

HIGH-DIMENSIONAL DYNAMIC STOCHASTIC MODEL REPRESENTATION *

ARYAN EFTEKHARI [†] AND SIMON SCHEIDEGGER [‡]

Abstract. We propose a scalable method for computing global solutions of nonlinear, high-dimensional dynamic stochastic economic models. First, within a time iteration framework, we approximate economic policy functions using an adaptive, high-dimensional model representation scheme, combined with adaptive sparse grids to address the ubiquitous challenge of the curse of dimensionality. Moreover, the adaptivity within the individual component functions increases sparsity since grid points are added only where they are most needed, that is, in regions with steep gradients or at nondifferentiabilities. Second, we introduce a performant vectorization scheme for the interpolation compute kernel. Third, the algorithm is hybrid parallelized, leveraging both distributed- and shared-memory architectures. We observe significant speedups over the state-of-the-art techniques, and almost ideal strong scaling up to at least 1,000 compute nodes of a Cray XC50 system at the Swiss National Supercomputing Center. Finally, to demonstrate our method’s broad applicability, we compute global solutions to two variates of a high-dimensional international real business cycle model up to 300 continuous state variables. In addition, we highlight a complementary advantage of the framework, which allows for a priori analysis of the model complexity.

Key words. High-Dimensional Model Representation, Sparse Grids, High-Performance Computing, International Real Business Cycles

AMS subject classifications. 41A63, 41A58, 68W25, 68W10, 91B70

1. Introduction. Motivated by empirical observations, features such as heterogeneity, interconnectedness, and uncertainty have become vital ingredients for capturing the salient features in contemporary dynamic economic models. In macroeconomics for instance, heterogeneity between types of agents, such as hand-to-mouth and non-hand-to-mouth consumers (see, e.g., [23]), financial frictions such as collateral constraints (see, e.g., [29]), or distributional channels (see, e.g., [27]) have become widely recognized as essential components for modern macroeconomic models. The need for heterogeneity has recently become even more evident during the current *COVID-19* pandemic, where unprecedented policy actions have to be taken to mitigate this once-in-a-century event (see, e.g., [19]).

To address today’s key questions in economics quantitatively, one quickly ends up with an intricate formal structure that relies on considering so-called *recursive equilibria* [50, 33]. In such equilibria, a potentially high-dimensional *state variable* $\mathbf{x} \in X \subset \mathbb{R}^d$ represents the state of the economy, d is the dimensionality of the state space, and a time-invariant *optimal policy function* $p : X \rightarrow Y \subset \mathbb{R}^m$, the desired unknown, captures the model dynamics and can be characterized as the solution to a functional equation that reads [14]:

$$(1.1) \quad \mathcal{H}(p) = \mathbf{0}.$$

*Submitted to the editors on January 6, 2021.

Funding: This work was generously supported by grants from the Swiss National Supercomputing Centre (CSCS) under project IDs s885 and s995, the Swiss Platform for Advanced Scientific Computing (PASC) under project ID “Computing equilibria in heterogeneous agent macro models on contemporary HPC platforms”, and the Swiss National Science Foundation under project IDs “Can Economic Policy Mitigate Climate-Change?” and “New methods for asset pricing with frictions”.

[†]Institute of Computational Science, Università della Svizzera italiana, Lugano, Switzerland, and Department of Economics, University of Lausanne, Switzerland (aryan.eftekhari@usi.ch).

[‡]Department of Economics, University of Lausanne, Switzerland, and Enterprise for Society (E4S) (simon.scheidegger@unil.ch).

This abstract description nests various characterizations of recursive equilibria, and in particular, the widespread case where the operator \mathcal{H} captures discrete-time first-order equilibrium conditions, the focal point of this paper.

A standard method for solving such *dynamic stochastic economic models* is the so-called *time iteration* algorithm [9], which computes the recursive equilibrium of a dynamic economic model by guessing a policy function and iteratively updating it using the first-order equilibrium conditions of the model. However, solving for global solutions¹ to models with substantial heterogeneity and highly nonlinear policy functions is very costly for two key reasons. First, no matter which model characterization is used, the *curse of dimensionality* [2] imposes a roadblock as soon as X is of a higher dimension (say $d > 3$), and a global solution has to be computed, where the equilibrium conditions need to be satisfied throughout the entire computational domain. A grid-based solution technique that relies on a *naive* tensor-product construction will require $\mathcal{O}(M^d)$ points when M points are needed in each dimension. This exponential growth makes tensor-product grids infeasible as soon as M and d reach moderate levels. Second, at each grid point, a system of nonlinear equations must be solved. When solving the system of equations at a given grid point, one needs to frequently interpolate from the function computed in the previous iteration step. These interpolation operations can account for up to 99% of the overall compute time for solving the system of equations [47]. These two impediments make it difficult to achieve an acceptable time-to-solution which can quickly reach the order of days on modern supercomputing facilities. As a result, present methods frequently fall well short of including as much heterogeneity as a reasonable modeling choice would imply.²

To deal with the ever-increasing complexity of state-of-the-art dynamic stochastic economic models, we propose in this work a generic, scalable, and flexible computational framework which can efficiently address the previously noted bottlenecks. Building on [47, 6, 5] and [12], we specifically contribute (i) an adaptive high-dimensional model representation scheme that is coupled with an adaptive sparse grid algorithm applicable for recursively formulated economic models that significantly reduces the number of grid points in the approximation and the time needed for each function evaluation, (ii) adaptivity criteria which can be used as an on-the-fly analysis tool elucidating the complexity of the model under consideration, (iii) a vectorized implementation for performant interpolation, and (iv) a hybrid parallelized time iteration solution framework fit for virtually any dynamic stochastic economic model. Finally (v), we deploy our solution framework at the Swiss National Supercomputing Center (CSCS) to solve highly nonlinear dynamic stochastic economic models of up to 300 dimensions globally.

The grid point reduction is achieved by combining adaptive sparse grids (adaptive SGs; see, e.g., [7, 42, 34]) with a dimensional decomposition (DD) framework that is

¹A *global solution* adheres to the model equilibrium conditions throughout the entire state space—that is, the computational domain, whereas a *local solution* is only concerned with the local approximation around a steady state.

²For example, [26] examined the welfare implications of social security reforms using a model in which one period amounted to six years rather than one, thus lowering the number of adult cohorts and therefore the problem’s dimensionality by a factor of six. Similarly, international real business cycle models often incorporate only a small number of countries. For instance, [3] studied cross-country risk-sharing at the business cycle frequency using a two-country model with one focus country and the rest of the world. By reducing the problem’s dimensionality in this way, valuable qualitative insights can be gained. However, in order to obtain reliable quantitative results or simply to assess the robustness of qualitative findings, it is frequently necessary to consider problems of larger dimensions.

based on high-dimensional model representation (see, e.g., [32, 35, 53]). Finally, the time-to-solution is substantially accelerated by using a hybrid parallelization scheme (i.e., using both distributed- and shared-memory hardware) combined with a novel vectorization approach for fast interpolation on the policy functions.

SGs can alleviate the curse of dimensionality to some extent, allowing one to tackle models that incorporate rich economic settings, including international real business cycle (IRBC) models of up to 50 countries, that is, 100 dimensions [6]. Furthermore, adaptive SGs can resolve steep gradients or nondifferentiabilities efficiently, making them useful in the context of solving a broad range of mid-scale economic models (say $d < 20$) with non-smooth policy functions that arise, for example, in the presence of collateral constraints on borrowing (see [4] for a review on the use of adaptive SGs in economics and finance).

However, the limitations of adaptive SGs become evident when one considers highly nonlinear economic models of much more heterogeneity, for example, IRBC models that consist of dozens of countries that face irreversible investment constraints. In such situations, adaptive SGs are no longer an applicable solution method, as the number of grid points increases substantially with the approximation quality or, equivalently, with the resolution of the grid. Furthermore, in high-dimensional SGs, access times of the data structures become computationally expensive (see, e.g., [40]).

