigate the two ideas in more detail.

Adaptive sparse grids to get monotone interpolation We can adapt the sparse grid, using the hierarchical coefficients as error indicator, as it is done in standard adaptivity approaches. A better criterion to overcome non-monotonicity is to use the computed derivatives in the points, i.e. one can use the computed derivative approximations and check if they are non-negative since this indicates that the interpolation is not monotone. By marking such points for adaption, one can iterate until all derivative approximations are non-negative or below a certain error threshold. The plots in Figure 16 show the improved interpolation for such an adapted sparse grid. Note that the contour plot already indicates that the interpolation is monotone, but to show that it really is monotone, see Figure 17. Here we can see that the computed sparse grid forward differences in the x-dimension are clearly positive and thus the interpolation is monotone in required sense. Additionally, we point out that the backward difference is also positive. Note that due to symmetry we do not additionally show this for the y direction. Even though the approximated derivative is clearly positive and thus the function approximation itself is monotone,

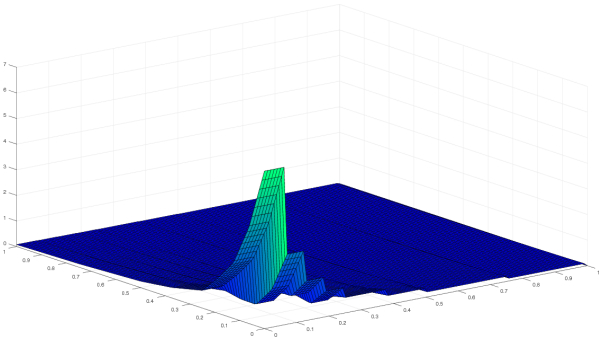


(a) Interpolation plot of f_4 for adapted sparse grid

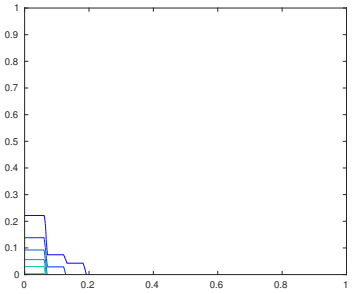


(b) Contour plot of f_4 for adapted sparse grid

Figure 16: Plot of interpolation of f_4 for an adapted sparse grid on the left and its contour plot on the right

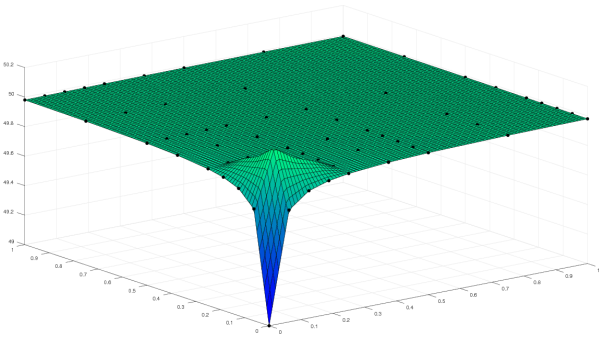


(a) Forward difference plot of f_4 for adapted sparse grid

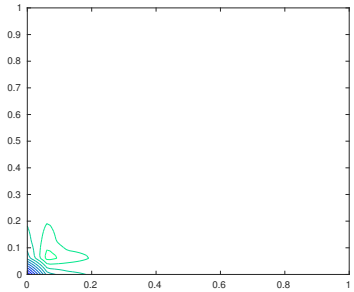


(b) Contour plot of the forward difference of f_4 for adapted sparse grid

Figure 17: Plot of the sparse grid forward difference in x-dimension of f_4 for an adapted sparse grid on the left and its contour plot on the right



(a) Interpolation plot of f_7 for adapted sparse grid



(b) Contour plot for the sparse grid interpolation of f_7 for adapted sparse grid

Figure 18: Plot of the sparse grid interpolation of f_4 for an adapted sparse grid on the left and its contour plot on the right

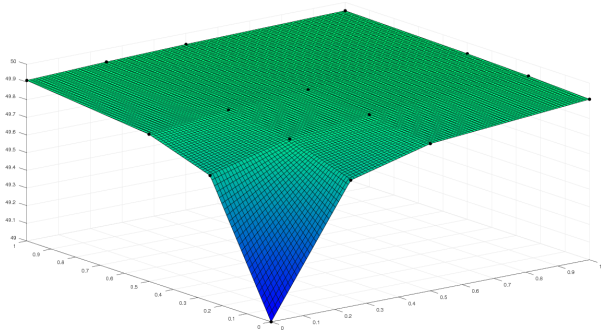
we point out that the derivative approximation itself is not monotone.

Moreover, notice that the above approach is a "two-way street". By that we mean that, even if one adapts the sparse grid such that there are no non-monotonicities, one can find a function for which the sparse grid interpolation is not monotone anymore. For example, in our case by simply increasing the factors in front of x and y in the function f_4, e.g.

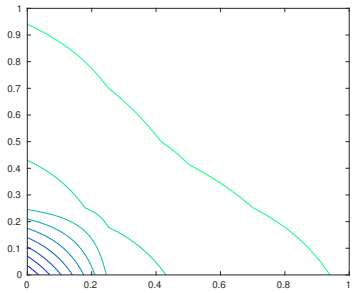
$$f_7(x, y) = \frac{-1}{1 + 50x + 50y} + 50,$$

the interpolation after adaption is again non-monotone, see Figure 18. Thus, one may have to readapt the sparse grid if the approximated function or the approximation changes as it is the case for iterates in an iterative algorithm.

Partial full grid to get monotone interpolation Now let us discuss the second idea — the partial full grid. The idea is to simply use the full grid



(a) Interpolation plot of f_4 for partial full grid



(b) Contour plot for interpolation of f_4 for partial full grid

Figure 19: Plot of the sparse grid interpolation of f_4 for partial full grid on the left and its contour plot on the right

rule instead of the sparse grid rule in the critical area, see subsection 3.1 for a description of the rules. We give a visualized example in Figure 19. Notice that the contour plot shows that the interpolation is not concave in the multi-dimensional sense even when we use a partial full grid. In our application however, we are particularly interested in the properties reflected in the points used for the (one-dimensional) finite differences. Obviously, this approach is monotone with respect to the inner full grid points. Additionally, for the restriction to one dimension, we have a concave interpolation and thus the backward difference difference is bigger than or equal to the forward difference in every point and thus it is also possible to use an upwind scheme based using the concavity property. The only issue is choosing the area for the partial full grid so that at its boundaries we also have the desired properties. Further, notice that we often do not need a partial full grid with respect to every dimension but only in the dimensions where the interpolation approach would fail otherwise. Notice though that for the newly introduced points one has to have the same properties again, and to be consistent one has to insert the ancestors of the respective points. Thus, such an adaption is iterative.
The obvious drawback of using a partial full grid is that one reintroduces the curse of dimensionality to some extend. Thus, this approach is only useful if the size of the full grid within the sparse grid is of limited size.

5.2 Comments on convergence in our setting

Note that we just showed that we cannot use the results of Barles and Souganidis. It may still be possible to prove convergence without using this theory. Furthermore, it would be useful to show that in the case that Algorithm 2 presented in Section 7 converges, it converges to the correct solution.

Under very strict assumptions which yield monotone sparse grid interpolation, it could be possible to prove convergence using Barles and Souganidis. However, this is not possible for our models with specific parameters.

As we explain in Section 7, the sparse grid approach that we use is based on the ideas presented in Section 4.3 for the simple model example. Thus, we can already discuss the issues which can arise in our setting. For a detailed explanation of our algorithmic approach in the sparse grid setting, see Section 7.

5.2.1 Overcoming non-monotonicity in our setting

The drawback of the approaches presented in the last subsection is that in our solution method we do not have a function evaluation (apart from the initial guess) but just iterates that are given by hierachical or nodal coefficients. Thus, to get function values at the grid points after adaption, we interpolate to newly inserted points. Since the basis functions are linear, the hierarchical values at the new points are zero directly after refinement and thus this does not instantly lead to an improved approximation but just allows for better ones in the following iterations with the refined grid.

Thus, to use the above approaches to get monotone interpolation, one normally cannot just continue after adaption with the current approximation on the refined grid since it is already non-monotone. One could either store the results of the prior iteration and recompute everything on the refined grid, or one could restart the whole algorithm on the refined grid. Notice though that with both approaches the approximation in the different iterations still changes and thus new non-monotonicities could arise and one would need to readapt the sparse grid. We point out that this could yield a lot of restarts and for the partial full grid approach one could end up with a real full grid in the worst case.

If one knows a priori in which areas non-monotonicities arise, one can overcome these issues, e.g. by just starting with a partial full grid in this area.

5.2.2 Issues arising in our setting without grid correction

Notice that for our models it depends on the model parameters if the arising interpolations are monotone (and concave). In our numerical results presented in Section 8 we do not use parameters that yield the extreme cases shown above. Thus, we can compute the solution of the HJB equation without using one of the presented grid correction approaches but simply following the algorithm presented in section 7.

We now want to explain what happens for our solution approach when non-monotonicities arise and what one can do to overcome the arising issues without using one of the grid correction approaches presented in 5.1.4 to get monotone interpolation.

Algorithm behavior for non-monotone interpolation In case of strong non-monotonicities the algorithm may not converge. Notice though that it never converges to a wrong solution (in our numerical studies).

Sometimes slight non-monotonicities show up in the first HJB iterations but the algorithm "regulates itself" in the sense that in later iterations no non-monotonicities arise anymore. The same holds true for non-concavities.

Handling arising issues in practice In case that the algorithm does not converge due to non-monotonicities in the first iteration(s) this may be because of the initial guess of the value function for starting the algorithm. The problem here is that when the initial guess is chosen such that strong non-monotonicities arise, even though the solution of the HJB equation can be approximated without non-monotonicities, the algorithm can fail. To overcome this issue, one should adapt the initial guess in such a way that this does not happen (e.g. by adding a constant in the denominator of the standard initial guess of staying put).

To handle the above noted problems with using a scheme that relies on concavity, we simply set the used finite difference to either the forward or the backward difference in the critical points. This works well in practice but that a scheme that does not rely on concavity would be a better approach for sparse grids.

6 Model Examples

In this section we explain the economic models that we implemented for our numerical experiments. We closely follow the presentation of a supplement to [KMV16] for the two asset model. The other models are natural extensions of this two-dimensional model to higher dimensions and can be found in Appendix A. For explanations of the model in the economic context, see Section 2.2, and for some further numerical explanations, see Section 4.3.

6.1 A model with two state variables – a two-asset model

In this subsection we describe a two-dimensional problem, give the arising first order conditions and present the numerical scheme.

6.1.1 Model formulation

We want to solve the following maximization problem

$$\max_{\{c_t,d_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{40}$$

subject to

$$\begin{aligned} \dot{b}_t &= w z_t r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t \\ z_t &= \text{Poisson with intensities } \lambda(z, z') \\ b_t &\geq b, \ a_t \geq 0. \end{aligned} \tag{41}$$

Here b_t denotes liquid assets and a_t illiquid assets. The respective returns on these assets are r^b and r^a. Further, we have consumption c_t, deposits d_t, the transaction cost function χ and wage w. The idiosyncratic productivity z_t follows a Poisson process with intensities $\lambda(z, z')$. The setting can easily be extended to diffusion type stochastic processes. We use the standard CRRA-utility function already stated in Section 2.2 given by

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

and the transaction cost function χ given by

$$\chi(d, a) = \chi_0 |d| + \frac{\chi_1}{2} \left(\frac{d}{a}\right)^2 a + \chi_2 \mathbb{1}_{\{d \neq 0\}}$$

with derivative w.r.t. d given by

$$\chi_d(d, a) = \chi_0 \mathbb{1}_{\{d>0\}} - \chi_0 \mathbb{1}_{\{d<0\}} + \chi_1 \frac{d}{a}.$$

6.1.2 HJB equation and first order conditions

By the standard steps, we get the HJB equation

$$\begin{aligned}\rho v(b,a,z) = &\max_{c,d} u(c)\\ &+ v_b(b,a,z)(wz + r^b(b)b - d - \chi(d,a) - c)\\ &+ v_a(b,a,z)(r^a + d)\\ &+ \sum_{z'} \lambda(z,z')(v(b,a,z') - v(b,a,z)).\end{aligned}$$

The first order conditions w.r.t. c and d yield

$$\begin{aligned}u_c(c) &= v_b(b,a,z),\\ v_a(b,a,z) &= v_b(b,a,z)(1+\chi_d(d,a))\end{aligned}$$

and thus we can simply compute the optimal consumption and optimal deposits given the value function derivatives. The optimal consumption is then given by

$$c = (v_b(b,a,z))^{-\frac{1}{\gamma}}.$$

Using our cost function, we get the optimal deposits for illiquid assets

$$\begin{aligned}d &= \underbrace{d^+}_{\text{case } d>0} + \underbrace{d^-}_{\text{case } d<0}\\ &= \left((\frac{v_a}{v_b} - 1 - \chi_0)\frac{a}{\chi_1}\right)^+ + \left((\frac{v_a}{v_b} - 1 + \chi_0)\frac{a}{\chi_1}\right)^-.\end{aligned} \tag{42}$$

6.1.3 Numerical approach using an upwind scheme

We are using an upwind scheme to solve the HJB equation numerically on sparse grids. The idea is to use an approach with proven convergence on full grids and show the approximation quality for the sparse grid finite difference method following the same upwind scheme.

As an initial guess for the value function, we use

$$v_0 = \frac{\left(\frac{wz + r^b(b)b + r^a a}{1-\gamma}\right)^{1-\gamma}}{\rho}, \tag{43}$$

where we follow the standard approach to start by staying "put", i.e. with controls equal to zero.

Notice that the optimal deposits d are computed with both the derivative with respect to b and with respect to a. Thus, to get a monotone scheme for this model, we use the trick to upwind such that there are no terms with different controls together and the respective forward and backward differences are used correctly. We split the drift of b,

$$s_b = wz + r^b(b)b - d - \chi(d,a) - c,$$

into different parts that do not have this type of interaction. We use the split

$$s_b = s^c + s^d$$

with

$$\begin{aligned} s^c &= wz + r^b(b)b - c, \\ s^d &= -d - \chi(d, a). \end{aligned}$$

We denote the optimal consumption that is computed with v_b^B by c^B and the optimal consumption that is computed with v_b^F by c^F. Then we denote the respective drifts by

$$\begin{aligned} s^{c,B} &= wz + r^b(b)b - c^B, \\ s^{c,F} &= wz + r^b(b)b - c^F. \end{aligned}$$

Using this notation, we approximate

$$v_b s^c \approx v_b^B (s^{c,B})^- + v_b^F (s^{c,F})^+.$$

For the optimal deposits, we follow the same scheme. Hence, we define

$$d^{m_1 m_2} \text{ with } m_1, m_2 \in \{B, F\}$$

where m_1 denotes the used difference approximation with respect to b, and m_2 denotes the used difference approximation with respect to a. Note that the optimal deposits are computed via (42). For example, for d^{BF} we end up with

$$d^{BF} = \left(\left(\frac{v_a^F}{v_b^B} - 1 - \chi_0 \right)^+ + \left(\frac{v_a^F}{v_b^B} - 1 + \chi_0 \right)^- \right) \frac{a}{\chi_1}.$$

Notice that we have to check if the marginal value of depositing or withdrawing is non-negative since otherwise $d = 0$ is the superior control. Therefore, we set d^{BF} to zero if

$$(v_a^F d^{BF} - v_b^F (d^{BF} + \chi(d, a)) < 0.$$

and refer to this as inaction region. Note further that the convex part of the cost function assures finite deposit rates. We can now define

$$\begin{aligned} d^B &= (d^{BF})^+ + (d^{BB})^-, \\ s^{d,B} &= -d^B - \chi(d^B, a), \end{aligned}$$

where v_b^B is used and

$$\begin{aligned} d^F &= (d^{FF})^+ + (d^{FB})^-, \\ s^{d,F} &= -d^F - \chi(d^F, a), \end{aligned}$$

where v_b^F is used. Using this notation, we approximate

$$v_b s^d \approx v_b^B (s^{d,B})^- + v_b^F (s^{d,F})^+.$$

Let us turn to our upwind approach for the a-dimension and its drift

$$s_a = r^a + d.$$

For the drift of a, we compute the deposits differently and define

$$\begin{aligned}\tilde{d}^B &= d^{BB}\mathbb{1}_{\{s^{d,B}<0\}} + d^{FB}\mathbb{1}_{\{s^{d,F}>0\}},\\ \tilde{d}^F &= d^{BF}\mathbb{1}_{\{s^{d,B}<0\}} + d^{FF}\mathbb{1}_{\{s^{d,F}>0\}},\end{aligned}$$

where we use the backward finite difference with respect to a, v_a^B, for $\tilde{d}^B$ and the forward finite difference with respect to a, v_a^F, for $\tilde{d}^F$.

Then we upwind the drift of a by

$$v_a s_a \approx v_a^B(\tilde{d}^B)^- + v_a^F((\tilde{d}^F)^+ + r^a).$$

All together we end up with the following finite difference upwind approximation for the HJB equation

$$\begin{aligned}\frac{v^{n+1}-v^n}{\Delta} + \rho v^{n+1} =& u(c^n)\\ &+ v_b^{B,n+1}(s^{c,B,n})^- + v_b^{F,n+1}(s^{c,F,n})^+\\ &+ v_b^{B,n+1}(s^{d,B,n})^- + v_b^{F,n+1}(s^{d,F,n})^+\\ &+ v_a^{B,n+1}(\tilde{d}^{B,n})^- + v_a^{F,n+1}((\tilde{d}^{F,n})^+ + r^a)\\ &+ \sum_{k'\neq k}^{K} \lambda_{k,k'}(v^{k',n+1} - v^{n+1}),\end{aligned}$$

where K denotes the number of Poisson states. Notice that we have omitted the grid point indices. Further, we point out that all value functions but the ones of state k', $v^{k',n+1}$, are value functions of state k. One can easily check that the above construction satisfies the monotonicity property given by Barles and Souganidis [BS90] (for full grids). Note that there are other numerical schemes which also work well. We further point out that this system is implicit in b, a and z. It is also possible to formulate a semi-implicit equation that is explicit in the productivity state z but still implicit in b and a, which allows to split the problem in K subproblems that one can solve simultaneously using parallelization.

Last but not least, we point out that our implementation is done for a two-state Poisson process, i.e. for $K = 2$.

6.2 Higher dimensional models

We refer to Appendix A for the implemented higher dimensional models in which we either add assets like housing ones or multiple diffusion type stochastic processes for different types of productivity. Notice that it comes down to a similar numerical scheme as the one presented in the last subsection and the same top level algorithm presented in the following section.

7 Algorithm and Implementation

In this section we present a sparse grid finite difference upwind algorithm to solve the HJB equation arising from models such as the ones presented in Section 6.1 and Appendix A. We start by explaining the approach for regular sparse grids, then we extend it to adaptive sparse grids. Finally, we explain our approach for solving the arising linear systems.

7.1 Algorithm for regular sparse grids

To state the algorithm for solving the HJB equation arising from our models on sparse grids, let us use the matrix notation introduced in Section 4.3.6. We recall (35),

$$((\frac{1}{\Delta} + \rho) - \mathbf{\Lambda} - \mathbf{A}^n)\mathbf{v}^{n+1} = \mathbf{u}^n + \frac{1}{\Delta}\mathbf{v}^n,$$

i.e. the linear system that we aim to solve on full grids.

The approach is very similar for the sparse grid setting. The non-stochastic drift matrix $\mathbf{A}$ can be build using the sparse grid finite difference operators introduced in Section 3.2.2. We further use these operators for the intensity matrix $\mathbf{\Lambda}$ in case of diffusion processes. Note that no derivative approximations are required for Poisson type processes.

As explained in Section 3.2.2, we use sparse grid finite difference operators operating on hierarchical coefficients. Since we require derivative approximations of the value function, we now denote $\mathbf{v}$ as the vector storing hierarchical coefficients of the value function approximation. For the utility function approximation, we already have nodal values and thus $\mathbf{u}$ now describes the vector containing nodal coefficients of the utility function approximation. To solve the linear system with consistent basis representations, we use the hierarchical to nodal basis transformations $\mathbf{E}$ and get

$$((\frac{1}{\Delta} + \rho)\mathbf{E} - \mathbf{\Lambda} - \mathbf{A}^n)\mathbf{v}^{n+1} = \mathbf{u}^n + \frac{1}{\Delta}\mathbf{E}\mathbf{v}^n \tag{44}$$

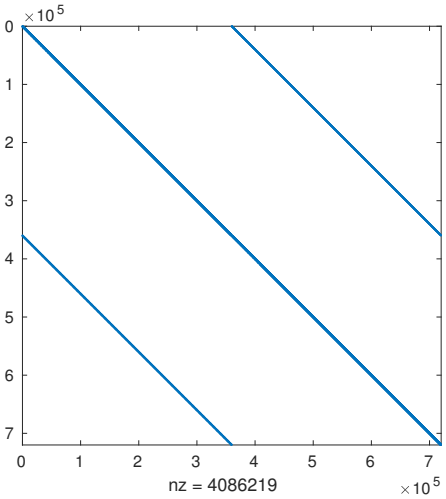
for diffusion processes where $\mathbf{\Lambda}$ is built with difference operators and thus works on hierarchical coefficients, whereas for Poisson processes we get

$$((\frac{1}{\Delta} + \rho)\mathbf{E} - \mathbf{E}\mathbf{\Lambda} - \mathbf{A}^n)\mathbf{v}^{n+1} = \mathbf{u}^n + \frac{1}{\Delta}\mathbf{E}\mathbf{v}^n \tag{45}$$

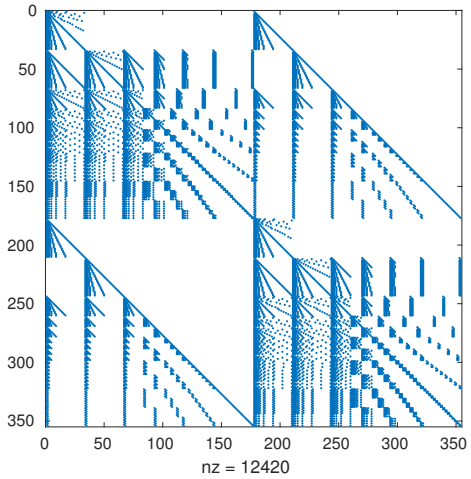
where $\mathbf{\Lambda}$ models the Poisson process. Notice that the resulting vectors on both sides of the equation are given in nodal values and the solution $\mathbf{v}^{n+1}$ is given in hierarchical values again such that we can simply use it in the next iteration for the computation of its derivatives.

We point out that the above formulation is general and can thus also be used for the models presented in Section 6.1 and Appendix A.

Notice that sparse grids have less structure than full grids and thus, by stacking the values of the respective grid points into a vector, we get a linear system which has no easily described sparsity structure. For a visualization of



Sparsity pattern of the matrix in the full grid approach arising for the $2d$ model with 600×600 full grid



Sparsity pattern of the matrix in the sparse grid approach arising for the $2d$ model for level $l = 5$

Figure 20: Sparsity patterns of the matrices arising for the $2d$ model (40) - (41) for the full grid on the left and for the sparse grid on the right

the sparsity pattern for both the full grid and the sparse grid case arising for the model presented in Section 6.1, see Figure 20.

Moreover, note that it would also be possible to use a nodal representation for the value function if we change the difference operators as operators working on nodal coefficients.

