Institut für Numerische Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn
Endenicher Allee 19b • 53115 Bonn • Germany
phone +49 228 73-69828 • fax +49 228 73-69847
www.ins.uni-bonn.de

J. Garcke, S. Ruttscheidt

Finite Differences on Sparse Grids for Continuous Time Heterogeneous Agent Models

INS Preprint No. 1906

September 2019

Finite Differences on Sparse Grids for Continuous Time Heterogeneous Agent Models

Jochen Garcke and Steffen Ruttscheidt

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 [4], 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. We analyze the convergence behavior for higher dimensional models and experiment with different sparse grid adaptivity types.

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

Jochen Garcke
Institut für Numerische Simulation, Universität Bonn, e-mail: garcke@ins.uni-bonn.de
Fraunhofer Center for Machine Learning and SCAI, Schloss Birlinghoven, Sankt Augustin

Steffen Ruttscheidt
Institut für Numerische Simulation, Universität Bonn

pricing models may feature many different assets, while 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. With standard discretizations, one faces the problem that one cannot introduce many variables due to the *curse of dimensionality*, a terminology coined by Bellman in [6] that describes the exponential dependence of the overall computational effort on the number of dimensions. In this work we study how to solve continuous time heterogeneous agent models with multiple assets in higher dimensions with a finite difference approach on sparse grids.

In [10] a finite difference method is used to solve the Hamilton-Jacobi-Bellman (HJB) equation in the economic context. This approach was improved in [2] to handle borrowing constraints by mathematically recasting them as state constraints. The computational method was further adapted in [20] to handle non-convexities and multiple assets.

In [31] finite differences on sparse grids were introduced and several theoretical results regarding consistency, stability and convergence are shown. Further studies have been made in [14, 15, 36]. 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 employ a finite difference method following [3], which is based on interpolation and can be implemented more easily. Notice that interpolation based ideas were already presented in [21].

With this approach, we are able to solve continuous time heterogeneous agent models in higher dimensions. Note that we use the same finite difference approach as in [2], but instead of implementing it on full grids we do it on sparse grids.

This work is structured as follows. The setup and motivation is given in Section 2, where we present a two-dimensional model problem. In Section 3 we shortly describe sparse grids and the employed finite differences on sparse grids [3]. An investigation of sparse grid interpolation is done in Section 4 to explain why we do not simply get convergence by means of Barles and Souganidis [4], which comes down to the non-monotonicity of sparse grid interpolation. We further present some approaches to overcome some of the arising issues regarding non-convergence. It is followed by a detailed presentation of the algorithm and its implementation in Section 5. Even though we do not have a theoretical convergence result, we give numerical results in Section 6 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. We conclude this work with an outlook in Section 7. The Appendix contains information about the implemented higher dimensional models and the choice of parameters for the numerical experiments.

2 Heterogeneous agent models as optimal control problems

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. Several well known heterogeneous agent models (e.g. Bewley, Huggett, and Aiyagari models) were recasted in continuous time by [2]. Even though we restrict ourselves to certain types of model in our experiments, we point out that the presented framework is basically applicable to any heterogeneous agent model. We here mainly follow [26]. For mathematical descriptions and proofs, see [23].

2.1 Optimal control problems

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

$$\max_{\{\alpha(t)\}_{t\geq 0}} J(x,\alpha), \text{ with } J(x,\alpha) := \int_0^\infty e^{-\rho t} h(x(t),\alpha(t))dt \tag{1}$$

such that the law of motion for given state x and control α

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

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 $f : X \times A \to \mathbb{R}^m$.

Note that 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.

The *value function* associated to the problem (1) is defined 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 solve optimal control problems, we use the *dynamic programming principle* (DPP) introduced by Bellman, see [5]. It is based on the recursive structure of the problem. By using this principle one can show that the value function satisfies the HJB equation

$$\rho v(x) = \max_{\alpha \in A} h(x,\alpha) + D_x v(x) \cdot f(x,\alpha), \quad \forall x \in X. \tag{2}$$

To compute the optimal controls, one uses the *first order conditions* (FOCs) on the HJB equation. For that, one computes the derivatives with respect to the different controls and sets them to zero.

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. Note that we omit the initial state in the following and denote the time dependence of the states by a subscript t.

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

$$\max_{\{c_t\}_{t\geq 0}} \int_0^\infty e^{-\rho t} u(c_t) dt \tag{3}$$

subject to

$$\dot{b}_t = w + r^b b_t - c_t. \tag{4}$$

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 [20], 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 (3). Note again that this choice directly reflects in the change of the state. A standard choice for the return function h is the Constant Relative Risk Aversion (CRRA)-utility function given by

$$u(c) = \frac{c^{1-\gamma}}{1-\gamma} \tag{5}$$

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

For the model (3) - (4), 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.

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

$$b_t \geq \underline{b}, \tag{6}$$

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 [2].

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

$$\begin{aligned} \dot{b}_t &= w z_t r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t \\ b_t &\geq b,\ a_t \geq 0 \end{aligned} \tag{7}$$

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

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

$$\max_{\{c_t\}_{t \geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{8}$$

subject to

$$\begin{aligned} \dot{b}_t &= wz_t + r^b b_t - c_t \\ \dot{z}_t &= \mu(z_t)dt + \sigma(z_t)dW_t \\ b_t &\geq \underline{b}. \end{aligned} \tag{9}$$

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 (8) 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.

Using Itô's lemma, see e.g. [29], one can show that the model leads to the HJB equation

$$\begin{aligned} \rho v(b,z) = \max_c u(c) &+ \partial_b v(b,z)(wz + r^b b - c) \\ &+ \partial_z v(b,z)\mu(z) + \frac{1}{2}\partial_{zz} v(b,z)\sigma^2(z) \end{aligned} \tag{10}$$

for the modified model.

If we instead replace $\dot{z}_t = \mu(z_t)dt + \sigma(z_t)dW_t$ in (9) by

$$z_t \in \{z_1, z_2\} \text{ Poisson with intensities } \lambda_1, \lambda_2, \tag{11}$$

we have a two-state Poisson type process instead of continuous stationary diffusion process for productivity z. The HJB equation for model (8)-(9) modified by (11) is then

$$\rho v(b,z) = \max_c u(c) + \partial_b v(b,z)(wz + r^b b - c) + \lambda_i(v(b,z_j) - v(b_1,z_i)) \tag{12}$$

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

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

We now explain the economic models that we employed for our numerical experiments, following closely the presentation in a supplement to [20] for the two asset model. Specifically, we want to solve the following maximization problem

$$\max_{\{c_t,d_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{13}$$

subject to

$$\begin{aligned} \dot{b}_t &= w z_t r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t \\ z_t &= \text{Poisson with intensities } \lambda(z, z') \\ b_t &\geq b,\ a_t \geq 0. \end{aligned} \tag{14}$$

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 (5) 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

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

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

$$\begin{aligned} d &= \underbrace{d^+}_{\text{case } d>0} + \underbrace{d^-}_{\text{case } d<0} \\ &= \left(\left(\frac{v_a}{v_b} - 1 - \chi_0\right) \frac{a}{\chi_1} \right)^+ + \left(\left(\frac{v_a}{v_b} - 1 + \chi_0\right) \frac{a}{\chi_1} \right)^- . \end{aligned} \tag{15}$$

2.4 Higher dimensional models

We refer to the Appendix for higher dimensional models which are natural extensions of this two-dimensional model, where we add assets such as housing ones or multiple diffusion type stochastic processes for different types of productivity.

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

We refer to [32] 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 [18] 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 [22] sparse grids are used to replace the expensive tensor product grids. For perturbation methods that are feasible in higher dimensions, see [17] and [25].

Sparse grids in combination with a fixed point iteration on the Euler equation are proposed in [19] 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 [8] to solve high dimensional stochastic dynamic models where functions are interpolated on a sparse grid either within time or value iterations. Further, in [33], 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 [7] and the references therein.

3 Sparse Grids

Sparse grids were introduced in [35] and go back to [34]. We give only a short introduction here.

To construct sparse grids, one uses a tensor product construction to obtain a multi-dimensional basis on the d-dimensional unit cube $\bar{\Omega} := [0,1]^d$ from the one-dimensional hierarchical basis. We use the multi-index $\mathbf{l} = (l_1, \ldots, l_d) \in \mathbb{N}^d$ to denote the level. We then consider the set of d-dimensional rectangular grids $\Omega_{\mathbf{l}}$ with mesh size

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

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*

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

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

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

See [9, 12, 28] for details and approximation properties.