The noted issues can be surpassed by combining DD (see, e.g., [31, 44, 17]) with adaptive SGs, which we refer to as DDSG. The core idea of DD is to approximate a function by decomposing it into a series of lower-dimensional functions. This decomposition gives a way to represent, for example, in the simplest case, a 300-dimensional function as a summation 300 one-dimensional functions. We will focus on one variate of DD referred to in the literature as High-Dimensional Model Representation (HDMR). The HDMR technique has been applied to multiple fields including chemistry (see, e.g., [53]), physics (see, e.g., [35]), and machine-learning (see, e.g., [39]). With this said, and to the best of our knowledge, HDMR so far seems to have gone unrecognized in the field of economics until now.

The remainder of this paper is organized as follows. In Section 2, we provide a very brief review of the related literature on global solution techniques for high-dimensional dynamic economic models. In Section 3, we describe the abstract structure of the models we aim to solve with our proposed method and specify a conceptually simple yet computationally demanding economic test case—the IRBC model. The latter has become the de-facto workhorse for studying methods for solving high-dimensional economic models, as its dimensionality can be scaled up in a straightforward and meaningful way, as it just depends linearly on the number of countries considered. In Section 4, we detail the proposed DDSG method. Next, in Section 5, we embed DDSG in a time iteration algorithm and discuss the hybrid parallelization scheme of the complete method. In Section 6, we support the performance claims and applicability of the proposed solution framework by providing a series of unit tests and global solution results in two different configurations of high-dimensional IRBC models. Finally, in Section 7 we conclude.

2. A brief literature review on global solution methods. Over the past two decades, there have been significant advancements in the development of algorithms and numerical tools to compute global solutions for high-dimensional dynamic stochastic economic models (see [36, 14] for recent reviews). To overcome the inherent curse of dimensionality, the computational economics community has pursued two main strands of research: i) SG-based solution algorithms and ii) grid-free meth-

ods. SG methods [8] are a mathematically well-studied, systematic way to tackle the numerical difficulties that arise in dynamic economic models due to the high-dimensional state spaces. However, they typically fail in real applications if the dimensionality of a highly nonlinear model exceeds about 20 (see [4] for a recent review). However, such problem sizes are often required from a theoretical point of view, for instance, in annually calibrated multi-country overlapping generation models with borrowing constraints—a 240-dimensional problem in its minimal formulation. In contrast, DDSG can, as we show below, handle problem sizes of at least 300 continuous state variables. Grid-free approaches have been proposed, for example, by [11] and have lately become more powerful by leveraging the rapid developments in machine learning. In [45, 25] and [46, 28], the authors combine Gaussian processes with reinforcement learning and active subspaces, respectively. However, the combination of these methods typically cannot deal in a straightforward manner with frictions such as irreversible investments and the related nonlinearities (as presented in the IRBC model in Section 3.1.2), whereas DDSG can. Other research in computational economics has recently applied deep neural networks to compute global solutions to high-dimensional dynamic stochastic models (see, e.g., [1, 37, 13]). However, while these methods could be considered an alternative to the work presented here, they nowadays still suffer from several drawbacks that limit their general applicability. Their convergence properties, for example, with respect to the network architecture, are still poorly understood, thus often requiring a substantial amount of hyper-parameter tuning for a successful model solution. In contrast, DDSGs provide a transparent way to handle high-dimensional models. Finally, nesting the DD approach with the time iteration algorithm is not restricted to SGs. Thus, alternative solution methods for low-dimensional economic models could also be scaled up in a relatively straightforward way if embedded in DD, in turn providing a simple means to extend the boundaries for research.

3. Large-scale dynamic stochastic economic models. This section outlines the types of models and the recursive solution techniques that we use below to demonstrate the versatility, accuracy, and computational scalability of the method introduced in this paper. To do so, we proceed in two main steps. First, we begin in Section 3.1 by briefly describing *smooth* and *non-smooth* variants of the IRBC model as concrete test cases³, thereby closely following [5]. The latter has recently become a workhorse model for studying methods for solving high-dimensional dynamic stochastic models (see, e.g., [22, 10], and references therein). The IRBC model is straightforward to explain, has a unique solution [50], and its dimensionality can be scaled up in a meaningful way as it just depends linearly on the number of countries considered. This property of the model allows us to concentrate on the computational issues of handling high-dimensional state spaces. To demonstrate that we can also handle non-smooth problems, we also consider a version of the IRBC model where investment is irreversible. Second, we present in Section 3.2—based on the example of the IRBC models—how a recursive equilibrium can be computed by applying the time iteration algorithm and discuss the computational challenges associated with solving for global solutions of large-scale dynamic stochastic economic models, and which motivate the development of our proposed DDSG method.

³As custom in the literature, we denote an IRBC model with no kinks in the policies as *smooth*, whereas we name it *non-smooth* if there exist non-differentiabilities in the latter functions.

3.1. A scalable test case: the international real business cycle model.

In the IRBC model we are considering as test case, time t is discrete, there are N countries, $j = 1, \dots, N$, each using its accumulated capital stock, $k_{j,t} \in \mathbb{R}_+$, to produce an output good, which can be used for investment, $I_{j,t}$, and for consumption, $c_{j,t} \in \mathbb{R}_+$, generating utility, $u_j(c_{j,t})$. In the model formulation we follow, complete markets are assumed [24]. Thus, a social planner solves the following (infinite-horizon) optimization problem:

$$\begin{aligned} & \max_{\{c_{j,t}, k_{j,t+1}\}} \left\{ \mathbb{E}_0 \left[\sum_{j=1}^N \tau_j \sum_{t=0}^{\infty} \beta^t u_j(c_{j,t}) \right] \right\} \\ & \text{subject to: (1) } \sum_{j=1}^N R(a_{j,t}, k_{j,t}, k_{j,t+1}, c_{j,t}) \geq 0, \forall t \\ & \text{(3.1) (2) } I_{j,t}(k_{j,t}, k_{j,t+1}) \geq 0, \forall \{t, j\}, \end{aligned}$$

where $a_{j,t}$ denotes productivity, τ_j are the welfare weights, β is the discount factor, \mathbb{E}_0 is the expectation conditional on the information available at $t = 0$, and where the initial capital stocks $\mathbf{k}_0 \in \mathbb{R}_+^N$ and productivity levels $\mathbf{a}_0 \in \mathbb{R}_+^N$ are assumed to be given. The parameterization for both the smooth and non-smooth IRBC model follows [22, 6] and is reported in Table 1. The first constraint (1) is the so-called *aggregate resource constraint*, while the second constraint (2) enforces *irreversible investments*.⁴ We refer to the *smooth* IRBC model where only constraint (1) is used. In contrast, the *non-smooth* IRBC model requires the solution of (3.1) with both constraints (1) and (2). Furthermore, we assume an additive separable per-period utility function that is given by $u_j(c_{j,t}) = c_{j,t}^{1-\frac{1}{\gamma_j}} / (1 - \frac{1}{\gamma_j})$. The aggregate resource constraint is a function of the production function Y , investment $I_{j,t}$, the capital adjustment costs Γ , and consumption $c_{j,t}$:

$$\begin{aligned} R(a_{j,t}, k_{j,t}, k_{j,t+1}, c_{j,t}) &= Y(a_{j,t}, k_{j,t}) - \Gamma(k_{j,t}, k_{j,t+1}) - I(k_{j,t}, k_{j,t+1}) - c_{j,t}, \\ Y(a_{j,t}, k_{j,t}) &= A \cdot a_{j,t} \cdot k_{j,t}^\alpha, \\ \text{(3.2) } I_{j,t}(k_{j,t}, k_{j,t+1}) &= k_{j,t+1} - (1 - \delta) \cdot k_{j,t}, \quad \text{and} \end{aligned}$$

$$\text{(3.3) } \Gamma(k_{j,t}, k_{j,t+1}) = \frac{\phi}{2} \cdot k_{j,t} \cdot \left(\frac{k_{j,t+1}}{k_{j,t}} - 1 \right)^2.$$

The law of motion of productivity is the sole source of stochasticity in the model and is given by:

$$\text{(3.4) } \ln a_{j,t} = \rho \cdot \ln a_{j,t-1} + \sigma \cdot (e_{j,t} + e_t),$$

where $e_{j,t} \sim \mathcal{N}(0, 1)$ and $e_t \sim \mathcal{N}(0, 1)$ denote the country-specific, and the global shocks, respectively. Both are assumed to be independent from each other and across time. Thus far, we have considered an infinite horizon problem. However, as indicated previously, economics frequently focuses on recursive equilibria [50, 33], in which the state of the economy is represented by a state variable and the economy's dynamics are given by a time-invariant function of this state (cf. (1.1)). We now briefly describe the recursive structure, that is, the two IRBC model formulations' first-order optimality conditions (FOCs). We direct the reader to [6] for the derivations.