Let us explain our approach on solving the HJB equation on sparse grids by presenting Algorithm 2 in which we give references to the respective subsections for explanations. Comparing Algorithm 1 and 2, you can see that the algorithm for sparse grid case is very similar to the full grid one. It comes down to the same computations just on the sparse grid using sparse grid finite difference operators instead of the standard full grid approach.

In the initialization part of the algorithm we construct the sparse grid, compute the required nodal to hierarchical basis transformation matrix and construct the finite difference operators, see rows 2-4. Further, in row 5, we precompute the matrix that models the stochastic process since it models exogenous states that do not change depending on the other states and controls. Last but not least, in row 6, the initial guess is computed in hierarchical representation so that the finite difference operators can simply be applied to it.

The iterative part is very similar to full grid case with the slight difference that the finite difference operators are not set up explicitly for the standard full approach used by [AHL+17]. Note that this is more of an implementation detail than an algorithmic issue. Let us briefly summarize the iteration. In row 9, one computes the forward and back differences of the value function for all required dimensions. Using the resulting value function derivative approximations, we

Algorithm 2 Solving the HJB equation on sparse grids

Data: model parameters, sparse grid parameters
Result: solution $\mathbf{v}$ of HJB equation

1: **Initialization:**
2: Generate sparse grid ▷ see 3.1
3: Compute hierarchical to nodal basis transformation matrix $\mathbf{E}$ ▷ see 3.2.1
4: Generate finite difference operators ▷ see 3.2.2
5: Set up matrix $\mathbf{\Lambda}$ that models stochastic process ▷ see 4.3.6
6: Compute initial guess in hierarchical representation $\mathbf{v}^0$ ▷ see e.g. (43) for model (40) - (41)
7: **Iterative part:**
8: **for** $n = 0, 1, \ldots$ **do**
9: Compute forward and backward differences of $\mathbf{v}^n$
10: ▷ use finite difference operators
11: Compute optimal controls
12: ▷ e.g. consumption, deposits for model (40) - (41)), use forward and backward differences of $\mathbf{v}^n$
13: Build drift matrix $\mathbf{A}^n$
14: ▷ see 4.3.6, follow upwind scheme and use finite difference operators
15: Solve (45) respectively (44) for $\mathbf{v}^{n+1}$ ▷ see 4.3.4, linearized HJB equation
16: **if** $\mathbf{v}^{n+1}$ is close to $\mathbf{v}^n$ **then**
17: $\mathbf{v} \leftarrow \mathbf{v}^{n+1}$
18: STOP
19: **end if**
20: **end for**

can compute the optimal controls in row 11. Then, in row 13, we build the drift matrix by following the respective upwind scheme. In row 15, by using the drift and the intensity matrix, we can set up the linear system which yields the value function for the next iteration. We stop the algorithm if the value function only differs slightly between consecutive iterations, see row 16.

7.2 Adaptive refinement on sparse grids for the HJB equation

7.2.1 Adaptive refinement on sparse grids

As explained in Section 3.1.5, it is possible to further reduce the required amount of points with the main idea being to start with a low level and only add points in locations where the approximation quality should be improved.

We use different types of adaptivity criteria that are based on the hierarchical surpluses as an error indicator. For a description of similar algorithms for refinement and coarsening, see [GK17], and a general good description can be found in [Pfl10].

Let us denote the grid points by their corresponding indices $(\mathbf{l}, \mathbf{i})$. Collecting them in an index set $\mathcal{I}$ allows us to define the corresponding sparse grid by $Q_{\mathcal{I}}$. We start with a regular sparse grid with $\mathcal{I} = \{(\mathbf{l}, \mathbf{i}) | \Phi_{\mathbf{l},\mathbf{i}} \in V_n^s\}$ where n is small. Then we refine the grid by marking all points $(\mathbf{l}, \mathbf{i})$ which fulfill the refinement criterion

$$|a_{\mathbf{l},\mathbf{i}}| ||\Phi_{\mathbf{l},\mathbf{i}}|| > \varepsilon \tag{46}$$

where $a_{\mathbf{l},\mathbf{i}}$ is the hierarchical coefficient introduced in Section 3.1.2 and the refinement threshold $\varepsilon > 0$ is given. For our numerical studies, we use the maximum norm in the above inequality which reduces it to $|a_{\mathbf{l},\mathbf{i}}| > \varepsilon$. For each marked grid point, there exist two child nodes per dimension, i.e. an interior node has $2d$ children. We add all children of marked points that are not in the current grid. To get consistency, one has to check if all parents of newly added points are in the grid to maintain the hierarchical structure. Notice that we omit this part in Algorithm 3 but check for it later in Algorithm 5.

To reduce the amount of points further, we use a coarsening approach. The idea is to remove points that are not very helpful for a good approximation since they increase the computational costs unnecessarily. Thus, given a coarsening parameter $\nu > 0$ we remove an index $(\mathbf{l}, \mathbf{i})$ from the grid if

$$|a_{\mathbf{l},\mathbf{i}}| ||\Phi_{\mathbf{l},\mathbf{i}}|| \leq \nu \tag{47}$$

and no children of $(\mathbf{l}, \mathbf{i})$ are in $\mathcal{I}$. This is done for all grid points until no more grid points get removed. An equivalent formulation is to keep an index $(\mathbf{l}, \mathbf{i})$ in $\mathcal{I}$ if

$$|a_{\mathbf{l},\mathbf{i}}| ||\Phi_{\mathbf{l},\mathbf{i}}|| > \nu \tag{48}$$

Algorithm 3 Adaptive refinement

Data: initial index set $\mathcal{I}$, refinement threshold ε, hierarchical coefficients $a_{\mathbf{l},\mathbf{i}}$ of the function used for adaptive refinement
Result: refined index set $\mathcal{I}_r$

$\mathcal{J} \leftarrow \emptyset$ ▷ set of marked indices: add children of these
for all indices $(\mathbf{l}, \mathbf{i}) \subset \mathcal{I}$ **do** ▷ for all grid points
  if $|a_{\mathbf{l},\mathbf{i}}| ||\Phi_{\mathbf{l},\mathbf{i}}|| > \varepsilon$ **then** ▷ see (46)
    $\mathcal{J} \leftarrow \mathcal{J} \cup (\mathbf{l}, \mathbf{i})$ ▷ add index to set if criterion fulfilled
  end if
end for
$\mathcal{I}_r \leftarrow \mathcal{J} \cup$ children of $\mathcal{J}$

Algorithm 4 Adaptive coarsening

Data: initial index set $\mathcal{I}$, coarsening threshold ν, hierarchical coefficients $a_{\mathbf{l},\mathbf{i}}$ of the function used for adaptive coarsening
Result: coarsened index set $\mathcal{I}_c$

$\mathcal{I}_c \leftarrow \emptyset$ ▷ set of marked indices: keep these indices
for all indices $(\mathbf{l},\mathbf{i}) \subset \mathcal{I}$ **do** ▷ for all grid points
  if $|a_{\mathbf{l},\mathbf{i}}|||\Phi_{\mathbf{l},\mathbf{i}}|| > \nu$ **then** ▷ see (48)
    $\mathcal{I}_c \leftarrow \mathcal{I}_c \cup (\mathbf{l},\mathbf{i})$ ▷ add index to set if criterion fulfilled
  end if
end for

and additionally all of its ancestors to be consistent. Our approach is to identify all points that we aim to keep, and then to keep them together with all their ancestors. For our experiments, we use the coarsening parameter $\nu = \varepsilon/10$ which is a typical choice. We omit the consistency part in Algorithm 4 since we state it in Algorithm 5.

Let us put the adaptive refinement and the adaptive coarsening together in algorithm 5. Given an index set $\mathcal{I}$, a refinement parameter ε, a coarsening parameter ν and the approximated function v on $Q_{\mathcal{I}}$ we can refine and coarsen the grid and approximate the function on the new grid. Notice that it is not only possible to use different norms in (46) and (47) respectively (48) but also completely different criteria, e.g. percentage based ones or the ones we describe in the next subsection. Further, notice that we add parents of newly added points that are not in the grid already in order to be consistent with the hierarchical structure.

7.2.2 Adaptive refinement on sparse grids for the HJB equation

Now we have all ingredients to set up an algorithm that solves the HJB equation adaptively on sparse grids. We implement a self-adaptive version that lowers the refinement threshold and coarsening parameter automatically when no new

Algorithm 5 Adaptive steps combined

Data: data of **Algorithm 3** and **Algorithm 4**
Result: consistent refined and coarsened index set $\tilde{\mathcal{I}}$, v interpolated to $Q_{\tilde{\mathcal{I}}}$

Call **Algorithm 3** ▷ get refined index set $\mathcal{I}_r$
Call **Algorithm 4** ▷ get coarsened index set $\mathcal{I}_c$
$\tilde{\mathcal{I}} \leftarrow \mathcal{I}_r \cup \mathcal{I}_c$ ▷ new index set of newly added and kept points
$\tilde{\mathcal{I}} \leftarrow \tilde{\mathcal{I}} \cup$ ancestors of $\tilde{\mathcal{I}}$ ▷ add parents not in the set yet for consistency
Interpolate $v \in V_{\mathcal{I}}$ to $Q_{\tilde{\mathcal{I}}}$

points are added by adapting with the current parameters. This allows to use the algorithm with less knowledge about appropriate adaptivity parameters, and more importantly it focuses on the more important points in early iteration so that we do not add too many points in the first adaptive steps. Note that self-adaptivity is also used in [Sch98].

Let us briefly summarize Algorithm 6 which solves the HJB equation with adaptive sparse grids. We start with the same initialization as for the non-adaptive Algorithm 2, see row 2. Then we solve for the solution of the HJB equation on the given grid using 2 in row 5. If no stopping criterion is fulfilled, we refine the sparse grid with Algorithm 5 and call the initialization part of Algorithm 2 to compute all required operators and values for the refined grid, see rows 9 and 16 respectively. We point out that in case that no points are added in the refinement we lower the refinement threshold and adapt the sparse grid again in row 11. Note that there are several possibilities for stopping criteria in row 6 such as maximum number of points, maximum sparse grid

Algorithm 6 Solving the HJB equation with self-adaptive refinement on sparse grids

Data: model parameters, sparse grid parameters, self-adaptivity factor $\eta \leq 1$
Result: solution v of HJB equation

1: **Initialization:**
2: Call **Initialization** of **Algorithm 2**
3: **Iterative part:**
4: **while** True **do**
5: Call **Iterative part** of **Algorithm 2** ▷ get solution v of HJB equation
6: **if** stopping criterion fulfilled **then**
7: STOP
8: **end if**
9: Call **Algorithm 5**
10: ▷ refine/coarsen sparse grid and interpolate v to the new sparse grid
11: **if** no new points are added and $\eta \neq 1$ **then**
12: $\varepsilon \leftarrow \varepsilon \cdot \eta$, $\nu \leftarrow \nu \cdot \eta$ ▷ Decrease refinement threshold and coarsening parameter by multiplying with the self-adaptivity factor
13: Call **Algorithm 5**
14: ▷ refine/coarsen sparse grid and interpolate v to the new sparse grid
15: **end if**
16: Call **Initialization** of **Algorithm 2** for adapted sparse grid
17: **end while**

level, maximum number of adaptivity iterations or a maximum number of self-adaptivity applications.

7.2.3 Different types of adaptivity criteria

We experiment with several types of adaptivity criteria since there is no theoretical rule determining which criterion is optimal. The solution of the HJB equation — the value function — often does not give a lot of useful insight. Thus, we are more interested in a good approximation of the policy functions. These are computed by the approximated derivatives of the value function. Hence, it is not directly clear where the grid should be refined. That is the reason we study value function adaptivity and policy functions adaptivity. Additionally, we experiment with combinations of both.

We aim to investigate if it is better to refine the grid in areas where the policy functions are steep or in areas in which the value function is steep.

Adaptivity with a single criterion Note that we can represent the value function and the policy function on sparse grids by (18). Thus, for the value function adaptivity we use the hierarchical coefficients of the sparse grid value function approximation. Similarly, we use the hierarchical coefficients of the policy function approximation for the policy function adaptivity. Hence, for the two dimensional model (40)-(41) described in Section 6.1, we can use a value function adaptivity, a consumption function adaptivity or a deposit function adaptivity. Moreover, we analyze the combination of the above described adaptivity types to which we turn now.

Combination of different adaptivity criteria One possibility is a *logical combination*. By that we mean the use of a logical operator like OR or AND to combine adaptivity with respect to different functions, i.e. (46) is checked for all different functions and then has to be fulfilled by one of them, i.e. OR, or all of them, i.e. AND, to mark a point for adaptivity.
Moreover, one can implement a *weighted combination* by computing a weighted sum of the hierarchical coefficients of different functions on the same points. This way we combine value function and policy function adaptivity by just comparing the result of the weighted sum with the refinement threshold.

Alternative adaptivity types It is also possible to use percentage based adaptivity criteria. Above we just described an approach where we simply choose all points for refinement where the respective hierarchical coefficients are above a certain threshold. Contrarily, percentage based approaches use a fixed portion of the points for refinement that are the most likely to improve the accuracy, e.g. the points with the highest 50 percent of the hierarchical coefficients.

Furthermore, note that different norms in the adaptivity criterion are possible. Dimension adaptivity can be suitable in situations where certain variables are more important for an accurate solution than other ones. In our models

e.g. one could put less points in the dimensions of the stochastic processes. Of course this strongly depends on the model and its parameters.

7.3 Solving the linear system

To solve the linear system (37) which yields the updated value function, there are different approaches. The standard Matlab approach is to use the "backslash" operator which automatically chooses an appropriate direct solver by investigating several matrix properties like sparsity, symmetry and definiteness. As already explained in Section 4.3.6, the arising matrix in the sparse grid setting has no easily described structure which is in contrast to the full grid setting.

To solve the arising linear systems of our models, the Matlab backslash operator uses the SuitSparse Umfpack solver, see [Dav04] and [Dav07]. It is written in ANSI/ISO C and relies on Level-3-Basic Linear Algebra Subprograms. Umfpack uses the unsymmetric MultiFrontal method for solving unsymmetric sparse linear systems. For a survey of direct solvers and useful techniques to build those efficiently, see [DRSL16].

Even though this choice is efficient for moderately large matrices, direct solvers are no longer applicable if the linear system becomes very large. This is the case when we go to high dimensions.

Thus, we propose to use an iterative solver for high dimensional models. The matrix which arises from HJB equation discretizations is badly conditioned using a sparse grid discretization. Note further that it is also pointed out in [Sch98] that the condition number of the discretization matrix grows with h^{-2} with h being the grid point distance on the boundary. As a side note, we mention that experiments also show that HJB discretizations lead to badly conditioned systems for the full grid setting. For iterative solvers, we thus have to use an appropriate preconditioner to solve the system efficiently. A great source for iterative solvers and preconditioning is [Saa03]. Useful survey articles for preconditioning are [Ben02] for ILU factorizations and sparse approximate inverses, and the more recent [Wat15] for references. At the end of this section, we reference several approaches like multilevel ones which could lead to even better results. We now turn to our approach on solving the linear system.

7.3.1 Our approach for solving the linear system

Let us present our approach for solving the linear system, which is based on using a standard iterative solver in combination with a standard preconditioning approach.

Iterative solvers By looking into the Algorithm 2 to approximate the solution of the HJB equation, you can see two advantages of using iterative solvers in our setting. First, the matrix we get in the different iterations only changes slightly, which allows to compute a preconditioner and to use it multiple times (in the following iterations). Obviously, the preconditiong gets worse and one

should compute a new preconditioner if the iterative solvers do not converge (fast enough) anymore. Second, iterative solvers start with an initial guess $\mathbf{v}_{init}$. By simply setting the approximated solution of the last iteration as the starting point, we have a great initial guess, especially in the last iterations.

Several iterative solvers can be suitable and it depends on the chosen preconditioner and the model which one works best. Since the arising matrix has no easily described structure, we use iterative solvers which can handle nonsysmmetric matrices. We experiment with several Krylov subspace methods, namely GMRES, GMRES with restart and BiCGSTAB. BiCGSTAB yields the best, i.e. faster and more stable, results for most models and preconditioners and that is why we restrict the following presentation to BiCGSTAB.

Preconditioning For our models, BiCGSTAB without preconditioning does not converge due to quantities that get too small in the computation procedure. Thus, we have to use a preconditioner. We mainly experiment with standard preconditioning approaches that are already implemented in Matlab.

Jacobi preconditioning, SPAI sparse approximate inverses and normal equations to use symmetric approaches do not work (well) since either the computation time is really high or the iterative solver does not converge at all. Another standard preconditioning approach is the use of ILU factorizations. One gets an ILU factorization by dropping different entries in the factors in the LU factorization process. These dropping strategies can either be position or value based. We refer to [Saa03] and [Ben02] for detailed explanations. The Crout ILU (ILUC) version which is proposed in [LSC03] is based on using different loop orderings for the lower and the upper factors which allows for specific dropping rules.

Several ILU approaches could not be used due to some quantities that was to small in the iterations of preconditioned Krylov subspace methods. The by far most reliable approach in our experiments was ILUC, even though there may still be situations where ILUC preconditioning does not work or one would need to at least adjust the drop tolerance due to the above noted problem of quantities which get to small. Note that in some cases ILU with dropping threshold and pivoting (ILUTP) works well. We restrict the following presentation to ILUC due to the above noted insights.

Our approach: ILUC as preconditioner for BiCGSTAB Due to the above explanations we propose to use ILUC together with BiCGSTAB. Notice that even though in our tests we always achieved fast convergence by using an appropriate drop tolerance, there may be discretization matrices from other models where this is not possible.

Denote the drop tolerance by τ and the convergence flag by $FLAG$ which is 0 if the iterative solver converged.

Algorithm 7 Solving the linear system with ILUC preconditiong

Data: matrix $\mathbf{B}$ and right hand side $\mathbf{b}$, $maxit_1$, $maxit_2$, tol_1, tol_2, initial guess / solution of last HJB iteration $\mathbf{v}_{init}$, after first HJB iteration: $\mathbf{L}$ and $\mathbf{U}$ from last iteration
Result: solution $\mathbf{v}$ of $\mathbf{Bv} = \mathbf{b}$

1: **if** $n = 0$ **then**
2: $\quad \mathbf{L}, \mathbf{U} \leftarrow \text{ILUC}(\mathbf{B}, \mathbf{b}, \tau)$
3: **end if**
4: $\mathbf{v}, FLAG \leftarrow \text{precBiCGSTAB}(\mathbf{B}, \mathbf{b}, maxit_1, tol_1, \mathbf{L}, \mathbf{U}, \mathbf{v}_{init})$
5: **if** $FLAG \neq 0$ **then** ▷ If precBiCGSTAB did not converge
6: $\quad \mathbf{L}, \mathbf{U} \leftarrow \text{ILUC}(\mathbf{B}, \mathbf{b}, \tau)$
7: $\quad \mathbf{v}, FLAG \leftarrow \text{precBiCGSTAB}(\mathbf{B}, \mathbf{b}, maxit_2, tol_2, \mathbf{L}, \mathbf{U}, \mathbf{v}_{init})$
8: $\quad$ **if** $FLAG \neq 0$ **then** ▷ If precBiCGSTAB did not converge
9: $\quad\quad \mathbf{v} = \mathbf{B} \backslash \mathbf{b}$ ▷ Use direct solver
10: $\quad$ **end if**
11: **end if**

Algorithm 7 shows an approach to compute the preconditioner less often. The idea is to only recompute the ILUC factorization if the iterative solver did not converge in less than $maxit_1$ iterations. Notice that instead of using a direct solver in line 9 one can also change the drop tolerance in the ILUC computation or use another preconditioning approach.

7.3.2 Multilevel and other approaches used in related works

Instead of using standard algebraic (black-box) preconditioning methods like the ones presented above, one can use problem specific preconditioning. Note that multigrid solvers are optimal in many settings, see e.g. [BM$^+$00], [TOS00] and [Hac13]. We did not implement the following approaches but reference them for follow-up work since similar approaches could be superior to our ILUC factorization preconditioning, especially with regard to parallelization.

Problem specific diagonal scaling is proposed in [Sch98] by using the fact that the basis transformations do not change the diagonal entries of the sparse grid finite difference operators. They report good results for specific elliptic and parabolic PDEs. Moreover, it is pointed out that the eigenvalue range only grows with rate $\mathcal{O}(h^{-1})$ after preconditioning. These diagonal preconditioners are easily implemented, fast constructed and applied, and can be coupled with other preconditioners.

Another approach is the use of multilevel methods like proposed in [Spr01]. In [HS99] it is shown how the arising discrete equations can efficiently be solved in an iterative process. Further, in [HS] it is reported that BiCGStab shows fast convergence if it is applied to the hierarchical representation and combined

with nested iteration in a cascadic algorithm, although the convergence rate is not truly independent of the mesh size.

A combination of lifting-wavelet transformations and diagonal scaling is used in [Kos] for elliptic PDEs. Furthermore, in [GH14] a multilevel approach for the discretized Laplacian is presented and for sparse grid discretizations of elliptic PDEs optimal scaling parameters with a specific subspace solver are proposed in [GHO15].

Moreover, even though mostly more related to standard finite elements, there is a lot of research in the area of algebraic multigrid methods (AMG) which extract information just from the system matrix and mimic geometric multigrid methods. See e.g. [RS87] and [Stü01] for an introduction, [Y$^+$02] for a well known AMG software and [XZ17] for a recent overview article. Due to the high irregularity and the fact that AMG methods are mainly limited to symmetric positive definite matrices unlike the arising matrix in our setting, it is likely that most AMG methods do not converge. Note though that, even when they are non-convergent, they can often be used as an efficient preconditioner.

Further investigations are needed if similar approaches can be used effectively for our setting.

8 Numerical Results

Before we present the numerical results, let us define our error metrics denoting the reference solution by f_{ref} and the sparse grid solution by f_{SG}. We use three different discrete vector norms, i.e. we compute the absolute error in M points $x_1, \ldots, x_M$ by

$$\begin{aligned} e^f_{1,a}(x_1, \ldots, x_M) &= \frac{1}{M} \sum_{m=1}^{M} |f_{\text{ref}}(x_m) - f_{\text{SG}}(x_m)| \\ e^f_{2,a}(x_1, \ldots, x_M) &= \left(\frac{1}{M} \sum_{m=1}^{M} |f_{\text{ref}}(x_m) - f_{\text{SG}}(x_m)|^2 \right)^{\frac{1}{2}}, \\ e^f_{\infty,a}(x_1, \ldots, x_M) &= \max_m |f_{\text{ref}}(x_m) - f_{\text{SG}}(x_m)|. \end{aligned} \tag{49}$$

Using a normalization with respect to the respective reference solutions allows us to better compare the arising errors of different functions. Thus, we compute relative errors by

$$\begin{aligned} e^f_{1,r}(x_1, \ldots, x_M) &= \frac{1}{M} \sum_{m=1}^{M} \left| \frac{f_{\text{ref}}(x_m) - f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m) - \min_m f_{\text{ref}}(x_m)} \right| \\ e^f_{2,r}(x_1, \ldots, x_M) &= \left(\frac{1}{M} \sum_{m=1}^{M} \left| \frac{f_{\text{ref}}(x_m) - f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m) - \min_m f_{\text{ref}}(x_m)} \right|^2 \right)^{\frac{1}{2}}, \\ e^f_{\infty,r}(x_1, \ldots, x_M) &= \max_m \left| \frac{f_{\text{ref}}(x_m) - f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m) - \min_m f_{\text{ref}}(x_m)} \right|. \end{aligned} \tag{50}$$

We additionally compute convergence rates for level l by

$$\rho_e(l) = \log_2 \left(\frac{e_{l-1}}{e_l} \right),$$

where e_l denotes the error for level l and e is one of the errors defined above.