3.1 Finite difference schemes on sparse grids

Finite differences on sparse grids were introduced and studied in [31], where consistency proofs and convergence results are given, see also [14, 15, 30]. The construction of finite difference operators are based on a dimensional splitting combined with a nodal to hierarchical basis transformation and its respective back-transform.

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

- a basis transformation from nodal to hierarchical basis in all dimensions but the 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 the neighboring grid point in dimension j,
- a basis transformation from hierarchical to nodal basis in all dimensions but dimension j.

The finite difference operators use per dimension the neighboring grid points of the respective grid point, i.e. the closest grid points in the dimension. For the approximation of the derivative on regular sparse grids, one uses appropriate equidistant difference stencils. If adaptive refinement is used, the grid points in the different dimensions are no longer equidistant, that is 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.

We consider an alternative approach [3], which introduces additional points and interpolates using these. Instead of 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 could simply take any sparse grid library, such as SG++ [28], to implement this approach.

Following [3], we first define the above noted ghost points. Afterwards, we describe interpolation operators working on these points. Finally, we introduce the finite difference operators by using these interpolation operators.

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

Definition 1 (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. 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. Other difference operators are possible but we restrict ourselves for a simplified presentation.

Definition 2 (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

Fig. 1: 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.

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

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

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

An example for ghost nodes is given in Figure 1.

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

Definition 3 (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, that is the grid of ghost nodes, for the forward difference $\mathbf{I}_{h_{g_j}}^F$ and backward difference $\mathbf{I}_{h_{g_j}}^B$ in dimension $j,\ 1 \le j \le d$ by

$$\mathbf{I}_{h_{g_j}}^F : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}} \quad \text{and} \quad \mathbf{I}_{h_{g_j}}^B : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}}.$$

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

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

The *sparse grid backward difference operator* $\tilde{\mathbf{D}}_j^{S,B}$ is given 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 1 with red points on top of sparse grid points) since the function values for these grid 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, as is also pointed out in [31].

The operators are linear and can thus be represented by matrices, therefore a comparison of the approaches can be easily undertaken using the corresponding matrices. Observe that the version presented in [31] is working on nodal values, whereas the interpolation based version is working on hierarchical basis coefficients. One thus applies the nodal to hierarchical basis transformation as first operation in the interpolation based version to compare the arising discretization matrices of both sparse grid finite difference approaches. For regular sparse grids of level $l = 1, 2, 3$, it is confirmed that both approaches yield the same finite difference operator [30]. The relationship between the two approaches will be topic of the upcoming work [3].

3.2 Adaptive sparse grids

There are cases in which one wants to use an adaptive sparse grid. For example, this is the case if 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*. As a criterion for adaptation one typically 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). Vice versa, grid points whose corresponding basis functions do not contribute much are removed in a coarsening step. For a description of similar algorithms for refinement and coarsening, see [13], and a general view can be found in [28].

We use different types of adaptivity criteria that are based on the hierarchical surpluses as an error indicator. The overall algorithm uses both, the adaptive refinement and the adaptive coarsening together. Given an index set $\mathscr{I}$, a refinement parameter ε, a coarsening parameter ν and the approximated function v on $Q_{\mathscr{I}}$ we can refine and coarsen the grid and approximate the function on the new grid. For our experiments, we use the coarsening parameter $\nu = \varepsilon/10$ which is a typical choice. An additional component can be self-adaptivity as also used in [31], where the refinement threshold and coarsening parameter are automatically decreased when no new points are added by adapting with the current parameters.

We perform experiments 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 also 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 why we study value function adaptivity and policy functions adaptivity. Additionally, we experiment with combinations of both.

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 (13)-(14) described in Section 2.3, we can use a value function adaptivity, a consumption function adaptivity, or a deposit function adaptivity.

Moreover, we use the combination of the above described adaptivity types. 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. the criteria 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.

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

The requirements one needs to fulfill to obtain a convergent approximation scheme for HJB equations by means of Barles and Souganidis [4] involve a stable, consistent and monotone scheme. Whereas it is trivial to show that we need monotone inter-

polation in this theoretical framework, we face the problem that one cannot prove that interpolation on sparse grids is monotone in general, as it is already pointed out in [16], see also [28]. [3] notes that the introduced upwind finite difference scheme converges even though the scheme is not monotone. However, [3] does not give examples or justifications that it is not monotone.

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 that we only use one-dimensional monotonicity and thus just need monotonicity with respect to the different dimensions, in particular with respect to the used ghost points. We now investigate if by restricting ourselves to concave monotonically increasing functions, such as the value functions arising from our models, one can achieve monotonicity.

4.1 (Non-)monotonicity of interpolation on sparse grids

Let us begin by giving our definition of monotone interpolation.

Definition 4 (Monotone interpolation in one dimension). Let $x_1, \ldots, x_n$ be data points with $x_1 < \ldots < x_n$. A function f is called monotone, if it holds that $f(x_1) \leq \ldots \leq f(x_n)$ or $f(x_1) \geq \ldots \geq f(x_n)$. In case of strict inequalities f is strictly monotone. The interpolation f_I of f is *monotone*, if for every pair of two points $\tilde{x}_1 < \tilde{x}_2$, $\tilde{x}_1, \tilde{x}_2 \in [x_1, x_n]$ it holds

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

or

$$f_I(\tilde{x}_1) \geq f_I(\tilde{x}_2) \text{ for } f(\tilde{x}_1) \geq f(\tilde{x}_2),$$

with strict inequality for *strictly monotone interpolation*.

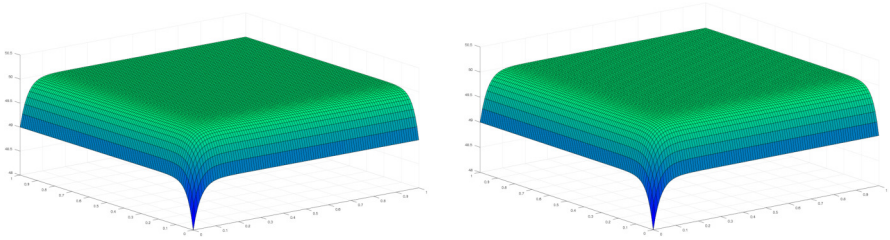
Note that we are interested in higher dimensions and aim for one-dimensional monotone interpolation for the restriction to the different dimensions respectively, and in particular with respect to the ghost points.

In the following examples we add a high constant to the functions 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. Let us begin our investigation by motivating the restriction to monotonically increasing concave functions.

In the following we consider concave functions which in $[0,1]^2$ are monotone.

4.1.1 Strictly increasing concave functions with positive coefficients

Notice that for the strictly increasing concave function



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

Fig. 2: Plots of the original function f_2 on the left and its sparse grid interpolation of level $l = 7$ on the right

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

we end up with just positive coefficients and we also get a monotone interpolation.

A more interesting case arises for the function

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

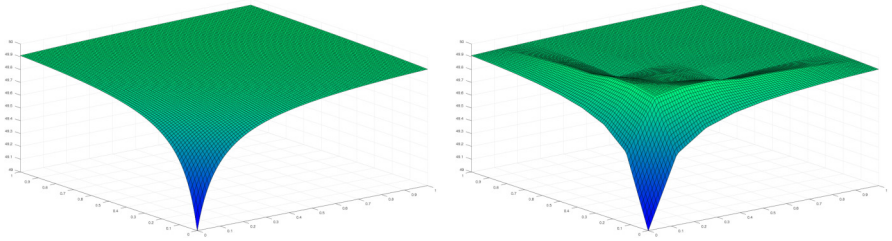
which has extremely steep gradients close to $(0,0)$ and is flat in the other parts of the domain. This function is also strictly concave and monotonically increasing. Notice further that all hierarchical coefficients are positive. In Figure 2 one can see that the interpolation looks monotone. Explicitly checking the monotonicity shows that the approximation is monotone up to machine accuracy.