⁴Investment is irreversible in the sense that it cannot be consumed or used for production in another country—an assumption that is more realistic than perfect reversibility, which is usually

TABLE 1
Parameterization of the smooth and non-smooth IRBC model.

Parameter	Symbol	Value
Discount factor	β	0.99
Elasticity of intertemporal substitution of country j	γ_j	$a + 0.75(j - 1)/(N - 1)$
Capital share	α	0.36
Depreciation	δ	0.01
Std. of log-productivity shocks	σ	0.01
Autocorrelation of log-productivity	ρ	0.95
Intensity of capital adjustment costs	ϕ	0.50
Aggregate productivity	A	$(1 - \beta(1 - \delta))/(\alpha \cdot \beta)$
Welfare weights	τ_j	A^{1/γ_j}

3.1.1. Smooth IRBC model - recursive structure. In order to obtain FOCs of the optimization problem stated in equation (3.1), we need to differentiate the Lagrangian with respect to $c_{j,t}$ and $k_{j,t+1}$. Denoting λ_t as the multiplier on the resource constraint at time t , and defining the growth rate of capital by $g_{j,t} = k_{j,t}/k_{j,t-1} - 1$, we obtain a system of N equilibrium conditions that have to hold at each t and for all countries j :

$$(3.5) \quad \lambda_t (1 + \phi g_{j,t+1}) - \beta \mathbb{E}_t \left[\lambda_{t+1} \left(a_{j,t+1} A \alpha (k_{j,t+1})^{\alpha-1} + (1 - \delta) + \frac{\phi}{2} g_{j,t+2} (g_{j,t+2} + 2) \right) \right] = 0.$$

where \mathbb{E}_t is the expectation conditional on the information available at t . Furthermore, the aggregate resource constraint (holding with equality due to strictly increasing per-period utility assumed) reads as follows:

$$(3.6) \quad \sum_{j=1}^N \left(a_{j,t} A (k_{j,t})^\alpha + k_{j,t} \left((1 - \delta) - \frac{\phi}{2} (g_{j,t+1})^2 \right) - k_{j,t+1} - \left(\frac{\lambda_t}{\tau_j} \right)^{-\gamma_j} \right) = 0,$$

where we use the fact that $c_{j,t} = (\lambda_t/\tau_j)^{-\gamma_j}$ holds at an optimal choice [6]. To explicitly point out the link to the abstract definition of a recursive equilibrium (1.1) (and the time iteration Algorithm 3.1 described in section 3.2 below), note that the smooth IRBC model presented here has a ($d = 2N$)-dimensional state space. As a reminder, the state variables are given by $\mathbf{x}_t = (\mathbf{a}_t, \mathbf{k}_t) \in \mathbb{R}^{2N}$, where $\mathbf{a}_t = (a_{1,t}, \dots, a_{N,t})$ and $\mathbf{k}_t = (k_{1,t}, \dots, k_{N,t})$, are the productivity and capital stock of country j . Furthermore, the optimal, time-invariant policy $p : \mathbb{R}^{2N} \rightarrow \mathbb{R}^{N+1}$ —the desired unknown—maps the current state into policies as $p(\mathbf{x}_t) = (\mathbf{k}_{t+1}, \lambda_t)$. Note that the investment choices determine the capital stock of the next period (i.e., $t + 1$) in a deterministic way through (3.2). In contrast, the law of motion of productivity, (3.4), is stochastic. Taken together, (3.2) and (3.4) specify the distribution of \mathbf{x}_{t+1} .⁵

3.1.2. Non-smooth IRBC model - recursive structure. To demonstrate the strength of our proposed algorithm to deal with high-dimensional and highly nonlinear models, we consider next a variant of the IRBC model where investment is irreversible, which in turn leads to non-smooth optimal policies. More precisely,

assumed to keep the model tractable.

⁵Throughout this paper, we compute the expectations by a simple yet fast monomial quadrature rule (see, e.g., [20], Sec. 7.5).

Algorithm 3.1 Time iteration algorithm.

Require: \tilde{p}' , ϵ .

- 1: **repeat**
 - 2: $\tilde{p} \leftarrow \tilde{p}'$
 - 3: **for all** $\mathbf{x}_t \in X$ **do**
 - 4: $\tilde{p}'(\mathbf{x}_t) \leftarrow \underset{\mathbf{k}_{t+1}, \boldsymbol{\mu}_t, \lambda_t}{\text{solve}} \{ \text{FOC}(\mathbf{k}_{t+1}, \boldsymbol{\mu}_t, \lambda_t) | \mathbf{x}_t, \tilde{p} \}$,
 - 5: **end for**
 - 6: **until** $\text{EulerEquationErrors}(\tilde{p}') < \epsilon$
 - 7: **return** \tilde{p}'
-

we assume that investment cannot be negative (cf. constraint (2) in equation (3.1)). As a direct consequence, we have to solve a system of $2N + 1$ equilibrium conditions. These conditions now include the Karush–Kuhn–Tucker (KKT) multiplier, $\mu_{j,t}$, for the irreversibility constraint. The optimality conditions for investment in capital as well as the irreversibility assumption for investment in each country j , and the associated complementary conditions, read as:

$$(3.7) \quad \lambda_t (1 + \phi g_{j,t+1}) - \mu_{j,t} - \beta \mathbb{E} \left[\lambda_{t+1} \left(a_{j,t+1} A \alpha (k_{j,t+1})^{\alpha-1} + 1 - \delta + \frac{\phi}{2} g_{j,t+2} (g_{j,t+2} + 2) \right) - (1 - \delta) \mu_{j,t+1} \right] = 0, \\ 0 \leq \mu_{j,t} \perp (k_{j,t+1} - k_{j,t}(1 - \delta)) \geq 0.$$

In addition, the aggregate resource constraint holds again. The state variables of this non-smooth IRBC model are again given by $\mathbf{x}_t = (\mathbf{a}_t, \mathbf{k}_t)$. The optimal, time-invariant policy $p : \mathbb{R}^{2N} \rightarrow \mathbb{R}^{2N+1}$ now maps the current state into policies as $p(\mathbf{x}_t) = (\mathbf{k}_{t+1}, \boldsymbol{\mu}_t, \lambda_t)$ where $\boldsymbol{\mu}_t = (\mu_{1,t}, \dots, \mu_{N,t})$. As in the previous Section 3.1.1, all the policies will have to be determined by iterating on (3.7) and the aggregate resource constraint.