We point out that for all adaptivity criteria we use a normalization of the hierarchical coefficients with respect to the range in nodal values. Further, we always coarsen the grid with respect to value function since this yields better results in our experiments.

8.1 Two-dimensional model: plots and accuracy analysis

Let us begin with the $2d$ model (40) - (41) presented in Section 6.1 to give some intuition and to show that our sparse grid algorithm converges to the solution of the full grid method.

We begin our accuracy analysis by presenting relative errors for the value function and all policy functions for regular sparse grids of different levels. We denote the degrees of freedom (number of grid points) by DOF. The reference solution is computed on a 600×600 full grid and we interpolate the sparse grid solution to this grid for the error computations. We denote the respective Poisson states by subscripts (1 and 2) for the respective functions.

8.1.1 Accuracy for regular sparse grids

We start by giving the respective e_r-errors and convergence rates for the value function v.

Level l	Dof	$e^{v_1}_{1,r}$	$\rho_{e^{v_1}_{1,r}}$	$e^{v_1}_{2,r}$	$\rho_{e^{v_1}_{2,r}}$	$e^{v_1}_{\infty,r}$	$\rho_{e^{v_1}_{\infty,r}}$
2	17	$1.16{\cdot}10^{-1}$	–	$1.22{\cdot}10^{-1}$	–	$2.14{\cdot}10^{-1}$	–
3	37	$6.81{\cdot}10^{-2}$	0.76	$7.26{\cdot}10^{-2}$	0.75	$1.44{\cdot}10^{-1}$	0.57
4	81	$3.62{\cdot}10^{-2}$	0.91	$3.89{\cdot}10^{-2}$	0.9	$8.81{\cdot}10^{-2}$	0.71
5	177	$1.96{\cdot}10^{-2}$	0.88	$2.12{\cdot}10^{-2}$	0.88	$2.92{\cdot}10^{-2}$	1.59
6	385	$9.70{\cdot}10^{-3}$	1.01	$1.06{\cdot}10^{-2}$	1	$2.92{\cdot}10^{-2}$	0
7	833	$4.60{\cdot}10^{-3}$	1.08	$5.00{\cdot}10^{-3}$	1.08	$1.49{\cdot}10^{-2}$	0.97
8	1 793	$1.70{\cdot}10^{-3}$	1.44	$1.90{\cdot}10^{-3}$	1.4	$6.00{\cdot}10^{-3}$	1.31

Table 2: Accuracy of different sparse grid levels: e_r-errors for the value function for state 1

Level l	Dof	$e^{v_2}_{1,r}$	$\rho_{e^{v_2}_{1,r}}$	$e^{v_2}_{2,r}$	$\rho_{e^{v_2}_{2,r}}$	$e^{v_2}_{\infty,r}$	$\rho_{e^{v_2}_{\infty,r}}$
2	17	$1.11{\cdot}10^{-1}$	–	$1.17{\cdot}10^{-1}$	–	$2.04{\cdot}10^{-1}$	–
3	37	$6.58{\cdot}10^{-2}$	0.75	$6.98{\cdot}10^{-2}$	0.74	$1.37{\cdot}10^{-1}$	0.57
4	81	$3.54{\cdot}10^{-2}$	0.89	$3.77{\cdot}10^{-2}$	0.89	$8.18{\cdot}10^{-2}$	0.74
5	177	$1.93{\cdot}10^{-2}$	0.87	$2.06{\cdot}10^{-2}$	0.87	$4.76{\cdot}10^{-2}$	0.78
6	385	$9.60{\cdot}10^{-3}$	1.01	$1.03{\cdot}10^{-2}$	1	$2.49{\cdot}10^{-2}$	0.93
7	833	$4.50{\cdot}10^{-3}$	1.09	$4.90{\cdot}10^{-3}$	1.07	$1.20{\cdot}10^{-2}$	1.05
8	1 793	$1.70{\cdot}10^{-3}$	1.4	$1.80{\cdot}10^{-3}$	1.44	$4.50{\cdot}10^{-3}$	1.41

Table 3: Accuracy of different sparse grid levels: e_r-errors for the value function for state 2

The following are the respective e_r-errors and convergence rates for the deposit function d.

Level l	Dof	$e^{d_1}_{1,r}$	$\rho_{e^{d_1}_{1,r}}$	$e^{d_1}_{2,r}$	$\rho_{e^{d_1}_{2,r}}$	$e^{d_1}_{\infty,r}$	$\rho_{e^{d_1}_{\infty,r}}$
2	17	$1.03{\cdot}10^{-1}$	–	$1.19{\cdot}10^{-1}$	–	$2.46{\cdot}10^{-1}$	–
3	37	$6.70{\cdot}10^{-2}$	0.61	$7.86{\cdot}10^{-2}$	0.6	$1.51{\cdot}10^{-1}$	0.71
4	81	$4.40{\cdot}10^{-2}$	0.61	$5.29{\cdot}10^{-2}$	0.57	$1.10{\cdot}10^{-1}$	0.46
5	177	$2.76{\cdot}10^{-2}$	0.67	$3.35{\cdot}10^{-2}$	0.66	$7.46{\cdot}10^{-2}$	0.55
6	385	$1.56{\cdot}10^{-2}$	0.82	$1.92{\cdot}10^{-2}$	0.8	$5.30{\cdot}10^{-2}$	0.49
7	833	$8.00{\cdot}10^{-3}$	0.96	$9.90{\cdot}10^{-3}$	0.96	$3.15{\cdot}10^{-2}$	0.75
8	1 793	$3.30{\cdot}10^{-3}$	1.28	$4.20{\cdot}10^{-3}$	1.24	$1.84{\cdot}10^{-2}$	0.78

Table 4: Accuracy of different sparse grid levels: e_r-errors for the deposit policy function for state 1

Level l	Dof	$e^{d_2}_{1,r}$	$\rho_{e^{d_2}_{1,r}}$	$e^{d_2}_{2,r}$	$\rho_{e^{d_2}_{2,r}}$	$e^{d_2}_{\infty,r}$	$\rho_{e^{d_2}_{\infty,r}}$
2	17	$1.04{\cdot}10^{-1}$	–	$1.21{\cdot}10^{-1}$	–	$2.48{\cdot}10^{-1}$	–
3	37	$6.93{\cdot}10^{-2}$	0.58	$8.06{\cdot}10^{-2}$	0.59	$1.55{\cdot}10^{-1}$	0.68
4	81	$4.65{\cdot}10^{-2}$	0.58	$5.49{\cdot}10^{-2}$	0.55	$1.07{\cdot}10^{-1}$	0.54
5	177	$2.97{\cdot}10^{-2}$	0.65	$3.56{\cdot}10^{-2}$	0.62	$8.82{\cdot}10^{-2}$	0.28
6	385	$1.71{\cdot}10^{-2}$	0.8	$2.06{\cdot}10^{-2}$	0.79	$6.30{\cdot}10^{-2}$	0.49
7	833	$8.80{\cdot}10^{-3}$	0.96	$1.08{\cdot}10^{-2}$	0.93	$3.79{\cdot}10^{-2}$	0.73
8	1 793	$3.60{\cdot}10^{-3}$	1.29	$4.50{\cdot}10^{-3}$	1.26	$1.70{\cdot}10^{-2}$	1.16

Table 5: Accuracy of different sparse grid levels: e_r-errors for the deposit policy function for state 2

The following are the respective e_r-errors and convergence rates for the consumption function c.

Level l	Dof	$e^{c_1}_{1,r}$	$\rho_{e^{c_1}_{1,r}}$	$e^{c_1}_{2,r}$	$\rho_{e^{c_1}_{2,r}}$	$e^{c_1}_{\infty,r}$	$\rho_{e^{c_1}_{\infty,r}}$
2	17	$2.27{\cdot}10^{-1}$	–	$2.32{\cdot}10^{-1}$	–	$3.62{\cdot}10^{-1}$	–
3	37	$1.23{\cdot}10^{-1}$	0.88	$1.28{\cdot}10^{-1}$	0.86	$2.46{\cdot}10^{-1}$	0.56
4	81	$6.22{\cdot}10^{-2}$	0.98	$6.64{\cdot}10^{-2}$	0.95	$1.55{\cdot}10^{-1}$	0.66
5	177	$3.26{\cdot}10^{-2}$	0.93	$3.56{\cdot}10^{-2}$	0.9	$1.28{\cdot}10^{-1}$	0.27
6	385	$1.59{\cdot}10^{-2}$	1.04	$1.80{\cdot}10^{-2}$	0.98	$1.03{\cdot}10^{-1}$	0.32
7	833	$7.50{\cdot}10^{-3}$	1.08	$8.90{\cdot}10^{-3}$	1.02	$7.71{\cdot}10^{-2}$	0.42
8	1 793	$2.80{\cdot}10^{-3}$	1.42	$3.60{\cdot}10^{-3}$	1.31	$5.21{\cdot}10^{-2}$	0.57

Table 6: Accuracy of different sparse grid levels: e_r-errors for the consumption policy function for state 1

Level l	Dof	$e^{c_2}_{1,r}$	$\rho_{e^{c_2}_{1,r}}$	$e^{c_2}_{2,r}$	$\rho_{e^{c_2}_{2,r}}$	$e^{c_2}_{\infty,r}$	$\rho_{e^{c_2}_{\infty,r}}$
2	17	$1.94{\cdot}10^{-1}$	–	$2.17{\cdot}10^{-1}$	–	$3.66{\cdot}10^{-1}$	–
3	37	$9.32{\cdot}10^{-2}$	1.06	$1.03{\cdot}10^{-1}$	1.08	$1.75{\cdot}10^{-1}$	1.06
4	81	$4.70{\cdot}10^{-2}$	0.99	$5.20{\cdot}10^{-2}$	0.98	$1.06{\cdot}10^{-1}$	0.73
5	177	$2.37{\cdot}10^{-2}$	0.99	$2.60{\cdot}10^{-2}$	1	$5.96{\cdot}10^{-2}$	0.83
6	385	$1.21{\cdot}10^{-2}$	0.97	$1.33{\cdot}10^{-2}$	0.97	$3.33{\cdot}10^{-2}$	0.84
7	833	$5.60{\cdot}10^{-3}$	1.11	$6.20{\cdot}10^{-3}$	1.1	$1.70{\cdot}10^{-2}$	0.97
8	1 793	$2.00{\cdot}10^{-3}$	1.49	$2.20{\cdot}10^{-3}$	1.49	$6.70{\cdot}10^{-3}$	1.34

Table 7: Accuracy of different sparse grid levels: e_r-errors for the consumption policy function for state 2

From the Tables 2 - 7 we get several insights. First of all we see that our sparse grid algorithm converges to the full grid solution for all sparse grid levels. We additionally note that the accuracy for both Poisson states is quite similar. Note that this may change if the resulting functions more different though.

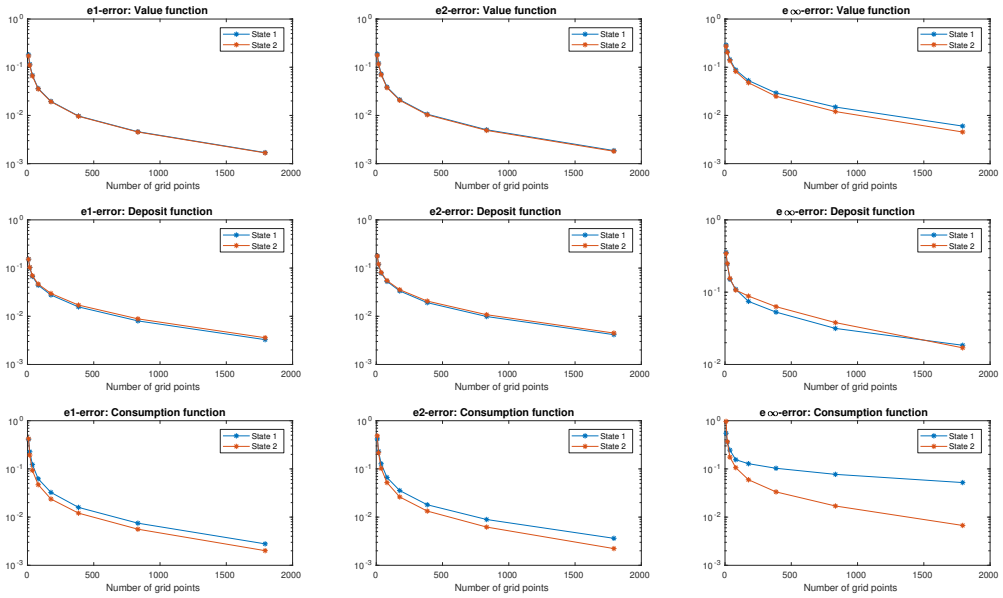


Figure 21: Accuracy of different sparse grid levels: e_r-errors for the value, deposit and consumption function for states 1 and 2 with stars indicating the levels $l = 1, \ldots, 8$

Furthermore, by going to higher sparse grid levels, we improve the accuracy. Notice that the convergence rate for all levels and error types is quite high and thus going to higher sparse grid levels pays off since convergence (in the sense of going to higher sparse grid levels) does not stagnate yet.

We additionally visualize the errors for the respective levels for the value, deposit and consumption functions in Figure 21. Note that we denote the respective e_r-errors on the y-axis and the number of grid points on the x-axis. You can see that the e_∞-error for the consumption function is quite high. This is due to that fact that the consumption function of state 1 is very steep close to the boundary and whence cannot be captured well by sparse grids. You can also see this in the following subsection in which we plot the solutions.

8.1.2 Plots for regular sparse grids

To get some insight into how the approximations and the arising errors look visually, let us present some plots of the respective approximations. We present the results for sparse grid level $l = 7$ and we again interpolate the sparse grid solution to the full grid to compare the respective solutions.

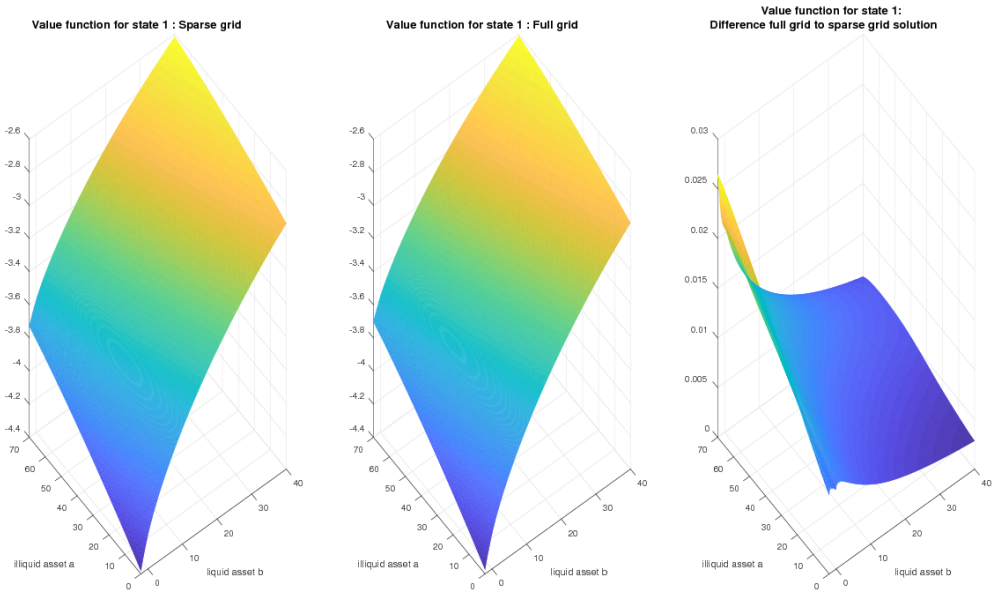


Figure 22: Value function for state 1: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

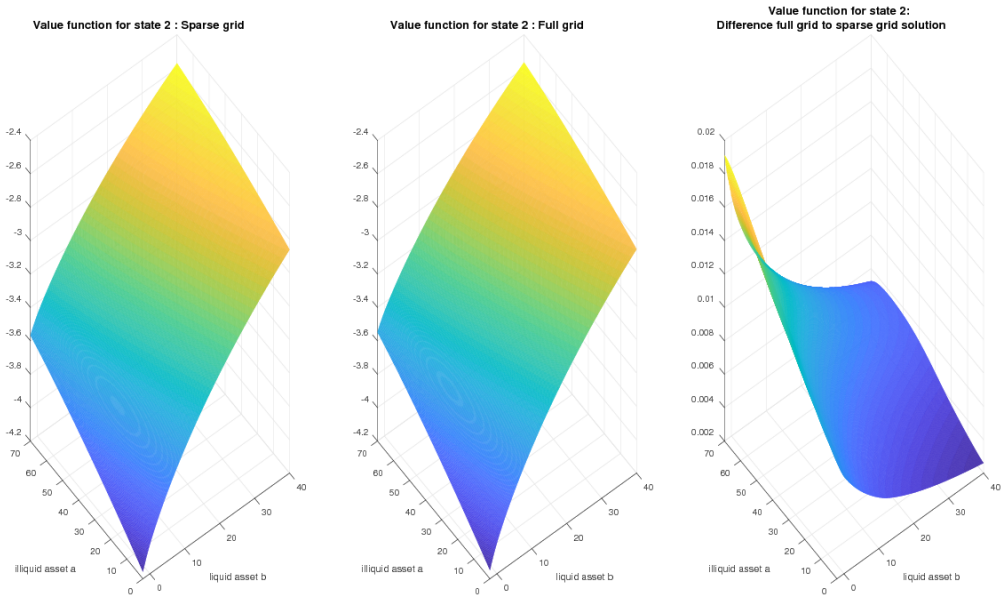


Figure 23: Value function for state 2: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

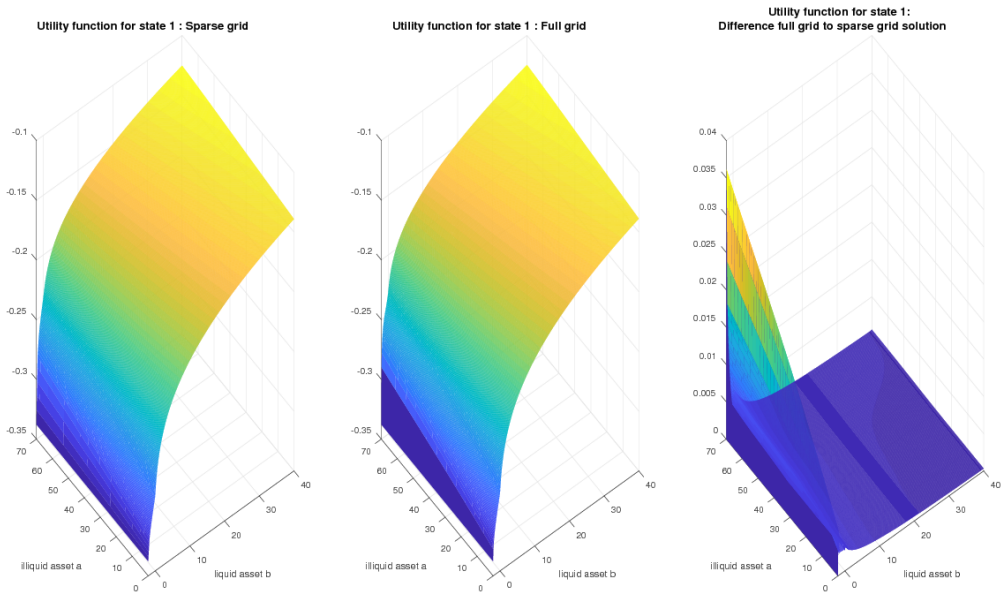


Figure 24: Utility function for state 1: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

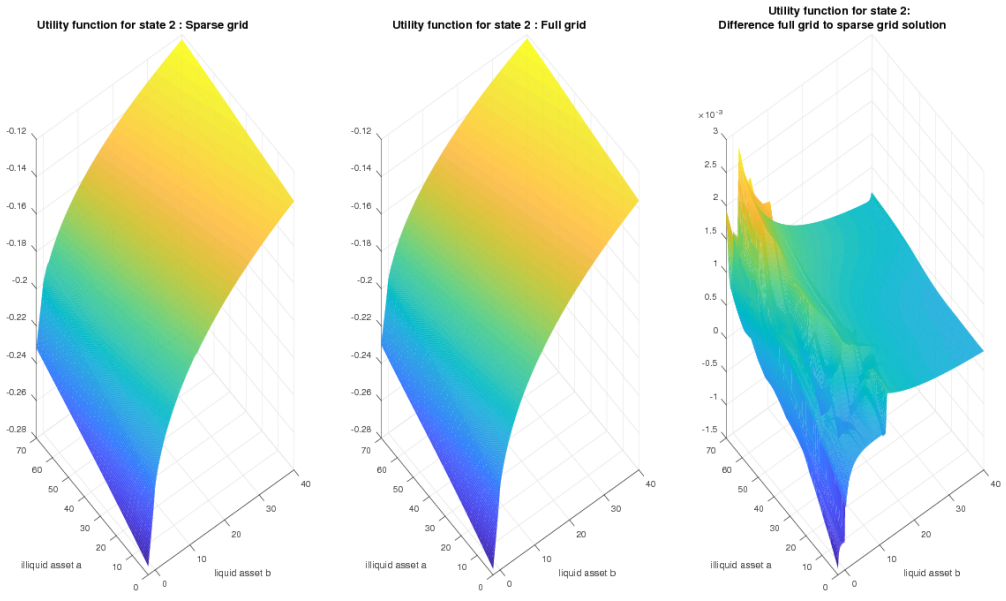


Figure 25: Utility function for state 2: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

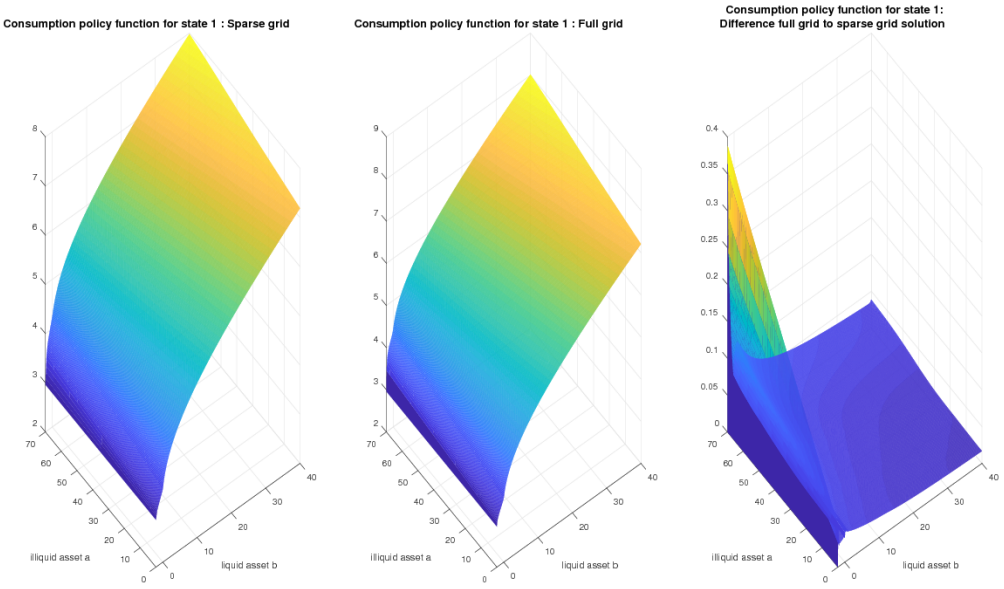


Figure 26: Consumption policy function for state 1: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

