algorithms

Article

Finite Differences on Sparse Grids for Continuous-Time Heterogeneous Agent Models

Jochen Garcke [1,2,*] and Steffen Ruttscheidt [1]

1 Institut für Numerische Simulation, Universität Bonn, 53111 Bonn, Germany
2 Fraunhofer SCAI, 53754 Sankt Augustin, Germany
* Correspondence: garcke@ins.uni-bonn.de

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. While one can show convergence for the used finite difference method on full grids by using the theory introduced by Barles and Souganidis, 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.

Keywords: sparse grids; Hamilton–Jacobi–Bellman equation; high-dimensional approximation

Academic Editors: Shuai Li and Dunhui Xiao

Received: 26 November 2024
Revised: 2 January 2025
Accepted: 6 January 2025
Published: 12 January 2025

Citation: Garcke, J.; Ruttscheidt, S. Finite Differences on Sparse Grids for Continuous-Time Heterogeneous Agent Models. *Algorithms* **2025**, *18*, 40. https://doi.org/10.3390/a18010040

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In recent years, advances in economic research 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, and asset 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. For this purpose we propose and investigate adaptive sparse grids. 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 [1] 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 (FD) approach on sparse grids that is based on standard interpolation and allows easy implementation.

In [2], a finite difference method is used to solve the Hamilton–Jacobi–Bellman (HJB) equation in the economic context. This approach was improved in [3] to handle borrowing constraints by mathematically recasting them as state constraints. The computational method was further adapted in [4] to handle non-convexities and multiple assets.

In [5], finite differences on sparse grids were introduced and several theoretical results regarding consistency, stability, and convergence are shown. Further studies have been made in [6–8]. 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 and employ a finite difference method following [9], which is based on interpolation on adaptive sparse grids and which can be implemented easily using existing sparse grid implementations. Notice that interpolation-based ideas were already presented in [10]. With the presented finite difference approach using adaptive sparse grid interpolation, we are able to solve continuous-time heterogeneous agent models in higher dimensions. Note that we use a finite difference approach following [3] to solve these agent models, but instead of implementing it on full grids we performed it on adaptive sparse grids. This allowed the numerical treatment of scenarios in up to six dimensions, thereby extending the range of possible numerical studies, in particular economic.

This article 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 briefly describe sparse grids and the employed finite differences on sparse grids [9]. An investigation of sparse grid interpolation is carried out in Section 3.3 to explain why we do not simply obtain convergence by means of Barles and Souganidis [11], 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. This is followed by a detailed presentation of the algorithm and its implementation in Section 4. Even though we do not have a theoretical convergence result, we give numerical results in Section 5 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 6. 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 has been progress using continuous-time formulations. Several well-known heterogeneous agent models (e.g., Bewley, Huggett, and Aiyagari models) were recasted in continuous time by [3]. They state computational advantages relative to discrete time, including the handling of borrowing constraints as a simple boundary condition on the value function, the numerical solution of first order conditions characterizing optimal policy functions, and the observation that continuous-time problems with discretized state space generate sparsity in the matrices characterizing the model's equilibrium conditions. Even though we restrict ourselves to certain types of models in our experiments, we point out that the presented framework is basically applicable to any heterogeneous agent model. We here mainly follow [12]. For mathematical descriptions and proofs, see [13].

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 $x(0) = x_0$ using the notation $\dot{x}(t) = \frac{d}{dt}x(t)$.

Here, $x \in X \subset \mathbb{R}^m$ denotes the *state vector*, $\alpha(t) \in A \subset \mathbb{R}^n$ 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. 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.

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 $\hat{\alpha}(t) \in A,\ t \geq 0$ such that

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

To solve optimal control problems, one uses the *dynamic programming principle* (DPP) introduced by Bellman; see [14]. 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 [15],

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

To compute the optimal controls, one typically 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. The optimal controls for all states are collected in the so-called *policy function*.

2.2. Optimal Control Problems in Economics

The above framework can be used to model economic settings. We present a slightly simplified two asset model from [4]; further details and economic background can be found therein. Note that we have omitted the initial state in the following and denoted the time dependence of the states by a subscript t.

Specifically, we want to solve the following maximization problem

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

subject to

$$\dot{b}_t = w z_t + r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \tag{4a}$$
$$\dot{a}_t = r^a a_t + d_t \tag{4b}$$
$$z_t = \text{Poisson with intensities } \lambda(z, z') \tag{4c}$$
$$b_t \geq \underline{b},\ a_t \geq 0. \tag{4d}$$

Here, b_t denotes liquid assets and a_t illiquid assets. The respective returns on these assets are r^b and r^a, where $r^b(b_t)$ summarizes the interest rate schedule faced by households. Further, we have consumption c_t, deposits d_t, and the transaction cost function χ. Instead of obtaining wage w, one receives uninsured income $w z_t$, where we assume that the exogenous productivity state z evolves stochastically over time, which is modeled as a Poisson or diffusion process.

While *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 conducted in [4], 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_t, this is a borrowing rate. A *borrowing constraint* is the maximum amount of money an agent can borrow, e.g., from banks, firms, or governments [3]. It is modeled as $b_t \geq \underline{b}$. Note that $\underline{b} = 0$ implies that the agent cannot borrow, but just save. Contrary to liquid assets b_t, illiquid assets a_t cannot be sold that easily without losing 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 [4], it is pointed out that in the equilibrium, illiquid assets pay a higher return than liquid assets due to the transaction costs, i.e., $r^a > r^b$.

In the framework given in Section 2.1, we have the state $x(t) = (b_t, a_t)$ consisting of liquid b_t and illiquid a_t assets. The control $\alpha(t)$ at time t reflects the consumption c_t and deposit d_t. Moreover, the state changes $f(x(t), \alpha(t)) = (wz_t + r^b(b_t)b_t - d_t - \chi(d_t, a_t) - c_t, r^a a_t + d_t)^T$ at time t follows (4a) and (4b). Thus, at time t, for the asset state (b_t, a_t), we want to choose an optimal control (c_t, d_t), i.e., how much we consume and deposit, to maximize (3). Note again that this choice directly results in a change in the state.

A standard choice for the return function $h(c)$ in (3) is the Constant Relative Risk Aversion (CRRA)-utility function u given by

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

with risk aversion $\gamma > 0$. Note that u is strictly convex and strictly monotonously increasing in c.

We use in (4a) the transaction cost function χ from [16] 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 the derivative with respect to d given by

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

Here, the linear component $\chi_0 > 0$ generates inaction in optimal deposit decisions. The quadratic term with $\chi_1 > 0$ ensures that deposit rates d/a are finite, so that households' asset holdings never jump.

By using standard arguments, one obtains the HJB equation

$$\begin{aligned}\rho v(b, a, z_i) = &\max_{c,d} u(c) \\ &+ \partial_b v(b, a, z_i)(wz + r^b(b)b - d - \chi(d, a) - c) \\ &+ \partial_a v(b, a, z_i)(r^a + d) \\ &+ \sum_{j=1}^{2} \lambda(i, j)(v(b, a, z_j) - v(b, a, z_i)),\end{aligned} \tag{6}$$

for $i = 1, 2$ for the two state Poisson process. The first order conditions with respect to c and d yield

$$\partial_c u(c) = \partial_b v(b, a, z), \tag{7a}$$
$$\partial_a v(b, a, z) = \partial_b v(b, a, z)(1 + \partial_d \chi(d, a)) \tag{7b}$$

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 obtain the optimal deposits for illiquid assets

$$\begin{aligned} d &= \underbrace{d^+}_{\text{case } d>0} + \underbrace{d^-}_{\text{case } d<0} \\ &= \left(\Big(\frac{v_a}{v_b} - 1 - \chi_0\Big) \frac{a}{\chi_1} \right)^+ + \left(\Big(\frac{v_a}{v_b} - 1 + \chi_0\Big) \frac{a}{\chi_1} \right)^- . \end{aligned} \tag{8}$$

Both can be collected per state to obtain the policy function given the value function.

In (4), the productivity z_t follows a Poisson process with intensities $\lambda(z, z')$. The setting can be easily extended to diffusion type stochastic processes. Thus, *productivity*, which is a measure for the output per unit of input, is modeled such that it influences the households income as wz_t. For the same model setup, one could also interpret z as a 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.