3.2. The time iteration algorithm and its computational challenges.

In this section, we briefly introduce the time iteration algorithm [9]. The latter solves for a recursive equilibrium of a dynamic economic model by guessing a policy function and iteratively updating it using the first-order equilibrium conditions of a given model such as those presented in Sections 3.1.1 and 3.1.2). For the sake of brevity, the discussion that follows only considers the non-smooth IRBC model. However, the algorithmic logic carries over to any model formulated analogously. Let $\mathbf{x}_t \in X \subset \mathbb{R}^d$ denote again the state of the economy at discrete time t .⁶ Recall that in the model under consideration, the economy is represented as a ($d = 2N$)-dimensional state variable $\mathbf{x}_t = (\mathbf{a}_t, \mathbf{k}_t)$, and that the optimal policy function $p : X \rightarrow \mathbb{R}^{2N+1}$ maps the state variable to the capital choice, the KKT multipliers for the irreversible investments constraint, and the Lagrange multiplier for the aggregate resource constraint, that is,

$$(3.8) \quad p(\mathbf{x}_t) = (\mathbf{k}_{t+1}, \boldsymbol{\mu}_t, \lambda_t), \text{ and } p(\mathbf{x}_{t+1}) = (\mathbf{k}_{t+2}, \boldsymbol{\mu}_{t+1}, \lambda_{t+1}).$$

The goal of the time iteration algorithm is to determine an approximate optimal policy function on the entire domain X , that is, to find a global solution to the problem stated in expression (3.1) (see, e.g., [21], Section 17.8 for further details).

⁶In practical applications, we restrict ourselves to the canonical domain, that is, $\mathbf{x}_t \in [0, 1]^d$.

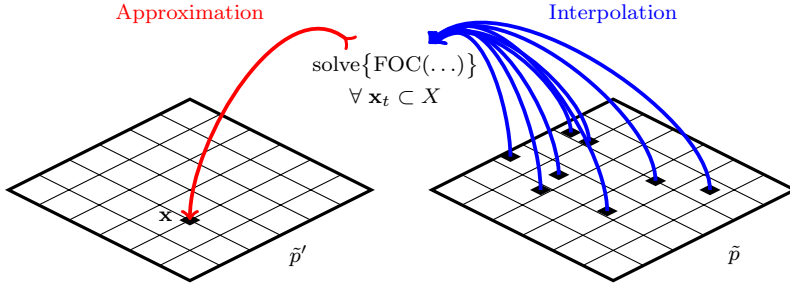


FIG. 1. This figure is a visual representation of one step of the time iteration algorithm. We solve the first-order conditions (FOC) of the model for the state variable \mathbf{x}_t in the updated policy function \tilde{p}' (left), using the policy function from the previous time iteration step \tilde{p} (right).

Such a policy function must satisfy the FOCs throughout the state space, that is,

$$(3.9) \quad p(\mathbf{x}_t) = \underset{\mathbf{k}_{t+1}, \boldsymbol{\mu}_t, \lambda_t}{\text{solve}} \left\{ \text{FOC}(\mathbf{k}_{t+1}, \boldsymbol{\mu}_t, \lambda_t | \mathbf{x}_t, p) \right\}, \forall \mathbf{x}_t \in X.$$

The time iteration procedure presented in Algorithm 3.1 iteratively computes the approximate optimal policy function \tilde{p}' —as a numerical proxy to the true p —by using the previous policy iterate (or initial guess) \tilde{p} to interpolate the values \mathbf{k}_{t+2} , $\boldsymbol{\mu}_{t+1}$, and λ_{t+1} (cf. expression (3.8)). The main components of the algorithm can be found in steps (3-5). In contrast to equation (3.9), we approximate \tilde{p}' here; thus, the FOCs hold exactly (up to numerical precision) for the discrete grid points $\mathbf{x}_t \subset X$. At each grid point where we solve (3.9), we require the solution for a system of $2N + 1$ nonlinear equations, that is, for the FOCs presented in section 3.1.2. In order to solve this nonlinear set of equations, a large number of interpolated values from \tilde{p} are required, as illustrated in Figure 1. Finally, in step (6), the time iteration algorithm is stopped conditional on satisfying that the so-called *Euler Equation Errors* are smaller than a given accuracy threshold. In economics, the latter criterion is a commonly used, unit-free measure of how accurately the equilibrium conditions are numerically satisfied (see, e.g., [6] Appendix C, or [14] Section 7.2 for further details). This process is repeated by swapping the policy function until the threshold ϵ is satisfied. The fundamental operations of Algorithm 3.1, which become computationally restrictive in a high-dimensional setting, are (i) generating the approximate policy function and (ii) costly interpolations from a policy function \tilde{p} for solving the system of nonlinear equations. Notice that the policy function we are interested in is high-dimensional and multivariate. In addition, due to the concavity assumptions on utility and production functions, the optimal policy p will also be nonlinear (cf. Section 3.1.2). Hence, approximating it only locally might provide misleading results. For such applications, we need a global solution and a performant interpolation method that approximates p over the entire state space X . To do this efficiently, we will employ DDSG in combination with a hybrid parallelization scheme in Sections 4 and 5, respectively.

4. Function approximation. The time iteration algorithm outlined in the previous section poses multiple computational challenges as it requires repeated function approximations and interpolation of a potentially high-dimensional policy function. This section outlines a function approximation method that addresses these concerns by using adaptive SGs and DD, which we discuss in Sections 4.1 and 4.2, respectively. With this background, we then combine the two numerical techniques in Section 4.3 resulting in the DDSG approximation scheme for high-dimensional functions. We note

that various forms of SG and DD exist; we will outline each method's key underlying concepts in a condensed fashion. For thorough introductions to SGs and DD, we refer to [8] and [32, 44, 35], respectively.

4.1. Adaptive sparse grids. Let $f : \mathbf{x} \rightarrow \mathbb{R}^n$ where $\mathbf{x} \in [0, 1]^d$, and d in our case is the number of continuous state variables in the economic model of interest, a potentially large number. For the sake of brevity, we assume that the function value is zero on the domain's boundary. This is not a necessary condition, and it can be easily changed by augmenting the basis function (see, e.g., [47]). First, consider a one-dimensional domain, discretized with grid spacing $h_l = 2^{-l}$. The grid points are located at $x_{l,i} = i \cdot h_l$, where $i \in \{1, \dots, 2^l\}$ and $l \in \mathbb{N}_+$ are the grid point indices and refinement level, respectively. Using the standard hat function, we define a family of univariate basis functions as

$$(4.1) \quad \phi_{l,i}(x) = \max\left(1 - \frac{1}{h_l}|x - x_{l,i}|, 0\right),$$

with support $[x_{l,i} - h_l, x_{l,i} + h_l]$. The one-dimensional basis functions can be extended to a d -dimensional domain by introducing the multi-indices $\mathbf{i} = (i_1, \dots, i_d)$ and $\mathbf{l} = (l_1, \dots, l_d) \in \mathbb{N}_+^d$. The grid points are now denoted as $\mathbf{x}_{\mathbf{l},\mathbf{i}} = (x_{l_1,i_1}, \dots, x_{l_d,i_d})$, and the corresponding d -linear hierarchical basis function is constructed by a tensor product, that is,

$$(4.2) \quad \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \prod_{j=1}^d \phi_{l_j,i_j}(x_j).$$

Next, we introduce the hierarchical index-set \mathbf{I}_1 and corresponding hierarchical subspace $W_{\mathbf{I}_1}$, which are given by:

$$(4.3) \quad \mathbf{I}_1 = \{\mathbf{i} : 0 < i_j < 2^{l_j}, i_j \text{ odd}, 1 \leq j \leq d\}, \quad W_{\mathbf{I}_1} = \text{span}\{\phi_{\mathbf{l},\mathbf{i}} : \mathbf{i} \in \mathbf{I}_1\}.$$

Notice that the odd increments of i_j result in the mutually disjoint support of the basis functions that cover the entire domain. For the space of piecewise linear functions,

$$(4.4) \quad V_\ell = \bigoplus_{\|\mathbf{l}\|_\infty \leq \ell} W_{\mathbf{l}},$$

we can construct a corresponding equidistant Cartesian grid, also called a *full grid*, with $M_\ell = 2^\ell$ number of grid points in each dimension, where ℓ denotes the *maximum refinement level*. Approximations using the full grid will have an L_2 interpolation error of $\mathcal{O}(M_\ell^{-2})$ and number of grid points are of $\mathcal{O}(M_\ell^d)$ [8]. It is clear that the full grid suffers from the curse of dimensionality and, thus, is not a scalable approach for high-dimensional function approximation.