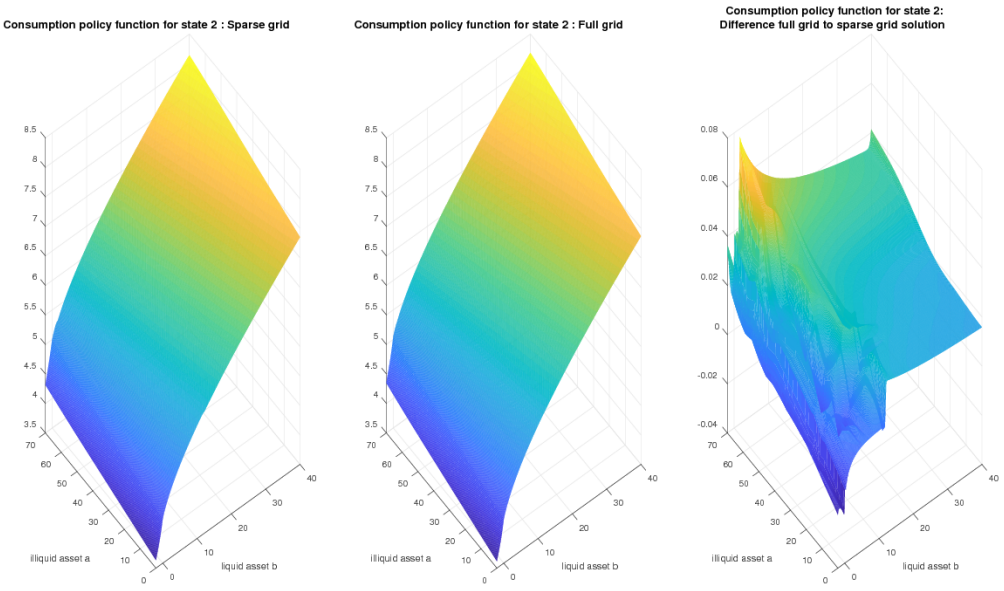


Figure 27: Consumption policy function for state 2: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

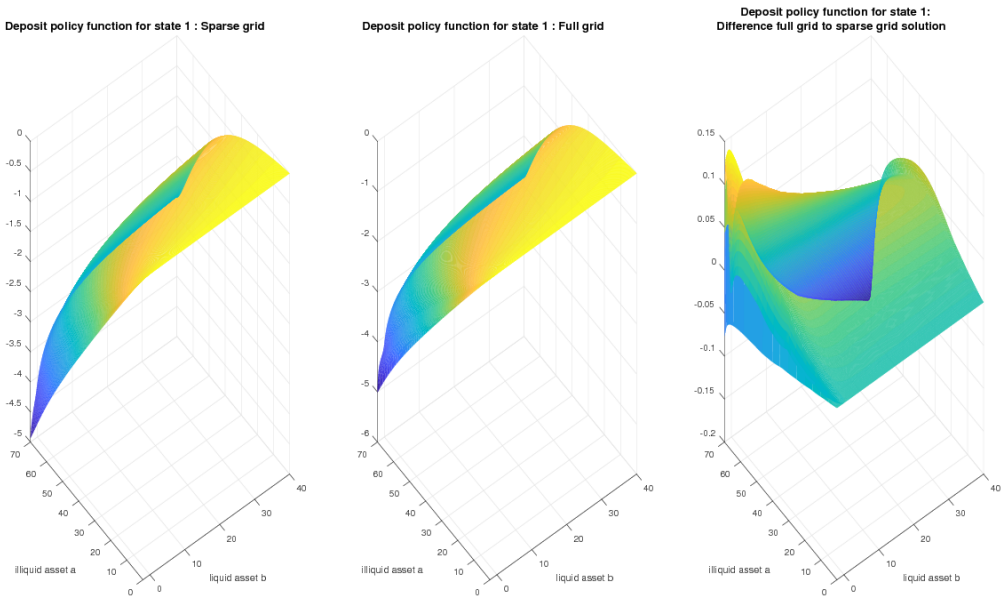


Figure 28: Deposit policy function for state 1: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

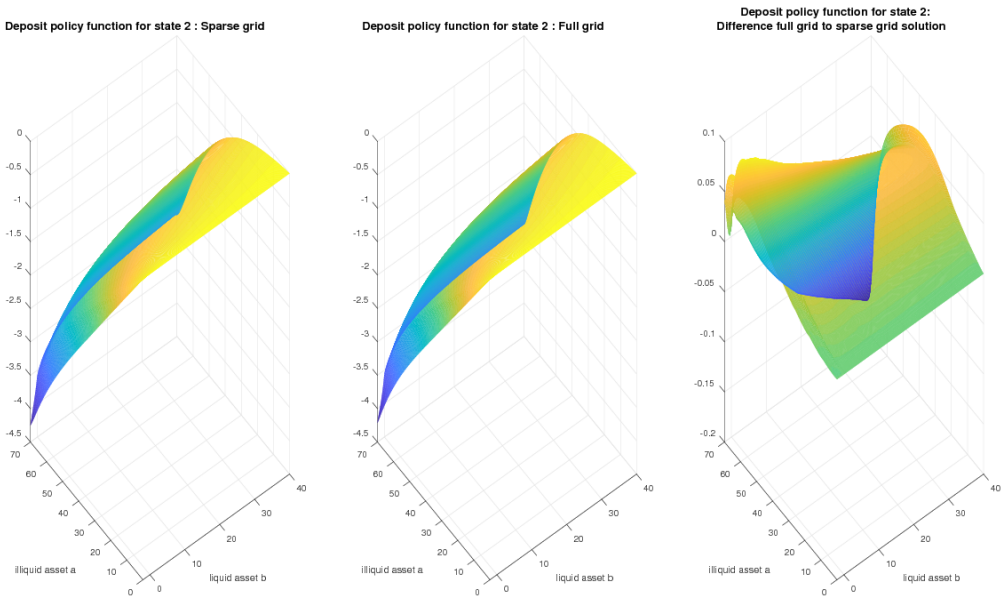


Figure 29: Deposit policy function for state 2: sparse grid approximation, full grid reference solution and the sparse grid approximation subtracted from the full grid solution

In the Figures 22 - 29 we can see that apart from the really steep parts of the functions the sparse grid approximation is able to capture the function behaviors quite well not only for the value function but also for the utility and policy functions.

To get a good approximation without using a really high sparse grid level (and thus many points), we want to use adaptive sparse grids. Our goal is to analyze if it is better to minimize the error of the value function approximation, which then implicitly leads to a better approximation of the policy functions, or if it is better to improve the value function approximation in the area where the policy functions are steep and thus normally not approximated that well. Let us focus our investigation on the deposit function since we observe the biggest errors here. Notice though that the analysis results can be transferred to policy functions in general.

8.1.3 Plots for adaptive sparse grids

Let us visualize the resulting sparse grids and sparse grid approximations for value and deposit function adaptivity respectively. Note that we indicate the grid points by their respective function values as bullet points.

In Figures 30 and 31 the resulting approximations and sparse grid after value function adaptivity are shown. You can see that the sparse grid is refined in the area in which the value function is steep. Note that this is not the area where the deposit function is steep.

The plots shown in Figures 32 and 33 visualize the sparse grid and the resulting functions after using deposit function adaptivity. Notice that the resulting sparse grid looks completely different to the one we obtained by adaptivity with respect to the value function. Now there are more grid points in the area where the deposit function is steep.

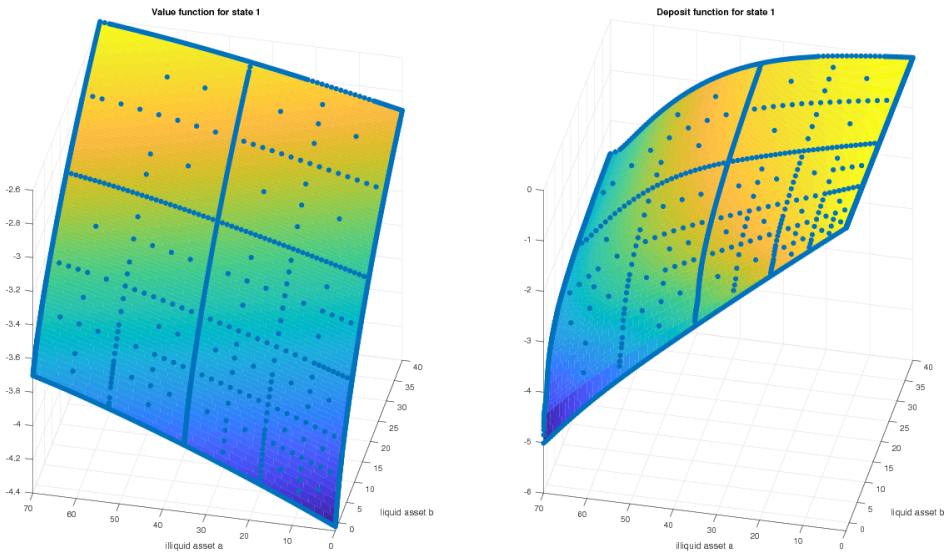


Figure 30: Scatter and surface plot of the value and the deposit function for state 1 for the adapted sparse grid with value function adaptivity

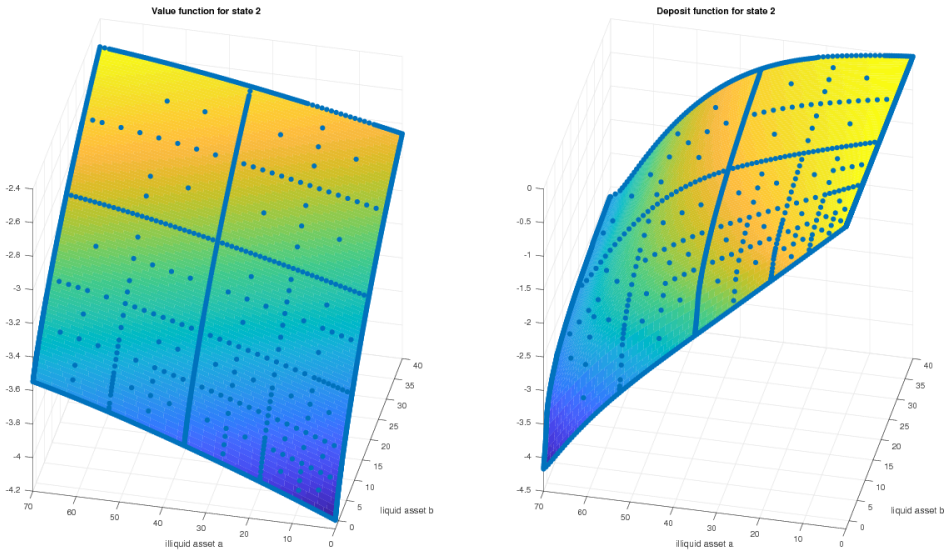


Figure 31: Scatter and surface plot of the value and the deposit function for state 2 for the adapted sparse grid with value function adaptivity

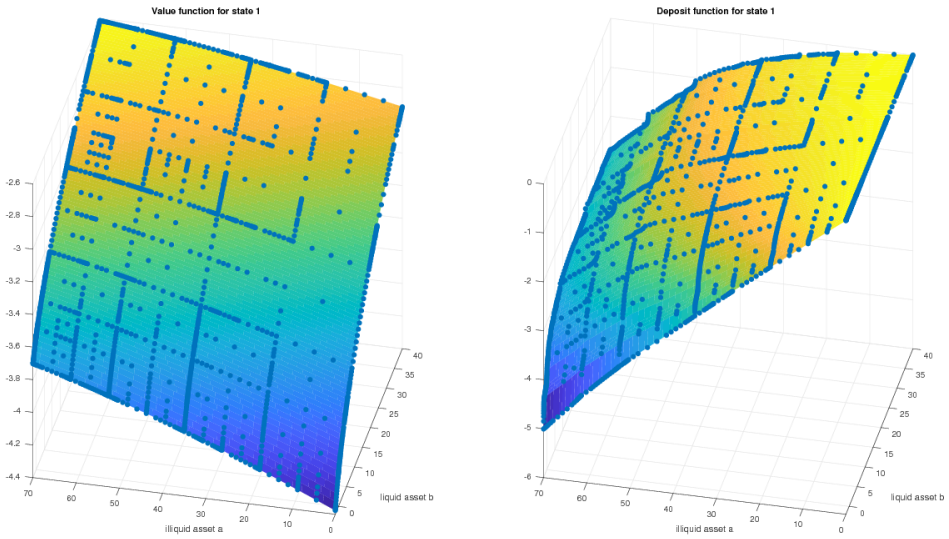


Figure 32: Scatter and surface plot of the value and the deposit function for state 1 for the adapted sparse grid with deposit function adaptivity

Since it is not clear a priori where the sparse grid should be adapted to get a good approximation of the deposit function or policy functions in general, we aim to compare the accuracies resulting from different types of adaptivity to which we turn now.

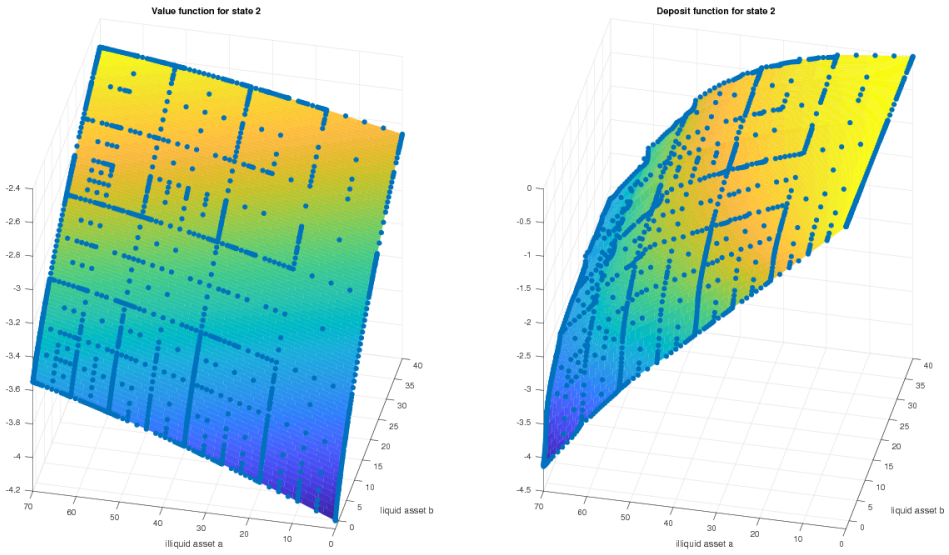


Figure 33: Scatter and surface plot of the value and the deposit function for state 2 for the adapted sparse grid with deposit function adaptivity

8.1.4 Accuracy for adaptive sparse grids

To get more insight into the approximation quality of different types of adaptivity (see Section 7.2 for descriptions) we look into the discrete relative $e_{1,r}$-, $e_{2,r}$- and $e_{\infty,r}$-errors noted in the beginning of this section, see (50). We compute a reference solution on a sparse grid of level $l = 11$ and interpolate both the reference solution and the approximations of the adapted sparse grids and lower level regular sparse grids to uniformly distributed points. We always use the coarsening parameter $\nu = \varepsilon/10$ with respect to the value function.

Note that in all accuracy plots we denote the respective e_r-errors on the y-axis and the number of grid points on the x-axis.

We start by analyzing the adaptivity process by looking at the resulting errors after the iterations of Algorithm 2 before the grid is refined as explained for Algorithm 6. Thus, these are intermediate results that would be final solutions if one stops the algorithm after the respective number of adaption steps.

To visualize the adaption process, we plot intermediate results with stars indicating the results before the respective refinement steps in Figures 34 -35 for different refinement thresholds ε. There are several things we can extract from the error plots. For the value function accuracy, it is easy to see that value function adaptivity works well. For the deposit function accuracy, it strongly depends on the starting level, the adaptive refinement threshold and the number of refinement steps. In Figure 34 we can see that the deposit function accuracy stagnates for the last adaption step for most adaptivity versions, whereas this is not the case for the latter iterations for $\varepsilon = 10^{-5}$ shown in Figure 35. This indicates that refinement with less focus on specific features works better here.

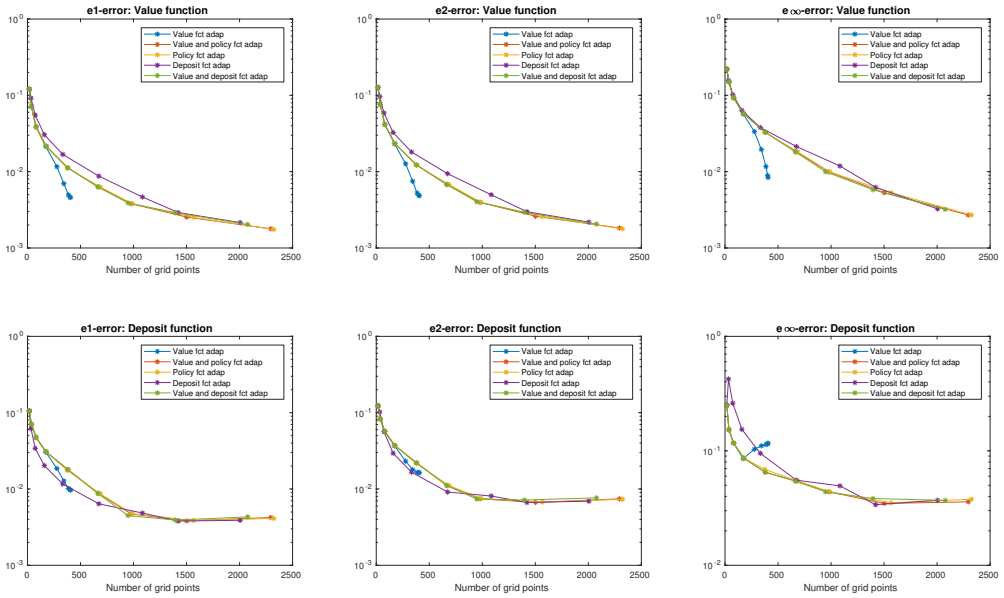


Figure 34: Adaptivity process: accuracy plots of different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-4}$ with stars indicating the results before the respective refinement steps

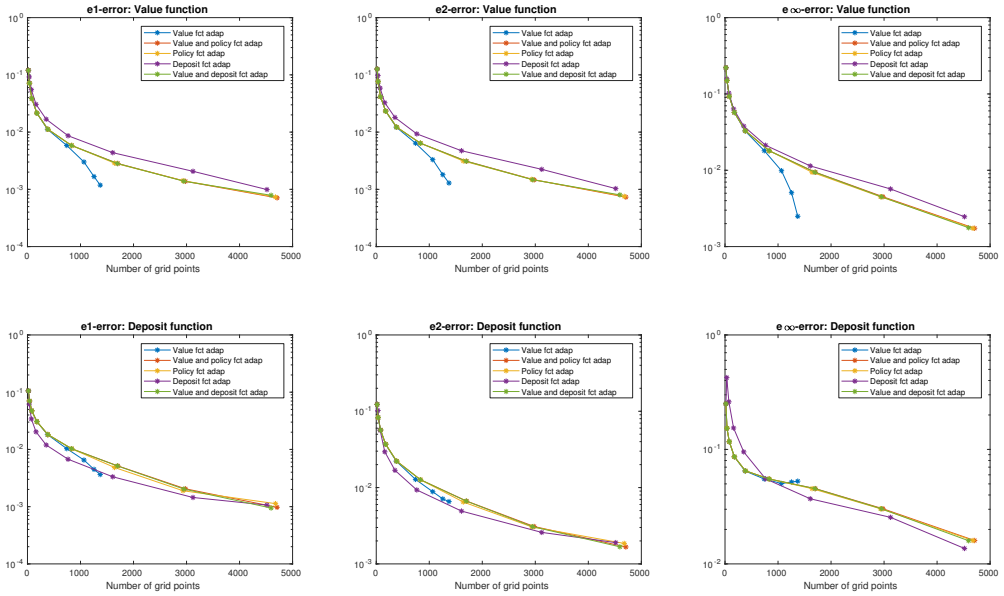


Figure 35: Adaptivity process: accuracy plots of different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-5}$ with stars indicating the results before the respective refinement steps

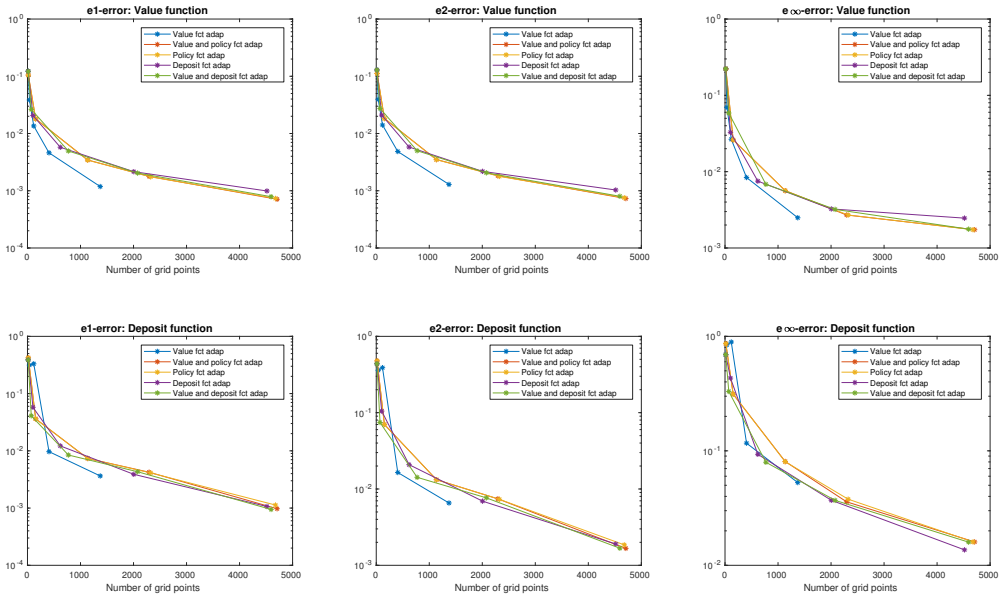


Figure 36: Accuracy plots of different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (stars on the respective lines) after using at most eight adaption steps

To compare the final results of using different refinement thresholds for a fixed maximum number of adaption steps, we present the plots given in Figure 36. We can extract again that value function adaptivity performs well for the value function approximation. In rare cases (here for small ϵ) other adaptivity versions outperform value function adaptivity for the deposit function accuracy.

We point out that it depends on the model parameters if value function adaptivity or deposit function adaptivity is better. Moreover, note that the combined criteria can be suitable in some situations. However, in general if one is not particularly interested in a specific policy function and if one does not want to spend a lot of time on parameter fine-tuning, we strongly recommend value function adaptivity which turns out to be the best approach in most situations. Further, notice that it requires a lot of fine-tuning and testing, or an algorithm for parameter optimization to find a good combination of parameters. Note that self-adaptivity is an approach to lower the refinement threshold automatically. Some resulting accuracies using self-adaptivity are presented in the following paragraph.