Higher Dimensional Models

In the appendix, we give higher dimensional models that 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.

2.3. Approaches Used in Economics to Handle High-Dimensional Discrete Time Model Problems

We refer to [17] 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. When 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 [19], sparse grids are used to replace the expensive tensor product grids. For perturbation methods that are feasible in higher dimensions, see [20,21].

Sparse grids in combination with a fixed point iteration on the Euler equation are proposed in [22] 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 used in [23] to solve high-dimensional stochastic dynamic models where functions are interpolated on a sparse grid either within time or value iterations. Further, in [24], dynamic portfolio choice models are solved with adaptive sparse grids, where in addition to an adaptive approximation of the value function, separate adaptive sparse grids for policy functions are also computed. A recent review on sparse grids for dynamic economic models can be found in [25]. The finite difference operators on sparse grids from [5] were recently adapted for economic applications in [26].

For a general overview of stochastic optimal control in the discrete time case, we refer to [27] and the references therein.

3. Sparse Grids

Sparse grids were introduced in [28] and date back to [29]. We give only a short introduction here. See [30–32] for details and approximation properties.

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 $x_{\mathbf{l},\mathbf{i}}$, where $\mathbf{i}$ indicates its spatial position, we associate a piecewise d-linear nodal basis function,

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

which is the product of the one-dimensional basis functions and has a support of size $2\mathbf{h_l}$. The one-dimensional $\phi_{l_j,i_j}(x_j)$ are the well-known hat functions. 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*

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

Let us define the *full grid spaces*

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

where each function $f \in V_n$ can be represented as

$$f(x) = \sum_{|\mathbf{l}|_\infty \leq n} \sum_{\substack{\mathbf{1} \leq \mathbf{i} \leq 2^\mathbf{l} - \mathbf{1}, \\ i_j \text{ odd } \forall \mathbf{1} \leq j \leq d}} \alpha_{\mathbf{l},\mathbf{i}} \cdot \Phi_{\mathbf{l},\mathbf{i}}(x), \tag{9}$$

and $\alpha_{\mathbf{l},\mathbf{i}} \in \mathbb{R}$ are the coefficients of the representation in the hierarchical tensor product basis. We can generalize the hierarchical representation (9) to different levels of discretization $\mathbf{k}$ per dimension, i.e., $f \in V_\mathbf{k}$, by replacing $|\mathbf{l}|_\infty \leq n$ in the first sum by $l_i \leq k_i, \forall \mathbf{1} \leq j \leq d$.

Now consider the d-linear interpolation of a function f by a $f_n \in V_n$, i.e., a representation as in (9). For illustration, we look at the linear interpolation in one dimension; for the hierarchical coefficients $\alpha_{l,j}$, $l \geq 1$, i odd, it holds

$$\begin{aligned}\alpha_{l,i} &= f(x_{l,i}) - \frac{f(x_{l,i} - h_l) + f(x_{l,i} + h_l)}{2} = f(x_{l,i}) - \frac{f(x_{l,i-1}) + f(x_{l,i+1})}{2} \\ &= f(x_{l,i}) - \frac{f(x_{l-1,(i-1)/2}) + f(x_{l-1,(i+1)/2})}{2}.\end{aligned}$$

This illustrates why the $\alpha_{l,i}$ are also called *hierarchical surplus*: they specify what has to be added to the hierarchical representation from level $l-1$ to obtain the one of level l. We can rewrite this in the following operator form

$$\alpha_{l,i} = \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}_{l,i} f$$

and with that we generalize to the d-dimensional hierarchization operator as follows,

$$\alpha_{\mathbf{l},\mathbf{i}} = \left(\prod_{t=1}^{d} \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}_{l_t,i_t} \right) f. \tag{10}$$

We denote $\mathbf{H}$ with the *hierarchization operator* that performs the transformation (10) from the nodal function values of $f \in V_n$ to obtain all the arising hierarchical values $\alpha_{\mathbf{l},\mathbf{i}}$. The inverse operator $\mathbf{E} = \mathbf{H}^{-1}$ is called the *dehierarchization operator* and computes from the hierarchical values $\alpha_{\mathbf{l},\mathbf{i}}$ the corresponding (nodal) function values f on all the grid points.

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

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

where instead of the maximum of the level indices their sum is used. Here, hierarchical basis functions with a small support, and therefore a small contribution to the function representation [30,31], are not included in the discrete space $\hat{V}_n$ of level n anymore. Note that the change from n to $n+d-1$ has to do with the boundary treatment, the underlying aspects are not relevant in the scope of this work, they can be found, e.g., in [31]. A function in $\hat{V}_n$ is represented in the hierarchical basis analogue to (9) as

$$f(x) = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\substack{\mathbf{1} \leq \mathbf{i} \leq 2^{\mathbf{l}} - \mathbf{1}, \\ i_j \text{ odd } \forall \mathbf{1} \leq j \leq d}} \alpha_{\mathbf{l},\mathbf{i}} \cdot \Phi_{\mathbf{l},\mathbf{i}}(x). \tag{11}$$

We define the set I_n of all indices of functions in $\hat{V}_n$ by $\mathsf{I}_n := \{(\mathbf{l},\mathbf{j}) | |\mathbf{l}|_1 \leq n+d-1, \mathbf{1} \leq \mathbf{i} \leq 2^{\mathbf{l}} - \mathbf{1},\ i_j \text{ odd } \forall \mathbf{1} \leq j \leq d\}$.

For a simplified exposition, we have so far only considered functions that are zero on the boundary of the domain. To allow non-zero boundary values, one introduces additional nodes on the boundary. This can be achieved by adding two boundary points on level $l = 1$ in the construction. By conducting this, we can obtain a modified set of subspaces $\tilde{W}_{\mathbf{l}}$ by the construction explained before.

See [25,26,30–32] for further background and details.

3.1. Finite Difference Schemes on Sparse Grids

Finite differences on sparse grids were introduced and studied in [5], where consistency proofs and convergence results are given; see also [6,7,33]. The construction of finite difference operators is based on a dimensional splitting combined with the nodal to hierarchical basis transformation (10) and its respective back-transform.

Let us first describe the original sparse grid finite difference approach, where operators are a composition of three partial operators [5]:

- A basis transformation from nodal to hierarchical basis (10) 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, i.e., reverse of (10), 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, as later explained in Section 3.2, is used, the grid points in the different dimensions are no longer equidistant, that is the distance can no longer be defined based on the grid refinement level, but the stencil is still chosen such that the closest neighbors in the respective dimensions are used.

We consider an alternative approach [9], which employs additional points and is simpler to implement since it only involves function evaluations. 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 basis transformations such as (10) and one could simply take any sparse grid library, such as SG++ [32], to implement this approach.

To describe the approach from [9], 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*. Examples for forward difference ghost nodes are given in Figure 1. Notice that for the second derivative finite difference, we can, as usual, employ the first derivative operators twice. 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*

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

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

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

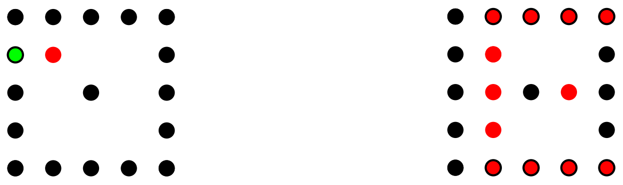


Figure 1. Visualization of the ghost points for the forward difference in x-dimension. **Left**: the ghost node (red) that is used for the sparse grid forward finite differences in x-dimension in the green grid point. **Right**: all forward difference ghost nodes that are used for the sparse grid are drawn in red. On the boundary, the red ghost nodes coincide with existing grid points, while in the interior function interpolation has to be used on the ghost nodes.

The main idea of the construction of finite difference operators is simply to collect the terms that are considered constants under partial differentiation. When one computes partial derivatives, other variables are held constant by definition. Therefore, if a function is multiplicatively separable in other variables, one can just compute the derivative of the variable as in one dimension. As the set of basis functions are built from hierarchical functions in one dimension, the basis functions lead to separable functions.