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

We now give a counter example to show that sparse grid interpolation for monotonically increasing concave functions is not monotone in general. To achieve this, we give a concave monotonically increasing functions for which negative hierarchical coefficients arise. Let us consider the interpolation of the function

$$f_3(x,y) = \frac{-1}{1+10x+10y} + 50.$$

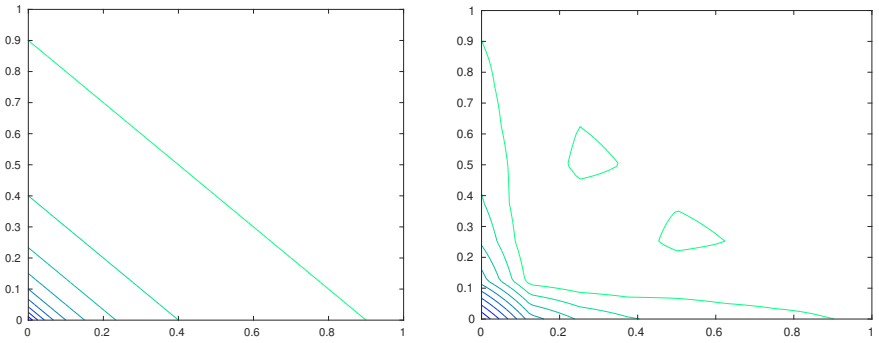
The reason that we chose this function is that it is similar to functions that arise as value function for some models. Computing the eigenvalues of the Hessian shows that it is negative semidefinite in $[0,1]^2$ and thus the function is concave. Unfortunately, we see in Figure 3 and Figure 4 that the 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, where you see the non-monotonicity of the interpolation in the interpolation contour



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

Fig. 3: Plots of the original function f_3 on the left and its sparse grid interpolation of level $l = 3$ on the right

plot. We use a sparse grid level $l = 3$ in the following, but corresponding counter examples can be constructed for other levels.



(a) Contour plot of function f_3 (b) Contour plot of the interpolation of f_4

Fig. 4: Contour plots of the function f_3 on the left and its sparse grid interpolation of level $l = 3$ on the right

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 5 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.

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. $f_4(x,y) = \frac{-1}{1+(x+0.01)^{0.2}+(y+0.01)^{0.2}} + 50$. 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 coefficients to zero to get a monotone approximation.

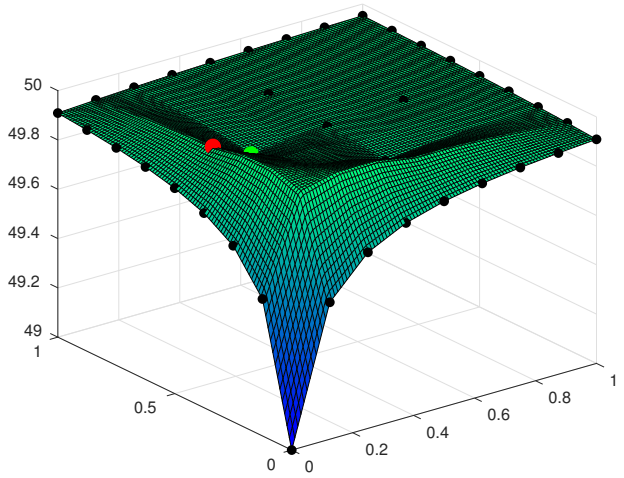
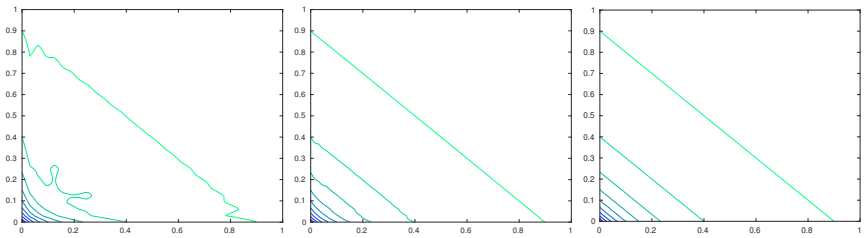


Fig. 5: For the backward difference at $(0.5/0.25)$, one uses the values drawn as the green and the red point

4.1.3 Overcoming the non-monotonicity of sparse grid interpolation



(a) Contour plot for the interpolation of f_4 for level $l = 5$ (b) Contour plot for the interpolation of f_4 for level $l = 7$ (c) Contour plot for the interpolation of f_4 for level $l = 9$

Fig. 6: Plots of contour plots for the sparse grid interpolation of f_4 for different levels.

There are several approaches to get monotone interpolation on sparse grids. The most trivial way is to go to a higher sparse grid level, this is visualized in Figure 6 where the interpolation of f_4 is presented for sparse grid levels $l = 5,7,9$ instead of $l = 3$. The approach is simple, but one cannot go to arbitrarily high levels in higher dimensions, besides for higher levels corresponding counter examples can be constructed as well.

Alternatively, one can identify the areas where non-monotonicities arise and insert points only in these areas. For that, one can adapt the sparse grid using the hierarchical coefficients as error indicator, as it is done in standard adaptive approaches. Another

approach is to go to a ”full” grid in the critical area. Here, one determines the maximal one-dimensional level used in the critical area and then simply uses the full grid level. Thereby non-monotonicity cannot arise in this area. For investigations on these approaches for monotone functions see [30]. Specific to our situation, a promising alternative refinement is to use the computed derivatives. That is, one can use the approximations of the derivative and check if these are non-negative since that indicates a non-monotone interpolation. By marking such points for adaption, one can iterate until all derivative approximations are non-negative or below a certain error threshold.

5 Numerical approach using an upwind scheme

Let us follow the very recent work of [2] 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. The approach was extended to sparse grids and applied to economic models in [3]. You can find a more mathematical description of finite difference schemes for solving the HJB equation that is not targeted to economic models in [1].

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 difference operators introduced in [3], as described in Section 3.1. 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. At the end of this subsection, we generalize the matrix notation so that it can easily be extended to the sparse grid setting.

For ease of presentation, let us consider the simple one-dimensional deterministic model already presented and explained in Section 2.2 consisting of equations (3),(4) and (6). 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.

5.1 Discretization

The finite difference approximation of the HJB equation (2), using J points, is

$$\rho v(b_j) = u(c_j) + v'(b_j)(w + r^b b_j - c_j), \quad c_j = (u')^{-1}(v'(b_j)),\ j = 1,\ldots,J \quad (16)$$

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.

An issue when 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 [2]. Furthermore, it is possible to construct other monotone schemes but this one is perfectly suited to implement borrowing constraints which we turn to now.

5.1.1 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 [2] for a description.

5.1.2 Numerical approach for overcoming the non-linearity

The HJB equation is highly nonlinear due to the maximum operator. We use an iterative scheme to solve this equation, i.e. *policy iteration* as for example explained in [11]. 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

$$\frac{v_j^{n+1} - v_j^n}{\Delta} + \rho v_j^n = u(c_j^n) + \frac{v_{j+1}^n - v_j^n}{\Delta b}(s_j^{F,n})^+ + \frac{v_j^n - v_{j-1}^n}{\Delta b}(s_j^{B,n})^-, \qquad (17)$$

where n denotes the iteration step.

Let us describe the algorithmic approach. 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. We then 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. Doing this, we stop when the value function approximation of consecutive iterations does not differ much.

There are two main interpretations for (17). The first one is the use of the Newton method for solving the system of non-linear equations (16). The other one is that the iterative scheme is equivalent to solving the HJB equation backward in time. Hence, (17) 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 (17) 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

$$\frac{v_j^{n+1}-v_j^n}{\Delta}+\rho v_j^n = u(c_j^n)+\frac{v_{j+1}^{n+1}-v_j^{n+1}}{\Delta b}(s_j^{F,n})^+ + \frac{v_j^{n+1}-v_{j-1}^{n+1}}{\Delta b}(s_j^{B,n})^- \tag{18}$$

where the value functions on the right hand side are of step $n+1$. Thus, one has to solve a linear system to solve (18) 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 often 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 (18), 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 [27].