Self-adaptivity To give the reader some insight into the effect of using self-adaptivity, we present Table 8 which contains the results of different self-adaptivity processes with the self-adaptivity parameter η, the starting refinement threshold ε_{start}, the final refinement threshold ε_{final}, the resulting number of grid points and the respective e_r-errors. Note that $\eta = 1$ means that we do not use self-adaptivity and thus ε_{final} is equal to ε_{start}. We use the same approach as above for the error computations (with the reference solution of level $l = 11$). You can get multiple insights. First of all notice that by using self

η	ε_{start}	ε_{final}	DOF	$e_{1,r}^{v_1}$	$e_{2,r}^{v_1}$	$e_{\infty,r}^{v_1}$
1	$1.00{\cdot}10^{-1}$	$1.00{\cdot}10^{-2}$	13	$1.22{\cdot}10^{-1}$	$1.29{\cdot}10^{-1}$	$2.23{\cdot}10^{-1}$
0.25	$1.00{\cdot}10^{-1}$	$3.91{\cdot}10^{-4}$	180	$9.40{\cdot}10^{-3}$	$9.90{\cdot}10^{-3}$	$1.82{\cdot}10^{-2}$
1	$1.00{\cdot}10^{-2}$	$1.00{\cdot}10^{-3}$	37	$3.85{\cdot}10^{-2}$	$3.95{\cdot}10^{-2}$	$6.96{\cdot}10^{-2}$
0.25	$1.00{\cdot}10^{-2}$	$1.56{\cdot}10^{-4}$	322	$5.60{\cdot}10^{-3}$	$5.90{\cdot}10^{-3}$	$1.08{\cdot}10^{-2}$
1	$1.00{\cdot}10^{-3}$	$1.00{\cdot}10^{-4}$	118	$1.36{\cdot}10^{-2}$	$1.41{\cdot}10^{-2}$	$2.66{\cdot}10^{-2}$
0.25	$1.00{\cdot}10^{-3}$	$6.25{\cdot}10^{-5}$	254	$6.80{\cdot}10^{-3}$	$7.10{\cdot}10^{-3}$	$1.16{\cdot}10^{-2}$
1	$1.00{\cdot}10^{-4}$	$1.00{\cdot}10^{-4}$	406	$4.50{\cdot}10^{-3}$	$4.80{\cdot}10^{-3}$	$8.20{\cdot}10^{-3}$
0.25	$1.00{\cdot}10^{-4}$	$1.00{\cdot}10^{-4}$	406	$4.50{\cdot}10^{-3}$	$4.80{\cdot}10^{-3}$	$8.20{\cdot}10^{-3}$

Table 8: Accuracy results for different (self-)adaptive sparse grids starting at level $l = 2$ with at most eight adaption steps: self-adaptivity parameter η, starting refinement threshold ε_{start}, final refinement threshold ε_{final}, e_r-errors for the value function for state 1

adaptivity we are not stuck with far less points than initially aimed since the refinement threshold is automatically lowered. Thus, one can stop the algorithm by using the maximum amount of points as stopping criterion instead of e.g. using the maximum number of adaption steps. By starting with a higher refinement threshold ε_{start} in combination with self-adaptivity, one does not run into the risk of adding non-important points due to a refinement threshold that was chosen too low. We point out though that by starting with a low refinement threshold one faces the risk of focusing on a single feature of the function. Second, notice that the adaption yields more points using self-adaptivity with $\varepsilon_{start} = 10^{-2}$ in comparison to $\varepsilon_{start} = 10^{-3}$ even with a fixed number of adaption steps. This shows that an earlier refinement can result in more points at the end. Third, notice that self-adaptivity did not have any effect for the first eight adaption steps with starting refinement threshold $\varepsilon_{start} = 10^{-4}$. Even though self-adaptivity can work well in practice, we restrict our following presentation to adaptivity without self-adaption since it allows a better analysis of different refinement thresholds.

8.2 Four-dimensional model: accuracy analysis and speed comparisons

Let us present our results for the $4d$ model (51) - (52) explained in Appendix A. We give different relative errors for different sparse grid levels and we again compare different adaptivity versions, see subsections 7.1 and 7.2 for the algorithmic approach. Further, we investigate the runtime of Matlabs backslash operator (SuitSparse Umfpack) in comparison to the one of ILUC preconditioned BiCGSTAB, see Section 7.3 for descriptions.

8.2.1 Accuracy

We compute the accuracy for different sparse grid levels and adaptivity versions by using a reference solution that we compute on a higher sparse grid level.

Instead of computing the error on the full grid of the full grid reference solution we compute the error by interpolating on uniformly distributed points for both the reference and the analyzed solutions.

Regular sparse grids Let us begin our accuracy analysis by presenting errors for the value function and all policy functions for regular sparse grids of different levels.

We start by giving the respective e_r-errors and convergence rates for the value function, the deposit a function, the deposit h function and the consumption function. We compute the reference solution on a sparse grid of level $l = 8$.

Level l	Dof	$e^v_{1,r}$	$\rho_{e^v_{1,r}}$	$e^v_{2,r}$	$\rho_{e^v_{2,r}}$	$e^v_{\infty,r}$	$\rho_{e^v_{\infty,r}}$
2	136	$7.12{\cdot}10^{-2}$	–	$7.64{\cdot}10^{-2}$	–	$1.76{\cdot}10^{-1}$	–
3	368	$3.79{\cdot}10^{-2}$	0.91	$4.00{\cdot}10^{-2}$	0.93	$1.05{\cdot}10^{-1}$	0.75
4	961	$8.40{\cdot}10^{-3}$	2.17	$1.02{\cdot}10^{-2}$	1.97	$3.93{\cdot}10^{-2}$	1.42
5	2 441	$5.20{\cdot}10^{-3}$	0.69	$6.20{\cdot}10^{-3}$	0.72	$1.92{\cdot}10^{-2}$	1.03
6	6 065	$2.40{\cdot}10^{-3}$	1.12	$3.00{\cdot}10^{-3}$	1.05	$1.02{\cdot}10^{-2}$	0.91
7	14 801	$1.20{\cdot}10^{-3}$	1	$1.40{\cdot}10^{-3}$	1.1	$4.60{\cdot}10^{-3}$	1.15

Table 9: Accuracy of different sparse grid levels: e_r-errors for the value function

Level l	Dof	$e^{d^a}_{1,r}$	$\rho_{e^{d^a}_{1,r}}$	$e^{d^a}_{2,r}$	$\rho_{e^{d^a}_{2,r}}$	$e^{d^a}_{\infty,r}$	$\rho_{e^{d^a}_{\infty,r}}$
2	136	$1.98{\cdot}10^{-1}$	–	$2.29{\cdot}10^{-1}$	–	$5.55{\cdot}10^{-1}$	–
3	368	$9.80{\cdot}10^{-2}$	1.02	$1.35{\cdot}10^{-1}$	0.76	$5.31{\cdot}10^{-1}$	$6.37 \cdot 10^{-2}$
4	961	$4.53{\cdot}10^{-2}$	1.11	$8.10{\cdot}10^{-2}$	0.74	$4.64{\cdot}10^{-1}$	0.2
5	2 441	$2.96{\cdot}10^{-2}$	0.61	$5.33{\cdot}10^{-2}$	0.6	$3.67{\cdot}10^{-1}$	0.34
6	6 065	$1.18{\cdot}10^{-2}$	1.33	$2.71{\cdot}10^{-2}$	0.98	$2.97{\cdot}10^{-1}$	0.31
7	14 801	$6.50{\cdot}10^{-3}$	0.86	$1.35{\cdot}10^{-2}$	1.01	$1.63{\cdot}10^{-1}$	0.86

Table 10: Accuracy of different sparse grid levels: e_r-errors for the deposit a policy function

Level l	Dof	$e^{d^h}_{1,r}$	$\rho_{e^{d^h}_{1,r}}$	$e^{d^h}_{2,r}$	$\rho_{e^{d^h}_{2,r}}$	$e^{d^h}_{\infty,r}$	$\rho_{e^{d^h}_{\infty,r}}$
2	136	$1.23{\cdot}10^{-1}$	–	$1.64{\cdot}10^{-1}$	–	$5.51{\cdot}10^{-1}$	–
3	368	$5.89{\cdot}10^{-2}$	1.06	$9.26{\cdot}10^{-2}$	0.82	$4.45{\cdot}10^{-1}$	0.31
4	961	$3.07{\cdot}10^{-2}$	0.94	$5.57{\cdot}10^{-2}$	0.73	$3.94{\cdot}10^{-1}$	0.17
5	2 441	$2.15{\cdot}10^{-2}$	0.51	$3.77{\cdot}10^{-2}$	0.56	$3.41{\cdot}10^{-1}$	0.21
6	6 065	$9.20{\cdot}10^{-3}$	1.22	$1.87{\cdot}10^{-2}$	1.01	$2.33{\cdot}10^{-1}$	0.55
7	14 801	$5.00{\cdot}10^{-3}$	0.88	$8.70{\cdot}10^{-3}$	1.1	$1.12{\cdot}10^{-1}$	1.06

Table 11: Accuracy of different sparse grid levels: e_r-errors for the deposit h policy function

Level l	Dof	$e^{c}_{1,r}$	$\rho_{e^{c}_{1,r}}$	$e^{c}_{2,r}$	$\rho_{e^{c}_{2,r}}$	$e^{c}_{\infty,r}$	$\rho_{e^{c}_{\infty,r}}$
2	136	$2.98{\cdot}10^{-1}$	–	$3.29{\cdot}10^{-1}$	–	$9.16{\cdot}10^{-1}$	–
3	368	$1.13{\cdot}10^{-1}$	1.4	$1.45{\cdot}10^{-1}$	1.18	$6.58{\cdot}10^{-1}$	0.48
4	961	$3.41{\cdot}10^{-2}$	1.72	$6.82{\cdot}10^{-2}$	1.09	$5.38{\cdot}10^{-1}$	0.29
5	2 441	$2.07{\cdot}10^{-2}$	0.72	$4.09{\cdot}10^{-2}$	0.74	$4.34{\cdot}10^{-1}$	0.31
6	6 065	$9.60{\cdot}10^{-3}$	1.11	$2.24{\cdot}10^{-2}$	0.87	$3.27{\cdot}10^{-1}$	0.41
7	14 801	$4.20{\cdot}10^{-3}$	1.19	$9.60{\cdot}10^{-3}$	1.22	$1.31{\cdot}10^{-1}$	1.32

Table 12: Accuracy of different sparse grid levels: e_r-errors for the consumption policy function

First, notice that for all of the sparse grid levels presented in Tables 9 - 12 the algorithm converges to the reference solution computed for level $l = 8$. Notice that by adding levels we get a higher accuracy for all error measures for all functions. Thus, by adding more levels the algorithm converges (in the sense of going to a higher sparse grid level) to the reference solution (for our model and algorithm parameters). Further, notice that the convergence rates vary for the different functions, e.g. going from level $l = 6$ to level $l = 7$ increases the accuracy for the consumption function more than the ones for the deposit functions. Moreover, notice that the e_∞-error of the policy functions only is reduced slightly by going to higher sparse grid levels (apart from going from level $l = 6$ to level $l = 7$). Additionally, we can see that the convergence is not stagnating for the presented levels and it thus pays off to go to higher sparse grid levels.

Adaptive sparse grids Let us turn to the analysis of different adaptivity approaches. We again restrict our presentation of adaptivity to the value function and the deposit functions. We compare the results for value function adaptivity, deposit function adaptivity and by logical OR combined value and deposit function adaptivity. We compute the reference solution on a sparse grid of level $l = 8$ and do not add points which are not in this grid in our adaption by limiting the maximum number of adaption steps. We do this because we do not want our adaptive methods to outperform our reference solution which could yield

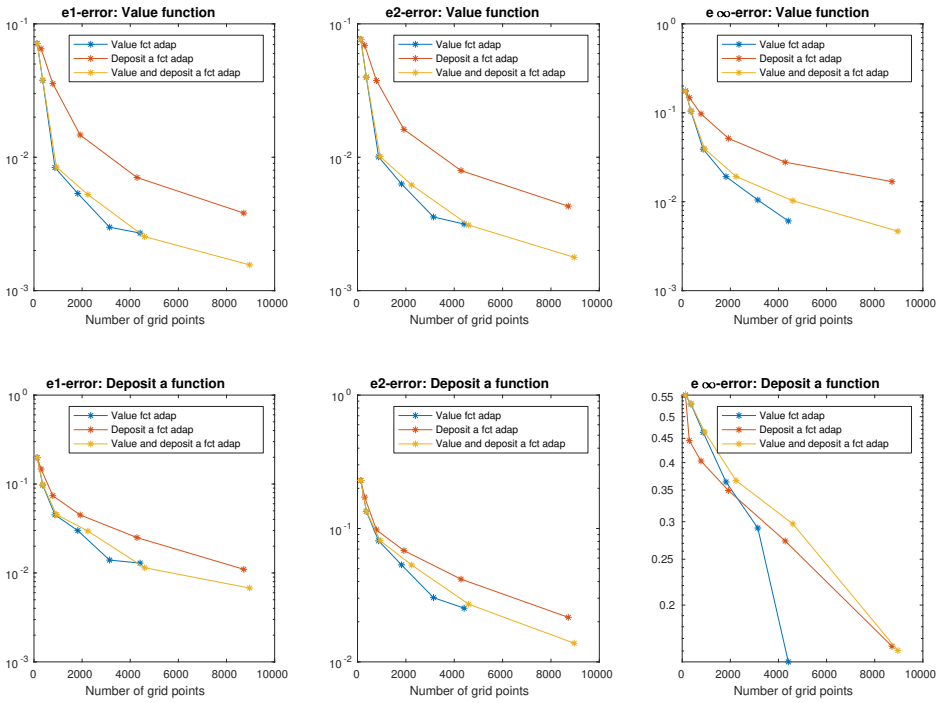


Figure 37: Adaptivity process: accuracy plots for different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-4}$

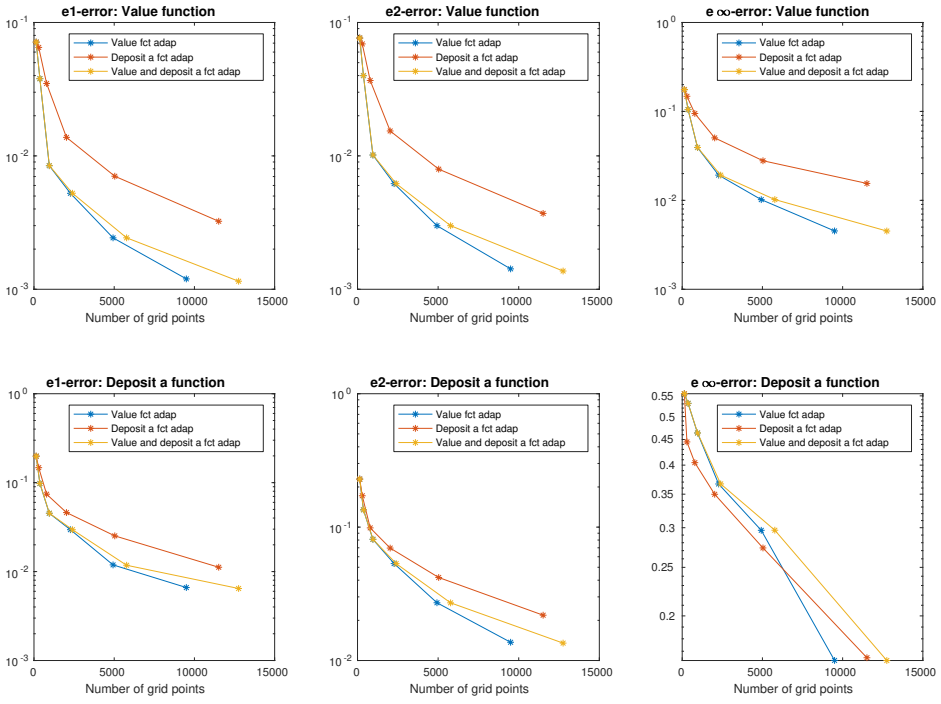


Figure 38: Adaptivity process: accuracy plots for different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-5}$

wrong insights. Note though that due to better "initial" guesses after refining the grid, the adaptive algorithm could still slightly outperform the higher level regular sparse grid one in rare situations.

For all adaptivity versions, we start with level $l = 2$ and do not use self-adaptivity to give a better presentation of different refinement thresholds ε. Notice that for all experiments we use the coarsening parameter $\nu = \varepsilon/10$ with respect to the value function. We again denote the relative errors on the y-axis and the number of grid points on the x-axis.

We compare different adaptivity versions, in particular value function adaptivity, deposit function adaptivity and with logical OR combined value and deposit function adaptivity. Figures 37 and 38 contain plots that show the e_r-errors for different refinement thresholds ε arising in the respective adaptions. To visualize the adaption process, we plot intermediate results with stars indicating the results before the respective refinement steps. You can see that in most cases value function adaptivity outperforms the other adaptivity versions for both the value function and the deposit function. We should point out that deposit function adaptivity works well to get a low e_∞-error of the deposit function with a small number of points.

To compare the final results of using different refinement thresholds for

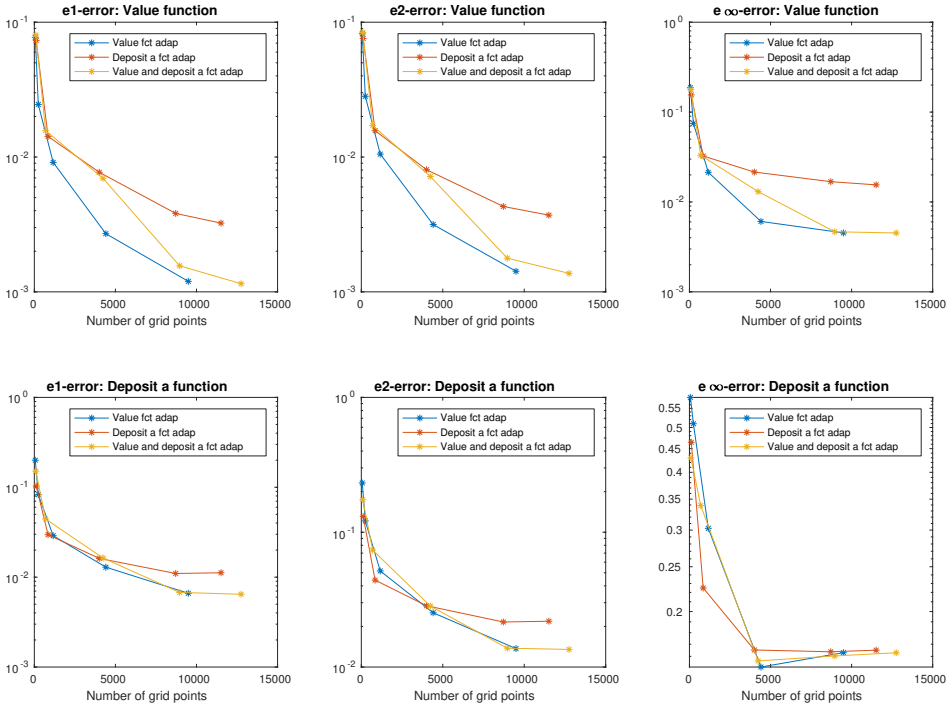


Figure 39: Accuracy plots of different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (indicated by stars on the respective lines) after using at most five adaption steps

a fixed number of maximum number of adaption steps, we present the plots in Figure 39. Notice that all adaptivity versions work well for the deposit function approximation. However, for the value function approximation, one can see that value function adaptivity works better than the other adaptivities. We further point out that that the error is decreasing for lower refinement thresholds and thus more points, i.e. the additional points pay off to get a better approximation. Thus, either adding even more points by using a lower refinement threshold or using more adaption steps could pay off.

8.2.2 Runtime

All of the following experiments are done on a machine of type PowerEdge R900 with 24x Intel(R) Xeon(R) CPU X7460 with 2.66 GHz, 256 GB Ram and an ATI Technologies Inc ES1000 (rev 02) graphics card.

As explained in Section 7.3, we restrict our runtime analysis to the comparison of Matlabs backslash operator and ILUC preconditioned BiCGSTAB.

There is no descriptive convergence bound for BiCGSTAB (and also for other Krylov subspace solvers) for non-symmetric matrices. Nevertheless, often the distribution of the eigenvalues and the condition numbers gives some hindsight into the performance of an iterative solver. Let us present the eigenvalues of the system matrix for different sparse grid levels. The eigenvalues arise that for the first iteration of Algorithm 2 for the $4d$ model (51) - (52) are shown in Figure 40. One can easily see in that there is a cluster of eigenvalues around $(0, 0)$ but more importantly notice that the eigenvalue range grows with rate $\mathcal{O}(h^{-2})$ where h denotes the mesh width on the boundary. The condition numbers are given in Table 13.

Now let us look at the effect of preconditioning with the ILUC factorization by investigating the condition numbers for the 2-norm. You can see in Table 14 that with ILUC the condition numbers are far lower. We are now interested in the convergence of BiCGSTAB, in particular in the computation time but

Level l	Condition: 1-norm	Condition: 2-norm
0	200.6458	67.4618
1	644.2919	143.7569
2	$3.2046 \cdot 10^3$	484.1335
3	$1.4978 \cdot 10^4$	$2.0532 \cdot 10^3$
4	$1.3034 \cdot 10^5$	$1.6303 \cdot 10^4$
5	$1.9338 \cdot 10^6$	$1.7799 \cdot 10^5$
6	$3.5727 \cdot 10^7$	$1.9734 \cdot 10^6$
7	$6.8861 \cdot 10^8$	$2.3317 \cdot 10^7$

Table 13: Condition numbers of the system matrix for different sparse grid levels arising in the first iteration of Algorithm 2 for the $4d$ model (51) - (52)

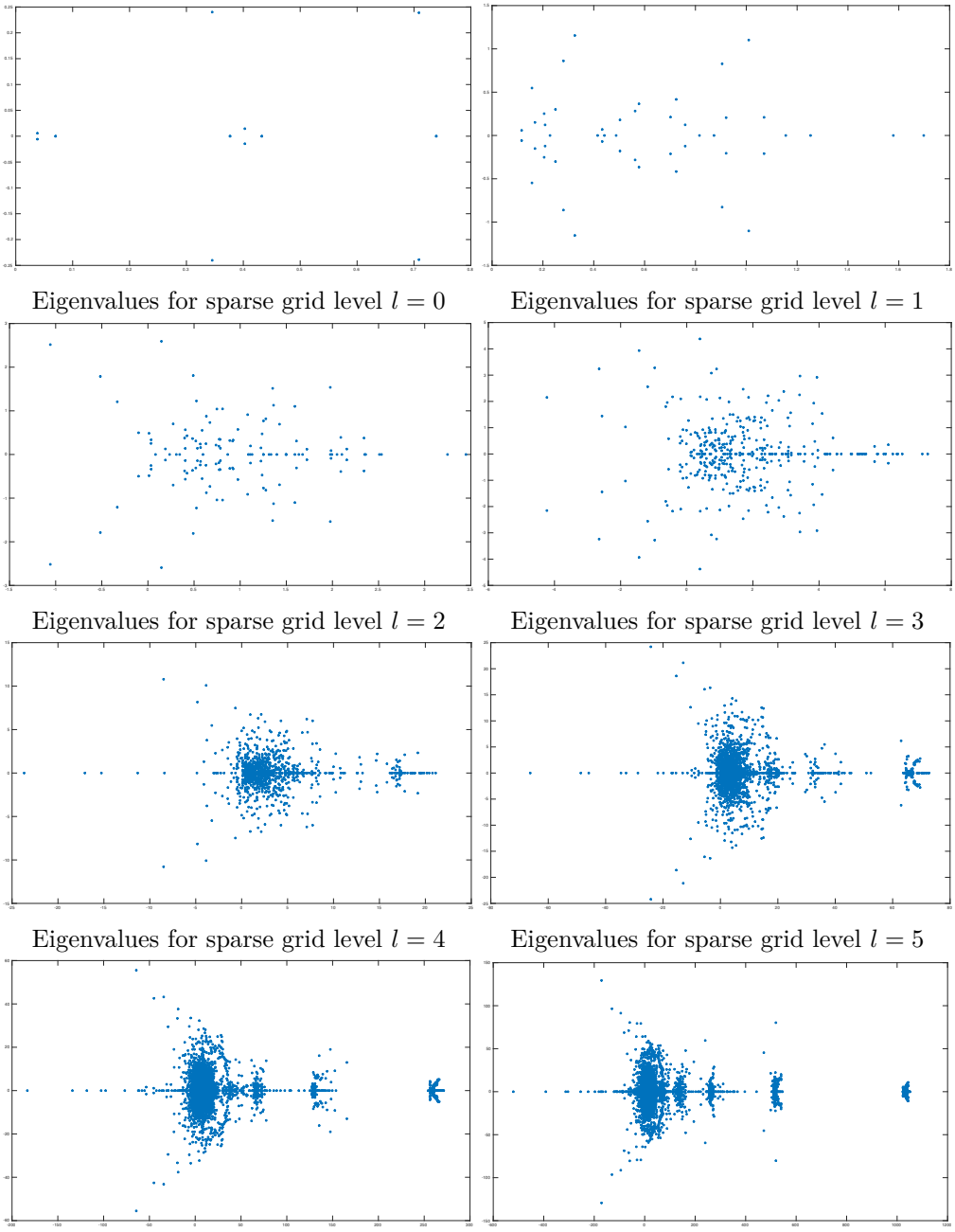


Figure 40: Eigenvalues for different sparse grid levels arising in the first iteration of Algorithm 2 for the $4d$ model (51) - (52)

also in the number of iterations. Even though we presented the eigenvalues for lower levels we are especially interested in higher dimensions. Let us thus look at the convergence behavior of preconditioned BiCGSTAB (precBiCGSTAB) and compare the computation times with the ones of the direct solver.