Without a loss of generality, we describe the finite difference operators in two dimensions with the derivative taken in the first dimension. The generalization to higher dimensions is straightforward using j for the dimension in which the derivative is taken and a multi-index $-j$ represents all other dimensions besides j. Using the fact that $\phi_{l_2,i_2}(x_2)$ from $\Phi_{(l_1,l_2),(i_1,i_2)}(x_1,x_2) = \phi_{l_1,i_1}(x_1)\phi_{l_2,i_2}(x_2)$ does not depend on x_1, and the linearity of the differential operator, we obtain

$$\begin{aligned}
\frac{\partial}{\partial x_1} f(x) &= \frac{\partial}{\partial x_1} \sum_{(l_1,l_2),(i_1,i_2)\in \mathsf{I}_n} \alpha_{(l_1,l_2),(i_1,i_2)} \phi_{l_1,i_1}(x_1)\phi_{l_2,i_2}(x_2) \\
&= \frac{\partial}{\partial x_1}\left[\sum_{(\cdot,l_2),(\cdot,i_2)\in \mathsf{I}_n} \phi_{l_2,i_2}(x_2) \cdot \left(\sum_{\{(l_1,i_1):(l_1,l_2),(i_1,i_2)\in \mathsf{I}_n\}} \alpha_{(l_1,l_2),(i_1,i_2)} \phi_{l_1,i_1}(x_1) \right) \right] \\
&= \sum_{(\cdot,l_2),(\cdot,i_2)\in \mathsf{I}_n} \phi_{l_2,i_2}(x_2) \cdot \frac{\partial}{\partial x_1} \left(\sum_{\{(l_1,i_1):(l_1,l_2),(i_1,i_2)\in \mathsf{I}_n\}} \alpha_{(l_1,l_2),(i_1,i_2)} \phi_{l_1,i_1}(x_1) \right).
\end{aligned}$$

Now, the expression in the ()-brackets to be differentiated is an one-dimensional function of x_1, on which any finite difference scheme can be applied to approximate the derivative.

Using, e.g., the forward difference,

$$\frac{\psi(x+h) - \psi(x)}{h}, \text{ where } \psi(x) := \sum_{\{(l_1,i_1):(l_1,l_2),(i_1,i_2)\in \mathsf{I}_n\}} \alpha_{(l_1,l_2),(i_1,i_2)} \phi_{l_1,i_1}(x_1), \tag{12}$$

we can now define a finite difference operator on sparse grids that is based on interpolation at (x_1, x_2) and $(x_1 + h, x_2)$. For a given grid and the desired difference operator, we can simply consider all respective ghost nodes arising from $(x_1 + h, x_2)$ and interpolate on these.

Definition 3 (Interpolation-based sparse grid finite difference operator)**.** *Let us formally denote the interpolation operator to be evaluated on $x_{\mathbf{l},\mathbf{i}}$ on the sparse grid by*

$$\mathbf{I}_s : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}},$$

which is essentially (11) *for $V_{\mathbf{l}}$. We define the corresponding interpolation operator for the by h_{g_j} shifted sparse grid, that is the grid of ghost nodes, needed for the forward difference $\mathbf{I}^F_{h_{g_j}}$ and backward difference $\mathbf{I}^B_{h_{g_j}}$, respectively, in dimension j, $1 \leq j \leq d$ by*

$$\mathbf{I}^F_{h_{g_j}} : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}} \quad \textit{and} \quad \mathbf{I}^B_{h_{g_j}} : \bigoplus_{\mathbf{k}=\mathbf{1}}^{\mathbf{l}} W_{\mathbf{k}} \to V_{\mathbf{l}},$$

respectively. We define the sparse grid forward difference operator $\tilde{\mathbf{D}}^{S,F}_j$ reflecting (12) *by*

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

The corresponding sparse grid backward difference operator $\tilde{\mathbf{D}}^{S,B}_j$ is given by

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

We point out that one does not need 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.

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 recomputation. First, one can use the interpolated values that one used for the first derivative finite differences also for the second derivative finite differences. Second, as is pointed out in [5], 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

$$\tilde{\mathbf{D}}^S_{jj} = \tilde{\mathbf{D}}^{S,B}_j \circ \tilde{\mathbf{D}}^{S,F}_j = \tilde{\mathbf{D}}^{S,F}_j \circ \tilde{\mathbf{D}}^{S,B}_j$$

which is a well-known identity for the full grid operators. This is due to the observation that locally the operator works on one-dimensional full grids [5].

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 [5] 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 the 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 in [33] that both approaches yield the same finite difference operator. The formal relationship between the two approaches needs further investigation.

3.2. Adaptive Sparse Grids

One can further reduce the computational complexity by using an adaptive sparse grid. For example, this is the case if the function has changes in its steepness, i.e., large second derivatives. The idea is to add new points to the sparse grid if it is likely that they increase the obtained accuracy. This is called *adaptive refinement*. As a criterion for adaptation, one typically uses a local error estimation based on the hierarchical surplus (coefficient), and then adds child nodes (in the hierarchical structure) to those points with a large estimate. Vice versa, grid points whose corresponding basis functions do not contribute much can be removed in a coarsening step. For a description of similar algorithms for refinement and coarsening in the optimal control setting, see [34,35], and a general view can be found in [32].

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 $\mathcal{I}$, 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. For our experiments, we use the coarsening parameter $\nu = \varepsilon/10$, which is a typical choice [34,35]. An additional component can be self-adaptivity as also used in [5], where the refinement threshold and coarsening parameter are automatically decreased when no new points are added by adapting with the current parameters.

3.3. (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 [11] involve a stable, consistent, and monotone scheme. While it is trivial to show that one needs monotone interpolation, interpolation on sparse grids is not monotone in general, as it is already pointed out in [36]; see also [32]. Since we are approximating value functions arising from economic models, it is of interest if one can achieve monotonicity by restricting ourselves to concave monotonically increasing functions as they arise for the employed models. Ref. [9] notes that the introduced upwind finite difference scheme converges even though the scheme is not monotone. However, Ref. [9] does not give examples or justifications that it is not monotone.

In just one dimension, the scheme is monotone since it is uses standard linear interpolation. Observe that we only need one-dimensional monotonicity per dimension, that is with respect to the used ghost points.

Let us give a definition of monotone interpolation.

Definition 4 (Monotone interpolation in one dimension)**.** *Let $x_1, \dots, x_n$ be data points with $x_1 < \dots < x_n$. A function f is called monotone if it holds that $f(x_1) \le \dots \le f(x_n)$ or $f(x_1) \ge \dots \ge f(x_n)$. 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*

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

or

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

with strict inequality for strictly monotone interpolation.

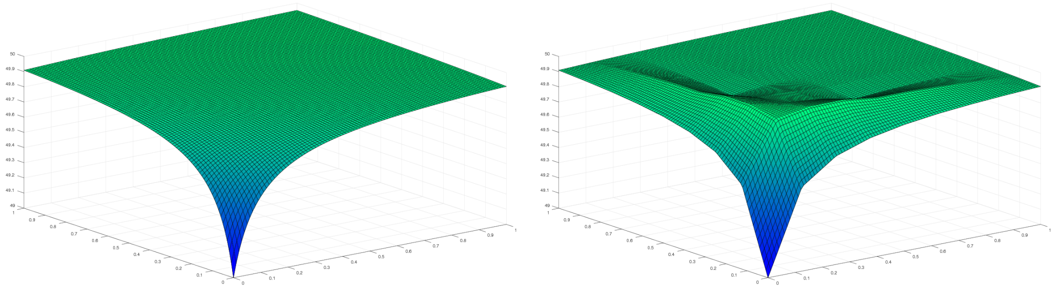
Note that we are interested in higher dimensions and aim for one-dimensional monotone interpolation for the restriction to the different dimensions.