SGs can alleviate this issue; their underlying construction principle is to systematically eliminate those hierarchical increment spaces, which contribute only little to the overall quality of the approximation [8]. It can be shown that the SG space is given by

$$(4.5) \quad V_\ell^{\text{SG}} = \bigoplus_{\|\mathbf{l}\|_1 \leq \ell + d - 1} W_{\mathbf{l}}.$$

In contrast to the full grid space in (4.4), where the maximum grid refinement levels in any dimension are restricted to ℓ , now the sum is restricted. The SG-based interpolation of a function f at point \mathbf{x} , for a maximum refinement level ℓ , can be uniquely expressed as

$$(4.6) \quad \mathcal{I}_\ell f(\mathbf{x}) = \sum_{\|\mathbf{1}\|_1 \leq \ell + d - 1} \sum_{\mathbf{i} \in \mathbf{I}_1} \alpha_{\mathbf{1}, \mathbf{i}} \phi_{\mathbf{1}, \mathbf{i}}(\mathbf{x}),$$

where the coefficients $\alpha_{\mathbf{1}, \mathbf{i}} \in \mathbb{R}$, commonly referred to as the *hierarchical surpluses*, can be readily computed (see, e.g., [7] for details). Under certain technical assumptions—the function to be approximated needs to exhibit bounded second-order mixed derivatives—the SG approximation error is $\mathcal{O}(M_\ell^{-2}(\log M_\ell)^{d-1})$, whereas the number of grid points grows as $\mathcal{O}(M_\ell \log(M_\ell)^{d-1})$. Thus, the number of grid points in an SG is significantly reduced, whereas the error has only slightly deteriorated. Finally, quadrature on SGs can be performed very effectively, that is,

$$(4.7) \quad \mathcal{Q}_\ell f = \sum_{\|\mathbf{1}\|_1 \leq \ell + d - 1} \sum_{\mathbf{i} \in \mathbf{I}_1} \alpha_{\mathbf{1}, \mathbf{i}} \int \phi_{\mathbf{1}, \mathbf{i}}(\mathbf{x}) d\mathbf{x},$$

can be readily evaluated by integrating the basis functions in (4.2)—a key of SGs that we will leverage in the sections to come. In summary, the SG approximation method is a computationally efficient method for interpolation and or quadrature of sufficiently smooth functions on moderately high-dimensional domains.

However, in situations where f are highly nonlinear and show distinct local features, a high resolution level is required, but only in particular locations of the domain, which renders the ordinary SG inefficient. Adaptive SGs can cope with this issue to some extent. Unlike the ordinary SG, which has an a priori selection of grid points, adaptive SGs utilize an a posteriori refinement, which, based on a local error estimator, selects which grid points in the SG structure to be refined further [41, 34]. For a predefined threshold $\epsilon_\gamma \in \mathbb{R}_+$ and

$$(4.8) \quad \gamma = \|\boldsymbol{\alpha}_{\mathbf{1}, \mathbf{i}}\|_\infty,$$

we deem a grid point to be significant if $\gamma > \epsilon_\gamma$. The refinement choice is governed by (4.8); however, depending on the application, more sophisticated criteria may need to be imposed for an efficient refinement [48]. If a grid point is not accepted, all grid points that fall in its support will be excluded in the higher refinement level.

A parallel implementation of adaptive SGs is a computationally efficient technique to tackle nonlinear economic models of moderately high dimensions, say $d \leq 100$ in case the smooth IRBC models, or $d \leq 20$ in the non-smooth IRBC models [6]. However, when working with models of significantly higher complexity, such as those with state-space dimensions > 100 or policy functions that exhibit high gradients, SGs, adaptive or not, are no longer a practical tool for function approximation. In particular, the data structure becomes computationally demanding to operate on (see, e.g., [40]). Furthermore, increasing refinement levels to resolve non-smooth features in high-dimensional settings significantly increases the number of grid points, thus quickly rendering the approximation uncomputable [6].

4.2. Dimensional decomposition. In this section, we outline the DD approach that directly targets the curse of dimensionality by attempting to model the input-output behavior of a high-dimensional function as a sum of low-dimensional

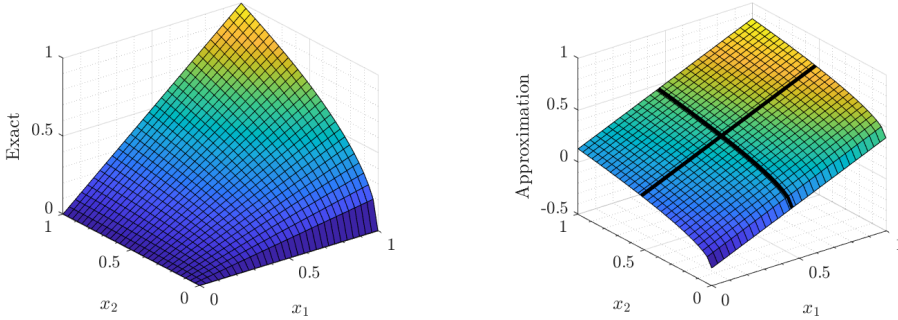


FIG. 2. A plot of $f(\mathbf{x}) = \mathbf{x}_1\sqrt{\mathbf{x}_2}$ (left), and the corresponding first-order cut-HDMR approximation with its one-dimensional component functions shown with thick black lines (right).

functions. As in the previous sections, we consider $f : \mathbf{x} \rightarrow \mathbb{R}$, where $\mathbf{x} \in [0, 1]^d$. Denote $\mathbf{u} \subseteq \mathcal{S} = \{1, 2, \dots, d\}$ as the *component index*, and $f_{\mathbf{u}} : \mathbf{x}_{\mathbf{u}} \rightarrow \mathbb{R}$ as the *component function*, where $\mathbf{x}_{\mathbf{u}}$ is the vector comprising of the values \mathbf{x}_i for $i \in \mathbf{u}$. The function $f(\mathbf{x})$ can be expressed as the hierarchical expansion

$$(4.9) \quad f(\mathbf{x}) = f_{\emptyset} + \sum_{1 \leq i \leq d} f_i(x_i) + \sum_{1 \leq i < j \leq d} f_{ij}(x_i, x_j) + \dots + f_{12\dots d}(x_1, x_2, \dots, x_d)$$

where f_{\emptyset} is a constant, $f_i(x_i)$ models the independent contribution, $f_{ij}(x_i, x_j)$ the pairwise dependent contribution, and so on, up to the last term $f_{12\dots d}(x_1, x_2, \dots, x_d)$, which accounts for the residual contributions. In its complete form, the summation in (4.9) is exact, as the last term accounts for all contributions of all the input variables. This representation is referred to as *High-Dimensional Model Representation* (HDMR) and is a general method for a function decomposition that captures high-dimensional input-output system behavior [44, 32, 18, 31]. The summation in (4.9) can be compactly written as

$$(4.10) \quad f(\mathbf{x}) = \sum_{\mathbf{u} \subseteq \mathcal{S}} f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}).$$

The terms in the summation in (4.10) are categorized by the *expansion order* $k = |\mathbf{u}|$ or equivalently by the dimension of $\mathbf{x}_{\mathbf{u}}$. Suppose this summation can be truncated to some maximum expansion order $\mathcal{K} \ll d$ without significant degradation in the approximation quality. In that case, one can reduce the overall dimensionality of the function. For example, if we consider a 100-dimensional function, its first-order expansion $k = 1$, will result in 100 1-dimensional functions. With this said, selecting the appropriate \mathcal{K} for the decomposition will undoubtedly depend on f .