5.1.3 Numerical approach for stochastic settings

For the heterogeneous agent model (8) - (9) 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 (10) which arises for this model by

$$\begin{aligned}\frac{v_{j,k}^{n+1}-v_{j,k}^n}{\Delta}+\rho v_{j,k}^n =& u(c_{j,k}^n)+\frac{v_{j+1,k}^{n+1}-v_{j,k}^{n+1}}{\Delta b}(s_{j,k}^{F,n})^+ + \frac{v_{j,k}^{n+1}-v_{j-1,k}^{n+1}}{\Delta b}(s_{j,k}^{B,n})^- \\ &+ \frac{v_{j,k+1}^{n+1}-v_{j,k}^{n+1}}{\Delta z}(\mu_k)^+ + \frac{v_{j,k}^{n+1}-v_{j,k-1}^{n+1}}{\Delta z}(\mu_k)^- \\ &+ \frac{\sigma_k^2}{2}\frac{v_{j,k+1}^{n+1}-2v_{j,k}^{n+1}+v_{j,k-1}^{n+1}}{(\Delta z)^2}.\end{aligned} \tag{19}$$

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 (8) - (9) 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 (12) which arises for this model with a two-state Poisson process by

$$\frac{v_{j,k}^{n+1}-v_{j,k}^{n}}{\Delta}+\rho v_{j,k}^{n} = u(c_{j,k}^{n})+\frac{v_{j+1,k}^{n+1}-v_{j,k}^{n+1}}{\Delta b}(s_{j,k}^{F,n})^{+}+\frac{v_{j,k}^{n+1}-v_{j-1,k}^{n+1}}{\Delta b}(s_{j,k}^{B,n})^{-} + \lambda_k(v_{j,-k}^{n+1}-v_{j,k}^{n+1}) \tag{20}$$

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

5.1.4 Matrix notation

After linearizing and discretizing the HJB equation, we can easily formulate the resulting equations as a linear system for the value function of size $m = J \cdot K$. 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

$$\frac{1}{\Delta}(\mathbf{v}^{n+1}-\mathbf{v}^{n})+\rho\mathbf{v}^{n+1} = \mathbf{u}^{n}+(\mathbf{A}^{n}+\boldsymbol{\Lambda})\mathbf{v}^{n+1} \tag{21}$$

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

$$\underbrace{((\frac{1}{\Delta}+\rho)\mathbf{I}-\mathbf{A}^{n}-\boldsymbol{\Lambda})}_{\mathbf{B}}\mathbf{v}^{n+1} = \underbrace{\mathbf{u}^{n}+\frac{1}{\Delta}\mathbf{v}^{n}}_{\mathbf{b}^{n}=\mathbf{b}(\mathbf{v}^{n})} \tag{22}$$

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

$$\mathbf{B}\mathbf{v}^{n+1} = \mathbf{b}^{n} \tag{23}$$

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

To define $\mathbf{A}^{n}$ and $\boldsymbol{\Lambda}$ more formally, one can denote the construction via finite difference operators and specific scalar matrix-row multiplications. Let us show this for equation (19). We denote the row-wise vector matrix scalar multiplication, i.e. scalar multiplication of vector entry i, $1 \le i \le m$ with matrix row i by $\star$. To denote (19) using matrix notation (21), we have

$$\mathbf{A}^{n} = (\mathbf{s}^{F,n})^{+} \star \mathbf{D}_{b}^{F} + (\mathbf{s}^{B,n})^{-} \star \mathbf{D}_{b}^{B} \tag{24}$$

and

$$\boldsymbol{\Lambda}^{n} = (\boldsymbol{\mu})^{+} \star \mathbf{D}_{z}^{F} + (\boldsymbol{\mu})^{-} \star \mathbf{D}_{z}^{B} + \frac{1}{2}\boldsymbol{\sigma}^{2} \star \mathbf{D}_{zz} \tag{25}$$

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 [24] for a description.

As explained in Section 3.1, 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

$$((\frac{1}{\Delta}+\rho)\mathbf{E}-\boldsymbol{\Lambda}-\mathbf{A}^n)\mathbf{v}^{n+1}=\mathbf{u}^n+\frac{1}{\Delta}\mathbf{E}\mathbf{v}^n \tag{26}$$

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

$$((\frac{1}{\Delta}+\rho)\mathbf{E}-\mathbf{E}\boldsymbol{\Lambda}-\mathbf{A}^n)\mathbf{v}^{n+1}=\mathbf{u}^n+\frac{1}{\Delta}\mathbf{E}\mathbf{v}^n \tag{27}$$

where $\boldsymbol{\Lambda}$ 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.

The overall procedure is given in Algorithm 1.

Algorithm 1 Solving the HJB equation on (adaptive) sparse grids

Data: model parameters, sparse grid parameters
Result: solution $\mathbf{v}$ of HJB equation
Initialization:
generate sparse grid
compute hierarchical to nodal basis transformation matrix $\mathbf{E}$
generate finite difference operators ▷ see 3.1
set up matrix $\boldsymbol{\Lambda}$ that models stochastic process ▷ see 5.1.4
compute initial guess in hierarchical representation $\mathbf{v}^0$ ▷ see e.g. (28) for model (13) - (14)
Iterative part:
for $n = 0, 1, \ldots$ **do**
    refine sparse grid and initialize the new sparse grid
    compute forward and backward differences of $\mathbf{v}^n$ ▷ use finite difference operators
    compute optimal controls ▷ e.g. consumption, deposits for model (13) - (14)),
        ▷ use forward and backward differences of $\mathbf{v}^n$
    build drift matrix $\mathbf{A}^n$ ▷ see 5.1.4, follow upwind scheme and use finite difference operators
    solve (26) respectively (27) for $\mathbf{v}^{n+1}$ ▷ see 5.1.2, linearized HJB equation
    if $\mathbf{v}^{n+1}$ is close to $\mathbf{v}^n$ according to stopping criteria **then**
        $\mathbf{v} \leftarrow \mathbf{v}^{n+1}$
        STOP
    end if
    coarsen sparse grid
end for

5.2 Further aspects

Considering the employed two-asset model problem (13), (14), 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 into different parts that do not have this type of interaction and approximate the value function using the split. For the optimal deposits we follow the same splitting idea.

The resulting 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. See [30] for the full technical details.

As an initial guess for the value function, we use

$$v_0 = \frac{\left(\frac{wz+r^b(b)b+r^a a}{1-\gamma}\right)^{1-\gamma}}{\rho}, \tag{28}$$

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

6 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 a normalization with respect to the reference solution to allow us to compare the arising errors of different functions. Thus, we compute relative errors by

$$\begin{aligned} e^f_{2,r}(x_1,\ldots,x_M) &= \left(\frac{1}{M}\sum_{m=1}^{M}\left|\frac{f_{\text{ref}}(x_m)-f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m)-\min_m f_{\text{ref}}(x_m)}\right|^2\right)^{\frac{1}{2}}, \\ e^f_{\infty,r}(x_1,\ldots,x_M) &= \max_m\left|\frac{f_{\text{ref}}(x_m)-f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m)-\min_m f_{\text{ref}}(x_m)}\right|. \end{aligned} \tag{29}$$

Note that for all adaptive refinements, we use a normalization of the hierarchical coefficients with respect to the range in nodal values. We always use the coarsening parameter $\nu = \varepsilon/10$ and always coarsen with respect to the value function, this yields in our experiments better results.

We solve the linear equation system with an ILUC preconditioned BiCGSTAB in Matlab.

6.1 Two-dimensional model

Let us begin with the $2d$ model (13) - (14) presented in Section 2.3 to give some intuition and to show that our sparse grid algorithm converges to the solution of the full grid method. We present relative errors for the value function and all policy functions for regular sparse grids of different levels. The reference solution is computed on a 600×600 full grid.

In Figure 7 we show the convergence behavior of regular sparse grids for the value function, the deposit policy, and the consumption policy. First of all, we see that the sparse grid algorithm converges to the full grid solution. We additionally note that the accuracy for both Poisson states is quite similar. Note that this may change if the resulting functions become more different.

One can see that the e_∞-error for one state of 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 hence cannot be captured well by sparse grids.

Note that in the following results, we give the errors for the multi-variate functions, where the different outputs are stacked into one vector.