Non-Monotone Sparse Grid Interpolation for Concave Monotonically Increasing Functions

We 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 concave monotonically increasing functions for which negative hierarchical coefficients arise. Let us consider the interpolation of the function

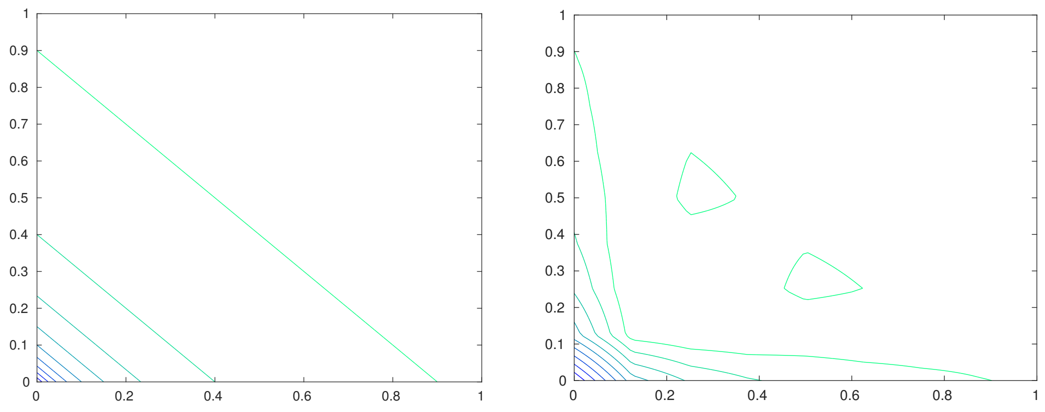
$$f_1(x, y) = \frac{-1}{1 + 10x + 10y} + 50,$$

which is similar to functions that arise as value functions 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, but note that it is not strictly concave. Unfortunately, we see in the function plots in Figure 2 and the contour plots in Figure 3 that the interpolation is not monotone. Increasing the factors in front of x and y increases this effect. Here, we use a sparse grid level $l = 3$, but corresponding counter examples can be constructed for other levels.



(**a**) Function plot of f_1. (**b**) Interpolation plot of f_1.

Figure 2. Plots of the original function f_1 on the left and its sparse grid interpolation of level $l = 3$ on the right.



(**a**) Contour plot of function f_1. (**b**) Contour plot of the interpolation of f_1.

Figure 3. Contour plots of the function f_1 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 4, 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.

Strictly concave functions can also yield non-monotone sparse grid interpolation, e.g., $f_2(x,y) = -1/(1 + (x+0.01)^{0.2} + (y+0.01)^{0.2}) + 50$. Here, one can check the concavity by using the leading principal minors criteria. Finally, we point out that one cannot simply set the negative hierarchical coefficients to zero to obtain a monotone approximation.

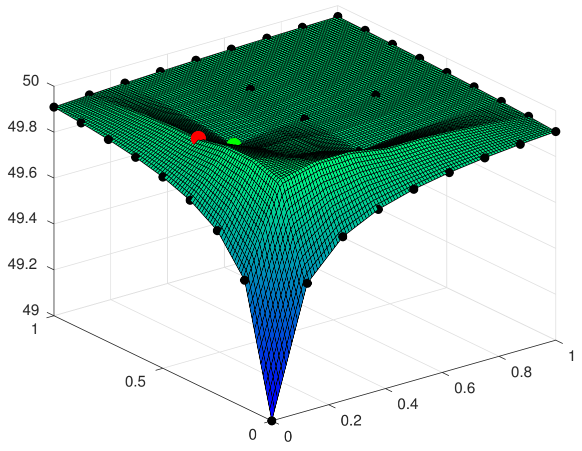


Figure 4. For the backward difference at $(0.5, 0.25)$, one uses the values drawn as the green and the red points.

3.4. Overcoming the Non-Monotonicity of Sparse Grid Interpolation

There are several possible approaches to obtain monotone interpolation on sparse grids. The most trivial way is to go to a higher sparse grid level; this is visualized in Figure 5, 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; moreover, for higher levels, corresponding counter examples can be constructed as well, therefore a guarantee of monotonicity cannot be expected.

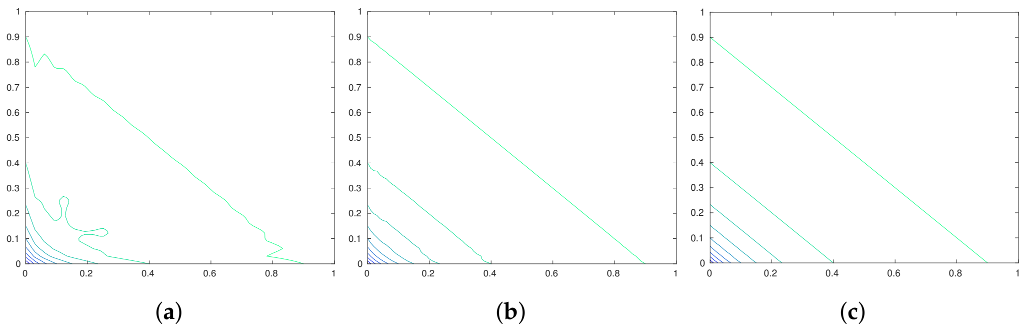


Figure 5. Plots of contour plots for the sparse grid interpolation of f_4 for different levels. (**a**) Contour plot for the interpolation of f_4 for level $l = 5$. (**b**) Contour plot for the interpolation of f_4 for level $l = 7$. (**c**) Contour plot for the interpolation of f_4 for level $l = 9$.

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 indicators, as is performed 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 [33]. Being of heuristic nature, we do not expect that monotonicity can be guaranteed. Specific to our situation, the most 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. With this approach, a guarantee of monotonicity might be achievable.

In our numerical experiments, we do not investigate these strategies and consider the enforcing of monotone interpolation on adaptive sparse grids a topic of future research.

4. Numerical Approach Using an Upwind Scheme

Let us follow the work of [3] and 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 [9]. You can find a more mathematical description of finite difference schemes for solving the HJB equation that is not targeted to economic models in [37].

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 [9], as described in Section 3.1. The idea in this work 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 section, we generalize the matrix notation so that it can easily be extended to the sparse grid setting.

We consider the model presented and explained in Section 2.2. Notice that (c_t, d_t) is our control and (b_t, a_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 in the state.

4.1. Discretization

To simplify the notation, we use here a reduced model derived from (4) that is without the illiquid assets a_t and deposit control d_t, i.e., it considers only the state b_t and control c_t. Therefore, the update in the state space is in the following $\dot{b}_t = wz_t + r^b(b_t)b_t - c_t$. The extension to the full model from (4) is commented upon in Section 4.2.

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

$$\rho v(b_j) = u(c_j) + v'(b_j)(w + r^b b_j - c_j), \quad c_j = (u')^{-1}(v'(b_j)),\ j = 1, \dots, J \tag{13}$$

where $v(b_j)' = v_j'$ is either the forward or the backward difference approximation. Note that the computation for c_j arises from the first order condition with respect to c_j and the equation is still highly nonlinear. It therefore has to be solved using an iterative scheme. In the following, whenever we state an equation for j, this holds true for $j = 1, \dots, J$.

An issue when constructing such a scheme is the monotonicity condition. We use an upwind scheme that gives a rule for the choice of the finite difference: (a) 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 (b) use the backward difference when it is negative. For a function v, let us 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. We define

$$s_j^F = w + r^b b_j - c_j^F \text{ and } s_j^B = w + r^b b_j - c_j^B \tag{14}$$

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 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 [3]. Furthermore, it is possible to construct other monotone schemes, but this one is perfectly suited to implement borrowing constraints that we now turn to.

4.1.1. 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 constrain, we need $s(\underline{b}) = w + r^b \underline{b} - c(\underline{b}) \geq 0$.

Since u is strictly monotonically increasing and concave, we obtain

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

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 [3].

4.1.2. Overcoming the Nonlinearity

HJB Equation (2) is nonlinear due to the maximum operator. We use an iterative scheme to solve this equation, i.e., *policy iteration*, as for the example explained in [15]. Its general idea is to linearize the HJB equation by omitting the maximum operator and using an iteration instead of searching for the maximum.

While explicit schemes are often easier to understand, 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. The implicit scheme is now given by

$$\frac{v_j^{n+1} - v_j^n}{\Delta} + \rho v_j^{n+1} = u(c_j^n) + \frac{v_{j+1}^{n+1} - v_j^{n+1}}{\Delta b}(s_j^{F,n})^+ + \frac{v_j^{n+1} - v_{j-1}^{n+1}}{\Delta b}(s_j^{B,n})^- \tag{15}$$

where the value functions on the right hand side are of step $n+1$. 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.