Two popular versions of HDMR are *cut-* and *ANOVA-HDMR* [43, 32].⁷ We will define the component function for both types of HDMR, but our focus will be on cut-HDMR. As we will see, the cut-HDMR approach is a more fitting approach for our application due to its reliance on function evaluations as opposed to ANOVA-HDMR, which requires high-dimensional numerical integration. Let $w(\mathbf{x}) = \prod_{i=1}^d w_i(x_i)$ be a product measure, with $w_i(x_i)$ having a unit volume. By sequentially ascending through the expansion orders, starting from the zeroth-order, the optimally and

⁷See [31] for a high-level review of variations of cut-HDMR such RS-HDMR, mp-cut-HDMR, Multicut-HDMR, and lp-RS-hDMR.

uniquely defined HDMR component function

$$(4.11) \quad f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) = \underset{g_{\mathbf{u}}}{\operatorname{argmin}} \int \left(\sum_{\mathbf{u} \subseteq \mathcal{S}} g_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) - f(\mathbf{x}) \right)^2 w(\mathbf{x}) d\mathbf{x},$$

subject to $\int g_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) w_i(\mathbf{x}_i) dx_i = 0, \quad \forall i \in \mathbf{u},$

will only be dependent on lower-order component functions $f_{\mathbf{v}}(\mathbf{x}_{\mathbf{v}})$ for $\mathbf{v} \subset \mathbf{u}$. This is particularly important, as the contrary would eliminate any reduction in dimensionality. This attribute is a result of the imposed orthogonality condition in (4.11) [44, 18]. The cut-HDMR component functions are based on the Dirac measure,

$$(4.12) \quad w(\mathbf{x}) d\mathbf{x} = \prod_{i=1}^d \delta(x_i - \bar{x}_i) dx_i.$$

where $\bar{\mathbf{x}} = (\bar{x}_1, \dots, \bar{x}_d)$ is a reference point in space, referred to as the *anchor point*. Several techniques exist for the selection of the anchor point [15]. A straightforward approach is to simply choose the anchor point randomly in the high-dimensional space. However, this can affect the approximation (i.e., for $\mathcal{K} < d$), and a careless selection may result in large errors (see, e.g., [52] for details on HDMR error analysis). As noted in [49] a suitable anchor point should satisfy

$$(4.13) \quad \min_{\bar{\mathbf{x}}} \|f(\bar{\mathbf{x}}) - \mathbb{E}[f(\mathbf{x})]\|_1.$$

An appropriate anchor point can be selected via sampling $\bar{\mathbf{x}}$ such that $f(\bar{\mathbf{x}})$ approximates the mean of the function over the domain (see, e.g., [35] for further details). It is important to highlight that $\bar{\mathbf{x}}$ is not unique since a given function value could attain the mean value in several parts of the domain. The cut-HDMR component functions can be explicitly evaluated from (4.11) as a telescopic summation [30]

$$(4.14) \quad f_{\mathbf{u}}^{\text{cut}}(\mathbf{x}_{\mathbf{u}}) = \sum_{\mathbf{v} \subseteq \mathbf{u}} (-1)^{|\mathbf{u}| - |\mathbf{v}|} f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_{\mathbf{v}}}, \quad \text{with } f_{\emptyset} = f(\bar{\mathbf{x}}).$$

We use the notation $\mathbf{x} = \bar{\mathbf{x}} \setminus \mathbf{x}_{\mathbf{v}}$ to refer to assigning \mathbf{x} the values of $\bar{\mathbf{x}}$ but excluding the indices of \mathbf{v} . For example, given $\mathbf{x} = (x_1, x_2, x_3)$, then $\bar{\mathbf{x}} \setminus \mathbf{x}_{1,2} = (x_1, x_2, \bar{x}_3)$. In Figure 2, we depict an example 2-dimensional function on the left panel and the first-order cut-HDMR approximation on the right panel.

Using the Lebesgue measure, in place of the Dirac in (4.12) the ANOVA-HDMR decomposition is recovered [16], with the component function

$$(4.15) \quad f_{\mathbf{u}}^{\text{ANOVA}}(\mathbf{x}_{\mathbf{u}}) = \sum_{\mathbf{v} \subseteq \mathbf{u}} (-1)^{|\mathbf{u}| - |\mathbf{v}|} \int f(\mathbf{x}) d\mathbf{x}_{\mathcal{S} \setminus \mathbf{u}}, \quad \text{with } f_{\emptyset} = \int f(\mathbf{x}) d\mathbf{x}.$$

Notice that on the k -th expansion order, (4.15) requires quadrature across $(d - k)$ dimensions. Computationally, high-dimensional quadrature is a prohibitive operation subject to the same computational challenges we wish to address: the curse of dimensionality. In contrast, the cut-HDMR component functions are easily computed, requiring only function evaluation, which in turn for an arbitrary input \mathbf{x} , necessitates interpolation of $f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_{\mathbf{v}}}$. As such, we proceed with adopting the cut-HDMR

decomposition for which we will simply refer to as $f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}})$. In principle, representing the cut-HDMR component functions is independent of the underlying numerical approach. In practice, however, we will need a very efficient numerical approach since the component functions with high DD expansion orders will, to some extent, be exposed to the curse of dimensionality.

4.3. Dimensional decomposition using adaptive sparse grids. To approximate an economic policy function globally, we follow [35] and [53], and embed adaptive SGs within DD to form the DDSG method.⁸ Utilizing adaptive SGs for the underlying numerical method has two desirable properties: (i) they can be applied to moderately high-dimensional component functions with non-smooth features, and (ii) quadrature operations can be carried out efficiently on them. Using the combined DDSG numerical approach, we can approximate the function f as

$$(4.16) \quad f(\mathbf{x}) \approx \sum_{\substack{\mathbf{u} \subseteq \mathcal{S} \\ |\mathbf{u}| \leq \mathcal{K}}} \sum_{\mathbf{v} \subseteq \mathbf{u}} (-1)^{|\mathbf{u}| - |\mathbf{v}|} \mathcal{I}_{\ell} f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_{\mathbf{v}}},$$

which is a summation of $|\mathbf{v}|$ -dimensional adaptive SGs, where here we truncate the DD maximum expansion order $\mathcal{K} \ll d$. The accuracy of the DDSG approximation is dependent on the underlying function, SG approximation, maximum expansion order \mathcal{K} , and in turn, the number of component functions.⁹

At a given expansion order, the DDSG component functions increase combinatorially. The number of grid points in the approximation is the summation over the grid points of lower-dimensional SGs,

$$(4.17) \quad \sum_{k=1}^{\mathcal{K}} |V_{\ell,k}^{\text{SG}}| \binom{N}{k} = \sum_{k=1}^{\mathcal{K}} |V_{\ell,k}^{\text{SG}}| \frac{d!}{(d-k)!k!}$$

where $|V_{\ell,k}^{\text{SG}}| = \mathcal{O}(M_{\ell} \log(M_{\ell})^{k-1})$ denotes the number of grid points for k -dimensional SG with maximum refinement level ℓ (see, e.g., Section 4.1 for further details).¹⁰ For $\mathcal{K} \ll d$, the number grid points for DDSG is given by $\mathcal{O}(M_{\ell} \log(M_{\ell})^{\mathcal{K}-1} d^{\mathcal{K}})$. In comparison to SG, we have removed the exponential dependence on d , but now instead, the problem is exponentially dependant on \mathcal{K} . Even for moderately sized problems, the number of component functions corresponding to an adaptive SG can pose a computational challenge for values of $\mathcal{K} > 1$.