Level l	Condition: ILUC(10^{-3})	Condition: ILUC(10^{-6})
0	1	1
1	1.0750	1.0348
2	1.0593	1
3	1.3483	1.0002
4	6.0024	1.0008
5	314.1479	1.0066
6	$1.6498 \cdot 10^4$	1.1531

Table 14: Condition numbers in the 2-norm of the ILUC-preconditioned system matrix for different sparse grid levels and different drop tolerances arising in the first iteration of Algorithm 2 for the $4d$ model (51) - (52)

We give the results for ILUC with BiCGSTAB with ILUC computed in every iteration in comparison to Umfpack. We denote the setup time of ILUC(10^{-3}) by SET, the iterative solver (BiCGSTAB) time by ITSOL, the total time TOT = SET + ITSOL and the computation time of the direct solver by DIR.

In Table 15 we can see that the setup time takes up a large part of the total computation time, and thus reducing this time by not recomputing the ILUC factorization every iteration may yield improvements. Let us look into this by comparing the average computation times for solving the linear system for the respective sparse grid levels (and the respective number of grid points). We use Algorithm 7 with parameters given in Appendix A to compute ILUC less often.

Let us turn to the solver runtime presented in Figure 41 where the stars indicate the different sparse grid levels $l = 1, \ldots, 8$. The plot shows that using an iterative solver instead of a direct one pays off for higher sparse grid levels. Additionally, notice that reducing the number of ILUC recomputations increases the speed only by a small margin.

The reduced runtime for solving the linear system also reflects in the total computation time of Algorithm 2 which is visualized in Figure 42. Notice that

Level l	SET	ITSOL	TOT	DIR
1	0.003	0.010	0.013	0.092
2	0.001	0.006	0.007	0.002
3	0.007	0.007	0.014	0.024
4	0.048	0.013	0.061	0.106
5	0.301	0.090	0.391	0.410
6	2.229	0.593	2.822	21.515
7	18.131	4.827	22.958	52.579
8	138.995	51.870	190.865	527.94

Table 15: Runtimes for different levels for ILUC preconditioned BiCGSTAB in comparison to the SuitSparse Umfpack direct solver

the total computation time grows less than exponentially with respect to the number of grid points. We point out that solving the linear system is the main bottleneck of Algorithm 2 for the $4d$ model (51)-(52). Since the linear system

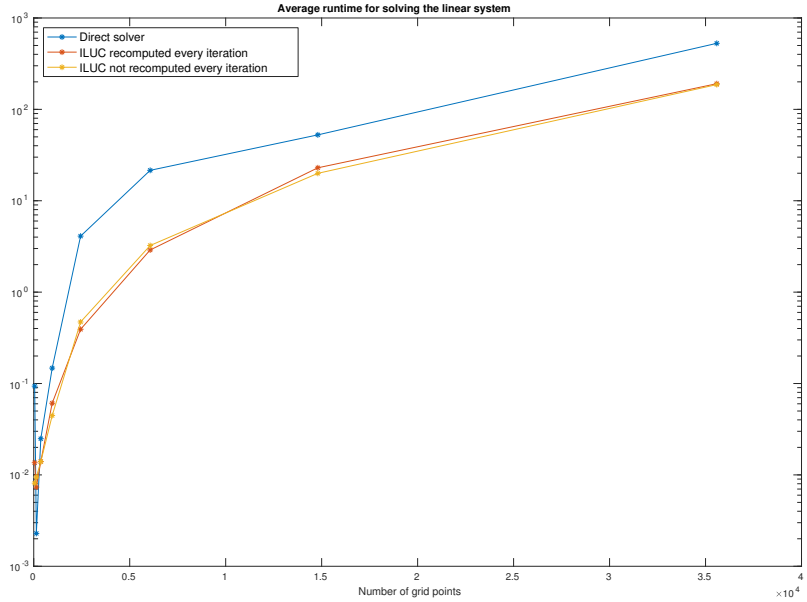


Figure 41: Average computation times of different methods for solving the arising linear system for levels $l = 1, \ldots, 8$ (indicated by stars on the respective lines)

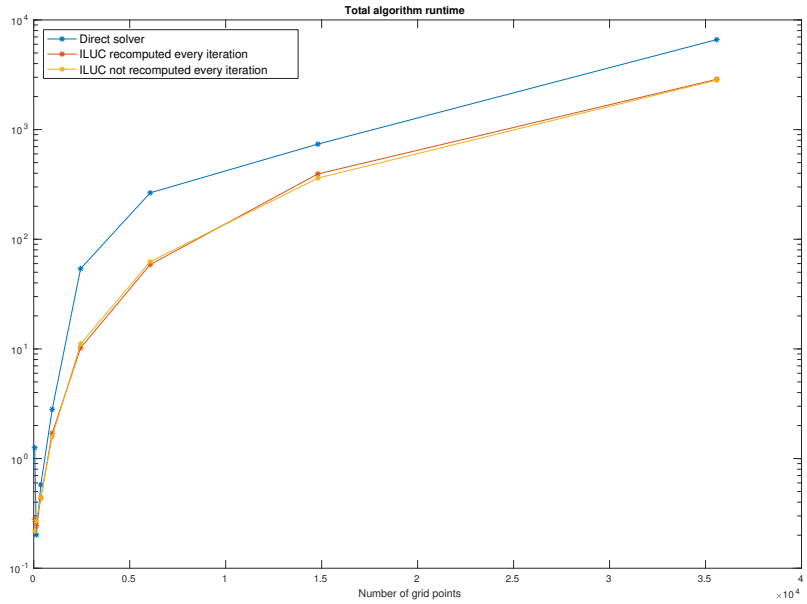


Figure 42: Total computation time for the Algorithm 2 for different methods for solving the arising linear system for levels $l = 1, \ldots, 8$ (indicated by stars on the respective lines)

has to be solved in every iteration the plot looks similar to the one in Figure 41.

Let us give the average number of iterations of precBiCGSTAB to share some insight into how well ILUC preconditioning works. First and most importantly, we can see in Figure 43 that the required number of iterations is always really low and thus the preconditioning works well in the sense that the preconditioned system is easy to solve. Thus, the main costs do not arise from a high number of iterative solver iterations but from the ILUC construction and its application within the iterative solver. Moreover, notice that the required amount of iterations grows with the number of grid points. On average the number of required BiCGSTAB iterations is higher when we recompute the ILUC factorization less often but the numbers are still acceptable. We point out that this number does not grow with the sparse grid level since it depends on when and how often ILUC is recomputed.

To give some insight when ILUC is recomputed, we show in which iterations it is done for the respective levels $l = 1, \ldots, 8$. Figure 44 shows that in the first iterations we have to recompute the ILUC more often, whereas in the last iterations we can use the same ILUC factorization for multiple iterations. We interpret this as this being mainly due to the better initial guess for the iterative solver in these iterations. Thus, it can be efficient to simply recompute ILUC in the first iterations and just follow Algorithm 7 for the remaining iterations. Further, note that we have to recompute ILUC more often when we go to higher sparse grid levels. Note that we never need to recompute ILUC in the last iterations and one could thus use more iterations of Algorithm 2 without

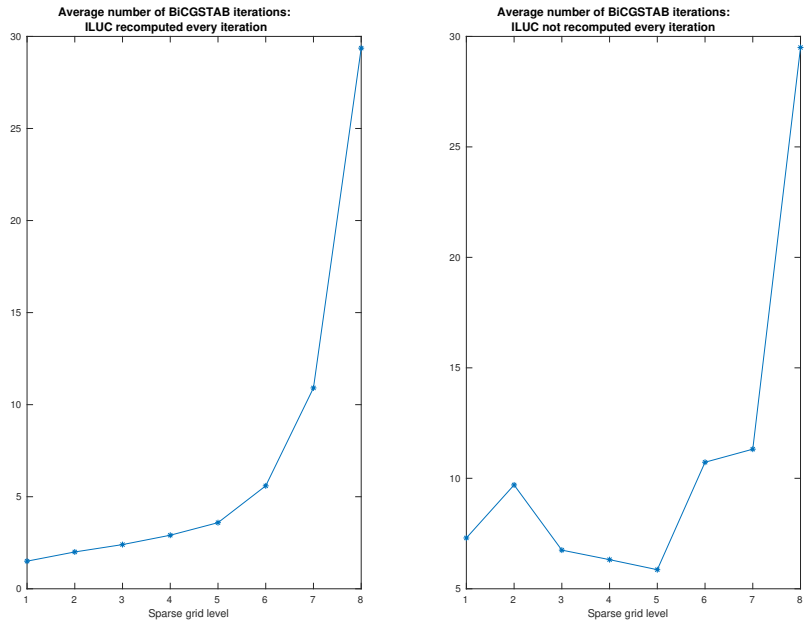
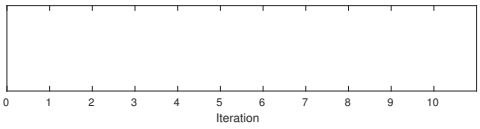
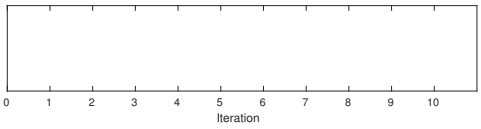


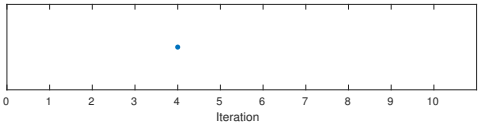
Figure 43: Average numbers of required iterations of precBiCGSTAB for different levels



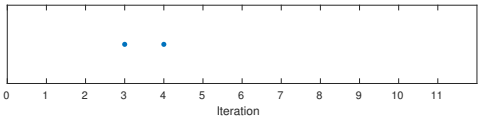
Iterations in which ILUC is recomputed for level $l = 1$



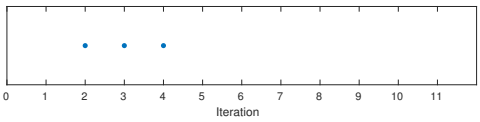
Iterations in which ILUC is recomputed for level $l = 2$



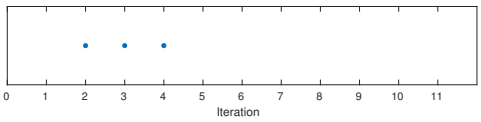
Iterations in which ILUC is recomputed for level $l = 3$



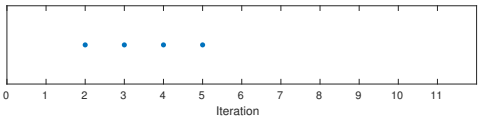
Iterations in which ILUC is recomputed for level $l = 4$



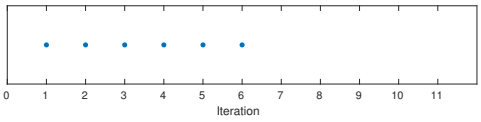
Iterations in which ILUC is recomputed for level $l = 5$



Iterations in which ILUC is recomputed for level $l = 6$



Iterations in which ILUC is recomputed for level $l = 7$



Iterations in which ILUC is recomputed for level $l = 8$

Figure 44: Iterations of Algorithm 2 in which ILUC is recomputed for level using Algorithm 7 for solving the linear system, given for sparse grid levels $l = 1, \ldots, 8$

high recomputation costs.

8.3 Six-dimensional model: accuracy analysis and speed comparisons

Let us present our results for the $6d$ model (53) - (54) explained in Appendix A. In the beginning, we do an accuracy analysis for regular sparse grids, see Algorithm 2 presented in Section 7.1 for our approach. We then compare the different adaptivity versions and the runtime of Matlabs backslash operator (SuitSparse Umfpack) to the one of ILUC preconditioned BiCGSTAB, see Section 7.2 and 7.3 respectively for descriptions. This section is similarly structured to Section 8.2 with numerical results for the $4d$ model (51) - (52) and we aim to particularly point out the differences and similarities of the results for the $6d$ model to the results presented therein.

8.3.1 Accuracy

We again compute the accuracy for different sparse grid levels and adaptivity versions by using a reference solution that we compute on a higher sparse grid

level. As in the last subsection, we interpolate on uniformly distributed points for both the reference and the analyzed function for the error computations.

Regular sparse grids Let us begin our accuracy analysis by presenting relative errors for the value function and all policy functions for regular sparse grids of different levels. We compute the reference solution on a sparse grid of level $l = 6$.

The respective e_r-errors and convergence rates for the value function, the deposit function and the consumption function are given in Tables 16 - 18. One can see that by adding levels we achieved a higher accuracy for all error measures for all functions. Thus, the algorithm converges, in the sense of going to higher sparse grid levels, to the reference solution (for our model and algorithm parameters). Further, note that the convergence rates indicate that higher sparse grid levels pay off since they are not stagnating for the presented

Level l	Dof	$e^v_{1,r}$	$\rho_{e^v_{1,r}}$	$e^v_{2,r}$	$\rho_{e^v_{2,r}}$	$e^v_{\infty,r}$	$\rho_{e^v_{\infty,r}}$
1	256	$1.42{\cdot}10^{-1}$	–	$1.45{\cdot}10^{-1}$	–	$2.25{\cdot}10^{-1}$	–
2	880	$8.62{\cdot}10^{-2}$	0.72	$8.90{\cdot}10^{-2}$	0.71	$1.55{\cdot}10^{-1}$	0.54
3	2 768	$4.52{\cdot}10^{-2}$	0.93	$4.72{\cdot}10^{-2}$	0.91	$9.32{\cdot}10^{-2}$	0.73
4	8 205	$1.79{\cdot}10^{-2}$	1.34	$1.88{\cdot}10^{-2}$	1.33	$4.32{\cdot}10^{-2}$	1.11
5	23 288	$8.30{\cdot}10^{-3}$	1.11	$8.80{\cdot}10^{-3}$	1.09	$1.98{\cdot}10^{-2}$	1.13

Table 16: Accuracy of different sparse grid levels: e_r-errors for the value function

Level l	Dof	$e^d_{1,r}$	$\rho_{e^d_{1,r}}$	$e^d_{2,r}$	$\rho_{e^d_{2,r}}$	$e^d_{\infty,r}$	$\rho_{e^d_{\infty,r}}$
1	256	$1.33{\cdot}10^{-1}$	–	$1.55{\cdot}10^{-1}$	–	$3.08{\cdot}10^{-1}$	–
2	880	$8.74{\cdot}10^{-2}$	0.61	$1.01{\cdot}10^{-1}$	0.61	$2.09{\cdot}10^{-1}$	0.56
3	2 768	$5.44{\cdot}10^{-2}$	0.68	$6.47{\cdot}10^{-2}$	0.64	$1.25{\cdot}10^{-1}$	0.74
4	8 205	$2.90{\cdot}10^{-2}$	0.91	$3.65{\cdot}10^{-2}$	0.83	$8.90{\cdot}10^{-2}$	0.49
5	23 288	$1.23{\cdot}10^{-2}$	1.23	$1.44{\cdot}10^{-2}$	1.34	$3.05{\cdot}10^{-2}$	1.54

Table 17: Accuracy of different sparse grid levels: e_r-errors for the deposit policy function

Level l	Dof	$e^c_{1,r}$	$\rho_{e^c_{1,r}}$	$e^c_{2,r}$	$\rho_{e^c_{2,r}}$	$e^c_{\infty,r}$	$\rho_{e^c_{\infty,r}}$
1	256	$2.70{\cdot}10^{-1}$	–	$2.80{\cdot}10^{-1}$	–	$4.13{\cdot}10^{-1}$	–
2	880	$1.53{\cdot}10^{-1}$	0.82	$1.58{\cdot}10^{-1}$	0.82	$2.59{\cdot}10^{-1}$	0.67
3	2 768	$7.52{\cdot}10^{-2}$	1.02	$8.02{\cdot}10^{-2}$	0.98	$1.85{\cdot}10^{-1}$	0.48
4	8 205	$2.92{\cdot}10^{-2}$	1.36	$3.20{\cdot}10^{-2}$	1.33	$9.72{\cdot}10^{-2}$	0.93
5	23 288	$1.31{\cdot}10^{-2}$	1.16	$1.52{\cdot}10^{-2}$	1.07	$5.78{\cdot}10^{-2}$	0.75

Table 18: Accuracy of different sparse grid levels: e_r-errors for the consumption policy function

levels.

Adaptive sparse grids Again, we compare different adaptivity approaches by computing a reference solution on a sparse grid of level $l = 6$. Note further that we do not add points which are not in this grid in our adaption by limiting the maximum number of adaption steps. Further, we again denote the relative errors on the y-axis and the number of grid points on the x-axis.

We compare value function adaptivity, deposit function adaptivity and with logical OR combined value and deposit function adaptivity. To visualize the adaption process, we again plot intermediate results with stars indicating the results before the respective refinement steps. The plots in the Figures 45 - 48 show the e_r-errors for different refinement thresholds ε arising in the respective adaptions. You can see that for the presented refinement thresholds value function adaptivity outperforms the other adaptivity versions for value function approximation accuracy. For deposit function accuracy, one sometimes gets a lower e_1-error with deposit function adaptivity for a low number of adaption steps. However, in general value function approximation works better than the other presented adaptivity types.

To compare the final results of using different refinement thresholds for a fixed number of maximum number of adaption steps, we present the plots in Figure 49. As in the lower dimensional experiments, value function adaptivity

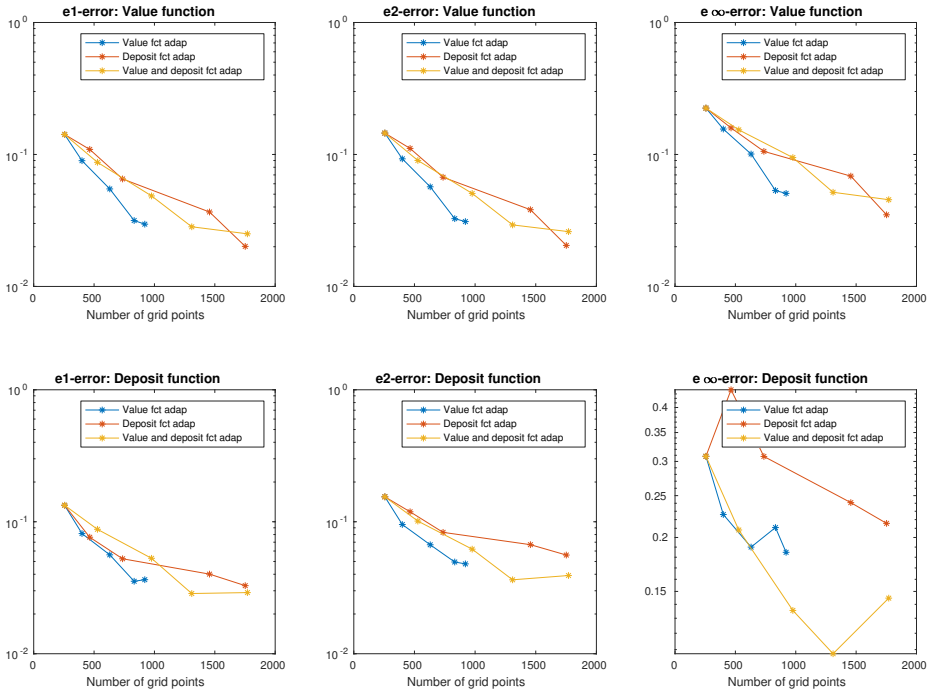


Figure 45: Adaptivity process: accuracy plots for different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-2}$

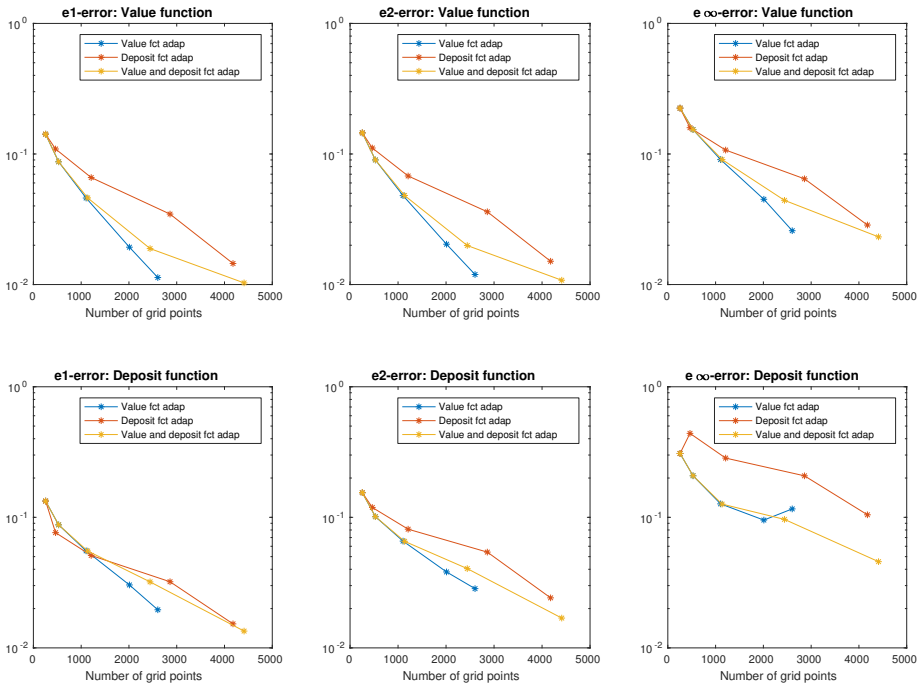


Figure 46: Adaptivity process: accuracy plots for different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-3}$