4.1.3. Stochastic Settings

For the heterogeneous agent model (3) and (4) featuring a Poisson process, we add another 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 HJB Equation (6) that arises for this model with a two-state Poisson process by

$$\begin{aligned}\frac{v_{j,k}^{n+1} - v_{j,k}^{n}}{\Delta} + \rho v_{j,k}^{n+1} = u(c_{j,k}^{n}) &+ \frac{v_{j+1,k}^{n+1} - v_{j,k}^{n+1}}{\Delta b}(s_{j,k}^{F,n})^{+} + \frac{v_{j,k}^{n+1} - v_{j-1,k}^{n+1}}{\Delta b}(s_{j,k}^{B,n})^{-} \\ &+ \lambda_k (v_{j,-k}^{n+1} - v_{j,k}^{n+1}),\end{aligned} \tag{16}$$

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

For heterogeneous agent models with a diffusion process instead of a Poisson one, e.g., (A1) and (A2), we add another grid dimension for the productivity state z. We discretize the HJB equation (A3) that arises for the above model by

$$\begin{aligned}\frac{v_{j,k}^{n+1} - v_{j,k}^{n}}{\Delta} + \rho v_{j,k}^{n+1} = u(c_{j,k}^{n}) &+ \frac{v_{j+1,k}^{n+1} - v_{j,k}^{n+1}}{\Delta b}(s_{j,k}^{F,n})^{+} + \frac{v_{j,k}^{n+1} - v_{j-1,k}^{n+1}}{\Delta b}(s_{j,k}^{B,n})^{-} \\ &+ \frac{v_{j,k+1}^{n+1} - v_{j,k}^{n+1}}{\Delta z}(\mu_k)^{+} + \frac{v_{j,k}^{n+1} - v_{j,k-1}^{n+1}}{\Delta z}(\mu_k)^{-} \\ &+ \frac{\sigma_k^2}{2}\frac{v_{j,k+1}^{n+1} - 2v_{j,k}^{n+1} + v_{j,k-1}^{n+1}}{(\Delta z)^2}.\end{aligned} \tag{17}$$

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.

4.1.4. Matrix Notation

After linearizing and discretizing the HJB equation, we can easily formulate the resulting equations as a system for the value function. 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 setup the respective matrices to formulate the discretized HJB equation in case of a diffusion process to compute an iterate of the value function $\mathbf{v}^n$ following the implicit scheme (17) by

$$\frac{1}{\Delta}(\mathbf{v}^{n+1} - \mathbf{v}^{n}) + \rho \mathbf{v}^{n+1} = \mathbf{u}^{n} + (\mathbf{A}^{n} + \boldsymbol{\Lambda})\mathbf{v}^{n+1} \tag{18}$$

where $\mathbf{A}^n \in \mathbb{R}^{m \times m}$ 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.

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 (17). 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 (17) using matrix notation (18), we have with $\mathbf{D}$ for a finite difference operator and $\mathbf{s}^F, \mathbf{s}^B$ for the vector with entries (14)

$$\mathbf{A}^n = (\mathbf{s}^{F,n})^+ \star \mathbf{D}_b^F + (\mathbf{s}^{B,n})^- \star \mathbf{D}_b^B \tag{19}$$

and

$$\mathbf{\Lambda} = (\mu)^+ \star \mathbf{D}_z^F + (\mu)^- \star \mathbf{D}_z^B + \frac{1}{2}\sigma^2 \star \mathbf{D}_{zz}, \tag{20}$$

where the standard operations should be understood entry-wise. The finite difference matrices $\mathbf{D}$, with sub- and superscripts indicating the taken derivative, are built using the standard full grid finite difference stencils; see any standard textbook such as [38] for a description. Note that, e.g., $((\mathbf{s}^{F,n})^+ \star \mathbf{D}_b^F)\mathbf{v}^n$ is nothing else but $(\mathbf{D}_b^F\mathbf{v}^n) \odot ((\mathbf{s}^{F,n})^+)$ where $\odot$ denotes the entry-wise vector multiplication.

By simple algebra, we obtain

$$\underbrace{((\frac{1}{\Delta} + \rho)\mathbf{I} - \mathbf{A}^n - \mathbf{\Lambda})}_{\mathbf{B}} \mathbf{v}^{n+1} = \underbrace{\mathbf{u}^n + \frac{1}{\Delta}\mathbf{v}^n}_{\mathbf{r}^n = \mathbf{r}(\mathbf{v}^n)} \tag{21}$$

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

$$\mathbf{B}\mathbf{v}^{n+1} = \mathbf{r}^n \tag{22}$$

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

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 approximation of the utility function (5), we already have nodal values and thus $\mathbf{u}$ now describes the vector containing nodal coefficients for the utility function. To solve the linear system with consistent basis representations, we use the hierarchical to nodal basis transformations $\mathbf{E}$ and obtain

$$\left(\left(\frac{1}{\Delta} + \rho\right)\mathbf{E} - \mathbf{\Lambda} - \mathbf{A}^n\right)\mathbf{v}^{n+1} = \mathbf{u}^n + \frac{1}{\Delta}\mathbf{E}\mathbf{v}^n \tag{23}$$

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

$$\left(\left(\frac{1}{\Delta} + \rho\right)\mathbf{E} - \mathbf{E}\mathbf{\Lambda} - \mathbf{A}^n\right)\mathbf{v}^{n+1} = \mathbf{u}^n + \frac{1}{\Delta}\mathbf{E}\mathbf{v}^n \tag{24}$$

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.

The overall procedure is given in Algorithm 1; see [34,35] for more details on the algorithmic details for the adaptive sparse grids approach, refinement, and coarsening, etc.

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

```
    Data: model parameters, sparse grid parameters
    Result: solution $\mathbf{v}$ of HJB equation
 1: Initialization:
 2: generate sparse grid
 3: compute hierarchical to nodal basis transformation matrix $\mathbf{E}$    ▷ see after (10)
 4: generate finite difference operators    ▷ see (3) in Section 3.1
 5: set up matrix $\mathbf{\Lambda}$ that models stochastic process    ▷ see (20) in Section 4.1.4
 6: compute initial guess in hierarchical representation $\mathbf{v}^0$    ▷ see e.g., (25) for model (3) and (4)
 7: Iterative part:
 8: for $n = 0, 1, \ldots$ do
 9:     refine sparse grid and initialize the new sparse grid
10:     compute forward and backward differences of $\mathbf{v}^n$    ▷ use FD operators
11:     compute optimal controls    ▷ consumption, deposits for model (3) and (4),
12:                                 ▷ use forward and backward differences of $\mathbf{v}^n$
13:     build drift matrix $\mathbf{A}^n$    ▷ (19) in Section 4.1.4, follow upwind scheme and use FD operators
14:     solve (23) respectively (24) for $\mathbf{v}^{n+1}$    ▷ see Section 4.1.2, linearized HJB equation
15:     if $\mathbf{v}^{n+1}$ is close to $\mathbf{v}^n$ according to stopping criteria then
16:         $\mathbf{v} \leftarrow \mathbf{v}^{n+1}$
17:         STOP
18:     end if
19:     coarsen sparse grid
20: end for
```

4.2. Further Aspects

Considering the employed two-asset model problem (3), (4), notice that the optimal deposits d are computed in (7) with the derivative with respect to both b and to a. Thus, to obtain 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 for splitting the problem in K subproblems that one can solve simultaneously using parallelization. See [33] for the full technical details.

As an initial guess for the value function we use

$$v_0 = \left(\frac{wz + r^b(b)b + r^a a}{1-\gamma} \right)^{1-\gamma} / \rho, \tag{25}$$

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

5. 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

$$
\begin{aligned}
e^f_{2,r}(x_1,\dots,x_M) &= \left(\frac{1}{M}\sum_{m=1}^{M}\left|\frac{f_{\text{ref}}(x_m)-f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m)-\min_m f_{\text{ref}}(x_m)}\right|^2\right)^{\frac{1}{2}},\\
e^f_{\infty,r}(x_1,\dots,x_M) &= \max_m\left|\frac{f_{\text{ref}}(x_m)-f_{\text{SG}}(x_m)}{\max_m f_{\text{ref}}(x_m)-\min_m f_{\text{ref}}(x_m)}\right|.
\end{aligned}
\tag{26}
$$

Here, the x_m are either the points of the full grid reference solution or random points for error measurement, where we use $M = 5000$.

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 better results in our experiments.

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

5.1. Two-Dimensional Model

Let us begin with the $2d$ model (3) and (4) presented in Section 2.2 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 6, 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 multivariate functions, where the different outputs are stacked into one vector.