To address this shortcoming, we outline two criteria that will efficiently truncate the expansion by ignoring its insignificant component functions. Specifically, we will outline two DDSG adaptivity criteria for the relevance of (i) the proceeding expansion order and (ii) of each component function within an expansion order. With these criteria, we can generate an adaptive variate of (4.16) by only proceeding to higher expansion orders when required and eliminating insignificant component functions within an expansion order. We emphasize that the expansion criterion, or the active dimension selection for that matter, is not a measure of convergence of the DD expansion. These criteria provide an assessment of the importance of the current

⁸The so-called *SG combination technique* (see, e.g., [8, 16], and references therein) provides an alternative construction of the basic DDSG method. However, we follow in our work the description of [35], where adaptive SGs are embedded within DD.

⁹For details on the approximation error of the DDSG method, see [35].

¹⁰The exact number of grid points for SG will depend on the type of SG used (see, e.g., [41] for a comparison of different variations of SGs).

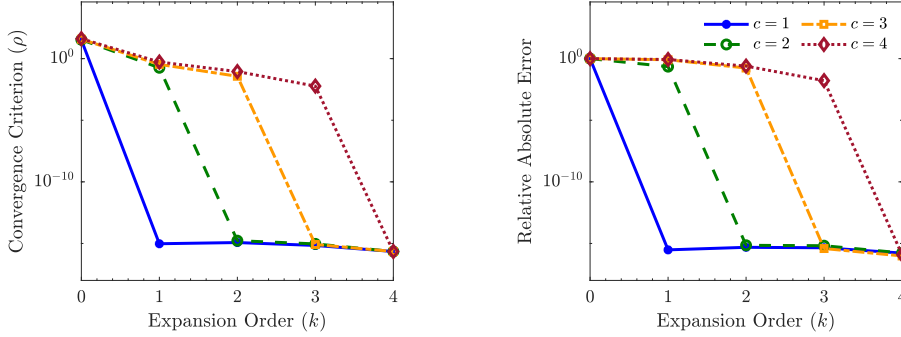


FIG. 3. A plot of the DDSG expansion criterion (left) and the relative absolute error of the DDSG approximation (right) with respect to the expansion order k are displayed. For both cases, the test function is a 4-dimensional polynomial of degree c ; i.e., $f(\mathbf{x}) = (\mathbf{x}_1 + \dots + \mathbf{x}_4)^c$. The component functions are represented using SGs, the absolute relative error is evaluated using 10^3 random samples, and $\bar{\mathbf{x}} = [0.5, 0.5, 0.5, 0.5]$.

expansion or component function with respect to the previously computed values. As such, the usage of the criteria, in particular with an aggressively high tolerance may lead to might lead to excessive truncation or pruning of the component functions. As discussed in more detail in Section 6.2, these criteria can also be used as an analysis tool to understand the impact of the different component functions and or expansion orders on the overall approximation.

4.3.1. Expansion criterion. The expansion criterion is the relative residual between two consecutive expansion orders k and $k - 1$. Given the current expansion order k , a predefined convergence threshold $\epsilon_\rho \in \mathbb{R}_+$, and a coefficient

$$(4.18) \quad \rho = \frac{\left\| \sum_{|\mathbf{u}| \leq k} \mathcal{Q}_\ell f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) d\mathbf{x} - \sum_{|\mathbf{u}| \leq k-1} \mathcal{Q}_\ell f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) d\mathbf{x} \right\|_2}{\left\| \sum_{|\mathbf{u}| \leq k-1} \mathcal{Q}_\ell f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) d\mathbf{x} \right\|_2},$$

we justify the progression to the next expansion order $k + 1$ if $\rho > \epsilon_\rho$. The SG quadrature operation \mathcal{Q}_ℓ is given by (4.7). With the assumption that $\mathcal{K} \ll d$, we can expect that the component function will not be high-dimensional, and thus, quadrature will not pose a computational bottleneck here.

It is important to highlight that a truncation of (4.16) is not necessarily an approximation. Indeed, the truncation can be error-free as long as the underlying function is additively separable up to the \mathcal{K} -th expansion order. This feature is shown in Figure 3, where we display the value of ρ and relative absolute error regarding the expansion order k for a 4-dimensional polynomial of varying degree c . Both ρ and the relative absolute error reach machine precision when the expansion order is $k \geq c$. We note that the expansion criterion is not a measure of convergence, as an increase in the expansion order does not necessarily translate into a decrease in the error.

4.3.2. Active dimension selection criterion. We look to identify insignificant component functions within an expansion order by using the active dimension selection criterion. Here, we assume that the underlying function is not constant with respect to a single variable. Thus, we only focus on component function in-

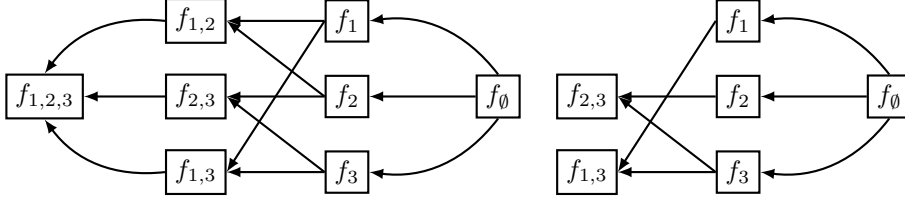


FIG. 4. A sketch of the component indices of a 3-dimensional function (left), and the same with active dimension coefficient $\eta_{1,2} < \epsilon_\eta$ (right) are shown. All component function indices which form a superset of $\{1, 2\}$ are ignored—that is, $\{1, 2\}$ and $\{1, 2, 3\}$. In both cases, arrows signify the computational dependence, for example, $f_{1,3}$ is dependent on the values of f_1 , f_3 and f_0 .

indices $|\mathbf{u}| \geq 2$. Given a predefined active component function threshold $\epsilon_\eta \in \mathbb{R}_+$ and coefficient

$$(4.19) \quad \eta_{\mathbf{u}} = \frac{\|\mathcal{Q}_\ell f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) d\mathbf{x}\|_2}{\left\| \sum_{\mathbf{v} \subset \mathcal{S}, |\mathbf{v}| \leq |\mathbf{u}|-1} \mathcal{Q}_\ell f_{\mathbf{v}}(\mathbf{x}_{\mathbf{v}}) d\mathbf{x} \right\|_2},$$

we deem the component function index \mathbf{u} as important if $\eta_{\mathbf{u}} > \epsilon_\eta$.¹¹ Component indices that do not satisfy this condition and any superset of these indices are not computed. In Figure 4, we schematically depict the active dimension selection criterion on a three-dimensional function and the resulting “pruning” effect on the combinatorial tree of component functions. In the left panel, we can see the component functions resulting from a full expansion. In the right panel, we display a scenario for which $\epsilon_\eta > \eta_{1,2}$ holds. In this case, the corresponding component index is removed from the computation, but also $\{1, 2, 3\}$ as $\{2, 3\} \subset \{1, 2, 3\}$. Notice that the values from the quadrature operations in (4.19) can also be shared with (4.18), and vice versa.

5. High-dimensional parallel time iteration framework. This section introduces a performant framework for high-dimensional DDSG function interpolation and approximation, leveraging an optimized adaptive SG framework [47]. In Section 5.1, we describe a vectorized approach to DDSG interpolation, which allows for a performant, cache-efficient execution of function calls. Next, we outline in Section 5.2 the parallelized DDSG time iteration framework for solving large-scale dynamic stochastic economic models.

5.1. Vectorized DDSG interpolation. In solving the system of nonlinear equations at a given point, the time iteration algorithm requires frequent interpolation on the policy functions \tilde{p} from the previous iteration (see Section 4.3 for further details). These interpolations typically take up the majority—often far beyond 90% [47]—of the computation time needed to solve the nonlinear set of equations. Therefore, the time-to-solution of the time iteration algorithm is highly sensitive to the performance of the DDSG interpolation function call. Direct implementation of (4.16) results in a massive number of repeated computations as many of the SG interpolants are identical. Consider for example, the component functions $f_{1,2}$ and $f_{1,2,3}$ both require the interpolation values of $\mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_1}$ and $\mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_2}$. Notice that the number of repeated computations increases nonlinearly with respect to the function’s dimensionality and DDSG expansion order. Furthermore, a simple

¹¹Note that different criteria for (4.19) have been proposed in the literature so far (see, e.g., [53]).

lookup-table approach would result in an erratic memory access pattern on a large array; thus, the computation would be plagued with cache misses.