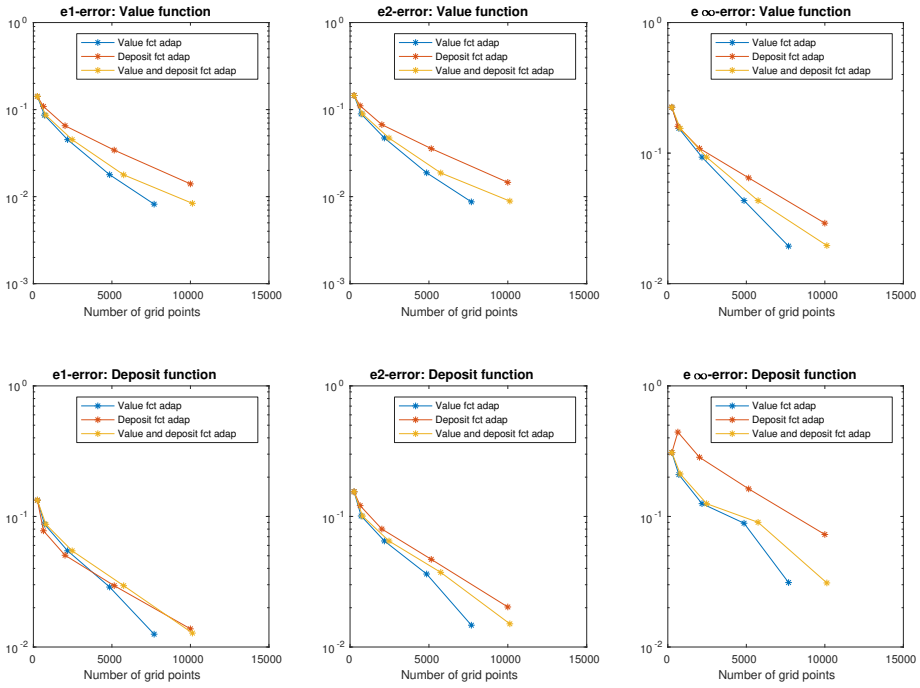


Figure 47: Adaptivity process: accuracy plots for different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-4}$

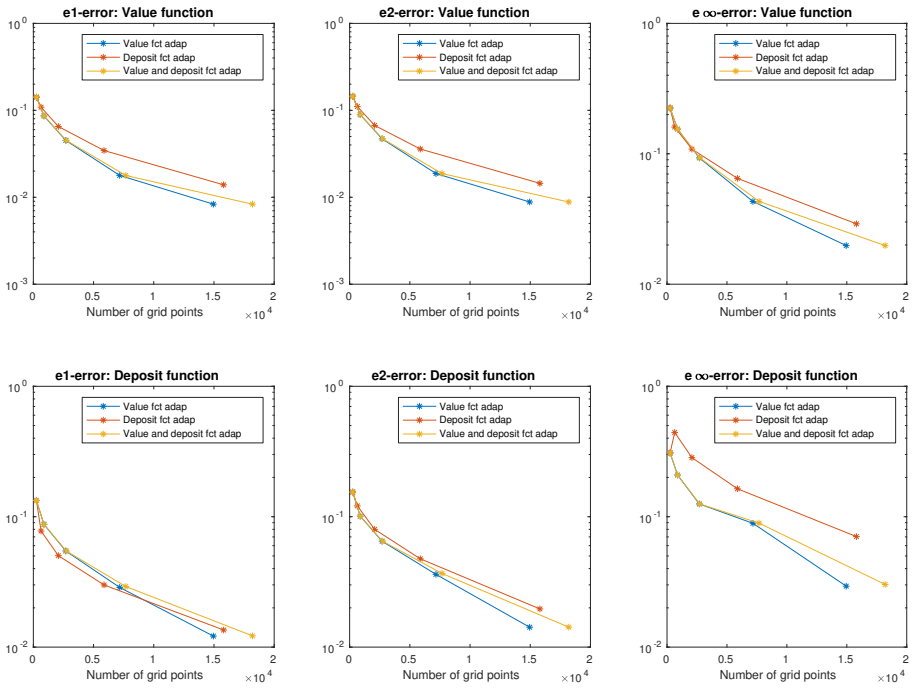


Figure 48: Adaptivity process: accuracy plots for different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-5}$

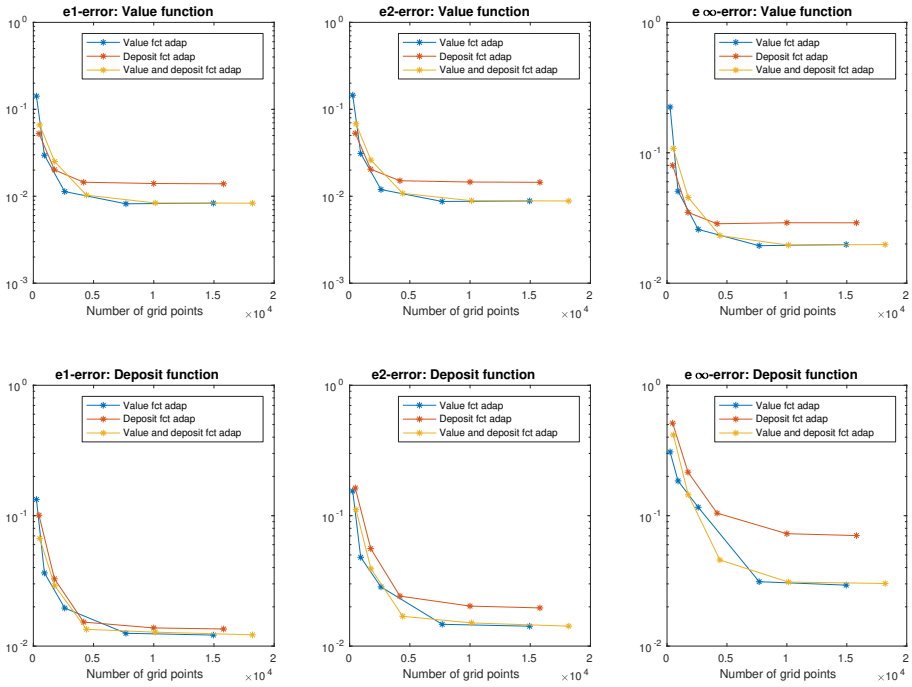


Figure 49: Accuracy plots of different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (indicated by stars on the respective lines) after using at most four adaption steps

works better than the other adaptivity types for the value function approximation. For the deposit function on the other hand, the combined adaptivity of value and deposit function adaptivity also yields good results. Additionally, we point out that for refinement thresholds $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-5}$ the accuracy does not increase even though many more points are used, i.e. the convergence is stagnating in this described sense. Therefore, one should start with a smaller refinement threshold and make use of self-adaptivity to add points that help for a better approximation quality.

8.3.2 Runtime

All of the following experiments are done on a machine of type PowerEdge R920 with 48x Intel(R) Xeon(R) CPU E7-4850 v2 with 2.3 GHz, 1.5 TB Ram and a Matrox Electronics Systems Ltd. G200eR2 graphics card.

In Table 19, results are given when we recompute the ILUC preconditioner in every iteration. All runtimes are given in seconds and as averages over all HJB iterations for the respective sparse grid levels. One can see that using ILUC preconditioned BiCGSTAB allows to compute solutions in higher dimensions in acceptable time. Notice that e.g. for level $l = 6$ we need less than a minute in comparison to multiple hours for the direct solver. As for the $4d$ model, we can see that the setup time takes a large part of the total computation time and thus reducing this time by not recomputing the ILUC factorization every iteration may yield improvements. Let us look into this by comparing the average computation times for solving the linear system for the respective sparse grid levels (and the respective number of grid points). We use Algorithm 7 with parameters given in Appendix A to compute ILUC less often.

You can see in Figure 50 that using an iterative solver instead of a direct one pays off for higher sparse grid levels. Notice that the difference is even bigger for $6d$ model than for the $4d$ model. In contrast to the solver runtime for the $4d$ model, reducing the number of ILUC recomputations increases the speed by quite a bit.

This average runtime of solving the linear system also reflects in the total

Level l	SET	ITSOL	TOT	DIR
1	0.011	0.033	0.044	0.087
2	0.014	0.006	0.021	0.056
3	0.142	0.031	0.173	0.575
4	1.105	0.158	1.263	33.929
5	6.302	1.022	7.324	514.093
6	35.615	7.128	42.743	9580.439

Table 19: Runtimes for different levels for ILUC preconditioned BiCGSTAB in comparison to the SuitSparse Umfpack direct solver

computation time of Algorithm 2. Notice that Figure 51 shows that the total computation time grows again less than exponentially with respect to the number of grid points. You can see that solving the linear system more efficiently pays off and results in a big reduction of the total computation time. This is due to the fact that solving the linear system is one of the main bottlenecks of Algorithm 2 for the $6d$ model (53)-(54). We point out that there is another bottleneck which is also on of the reasons that the difference of the different solver approaches is less notable than in Figure 50 besides the logarithmic scale. This bottleneck is the computation of the finite difference operators since we have to compute the forward and backward difference operator for every dimension and thus twelve operators for the first derivative approximations. Notice though that this is easily parallelizable by simply computing all of them at the same time (e.g. one per core) instead of one after another. Hence, solving the linear system is the only main bottleneck again.

Let us turn to the average number of iterations of precBiCGSTAB. First and most importantly, we can see in Figure 52 that the required number of iterations are always really low and thus the preconditioning works well in the sense that the preconditioned system is easy to solve. Thus, the main costs again do not arise from a high number of iterative solver iterations but from the ILUC construction and its application within the iterative solver. Further, notice that the required amount of iterations grows with the number of grid points. On average, the number of required BiCGSTAB iterations is higher when we recompute the ILUC factorization less often but the numbers are still acceptable. We point out that this number does not grow with the sparse grid

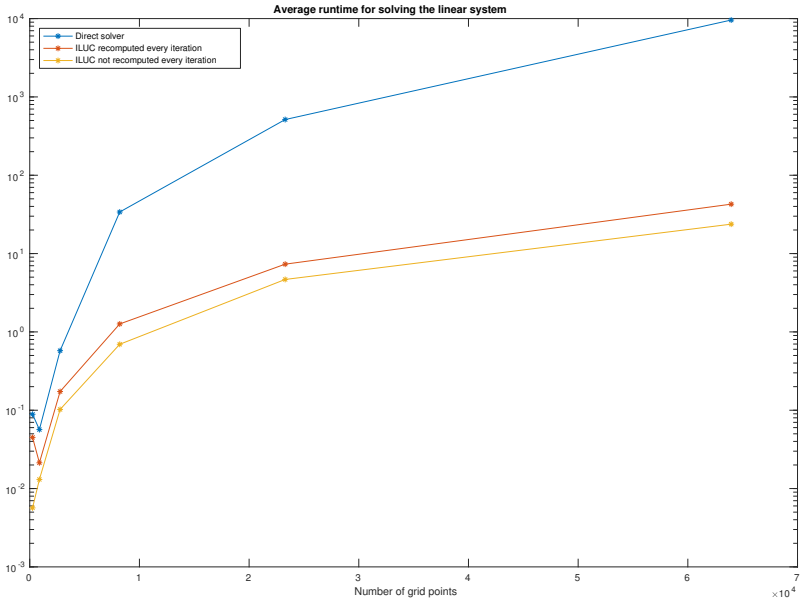


Figure 50: Average computation times of different methods for solving the arising linear system for levels $l = 1, \ldots, 6$ (indicated by stars on the respective lines)

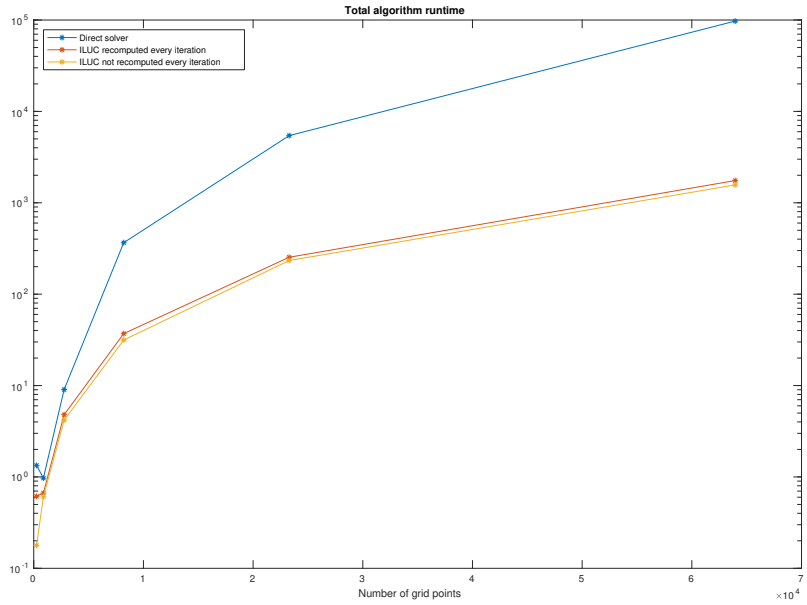


Figure 51: Total computation time for the Algorithm 2 for different methods for solving the arising linear system for levels $l = 1, \ldots, 6$ (indicated by stars on the respective lines)

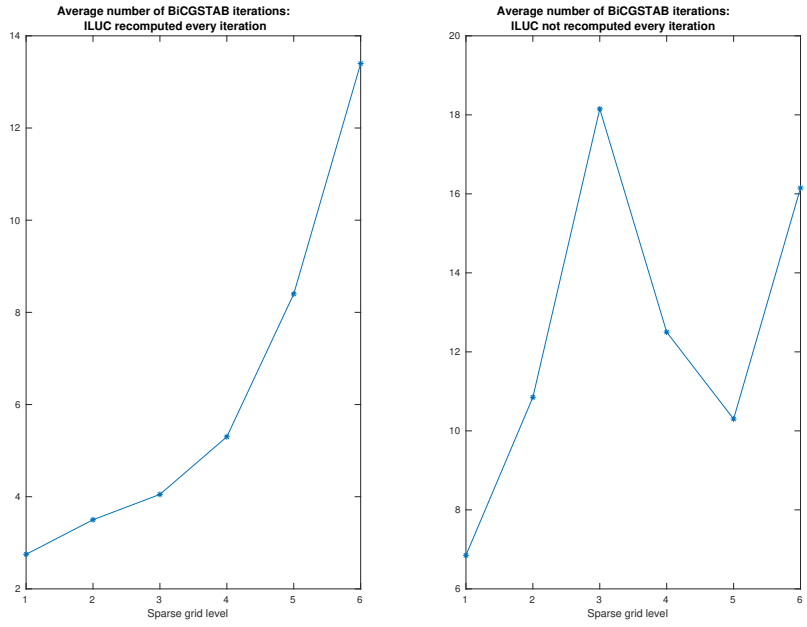
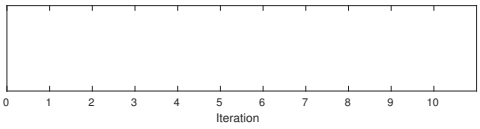


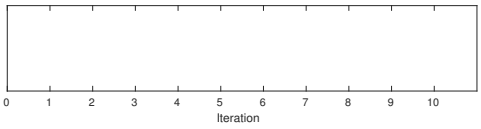
Figure 52: Average number of required iterations of precBiCGSTAB for different levels

level since it depends on when and how often ILUC is recomputed.

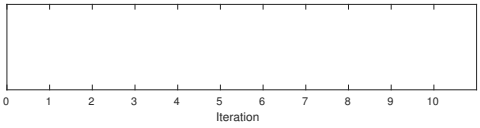
To give some insight when ILUC is recomputed, we show in which iterations it is done for the respective levels l. Figure 53 shows that in the first iterations



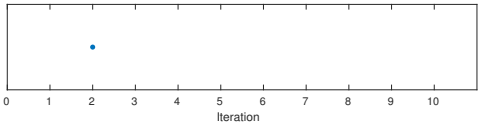
Iterations in which ILUC is recomputed for level $l = 1$



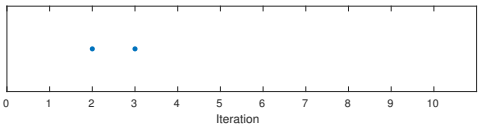
Iterations in which ILUC is recomputed for level $l = 2$



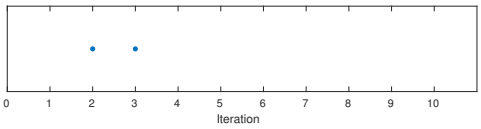
Iterations in which ILUC is recomputed for level $l = 3$



Iterations in which ILUC is recomputed for level $l = 4$



Iterations in which ILUC is recomputed for level $l = 5$



Iterations in which ILUC is recomputed for level $l = 6$

Figure 53: Iterations of Algorithm 2 in which ILUC is recomputed using Algorithm 7 for solving the linear system, given for sparse grid levels $l = 1, \ldots, 6$

we have to recompute the ILUC more often, whereas in the last iterations we can use the same ILUC factorization for multiple iterations. We interpret this as this being mainly due to the better initial guess for the iterative solver in these iterations. Thus, it can be efficient to simply recompute ILUC in the first iterations and just follow Algorithm 7 for the remaining iterations. In contrast to the $4d$ model, the number of recomputations does not grow as fast with the sparse grid level. This is due to the fact, that the precomputed stochastic matrix contributes more to the resulting system matrix which thus leads to smaller differences of the matrices between the iterations. Thus, the preconditioner can more easily be used in multiple iterations.

8.4 Remarks on our numerical results

We can extract multiple results from our numerical studies. First, sparse grid finite differences work quite well in practice for solving continuous time economic models. For a two-dimensional model, we showed that our numerical scheme converges to the full grid solution for which it is proven that it converges to the correct solution. Second, the experiments with different types of adaptivity indicate that value function adaptivity is performing well for approximating the value function. To get a good approximation of the policy functions, it can sometimes be better to use a criterion suited to this function or a combined criterion. Note though that for policy functions it strongly depends on the choice of parameters like starting sparse grid level or starting refinement threshold how well it performs. However, in general we recommend to use value function adaptivity since it leads to the best results in most cases. Third, by using BiCGSTAB together with ILUC we achieve a great runtime

reduction which allows it to compute solutions for higher dimensional models or higher sparse grid levels. The number of required ILUC recomputations is by far lower when a "bigger" part of the matrix is precomputed as it is the case for stochastic processes and thus does not change between the iterations. Note that this could be a drawback for high dimensional models where the system matrix changes completely in every iteration and thus one may have to recompute ILUC in every iteration. We point out that without preconditioning or several other preconditioning approaches we did not get convergence of Krylov subspace solvers for our models in most cases.

Note that with the current Matlab implementation we cannot go to higher sparse grid levels since it requires allocating large amounts of memory and we thus face memory constraints. Thus, for [Ahn18] several Matlab functions are rewritten using MEX-files.

9 Conclusion and Outlook

Conclusion In this work we explained a sparse grid finite difference approach for solving economic models following the numerical scheme of [AHL+17].

To get a general convergence result for sparse grids finite difference schemes for solving the HJB equation due to [BS90], we would need monotone sparse grid interpolation. However, we showed that interpolation on sparse grids is not monotone in general even if we restrict ourselves to one-dimensional monotonicity for concave monotonically increasing functions. A general theoretical result based on assumptions that are fulfilled by most economic models is hardly possible, since it often depends on model parameters if the arising interpolations are monotone for the used sparse grid. Thus, it depends on the model parameters if our approach works correctly without specific approaches to overcome non-monotonicity. We present two possible approaches to overcome non-monotonicity of sparse grid interpolation. Both approaches have the drawback of the necessity of recomputing a big part of the algorithm (or hoping for an automatic correction). Note further that we explain that one can also find model parameters for which the algorithm without correction fails and give some advices for these cases.

We analyzed the accuracy and runtime for our approach for economic models ranging from dimension $d = 2$ to dimension $d = 6$ without any correction and achieve good results for the used model parameters. For the $2d$ model, we showed that our sparse grid solution converges to the full grid solution for which it is proven that it converges against the correct solution. Moreover, we implemented different adaptivity criteria and self-adaptive refinement for adaptive sparse grids. The numerical experiments for our models indicate that for the value function accuracy an adaptivity criterion with respect to the value function works well. For the accuracy of the policy functions, we restricted our presentation to the deposit function, but all results are similar for other policy functions. Here we showed that it depends on the sparse grid parameters which adaptivity approach is preferable, but in most cases and if one is not specifically interested in a certain policy function, we recommend the use of value function adaptivity.

Furthermore, we compared different approaches for solving the arising linear system. The best approach for our models was ILUC in combination with BiCGSTAB. Note that we reference approaches of other works that could outperform ILUC preconditioned BiCGSTAB.

Outlook In [AHL+17] mean field games are solved, i.e. not only solving the HJB equation but also the KF equation. Solving the KF equation is not trivial with sparse grids since standard sparse grids do not preserve the function value range which is important for approximating density functions. An approach similar to the one proposed in work in progress [PF18] could be possible. Note though that their approach for limiting the function value range can yield full grids in the worst case and thus may not be feasible in high dimensions.

Additionally, an efficient approach to handle problems where non-monotonicities arise could be developed.

Furthermore, we are looking forward to theoretical results underlying our numerical studies. For example, proving the convergence of the approach under the assumption that interpolation is monotone for the arising functions (even though this can happen for heterogeneous agent models). Another useful result would be to show that the algorithm converges to the correct solution, in case that it converges at all.

Even when using ILUC with BiCGSTAB the majority of the computational costs remains in solving the linear system. A problem specific approach could outperform our algebraic one. Moreover, parallelization can further improve the runtime. Note that there exists e.g. a fix point version of ILU and several of the approaches presented in Section 7.3 are easily parallelizable. By using such a preconditioner, one could fully parallelize the process of solving the linear system to get a great runtime reduction. Additionally, one can easily do a parallel computation of the different sparse grid finite difference operators since there is no necessity of computing them one after another. Moreover, by using data structures like hash or tree based ones in a low level programming language, sparse grid computations could be speeded up. One possibility would be to use a library like SG++.

Further (numerical) studies for other adaptivity approaches (e.g. adaption with respect to certain dimensions), for higher sparse grid levels and for other economic models could be done. Moreover, the performance of our sparse grid finite difference approach could be compared with the sparse grid Semi-Lagrangian approaches cited in the introduction.

We are looking forward to further research in this area.