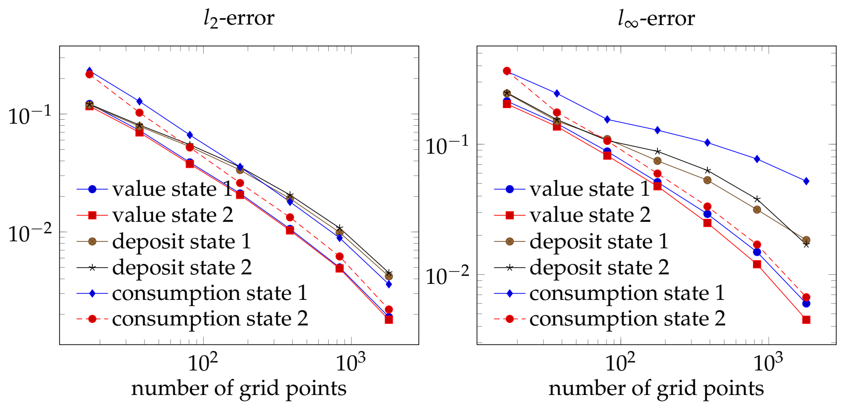


Figure 6. Accuracy of different sparse grid levels: e_2 and e_∞-errors for the value function; deposit policy and consumption policy for states 1 and 2.

5.1.1. Adaptive Sparse Grids

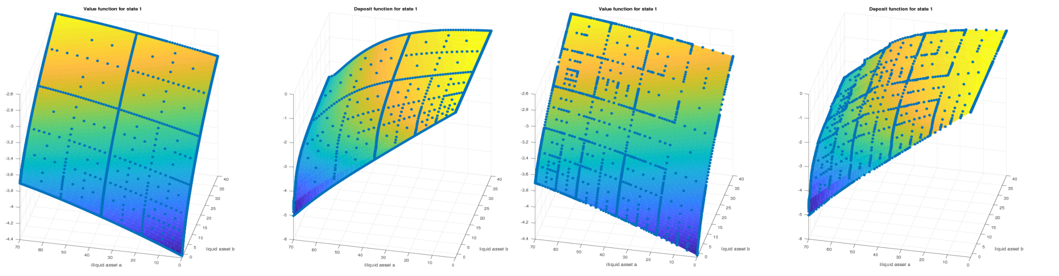
We performed experiments with several types of adaptivity criteria since there is no theoretical rule determining which criterion is optimal. The solution of the HJB equation—the value function—often does not give useful insight. Thus, we are also interested in a good approximation of the policy function describing the controls' consumption c_t and deposit d_t. These were computed by the approximated derivatives of the value function as given by (7). 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 used the hierarchical coefficients of the sparse grid value function approximation. Similarly, we used the hierarchical coefficients of the policy function approximation for the policy function adaptivity. Hence, for the two-dimensional model (3) and (4) described in Section 2.2, we can use a value function adaptivity, a consumption function adaptivity, or a deposit function adaptivity.

Moreover, we used 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 have 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.

First, in the two-dimensional case we could visualize the different grids obtained by value function adaptivity versus policy function. Here, we focused on the deposit function since we observed the biggest errors for it, but the observations could be transferred to policy functions in general.

In Figure 7, the approximation of the value function for state 1 and the sparse grid are shown. Note that we indicate the grid points by their respective function values as bullet points. One can see that, using value function adaptivity, 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. Using deposit function adaptivity, the resulting sparse grid looks completely different, now there are more grid points in the area where the deposit function is steep.



(**a**) Adaptivity based on the value function. (**b**) Adaptivity based on the deposit function.

Figure 7. Scatter and surface plot of both the value (**left**) and the deposit (**right**) functions for state 1 for the adaptive sparse grid using different refinement criteria in (**a**,**b**).

5.1.2. Accuracy for Adaptive Sparse Grids

For more insight into the approximation quality of different types of adaptivity, we aim to compare the accuracies resulting from different types of adaptivity. We use 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 8 the results for different refinement thresholds, where we limit the maximum number of adaption steps so that we do 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 outperformed the value function adaptivity for the deposit policy accuracy, in the beginning. Concerning the l_2-error, the other adaptivity criteria are not better, if at all, than a regular sparse grid. 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.

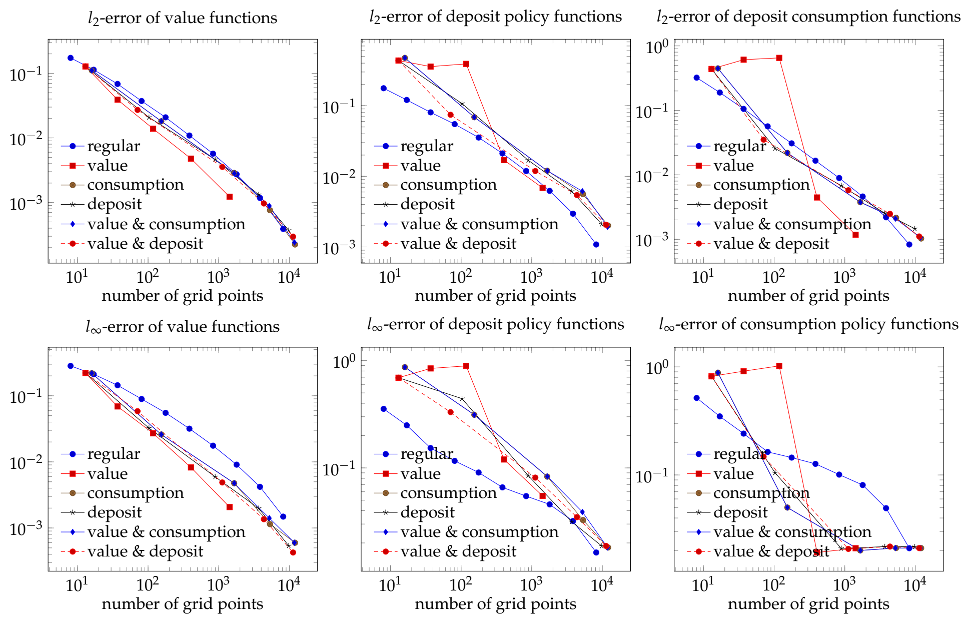


Figure 8. Accuracy plots for the two-dimensional problem using the regular sparse grid and different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (marked on the respective lines) after using at most ten adaption steps.

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.

5.2. Four-Dimensional Model

Let us present our results for the 4d model (A1) and (A2) explained in Appendix A.1. We computed the accuracy for different sparse grid levels and adaptivity versions by using a reference solution that we computed on a higher sparse grid level $l = 8$. Instead of computing the error on the grid of the reference solution, we computed the error by interpolating on uniformly distributed points for both the reference and the analyzed solutions.

We present in Figure 9 the results for different refinement thresholds, where we limit the maximum number of adaption steps so that we do 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.

Note that all adaptivity versions work for the policy function approximation, where the maximum error is still relatively large. However, for the value function approximation, one can see that value function adaptivity worked better than the other adaptation criteria. Overall, the adaptivity gave better results in comparison to a regular grid. We observed a stagnation in particular for the policy function; we assumed this was due to the limitation of the refinement level to the maximum level of the reference sparse grid. For a more detailed comparison of the convergence behavior of the different adaptivity approaches, other error estimations rather than comparing against a regular sparse grid of high level are needed.