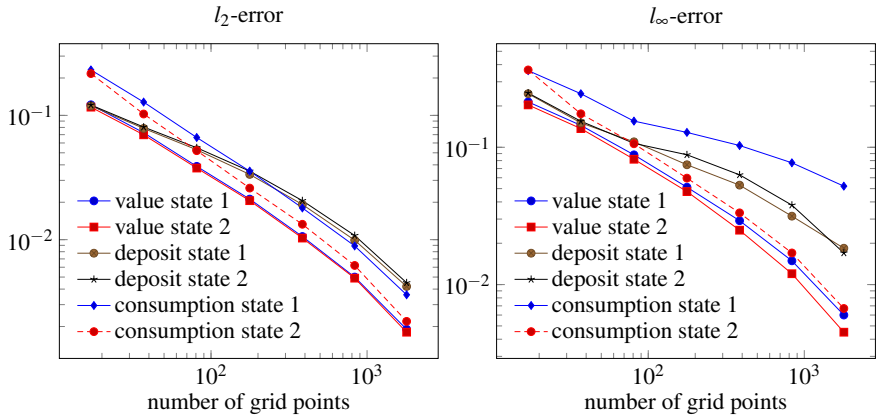


Fig. 7: Accuracy of different sparse grid levels: e_2 and e_∞-errors for the value function, deposit policy and consumption policy for states 1 and 2.

6.1.1 Plots for regular sparse grids

To get insight into how the approximations and the arising errors look, we present plots of the approximations for sparse grid level $l = 7$. Additionally we show the difference between sparse grid and full grid solution.

In the Figures 8 - 10 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.

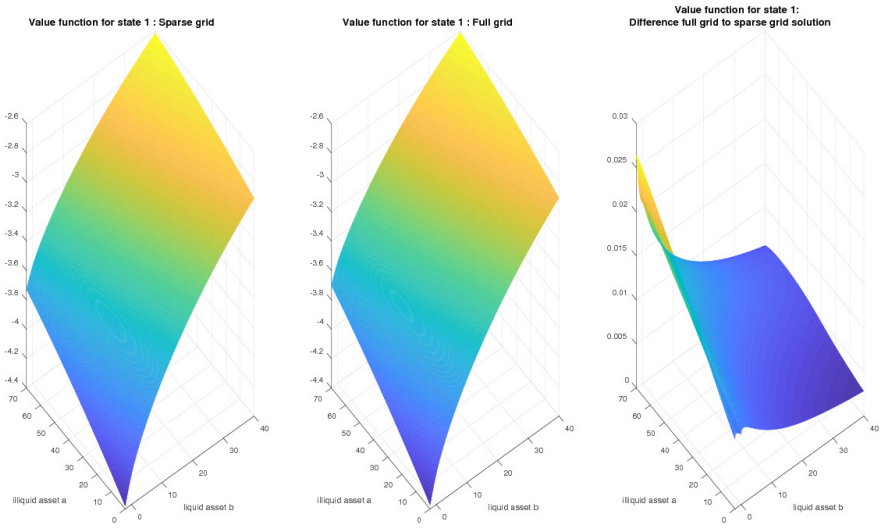


Fig. 8: Value function for state 1: sparse grid approximation, full grid reference solution, and the sparse grid approximation subtracted from the full grid solution

6.1.2 Plots for adaptive sparse grids

To get a good approximation without using a really high sparse grid level (and thus many points), we investigate adaptive sparse grids. One 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.

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 Figure 11 the approximation of the value function for state 1 and the sparse grid using value function adaptivity are shown. One 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.

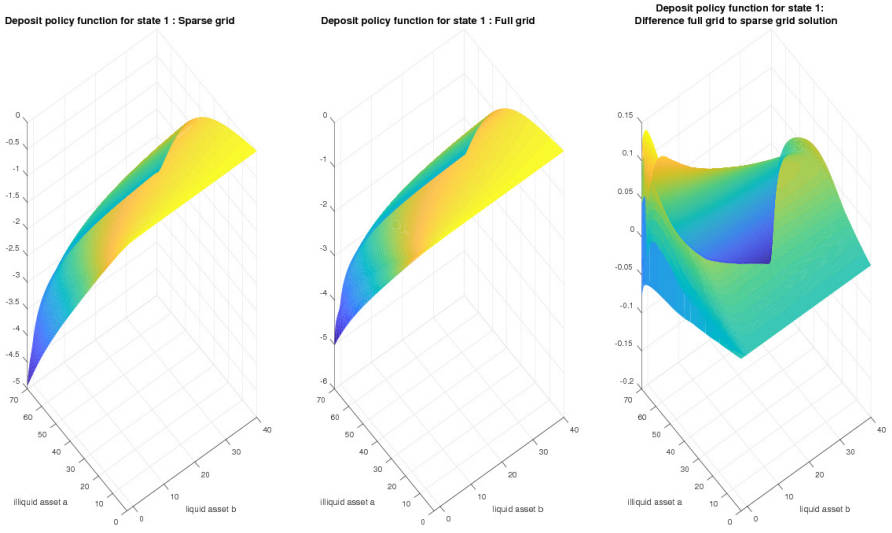


Fig. 9: Deposit policy function for state 1: sparse grid approximation, full grid reference solution, and the sparse grid approximation subtracted from the full grid solution

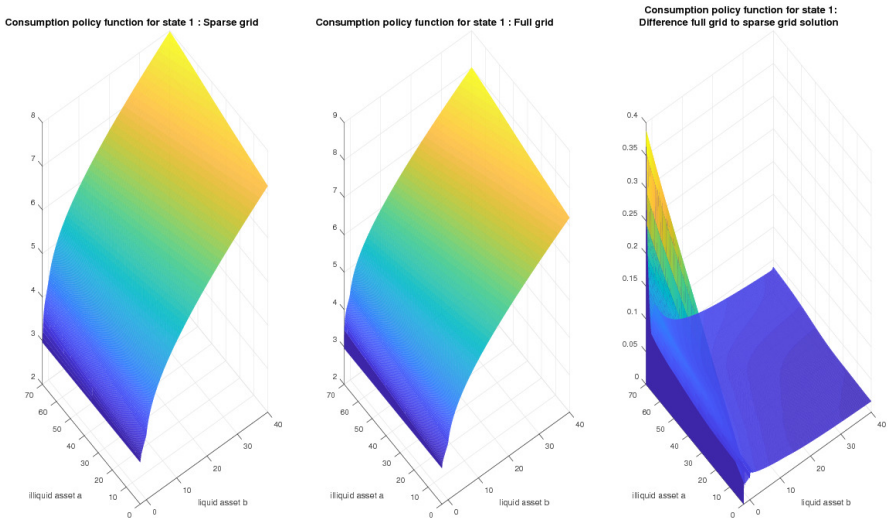


Fig. 10: Consumption policy function for state 1: sparse grid approximation, full grid reference solution, and the sparse grid approximation subtracted from the full grid solution

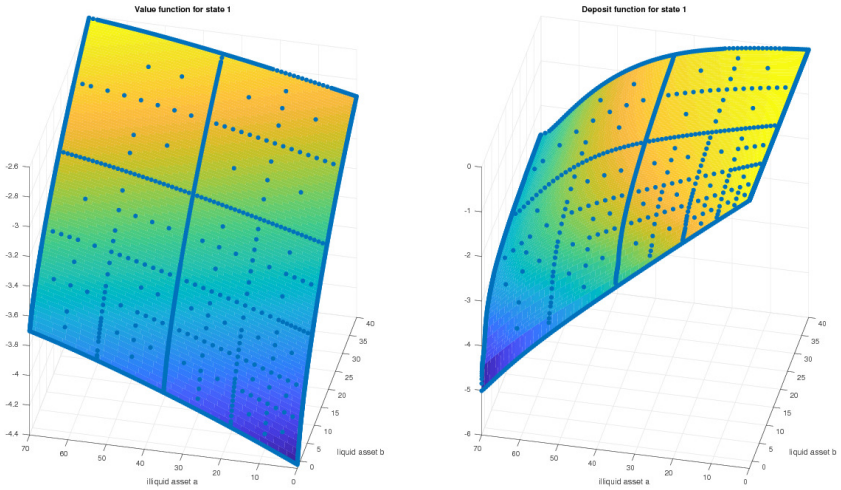


Fig. 11: Scatter and surface plot of the value and the deposit function for state 1 for the adapted sparse grid with value function adaptivity

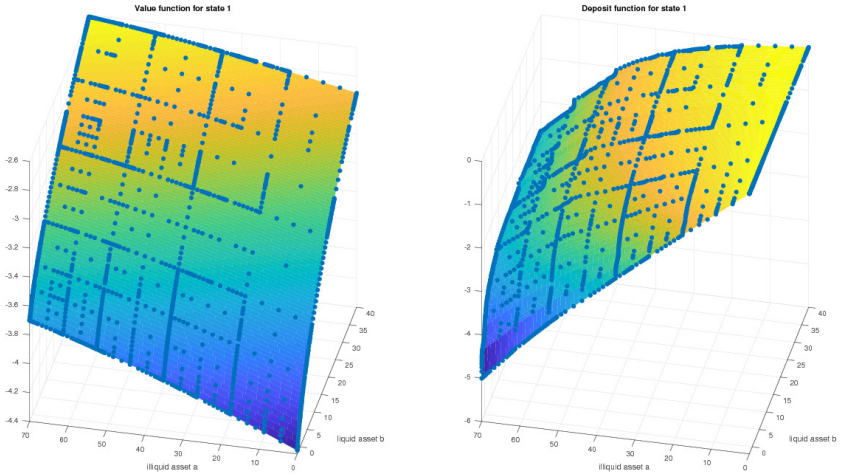


Fig. 12: Scatter and surface plot of the value and the deposit function for state 1 for the adapted sparse grid with deposit function adaptivity

The plots shown in Figure 12 visualize the sparse grid and the resulting value function for state 1 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.