References

[ABIL13] Achdou, Yves ; Barles, Guy ; Ishii, Hitoshi ; Litvinov, Grigorii L.: *Hamilton-Jacobi equations: approximations, numerical analysis and applications*. Springer, 2013

[AHL+17] Achdou, Yves ; Han, Jiequn ; Lasry, Jean-Michel ; Lions, Pierre-Louis ; Moll, Benjamin: Income and wealth distribution in macroeconomics: A continuous-time approach. (2017)

[Ahn18] Ahn, SeHyoun: *Sparse grid methods for economic models*. 2018

[Aiy94] Aiyagari, S R.: Uninsured idiosyncratic risk and aggregate saving. In: *The Quarterly Journal of Economics* 109 (1994), Nr. 3, S. 659–684

[Bar13] Barles, Guy: An Introduction to the Theory of Viscosity Solutions for First-Order Hamilton–Jacobi Equations and Applications. In: *Hamilton-Jacobi Equations: Approximations, Numerical Analysis and Applications: Cetraro, Italy 2011, Editors: Paola Loreti, Nicoletta Anna Tchou*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013, S. 49–109

[Bel57] Bellman, R. E.: *Dynamic programming*. Princeton University Press, 1957

[Bel61] Bellman, R. E.: *Adaptive Control Processes*. Princeton University Press, 1961

[Ben02] Benzi, Michele: Preconditioning techniques for large linear systems: a survey. In: *Journal of computational Physics* 182 (2002), Nr. 2, S. 418–477

[Bew86] Bewley, Truman: Stationary monetary equilibrium with a continuum of independently fluctuating consumers. In: *Contributions to mathematical economics in honor of Gérard Debreu* 79 (1986)

[BG04] Bungatrz, Hans-Joachim ; Griebel, Michael: Sparse grids. In: *Acta Numerica* 13 (2004), Nr. 1, S. 147–269

[BGGK13] Bokanowski, Olivier ; Garcke, Jochen ; Griebel, Michael ; Klompmaker, Irene: An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations. In: *Journal of Scientific Computing* 55 (2013), Nr. 3, S. 575–605

[BM+00] Briggs, William L. ; McCormick, Steve F. u. a.: *A multigrid tutorial*. Bd. 72. Siam, 2000

[BS90] Barles, G. ; Souganidis, P.E.: *Convergence of Approximation Schemes for Fully Nonlinear Second Order Equations*. 1990

[BS04] Bertsekas, Dimitir P. ; Shreve, Steven: *Stochastic optimal control: the discrete-time case*. 2004

[BS17] Brumm, Johannes ; Scheidegger, Simon: Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models. In: *Econometrica* 85 (2017), Nr. 5, S. 1575–1612

[Can99] Candler, G. V.: Finite-Difference Methods for Dynamic Programming Problems. In: *Computational Methods for the Study of Dynamic Economies. Cambridge University Press, Cambridge, England* (1999)

[CC] Caffarelli, L.A. ; Cabré, X.: *Fully Nonlinear Elliptic Equations*. American Mathematical Soc. (American Mathematical Society: Colloquium publications Bd. 43)

[CF15] Cacace, Simone ; Falcone, Maurizio: A dynamic domain decomposition for a class of second order semi-linear equations. In: *arXiv preprint arXiv:1502.01629* (2015)

[CFF04] Carlini, Elisabetta ; Falcone, Maurizio ; Ferretti, Roberto: An efficient algorithm for Hamilton-Jacobi equations in high dimension. In: *Computing and Visualization in Science* 7 (2004), Nr. 1, S. 15–29

[CILS92] Crandall, M.G. ; Ishii, H. ; Lions, P.L. ; Society, American M.: *User's Guide to Viscosity Solutions of Second Order Partial Differential Equations*. American Mathematical Society, 1992

[CL83] Crandall, Michael G. ; Lions, Pierre-Louis: Viscosity solutions of Hamilton-Jacobi equations. In: *Transactions of the American Mathematical Society* 277 (1983), Nr. 1, S. 1–42

[CP14] Camilli, Fabio ; Prados, Emmanuel: Viscosity Solution. In: Ikeuchi, Katsushi (Hrsg.): *Computer Vision: A Reference Guide*. Boston, MA : Springer US, 2014, S. 856–860

[Cra97] Crandall, Michael G.: Viscosity solutions: A primer. In: Dolcetta, Italo C. (Hrsg.) ; Lions, Pierre L. (Hrsg.): *Viscosity Solutions and Applications: Lectures given at the 2nd Session of the Centro Internazionale Matematico Estivo (C.I.M.E.) held in Montecatini Terme, Italy, June 12–20, 1995*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1997, S. 1–43

[Dav04] Davis, Timothy A.: Algorithm 832: UMFPACK V4. 3—an unsymmetric-pattern multifrontal method. In: *ACM Transactions on Mathematical Software (TOMS)* 30 (2004), Nr. 2, S. 196–199

[Dav07] Davis, Timothy A.: UMFPACK version 5.2. 0 user guide. In: *University of Florida* (2007)

[DRSL16] Davis, Timothy A. ; Rajamanickam, Sivasankaran ; Sid-Lakhdar, Wissam M.: A survey of direct methods for sparse linear systems. In: *Acta Numerica* 25 (2016), S. 383–566

[FF13] Falcone, Maurizio ; Ferretti, Roberto: *Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations*. SIAM, 2013

[GG10] Gerstner, Thomas ; Griebel, Michael: Sparse grids. In: *Encyclopedia of Quantitative Finance* (2010)

[GH14] Griebel, M. ; Hullmann, A.: On a Multilevel Preconditioner and its Condition Numbers for the Discretized Laplacian on Full and Sparse Grids in Higher Dimensions. In: *Singular Phenomena and Scaling in Mathematical Models*. Springer International Publishing Switzerland, 2014

[GHO15] Griebel, M. ; Hullmann, A. ; Oswald, P.: Optimal scaling parameters for sparse grid discretizations. In: *Numerical Linear Algebra with Applications* 22 (2015), Nr. 1, S. 76–100

[GK17] Garcke, Jochen ; Kröner, Axel: Suboptimal feedback control of PDEs by solving HJB equations on adaptive sparse grids. In: *Journal of Scientific Computing* 70 (2017), Nr. 1, S. 1–28

[GLW16] Gevret, Hugo ; Lelong, Jerome ; Warin, Xavier: *STochastic OPTimization library in C++*, EDF Lab, Diss., 2016

[Gri98] Griebel, Michael: Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. In: *Computing* 61 (1998), Nr. 2, S. 151–179

[GS99] Griebel, M ; Schiekofer, T: An adaptive sparse grid Navier–Stokes solver in 3D based on the finite difference method. In: *Proc. ENUMATH97*, 1999

[Hac13] Hackbusch, Wolfgang: *Multi-grid methods and applications*. Bd. 4. Springer Science & Business Media, 2013

[Hem00] Hemker, Pieter W.: Application of an adaptive sparse-grid technique to a model singular perturbation problem. In: *Computing* 65 (2000), Nr. 4, S. 357–378

[HS] In: Hemker, P. W. ; Sprengel, F.: *Experience with the Solution of a Finite Difference Discretization in Sparse Grids*, S. 402–413

[HS99] Hemker, Pieter W. ; Sprengel, Frauke: *On the representation of functions and finite difference operators on adaptive sparse grids*. Centrum voor Wiskunde en Informatica, 1999

[Hug93] Huggett, Mark: The risk-free rate in heterogeneous-agent incomplete-insurance economies. In: *Journal of economic Dynamics and Control* 17 (1993), Nr. 5-6, S. 953–969

[JJ02] Jin, Hehui ; Judd, Kenneth L.: Perturbation methods for general dynamic stochastic models / Mimeo April. 2002. – Forschungsbericht

[JMM09] Judd, Kenneth ; Maliar, Lilia ; Maliar, Serguei: Numerically Stable Stochastic Simulation Approaches for Solving Dynamic Economic Models. (2009), August, Nr. 15296

[JMMV14] Judd, Kenneth L. ; Maliar, Lilia ; Maliar, Serguei ; Valero, Rafael: Smolyak method for solving dynamic economic models. In: *Journal of Economic Dynamics and Control* 44 (2014), Nr. C, S. 92–123

[KD13] Kushner, Harold ; Dupuis, Paul G.: *Numerical methods for stochastic control problems in continuous time*. Bd. 24. Springer Science & Business Media, 2013

[KK04] Krueger, Dirk ; Kubler, Felix: Computing equilibrium in OLG models with stochastic production. In: *Journal of Economic Dynamics and Control* 28 (2004), Nr. 7, S. 1411–1436

[KK17] Kalise, Dante ; Kunisch, Karl: Polynomial approximation of high-dimensional Hamilton-Jacobi-Bellman equations and applications to feedback control of semilinear parabolic PDEs. In: *arXiv preprint arXiv:1702.04400* (2017)

[KMV16] Kaplan, Greg ; Moll, Benjamin ; Violante, Giovanni L.: Monetary policy according to HANK. (2016)

[Kos] Koster, Frank: *Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern*, Diss.

[KW15] Kang, Wei ; Wilcox, Lucas C.: Mitigating the curse of dimensionality: sparse grid characteristics method for optimal feedback control and HJB equations. In: *Computational Optimization and Applications* (2015), S. 1–27

[Lan13] Langtangen, Hans P.: *Computational partial differential equations: numerical methods and diffpack programming*. Bd. 2. Springer Science & Business Media, 2013

[LS88] Lions, P.-L. ; Souganidis, P. E.: Viscosity Solutions of Second-Order Equations, Stochastic Control and Stochastic Differential Games. In: Fleming, Wendell (Hrsg.) ; Lions, Pierre-Louis (Hrsg.): *Stochastic Differential Systems, Stochastic Control Theory and Applications*. New York, NY : Springer New York, 1988, S. 293–309

[LSC03] Li, Na ; Saad, Yousef ; Chow, Edmond: Crout versions of ILU for general sparse matrices. In: *SIAM Journal on Scientific Computing* 25 (2003), Nr. 2, S. 716–728

[MMV13] Maliar, Lilia ; Maliar, Serguei ; Villemot, Sébastien: Taking perturbation to the accuracy frontier: a hybrid of local and global solutions. In: *Computational Economics* 42 (2013), Nr. 3, S. 307–325

[Mol16a] Moll, Benjamin: *Lecture 1: Lecture 3:Hamilton-Jacobi-Bellman Equations*. `http://www.princeton.edu/~moll/ECO521_2016/Lecture3_ECO521.pdf`. Version: 2016

[Mol16b] Moll, Benjamin: *Lecture 4: Diffusion Processes, Stochastic HJB Equations and Kolmogorov Forward Equations*. `http://www.princeton.edu/~moll/ECO521_2016/Lecture4_ECO521.pdf`. Version: 2016

[Mol16c] Moll, Benjamin: *Supplement to Lecture 3: Viscosity Solutions for Dummies (including Economists)*. `http://www.princeton.edu/~moll/viscosity_slides.pdf`. Version: 2016

[PF18] Pflüger, Dirk ; Franzelin, Fabian: *Limiting function value ranges of sparse grid surrogates*. 2018

[Pfl10] Pflüger, Dirk: *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München : Verlag Dr. Hut, 2010

[Pha09] Pham, H.: *Continuous-time Stochastic Control and Optimization with Financial Applications*. Springer Berlin Heidelberg, 2009 (Stochastic Modelling and Applied Probability)

[PR55] Peaceman, Donald W. ; Rachford, Henry H. Jr: The numerical solution of parabolic and elliptic differential equations. In: *Journal of the Society for industrial and Applied Mathematics* 3 (1955), Nr. 1, S. 28–41

[Pro05] Protter, Philip E.: Stochastic differential equations. In: *Stochastic integration and differential equations*. Springer, 2005, S. 249–361

[RS87] Ruge, John W. ; Stüben, Klaus: Algebraic multigrid. In: *Multigrid methods*. SIAM, 1987, S. 73–130

[Saa03] Saad, Yousef: *Iterative methods for sparse linear systems*. SIAM, 2003

[Sch98] Schiekofer, Thomas: *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen*, PhD thesis, University of Bonn, Diss., 1998

[Sch18] Schober, Peter: Solving dynamic portfolio choice models in discrete time using spatially adaptive sparse grids. (2018)

[SJ13] Schmedders, K. ; Judd, K.L.: *Handbook of Computational Economics*. Elsevier Science, 2013 (Handbook of Computational Economics Bd. 3)

[Smo63] Smolyak, S. A.: Quadrature and interpolation formulas for tensor products of certain class of functions. In: *Dokl. Akad. Nauk SSSR* 148 (1963), Nr. 5, S. 1042–1053. – Transl.: Soviet Math. Dokl. 4:240-243, 1963

[Son86] Soner, Halil M.: Optimal control with state-space constraint I. In: *SIAM Journal on Control and Optimization* 24 (1986), Nr. 3, S. 552–561

[Spr01] Sprengel, Frauke: Multilevel algorithms for finite difference discretizations on sparse grids. In: *Numerical Algorithms* 26 (2001), Nr. 2, S. 111–121

[Sto08] Stokey, Nancy L.: *The Economics of Inaction: Stochastic Control models with fixed costs*. Princeton University Press, 2008

[Stü01] Stüben, Klaus: An introduction to algebraic multigrid. In: *Multigrid* (2001), S. 413–532

[TOS00] Trottenberg, Ulrich ; Oosterlee, Cornelius W. ; Schuller, Anton: *Multigrid*. Academic press, 2000

[War14] Warin, Xavier: Adaptive sparse grids for time dependent Hamilton-Jacobi-Bellman equations in stochastic control. In: *arXiv preprint arXiv:1408.4267* (2014)

[Wat15] Wathen, Andy J.: Preconditioning. In: *Acta Numerica* 24 (2015), S. 329–376

[XZ17] Xu, Jinchao ; Zikatanov, Ludmil: Algebraic multigrid methods. In: *Acta Numerica* 26 (2017), S. 591–721

[Y+02] Yang, Ulrike M. u. a.: BoomerAMG: a parallel algebraic multigrid solver and preconditioner. In: *Applied Numerical Mathematics* 41 (2002), Nr. 1, S. 155–177

[Yu08] Yu, Xinwei: *Viscosity Solutions (lecture 11)*. `http://www.math.ualberta.ca/~xinweiyu/527.1.08f/lec11.pdf`. Version: 2008

[Zen91] Zenger, Christoph: *Sparse Grids*. 1991

[Zum00] Zumbusch, Gerhard W.: A Sparse Grid PDE Solver; Discretization, Adaptivity, Software Design and Parallelization. In: *Advances in Software Tools for Scientific Computing* 10 (2000), S. 133–177

A Appendix

A.1 A model with four state variables – a three-asset model with productivity modeled by a continuous stochastic process

We are now turning to a model with four state variables that is an extension of our $2d$-model. The theory developed and used in the lower dimensional problem can be adapted to this problem. Thus, we only describe the differences to the $2d$-model. Hence, the basic idea here is again to derive an appropriate approach for full grid finite difference methods and then use sparse grid finite difference method to solve this model. Due to the higher dimensionality, the standard full grid approach is no longer useful and the main advantage of sparse grids shows off. We refer to Section 2.2 for descriptions of the model components, to Section 4.3 for explanations of the numerical approach and to Section 6.1 for more details, in particular with respect to creating a monotone scheme.

A.1.1 Model formulation

We are now interested in the following maximization problem

$$\max_{\{c_t, d_t^a, d_t^h\}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t, h_t) dt \tag{51}$$

subject to

$$\begin{aligned} \dot{b}_t &= w z_t r^b(b_t) b_t - d_t^a - \chi(d_t^a, a_t) - d_h^t - \chi(d_t^h, h_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t^a \\ \dot{h}_t &= d_t^h \\ \dot{z}_t &= \mu(z_t) dt + \sigma(z_t) dW_t \\ b_t &\geq b,\ a_t \geq 0,\ h_t \geq 0 \end{aligned} \tag{52}$$

The diffusion is reflected on the boundaries in dimension z, i.e.

$$\partial_z v(b, a, h, \underline{z}) = 0,\ \partial_z v(a, \bar{z}) = 0, \quad \text{for } b \in (\underline{b}, \infty), a \in (\underline{a}, \infty), h \in (\underline{h}, \infty).$$

We model housing assets h to pay a utility return added to the standard utility function instead of a monetary return, i.e.

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

Notice that we now have a stationary diffusion process instead of a two-state Poisson process for income z_t. We assume that a worker's efficiency evolves stochastically over time on a bounded interval $[\underline{z}, \bar{z}]$ with $\underline{z} \geq 0$.

The HJB equation for this model is

$$\begin{aligned}\rho v(b,a,h,z) = \max_{c,d^a,d^h} & \; u(c,h) \\ & + v_b(b,a,h,z)(wz + r^b(b)b - d^a - \chi(d^a,a) - d^h - \chi(d^h,h) - c) \\ & + v_a(b,a,h,z)(r^a + d^a) \\ & + v_h(b,a,h,z)(d^h) \\ & + \partial_z v(b,a,h,z)\mu(z) + \frac{1}{2}\partial_{zz} v(b,a,h,z)\sigma^2(z).\end{aligned}$$

A.2 A model with six state variables – a two-asset model with four skill types modeled by continuous stochastic processes

The following model is again an extension of the $2d$-model presented in Section 6.1. It is used to analyze the high-dimensional behavior of the sparse grid approach. Note that by introducing different weights and ranges of the different stochastic processes or different types of stochastic processes, this multi-dimensional modeling allows further analysis in the economic context, but we restrict our numerical analysis to this simplified version.

A.2.1 Model formulation

We are interested in the following maximization problem

$$\max_{\{c_t,d_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{53}$$

subject to

$$\begin{aligned}\dot{b}_t &= \frac{(z_t^1 + z_t^2 + z_t^3 + z_t^4)}{4} w r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t \\ \dot{z}_t^1 &= \mu(z_t^1)dt + \sigma(z_t^1)dW_t \\ \dot{z}_t^2 &= \mu(z_t^2)dt + \sigma(z_t^2)dW_t \\ \dot{z}_t^3 &= \mu(z_t^3)dt + \sigma(z_t^3)dW_t \\ \dot{z}_t^4 &= \mu(z_t^4)dt + \sigma(z_t^4)dW_t \\ b_t &\geq b, \; a_t \geq 0\end{aligned} \tag{54}$$

Here $z_t^i, \; i = 1, \ldots, 4$ can be interpreted as different types of skill or luck that evolve differently over time. We use the standard CRRA-utility function again and have reflecting boundary conditions again.

We get the HJB equation

$$
\begin{aligned}
&\rho v(b,a,z^1,z^2,z^3,z^4) \\
&\quad = \max_{c,d} u(c) \\
&\qquad + v_b(b,a,z^1,z^2,z^3,z^4)\left(\frac{(z^1+z^2+z^3+z^4)}{4}w + r^b(b)b - d - \chi(d,a) - c\right) \\
&\qquad + v_a(b,a,z^1,z^2,z^3,z^4)(r^a+d) \\
&\qquad + \partial_{z^1} v(b,a,z^1,z^2,z^3,z^4)\mu(z^1) + \frac{1}{2}\partial_{z^1z^1} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^1) \\
&\qquad + \partial_{z^2} v(b,a,z^1,z^2,z^3,z^4)\mu(z^2) + \frac{1}{2}\partial_{z^2z^2} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^2) \\
&\qquad + \partial_{z^3} v(b,a,z^1,z^2,z^3,z^4)\mu(z^3) + \frac{1}{2}\partial_{z^3z^3} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^3) \\
&\qquad + \partial_{z^4} v(b,a,z^1,z^2,z^3,z^4)\mu(z^4) + \frac{1}{2}\partial_{z^4z^4} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^4).
\end{aligned}
$$

A.3 Parameters

Let us denote the model and algorithm parameters that we used in our numerical studies. Notice that we do not state parameters here that differ in the experiments (like sparse grid levels or adaptivity parameters). These are stated for the respective numerical experiments in Section 8.

A.3.1 Parameters for the two-dimensional model

We use the values for the parameters given in Table 20 in our $2d$ model (40)-(41).

Parameter	Default value	Description
γ	2	CRRA utility parameter
ρ	0.06	discount rate
r^b_{pos}	0.03	returns on liquid asset b if positive
r^b_{neg}	0.12	returns on liquid asset b if negative
r^a	0.04	returns on illiquid asset a
r^h	0.0003	returns on illiquid asset h
χ_0	0.07	parameter of cost function
χ_1	3	parameter of cost function
χ_2	0	parameter of cost function (fix costs)
ξ	0	automatic deposit parameter
w	4	wage
z_1	0.8	Poisson state 1 (productivity)
z_2	1.3	Poisson state 2 (productivity)
λ	$\pm 1/3$	Poisson parameters

Table 20: Model parameters for the $2d$ model

For Algorithm 6 for solving the $2d$ model, we use the parameters given in Table 21.

Parameter	Default value	Description
$crit$	10^{-10}	algorithm stopping criterion (maximum absolute value function value of all grid points)
$maxit$	35	maximum number of iterations in Algorithm 2
Δ	100	Δ in HJB equation

Table 21: Algorithm parameters for the $2d$ model

We give the lower and upper bounds for the respective states in the $2d$ model in Table 22. Note that the lower bounds are actually model parameters, whereas the upper bounds for the assets are numerical bounds on the computational domain.

State	Lower bound	Upper bound	Description
b	-2	40	liquid asset
a	0	70	illiquid asset

Table 22: Bounds for the respective states of the $2d$ model

A.3.2 Parameters for the four-dimensional model

We use the values for the parameters in our $4d$ model (51) - (52) given in Table 23.

Parameter	Default value	Description
γ	2	CRRA utility parameter
ρ	0.06	discount rate
r^b_{pos}	0.03	returns on liquid asset b if positive
r^b_{neg}	0.12	returns on liquid asset b if negative
r^a	0.04	returns on illiquid asset a
r^h	0.0003	returns on illiquid asset h
χ_0	0.08	parameter of cost function
χ_1	3	parameter of cost function
χ_2	0	parameter of cost function (fix costs)
w	4	wage
σ	0.1414	standard deviation for productivity
$\hat{z}$	1	mean of z (used for computation of μ)
θ	0.3	persistence

Table 23: Model parameters for the $4d$ model

For Algorithm 6 for solving the $4d$ model, we use the parameters given in Table 24.

Parameter	Default value	Description
crit	10^{-7}	algorithm stopping criterion (maximum absolute value function value of all grid points)
maxit	35	maximum number of iterations in Algorithm 2
Δ	100	Δ in HJB equation

Table 24: Algorithm parameters for the $4d$ model

We give the lower and upper bounds for the respective states in the $4d$ model in Table 25 . Note that all lower bounds and the upper bound of productivity are actually model parameters, whereas the upper bounds for the assets are numerical bounds on the computational domain.

State	Lower bound	Upper bound	Description
b	-2	40	liquid asset
a	0	70	illiquid asset
h	0	70	housing asset
z	0.8	1.2	productivity

Table 25: Bounds for the respective states of the $4d$ model

A.3.3 Parameters for the six-dimensional model

We use the values for the parameters in our $6d$ model (53) - (54) given in Table 26.

Parameter	Default value	Description
γ	2	CRRA utility parameter
ρ	0.06	discount rate
r^b_{pos}	0.03	returns on liquid asset b if positive
r^b_{neg}	0.12	returns on liquid asset b if negative
r^a	0.04	returns on illiquid asset a
χ_0	0.07	parameter of cost function
χ_1	3	parameter of cost function
χ_2	0	parameter of cost function (fix costs)
w	5	wage
σ	0.1414	standard deviation for productivity
$\hat{z}$	1	mean of z (used for computation of μ)
θ	0.3	persistence

Table 26: Model parameters for the $6d$ model

For Algorithm 6 for solving the $6d$ model, we use the parameters given in Table 27.

Parameter	Default value	Description
$crit$	10^{-7}	algorithm stopping criterion (maximum absolute value function value of all grid points)
$maxit$	50	maximum number of iterations in Algorithm 2
Δ	100	Δ in HJB equation

Table 27: Algorithm parameters for the $6d$ model

We give the lower and upper bounds for the respective states in the $6d$ model in Table 28. Note that all lower bounds and the upper bound of productivity are actually model parameters, whereas the upper bounds for the assets are numerical bounds on the computational domain.

State	Lower bound	Upper bound	Description
b	-2	40	liquid asset
a	0	70	illiquid asset
h	0	70	housing asset
z^1	0.8	1.2	skill type 1
z^2	0.8	1.2	skill type 2
z^3	0.8	1.2	skill type 3
z^4	0.8	1.2	skill type 4

Table 28: Bounds for the respective states of the $6d$ model

A.3.4 Parameters for solving the linear system

Let us give the parameters for Algorithm 7 in Table 29.

Parameter	Default value	Description
$maxit_1$	30	maximum number of BiCGSTAB iterations using the "old" LU factors
$maxit_2$	300	maximum number of BiCGSTAB iterations using "new" LU factors
tol_1, tol_2	10^{-12}	convergence thresholds of BiCGSTAB

Table 29: Bounds for the respective states of the $6d$ model