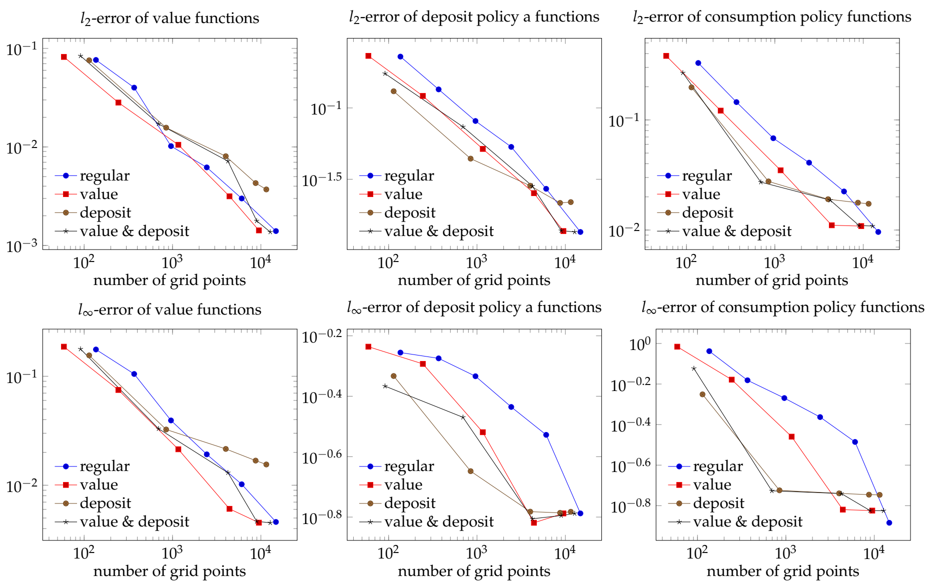


Figure 9. Accuracy plots for the four-dimensional problem using regular sparse grid and different adaptivity versions starting at level $l = 2$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (marked on the respective lines) after using at most five adaption steps.

5.3. Six-Dimensional Model

Finally we give results in Figure 10 for the 6d model (A4) and (A5) explained in Appendix A.2. We again compute the accuracy for different sparse grid levels and adaptivity versions by using a reference solution that we computed on a higher sparse grid level $l = 6$. Again, we did 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 interpolated 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 worked 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 yielded good results. The advantage of the adaptive approaches in comparison to the regular sparse grid further increases. As before for the four-dimensional model, a more detailed comparison of the convergence behavior would need other approaches for the estimation of the errors.

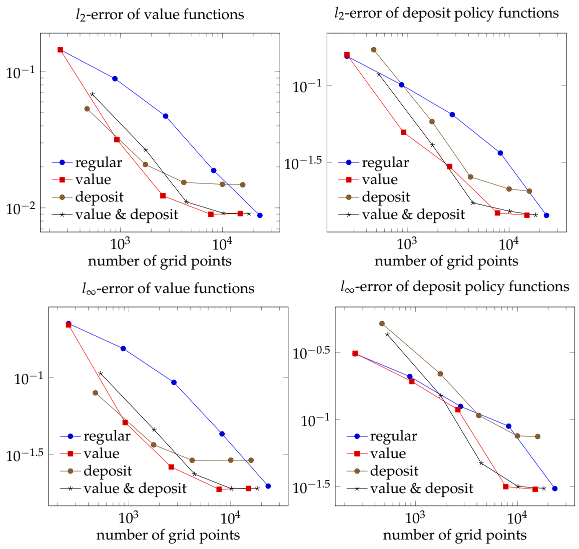


Figure 10. Accuracy plots for the six-dimensional problem for regular sparse grid and different adaptivity versions starting at level $l = 1$ with refinement threshold $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ (marked on the respective lines) after using at most four adaption steps.

6. Conclusions and Outlook

In this work we introduced a finite difference approach using interpolations on an adaptive sparse grid for solving economic models following the numerical scheme of [3]. We analyzed the accuracy for our approach for economic models ranging from dimension $d = 2$ to dimension $d = 6$ and achieved 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 correct solution. Second, the experiments with different types of adaptivity indicate that value function adaptivity performs well for approximating the value function. To achieve a good approximation of the policy functions, it can sometimes be better to use a criterion suited to this function or combined criteria. Note though that for policy functions it strongly depends on the choice of parameters, such as the starting level of the sparse grid or the starting refinement threshold, to observe how well it performs. Nevertheless, we recommend to use value function adaptivity since it leads to the best results in most cases.

For a convergence result for sparse grids finite difference schemes for solving the HJB equation according to [11], 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 therefore hardly possible, since it depends on model parameters if the arising interpolations on the value functions are monotone for the used adaptive sparse grid. Thus, it depends on the model parameters if our approach works correctly without specific approaches to overcome non-monotonicity. Nevertheless, the numerical results indicate that a non-monotone discretization approach can achieve convergence, which highlights the need for additional theoretical investigations.

While adaptive sparse grids allow the discretization of higher dimensional problems, we note that the computational costs for solving the arising linear equation system do increase with the number of dimensions. For the purpose of this study we used a standard ILUC preconditioned BiCGSTAB iterative solver, but we expect that there are preconditioners available that are more suitable for this problem class, which would be one aspect of future research. Additionally, parallelization can further improve the runtime.

Moreover, the computational performance of the different sparse grid approaches for solving Hamilton–Jacobi–Bellman equations could be compared. Besides the one presented in this work, this would involve, among others, parallel adaptive sparse grids from [23], finite difference operators on sparse grids [26], and sparse grid semi-Lagrangian approaches [34,35]. Furthermore, there are investigations on using deep neural networks for solving dynamic economic models, e.g., [39] casts these into nonlinear regression equations. An investigation on which numerical approach is better suited for which economic scenario is warranted.

Author Contributions: Conceptualization, J.G. and S.R.; methodology, J.G. and S.R.; software, S.R.; validation, J.G. and S.R.; writing—original draft preparation, J.G. and S.R.; writing—review and editing, J.G.; visualization, J.G. and S.R.; supervision, J.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: We thank SeHyoun Ahn and Benjamin Moll for their fruitful discussions, and SeHyoun Ahn for help with the Matlab implementation, which is based on his code written for [9].

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix 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 a 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 4 for explanations of the numerical approach.

We are now interested in the following maximization problem,

$$\max_{\{c_t, d_t^a, d_t^h\}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t, h_t) dt \tag{A1}$$

subject to

$$\begin{aligned}
&\dot{b}_t = wz_t + r^b(b_t)b_t - d_t^a - \chi(d_t^a, a_t) - d_h^t - \chi(d_t^h, h_t) - c_t \\
&\dot{a}_t = r^a a_t + d_t^a \\
&\dot{h}_t = d_t^h \\
&\dot{z}_t = \mu(z_t)dt + \sigma(z_t)dW_t \\
&b_t \geq b, \; a_t \geq 0, \; h_t \geq 0
\end{aligned} \tag{A2}$$

The diffusion is reflected on the boundaries in dimension z, i.e.,

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

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

$$\begin{aligned}
\rho v(b,a,h,z) = \max_{c,d^a,d^h} &\, u(c,h) \\
&+ v_b(b,a,h,z)(wz + r^b(b)b - d^a - \chi(d^a,a) - d^h - \chi(d^h,h) - c) \\
&+ v_a(b,a,h,z)(r^a + d^a) \\
&+ v_h(b,a,h,z)(d^h) \\
&+ \partial_z v(b,a,h,z)\mu(z) + \frac{1}{2}\partial_{zz} v(b,a,h,z)\sigma^2(z).
\end{aligned} \tag{A3}$$

Appendix 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 2.2. 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,

$$\max_{\{c_t,d_t\}_{t\geq 0}} \mathbb{E}_0 \int_0^\infty e^{-\rho t} u(c_t) dt \tag{A4}$$

subject to

$$\begin{aligned}
\dot{b}_t &= \frac{(z_t^1 + z_t^2 + z_t^3 + z_t^4)}{4} w + r^b(b_t) b_t - d_t - \chi(d_t, a_t) - c_t \\
\dot{a}_t &= r^a a_t + d_t \\
\dot{z}_t^1 &= \mu(z_t^1) dt + \sigma(z_t^1) dW_t \\
\dot{z}_t^2 &= \mu(z_t^2) dt + \sigma(z_t^2) dW_t \\
\dot{z}_t^3 &= \mu(z_t^3) dt + \sigma(z_t^3) dW_t \\
\dot{z}_t^4 &= \mu(z_t^4) dt + \sigma(z_t^4) dW_t \\
b_t &\geq b,\ a_t \geq 0
\end{aligned} \tag{A5}$$

Here, the diffusion processes $z_t^i,\ i = 1, \ldots, 4$ can be interpreted as different types of skills or luck that evolve differently over time. $W(t) \sim \mathcal{N}(0, t)$ is normally distributed, where $\mu(\cdot)$ is the *drift* and $\sigma(\cdot)$ is the *diffusion* of z. We use the standard CRRA-utility function again and have reflecting boundary conditions again.

We obtain the HJB equation

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

Appendix A.3. Parameters

Here, we give the model and algorithm parameters that we used in our numerical studies.