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.

6.1.3 Accuracy for adaptive sparse grids

To get more insight into the approximation quality of different types of adaptivity (see Section 3.2) we look into the discrete relative $e_{2,r}$- and $e_{\infty,r}$-errors noted in the beginning of this section. 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 present in Figure 13 the results for different refinement thresholds, where we limit the maximum number of adaption steps so that we not exceed the level of the reference grid. We can observe that value function adaptivity performs well for the value function approximation. In some cases other adaptivity versions outperform in the beginning the value function adaptivity for the deposit policy accuracy. Concerning the l_2-error, the other adaptivity critera are not better, if at all, than a regular sparse grids. For the maximum error this depends on the function under consideration, e.g. for the deposit policy function only with finer resolutions the adaptation based on the value function helps.

Note that it depends on the model parameters if value function adaptivity or policy function adaptivity is better. 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 recommend value function adaptivity, which turns out to be the best approach in most situations. Further, we observed that it requires fine-tuning and testing, or an algorithm for parameter optimization, to find a good combination of parameters improving on value function adaptivity.

6.2 Four-dimensional model

Let us present our results for the $4d$ model (30) - (31) explained in the Appendix. 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 $l = 8$. Instead of computing the error on the grid of the reference solution, we compute the error by interpolating on uniformly distributed points for both the reference and the analyzed solutions.

We present in Figure 14 the results for different refinement thresholds, where we limit the maximum number of adaption steps so that we not exceed the level of the reference grid. We compare the results for value function adaptivity, deposit function adaptivity, and by logical OR combined value and deposit function adaptivity.

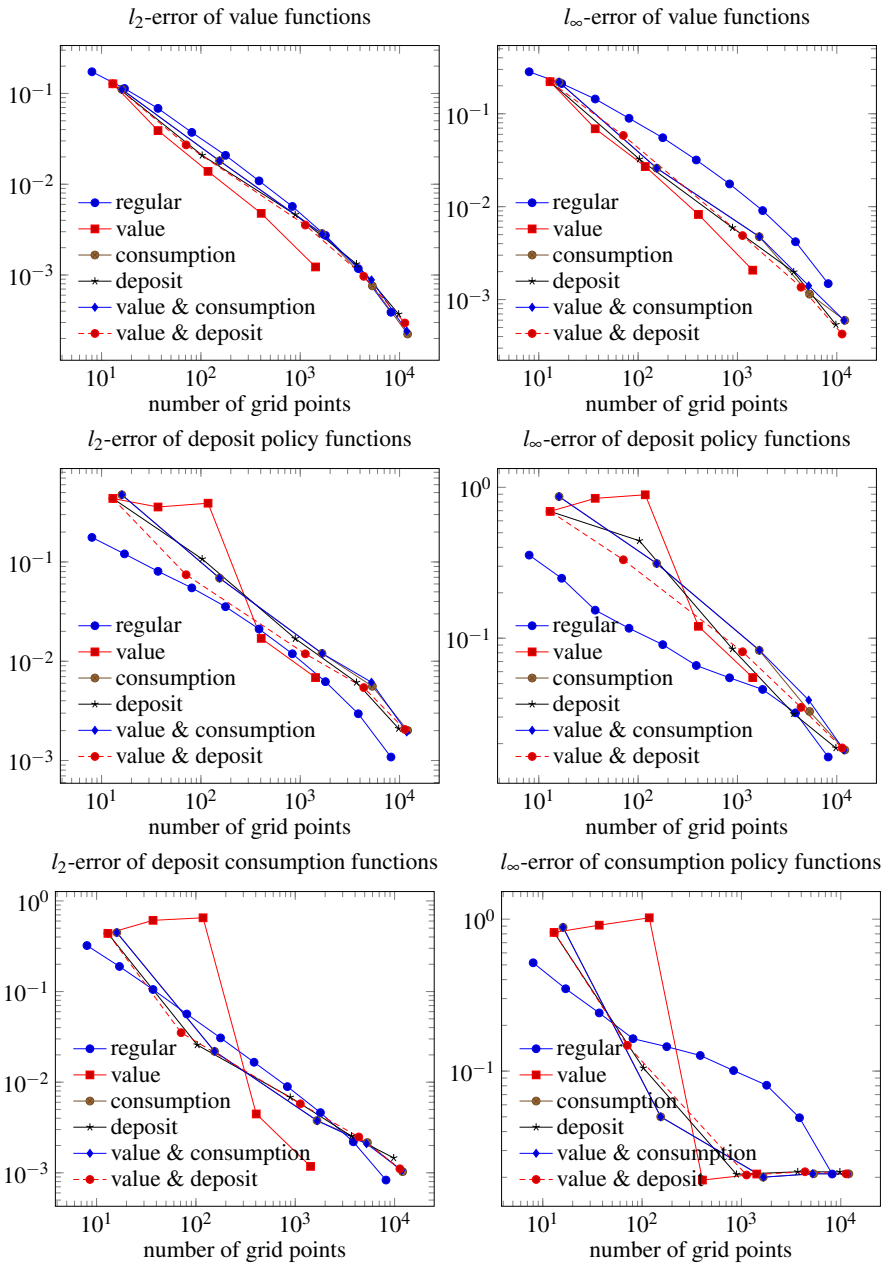


Fig. 13: Accuracy plots for the two-dimensional problem using different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (marked on the respective lines) after using at most ten adaption steps.

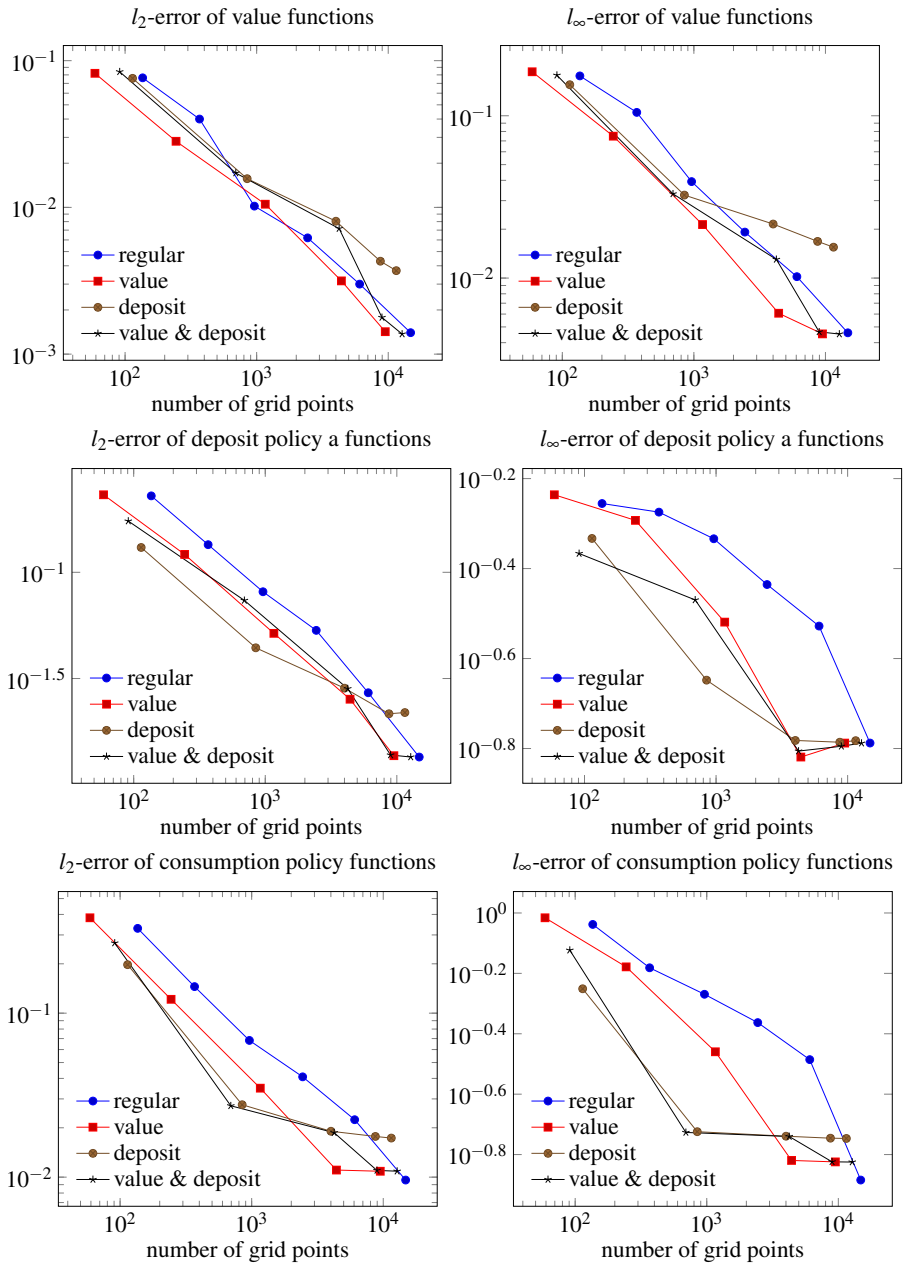


Fig. 14: Accuracy plots for the four-dimensional problem using different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (marked on the respective lines) after using at most five adaption steps.

Note that all adaptivity versions work for the policy function approximation, where for the maximum error is relatively large. However, for the value function approximation, one can see that value function adaptivity works better than the other adaptivities. Overall, the adaptivity gives better results in comparison to a regular grids. We observe a stagnation in particular for the policy function, we assume this is due to the limitation of the refinement level in this study.

6.3 Six-dimensional model

Finally we give results for the $6d$ model (32) - (33) explained in the Appendix. 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 $l = 6$. Again, we do not add points which are not in this grid in our adaptation by limiting the maximum number of adaptation steps. As in the last subsection, we interpolate on uniformly distributed points for both the reference and the analyzed function for the error computations.

As in the lower dimensional experiments, value function adaptivity 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. The advantage of the adaptive approaches in comparison to the regular sparse grid further increases.

7 Conclusion and Outlook

In this work we explained a sparse grid finite difference approach for solving economic models following the numerical scheme of [2].

To get a general convergence result for sparse grids finite difference schemes for solving the HJB equation due to [4], 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 analyzed the accuracy for our approach for economic models ranging from dimension $d = 2$ to dimension $d = 6$ and achieve good results for the used model parameters. 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

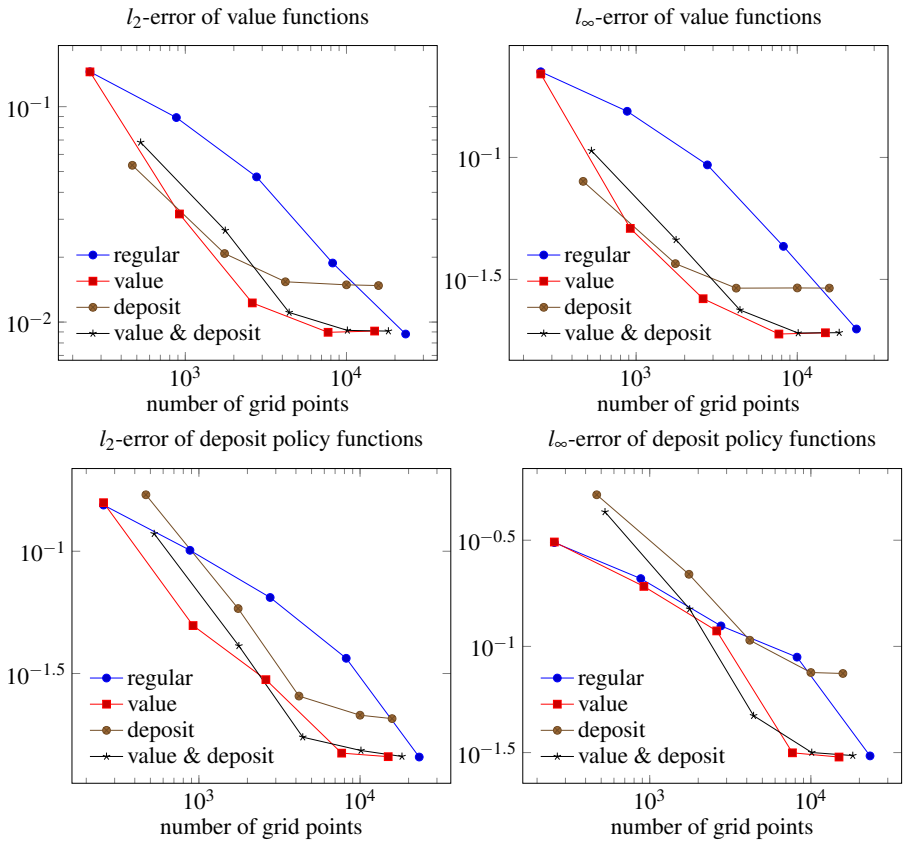


Fig. 15: Accuracy plots for the six-dimensional problem for different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (marked on the respective lines) after using at most four adaption steps.

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. Nevertheless, we recommend to use value function adaptivity since it leads to the best results in most cases.

Note that with the current Matlab implementation we cannot go to higher sparse grid levels and refinements since it requires allocating large amounts of memory and we thus face memory constraints. Thus, for [3] several Matlab functions are

rewritten using MEX-files, but also an implementation using other performant sparse grid libraries seems promising.

Acknowledgements We thank SeHyoun Ahn and Benjamin Moll for fruitful discussions, and SeHyoun Ahn for help with the Matlab implementation, which is based on his code written for [3].

Appendix

7.1 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 and to Section 5 for explanations of the numerical approach.

We are now interested in the following maximization problem

$$\max_{\{c_t,d_t^a,d_t^h\}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t,h_t)dt \tag{30}$$

subject to

$$\begin{aligned}
\dot{b}_t &= wz_t r^b(b_t)b_t - d_t^a - \chi(d_t^a,a_t) - d_h^t - \chi(d_t^h,h_t) - c_t \\
\dot{a}_t &= r^a a_t + d_t^a \\
\dot{h}_t &= d_t^h \\
\dot{z}_t &= \mu(z_t)dt + \sigma(z_t)dW_t \\
b_t &\geq \underline{b},\ a_t \geq 0,\ h_t \geq 0
\end{aligned} \tag{31}$$

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

$$\begin{aligned}\rho v(b,a,h,z) = \max_{c,d^a,d^h} & u(c,h) \\ & + v_b(b,a,h,z)(wz + r^b(b)b - d^a - \chi(d^a,a) - d^h - \chi(d^h,h) - c) \\ & + v_a(b,a,h,z)(r^a + d^a) \\ & + v_h(b,a,h,z)(d^h) \\ & + \partial_z v(b,a,h,z)\mu(z) + \frac{1}{2}\partial_{zz} v(b,a,h,z)\sigma^2(z).\end{aligned}$$

7.2 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 2.3. 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.