Appendix A.3.1. Parameters for the Two-Dimensional Model

Table A1. Model parameters for the 2d model (3) and (4).

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

Table A1. *Cont.*

Parameter	Default Value	Description
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 A2. Algorithm 1 parameters for the 2d model.

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 A3. Lower and upper bounds for the respective states in the 2d model. The lower bounds are 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

Appendix A.3.2. Parameters for the Four-Dimensional Model

Table A4. Model parameters for the 4d model (A1) and (A2).

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 A5. Algorithm 1 parameters for the 4d model.

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 A6. 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.

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

Appendix A.3.3. Parameters for the Six-Dimensional Model

Table A7. Model parameters for the 6*d* model (A4) and (A5).

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 A8. Algorithm 1 parameters for the 6*d* model.

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 A9. 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.

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

References

1. Bellman, R.E. *Adaptive Control Processes*; Princeton University Press: Princeton, NJ, USA, 1961.
2. Candler, G.V. Finite-Difference Methods for Dynamic Programming Problems. In *Computational Methods for the Study of Dynamic Economies*; Cambridge University Press: Cambridge, UK, 1999.
3. Achdou, Y.; Han, J.; Lasry, J.M.; Lions, P.L.; Moll, B. Income and wealth distribution in Macroeconomics: A continuous-time approach. *Rev. Econ. Stud.* **2022**, *89*, 45–86. [CrossRef]

4. Kaplan, G.; Moll, B.; Violante, G.L. Monetary policy according to HANK. *Am. Econ. Rev.* **2019**, *108*, 697–743. [CrossRef]
5. Schiekofer, T. Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung Elliptischer und Parabolischer Partieller Differentialgleichungen. Ph.D. Thesis, Institut für Angewandte Mathematik, Universität Bonn, Bonn, Germany, 1998.
6. Griebel, M. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. *Computing* **1998**, *61*, 151–179. [CrossRef]
7. Griebel, M.; Schiekofer, T. An adaptive sparse grid Navier–Stokes solver in 3D based on the finite difference method. In Proceedings of the ENUMATH97, Heidelberg, Germany, 28 September–3 October 1999.
8. Zumbusch, G.W. A Sparse Grid PDE Solver; Discretization, Adaptivity, Software Design and Parallelization. *Adv. Softw. Tools Sci. Comput.* **2000**, *10*, 133–177.
9. Ahn, S. Sparse Grid Methods for Economic Models. Unpublished Manuscript, Code. Available online: https://sehyoun.com/EXAMPLE_Aiyagari_HJB_Adaptive_Sparse_Grid_web.html (accessed on 12 November 2024).
10. Koster, F. Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern. Ph.D. Thesis, Institut für Angewandte Mathematik, Universität Bonn, Bonn, Germany, 2002.
11. Barles, G.; Souganidis, P.E. Convergence of approximation schemes for fully nonlinear second order equations. *Asymptot. Anal.* **1991**, *4*, 271–283. [CrossRef]
12. Moll, B. Lecture Notes of Income and Wealth Distribution in Macroeconomics. 2016. Princeton. Available online: https://benjaminmoll.com/lectures/ (accessed on 12 November 2024).
13. Kushner, H.; Dupuis, P.G. *Numerical Methods for Stochastic Control Problems in Continuous Time*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013; Volume 24.
14. Bellman, R.E. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.
15. Falcone, M.; Ferretti, R. *Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations*; SIAM: Philadelphia, PA, USA, 2013.
16. Ahn, S.H.; Kaplan, G.; Moll, B.; Winberry, T.; Wolf, C. When inequality matters for macro and macro matters for inequality. *NBER Macroecon. Annu.* **2018**, *32*, 1–75. [CrossRef]
17. Schmedders, K.; Judd, K. *Handbook of Computational Economics*; Number Bd. 3 in Handbook of Computational Economics; Elsevier Science: Amsterdam, The Netherlands, 2013.
18. Judd, K.; Maliar, L.; Maliar, S. Numerically Stable and Accurate Stochastic Simulation Methods for Solving Dynamic Models. *Quant. Econ.* **2011**, *2*, 173–210. [CrossRef]
19. Krueger, D.; Kubler, F. Computing equilibrium in OLG models with stochastic production. *J. Econ. Dyn. Control* **2004**, *28*, 1411–1436. [CrossRef]
20. Jin, H.; Judd, K.L. Perturbation Methods for General Dynamic Stochastic Models. Technical Report, Mimeo April. 2002. Available online: https://web.stanford.edu/~judd/papers/PerturbationMethodRatEx.pdf (accessed on 2 January 2025).
21. Maliar, L.; Maliar, S.; Villemot, S. Taking perturbation to the accuracy frontier: A hybrid of local and global solutions. *Comput. Econ.* **2013**, *42*, 307–325. [CrossRef]
22. Judd, K.L.; Maliar, L.; Maliar, S.; Valero, R. Smolyak method for solving dynamic economic models. *J. Econ. Dyn. Control* **2014**, *44*, 92–123. [CrossRef]
23. Brumm, J.; Scheidegger, S. Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models. *Econometrica* **2017**, *85*, 1575–1612. [CrossRef]
24. Schober, P. Solving Dynamic Portfolio Choice Models in Discrete Time Using Spatially Adaptive Sparse Grids. In *Proceedings of the Sparse Grids and Applications–Miami 2016*; Garcke, J., Pflüger, D., Webster, C.G., Zhang, G., Eds.; Springer: Cham, Switzerland, 2018; pp. 135–173.
25. Brumm, J.; Krause, C.; Schaab, A.; Scheidegger, S. Sparse Grids for Dynamic Economic Models. In *Oxford Research Encyclopedia of Economics and Finance*; Oxford University Press: Oxford, UK, 2022. [CrossRef]
26. Schaab, A.; Zhang, A. Dynamic Programming in Continuous Time with Adaptive Sparse Grids. 2022. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4125702 (accessed on 12 November 2023).
27. Bertsekas, D.P.; Shreve, S. *Stochastic Optimal Control: The Discrete-Time Case*; Athena Scientific: Nashua, NH, USA, 1996.
28. Zenger, C. Sparse Grids. In *Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, Germany, 18–21 January 1990*; Hackbusch, W., Ed.; Vieweg-Verlag: Wiesbaden, Germany, 1991; Volume 31, pp. 241–251.
29. Smolyak, S.A. Quadrature and interpolation formulas for tensor products of certain class of functions. *Dokl. Akad. Nauk SSSR* **1963**, *148*, 1042–1053; Transl. Soviet Math. Dokl. **1963**, *4*, 240–243.
30. Bungatrz, H.J.; Griebel, M. Sparse grids. *Acta Numer.* **2004**, *13*, 147–269.
31. Garcke, J. Sparse Grids in a Nutshell. In *Sparse Grids and Applications*; Garcke, J., Griebel, M., Eds.; Lecture Notes in Computational Science and Engineering; Springer: Berlin/Heidelberg, Germany, 2013; Volume 88, pp. 57–80. [CrossRef]
32. Pflüger, D. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*; Verlag Dr. Hut: München, Germany, 2010.

33. Ruttscheidt, S. Adaptive Sparse Grids for Solving Continuous Time Heterogeneous Agent Models. Master's Thesis, Institut für Numerische Simulation, Universität Bonn, Bonn, Germany, 2018.
34. Bokanowski, O.; Garcke, J.; Griebel, M.; Klompmaker, I. An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations. *J. Sci. Comput.* **2013**, *55*, 575–605. [CrossRef]
35. Garcke, J.; Kröner, A. Suboptimal feedback control of PDEs by solving HJB equations on adaptive sparse grids. *J. Sci. Comput.* **2017**, *70*, 1–28. [CrossRef]
36. Noordmans, J.; Hemker, P.W. Application of an Adaptive Sparse Grid Technique to a Model Singular Perturbation Problem. *Computing* **2000**, *65*, 357–378.
37. Achdou, Y.; Barles, G.; Ishii, H.; Litvinov, G.L. *Hamilton-Jacobi Equations: Approximations, Numerical Analysis and Applications*; Springer: Berlin/Heidelberg, Germany, 2013.
38. Langtangen, H.P. *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013; Volume 2.
39. Maliar, L.; Maliar, S.; Winant, P. Deep learning for solving dynamic economic models. *J. Monet. Econ.* **2021**, *122*, 76–101. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.