We are interested in the following maximization problem

$$\max_{\{c_t,d_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{32}$$

subject to

$$\begin{aligned}\dot{b}_t &= \frac{(z_t^1 + z_t^2 + z_t^3 + z_t^4)}{4} w r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \\ \dot{a}_t &= r^a a_t + d_t \\ \dot{z}_t^1 &= \mu(z_t^1)dt + \sigma(z_t^1)dW_t \\ \dot{z}_t^2 &= \mu(z_t^2)dt + \sigma(z_t^2)dW_t \\ \dot{z}_t^3 &= \mu(z_t^3)dt + \sigma(z_t^3)dW_t \\ \dot{z}_t^4 &= \mu(z_t^4)dt + \sigma(z_t^4)dW_t \\ b_t &\geq b,\ a_t \geq 0\end{aligned} \tag{33}$$

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

$$
\begin{aligned}
&\rho v(b,a,z^1,z^2,z^3,z^4) \\
&\quad = \max_{c,d} u(c) \\
&\qquad + v_b(b,a,z^1,z^2,z^3,z^4)\left(\frac{(z^1+z^2+z^3+z^4)}{4}w + r^b(b)b - d - \chi(d,a) - c\right) \\
&\qquad + v_a(b,a,z^1,z^2,z^3,z^4)(r^a+d) \\
&\qquad + \partial_{z^1} v(b,a,z^1,z^2,z^3,z^4)\mu(z^1) + \frac{1}{2}\partial_{z^1 z^1} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^1) \\
&\qquad + \partial_{z^2} v(b,a,z^1,z^2,z^3,z^4)\mu(z^2) + \frac{1}{2}\partial_{z^2 z^2} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^2) \\
&\qquad + \partial_{z^3} v(b,a,z^1,z^2,z^3,z^4)\mu(z^3) + \frac{1}{2}\partial_{z^3 z^3} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^3) \\
&\qquad + \partial_{z^4} v(b,a,z^1,z^2,z^3,z^4)\mu(z^4) + \frac{1}{2}\partial_{z^4 z^4} v(b,a,z^1,z^2,z^3,z^4)\sigma^2(z^4).
\end{aligned}
$$

7.3 Parameters

We here give model and algorithm parameters that we used in our numerical studies.

7.3.1 Parameters for the two-dimensional model

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 1: Model parameters for the $2d$ model (13)- (14).

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 1
Δ	100	Δ in HJB equation

Table 2: Algorithm 1 parameters for the 2*d* model.

State	Lower bound	Upper bound	Description
b	-2	40	liquid asset
a	0	70	illiquid asset

Table 3: Lower and upper bounds for the respective states in the 2*d* model. The lower bounds are model parameters, whereas the upper bounds for the assets are numerical bounds on the computational domain.

7.3.2 Parameters for the four-dimensional model

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 4: Model parameters for the 4*d* model (30) - (31).

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 1
Δ	100	Δ in HJB equation

Table 5: Algorithm 1 parameters for the 4*d* model.

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 6: Lower and upper bounds for the respective states in the 4*d* model. All lower bounds and the upper bound of productivity are model parameters, whereas the upper bounds for the assets are numerical bounds on the computational domain.

7.3.3 Parameters for the six-dimensional model

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 7: Model parameters for the 6*d* model (32) - (33).

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 1
Δ	100	Δ in HJB equation

Table 8: Algorithm 1 parameters for the 6*d* model.

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 9: Lower and upper bounds for the respective states in the 6*d* model. All lower bounds and the upper bound of productivity are model parameters, whereas the upper bounds for the assets are numerical bounds on the computational domain.

References

1. Y. Achdou, G. Barles, H. Ishii, and G. L. Litvinov. *Hamilton-Jacobi equations: approximations, numerical analysis and applications*. Springer, 2013.
2. Y. Achdou, J. Han, J.-M. Lasry, P.-L. Lions, and B. Moll. Income and wealth distribution in macroeconomics: A continuous-time approach. *Review of Economic Studies*, 2017. submitted.
3. S. Ahn. Sparse grid methods for economic models. unpublished manuscript, 2017.
4. G. Barles and P. Souganidis. *Convergence of Approximation Schemes for Fully Nonlinear Second Order Equations*. Lefschetz Center for Dynamical Systems and Center for Control Sciences, Division of Applied Mathematics, Brown University, 1990.
5. R. E. Bellman. *Dynamic programming*. Princeton University Press, 1957.
6. R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961.
7. D. P. Bertsekas and S. Shreve. *Stochastic optimal control: the discrete-time case*. 2004.
8. J. Brumm and S. Scheidegger. Using adaptive sparse grids to solve high-dimensional dynamic models. *Econometrica*, 85(5):1575–1612, 2017.
9. H.-J. Bungatrz and M. Griebel. Sparse grids. *Acta Numerica*, 13(1):147–269, 2004.
10. G. V. Candler. Finite-difference methods for dynamic programming problems. *Computational Methods for the Study of Dynamic Economies. Cambridge University Press*, 1999.
11. M. Falcone and R. Ferretti. *Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations*. SIAM, 2013.
12. J. Garcke. Sparse grids in a nutshell. In J. Garcke and M. Griebel, editors, *Sparse grids and applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 57–80. Springer, 2013.
13. J. Garcke and A. Kröner. Suboptimal feedback control of PDEs by solving HJB equations on adaptive sparse grids. *Journal of Scientific Computing*, 70(1):1–28, 2017.

14. M. Griebel. Adaptive sparse grid multilevel methods for elliptic pdes based on finite differences. *Computing*, 61(2):151–179, 1998.
15. M. Griebel and T. Schiekofer. An adaptive sparse grid navier–stokes solver in 3d based on the finite difference method. In *Proc. ENUMATH97*, 1999.
16. P. W. Hemker. Application of an adaptive sparse-grid technique to a model singular perturbation problem. *Computing*, 65(4):357–378, 2000.
17. H. Jin and K. L. Judd. Perturbation methods for general dynamic stochastic models. Technical report, Mimeo April, 2002.
18. K. Judd, L. Maliar, and S. Maliar. Numerically stable and accurate stochastic simulation methods for solving dynamic models. *Quantitative Economics*, 2(2):173–210, August 2011.
19. K. L. Judd, L. Maliar, S. Maliar, and R. Valero. Smolyak method for solving dynamic economic models. *Journal of Economic Dynamics and Control*, 44(C):92–123, 2014.
20. G. Kaplan, B. Moll, and G. L. Violante. Monetary policy according to HANK. *American Economic Review*, 108(3):697–743, 2019.
21. F. Koster. *Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern*. PhD thesis, Institut für Angewandte Mathematik, Universität Bonn, 2002.
22. D. Krueger and F. Kubler. Computing equilibrium in olg models with stochastic production. *Journal of Economic Dynamics and Control*, 28(7):1411–1436, 2004.
23. H. Kushner and P. G. Dupuis. *Numerical methods for stochastic control problems in continuous time*, volume 24. Springer Science & Business Media, 2013.
24. H. P. Langtangen. *Computational partial differential equations: numerical methods and diffpack programming*, volume 2. Springer Science & Business Media, 2013.
25. L. Maliar, S. Maliar, and S. Villemot. Taking perturbation to the accuracy frontier: a hybrid of local and global solutions. *Computational Economics*, 42(3):307–325, 2013.
26. B. Moll. Lecture Notes to Income and Wealth Distribution in Macroeconomics, 2016. Princeton.
27. D. W. Peaceman and H. H. Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for industrial and Applied Mathematics*, 3(1):28–41, 1955.
28. D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Verlag Dr. Hut, München, Aug. 2010.
29. P. E. Protter. Stochastic differential equations. In *Stochastic integration and differential equations*, pages 249–361. Springer, 2005.
30. S. Ruttscheidt. Adaptive Sparse Grids for Solving Continuous Time Heterogeneous Agent Models. Master thesis, Institut für Numerische Simulation, Universität Bonn, 2018.
31. T. Schiekofer. *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen*. PhD thesis, Institut für Angewandte Mathematik, Universität Bonn, 1998.
32. K. Schmedders and K. Judd. *Handbook of Computational Economics*. Number Bd. 3 in Handbook of Computational Economics. Elsevier Science, 2013.
33. P. Schober. Solving dynamic portfolio choice models in discrete time using spatially adaptive sparse grids. In J. Garcke, D. Pflüger, C. G. Webster, and G. Zhang, editors, *Sparse Grids and Applications - Miami 2016*, pages 135–173, Cham, 2018. Springer International Publishing.
34. S. A. Smolyak. Quadrature and interpolation formulas for tensor products of certain class of functions. *Dokl. Akad. Nauk SSSR*, 148(5):1042–1053, 1963. Transl.: Soviet Math. Dokl. 4:240-243, 1963.
35. C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, 1990*, volume 31 of *Notes on Num. Fluid Mech.*, pages 241–251. Vieweg-Verlag, 1991.
36. G. W. Zumbusch. A sparse grid pde solver; discretization, adaptivity, software design and parallelization. *Advances in Software Tools for Scientific Computing*, 10:133–177, 2000.