We can eliminate all redundant SG interpolation and achieve an ideal access pattern without significant overhead in the memory footprint. This is achieved by separating telescopic summation in (4.16) and storing the SG interpolation and coefficient values in two separate arrays

$$(5.1) \quad \left. \begin{aligned} \mathbf{a}_i(\mathbf{x}) &= \mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_i}, \\ \mathbf{b}_i &= \sum_{\mathbf{u} \subseteq \mathcal{S}} \sum_{\substack{\mathbf{v} \subseteq \mathbf{u} \\ \mathbf{i}=\mathbf{v}}} (-1)^{|\mathbf{u}|-|\mathbf{v}|} \end{aligned} \right\} \quad \{\forall \mathbf{i} \subseteq \mathcal{S} : |\mathbf{i}| \leq \mathcal{K}\}.$$

In addition, \mathbf{b} , as it is independent of \mathbf{x} , will only need to be computed once.¹² Notice that for notation clarity, the formulation above does not consider the two DDSG adaptivity criteria described in Section 4.3 and asserts the full expansion up to the maximum expansion order \mathcal{K} . The following section will provide an explicit algorithm describing how each component function is selected with the respective DDSG adaptivity criteria. The DDSG interpolation function call now reduces to a desirable dot product $f(\mathbf{x}) \approx \mathbf{a}(\mathbf{x})^\top \mathbf{b}$ with a contiguous data access pattern.

5.2. Parallel DDSG time iteration algorithm. We begin with the general description of the generic parallelized DDSG Algorithm 5.1 and proceed to outline the inclusion of DDSG in the time iteration Algorithm 3.1.

The parallelized DDSG algorithm takes as input: the function f to be approximated, maximum expansion order \mathcal{K} , expansion criterion tolerance ϵ_ρ , active dimension selection tolerance ϵ_η , anchor point $\bar{\mathbf{x}}$, maximum refinement level ℓ , and adaptivity SG tolerance ϵ_γ . We begin in step (1) with all compute instances initializing the empty, vectorized DDSG arrays defined in (5.1), the zeroth-order component function $f_\emptyset = f(\bar{\mathbf{x}})$, and the *reject index-set* $\mathbf{Z} = \emptyset$. The reject index-set collects all component function indices excluded from the DDSG expansion as per the active dimension selection criterion. We sequentially progress through the expansion orders in the body of the algorithm in steps (2–20). At expansion order k , the *current order index-set* \mathbf{C} is defined as the component indices of order k , which are not a superset of any of the indices in \mathbf{Z} . Expanding on the example in Figure 4 with expansion order $k = 3$, values of the reject and current order index set will be $\mathbf{Z} = \{\{1, 2\}\}$ and $\mathbf{C} = \{\emptyset\}$ as $\{1, 2, 3\} \supset \{1, 2\}$, respectively. Subsequently, at step(4), we rebalance the compute resources evenly based on the current order index set, and the computation is carried out in parallel in steps (5–14). For each parallel SG interpolant, the quadrature value, and the active dimension selection coefficient η_i are computed for component index \mathbf{i} . Next, in steps (9–13), we employ the active dimension selection criterion for each component index. If the component function is accepted, we assign the DDSG vector arrays as defined in (5.1). If not, the component index \mathbf{i} is added to the rejected index set, and the SG interpolant and its quadrature values are discarded. Notice that each compute instance has a local or partial version of the computed variables within the parallel section of the algorithm. We perform the global synchronization upon exiting the parallel region at step(15). In steps (16–19), with the globally available quadrature values available, we apply the DDSG expansion criterion (4.19). If the threshold is reached, the routine terminates. Otherwise, we proceed to the next ex-

¹²While the vectorized strategy is similar to the memoization technique [38], we are also concerned with data contiguity for increased cache performance.

Algorithm 5.1 Generic Parallel Adaptive DDSG Algorithm.

Require: $f, \mathcal{K}, \epsilon_\rho, \epsilon_\eta, \bar{\mathbf{x}}, \ell, \epsilon_\gamma$

- 1: initialize : $\{\mathbf{a}, \mathbf{b}, f_\theta, \mathcal{Z}\}$
- 2: for $k = 1$ to \mathcal{K} do
- 3: $\mathbf{C} \leftarrow \{\mathbf{C} \subseteq \mathcal{S} : \forall \mathbf{c} \in \mathbf{C}, \forall \mathbf{z} \in \mathbf{Z}, |\mathbf{c}| = k, \mathbf{c} \not\supseteq \mathbf{z}\}$
- 4: load_balance given \mathbf{C}
- 5: for all $\mathbf{i} \in \mathbf{C}$ parallel do
- 6: compute : $\mathcal{I} \ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_i}$ \triangleright Using SG adaptivity tolerance ϵ_γ .
- 7: compute : $\mathcal{Q} \ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_i}$ \triangleright Using SG adaptivity tolerance ϵ_γ .
- 8: compute : η_i
- 9: if $\eta_i \geq \epsilon_\eta$ then
- 10: compute : $\{\mathbf{a}_i(\mathbf{x}), \mathbf{b}_i\}$ \triangleright As defined in (5.1).
- 11: else
- 12: $\mathbf{Z} \leftarrow \{\mathbf{Z} \cup \mathbf{i}\}$
- 13: end if
- 14: end for
- 15: synchronize : $\{\mathbf{Z}, \mathcal{Q} \ell f \dots, \mathbf{a} \dots, \mathbf{b} \dots\}$
- 16: compute : $\{\rho\}$
- 17: if $\rho < \epsilon_\rho$ then
- 18: break
- 19: end if
- 20: end for
- 21: return \mathbf{a}, \mathbf{b}

pansion order. The return values of the routine at step(21) are the vectorized DDSG interpolation arrays.

In Figure 5 we show the schematic for the parallel DDSG time iteration algorithm. In particular, we display two levels of parallelism: the first is based on distributed-memory (dashed blue lines), whereas the second relies on mainly share-memory¹³(dotted red lines)¹⁴. Starting in step(1), we assign the newly computed policy function \tilde{p}' as the current policy function \tilde{p} , or in the case of the initial step, we assign a random (guessed) policy function. Next in step(2), we begin the HDMR decomposition starting from expansion order $k = 1$ and evenly assign the component functions amongst groups of distributed-memory processes, referred to as a *process-groups*. For each process group in step(3), the respective SGs of the component functions are computed in parallel using a shared-memory approach. At a given refinement level, first, the SG grid points are evenly partitioned amongst the compute resources (threads), and second, at each grid point, we solve the first-order conditions¹⁵ noted in Section 3.2. We incrementally ascend through the SG refinement levels based on the adaptivity criterion described in Section 4.1. Next in step(4), having computed the SG, we evaluate its quadrature for use in the DDSG adaptivity criteria noted in Section 4.3. In step(5), corresponding to step(15) in Algorithm 5.1, we globally synchronize the distributed computations. We proceed in step(6) by checking the expansion criterion noted in (4.18) and move to the next HDMR expansion order if required, that is, back to step (2). Finally, in step (7), we reconstruct the DDSG policy function for the next iteration in step(1). With this parallelization approach, we expect that the parallel efficiency would be higher in the DD component, the pri-

¹³Both distributed- and shared-memory parallelization can be used in adaptive SG, though it is more effective to allocated distributed memory processes to the DD portion of the code [12].

¹⁴MPI and Intel(R) TBB are used for distributed- and shared-memory parallelism, respectively.

¹⁵We use IPOPT [51] for solving the FOCs.

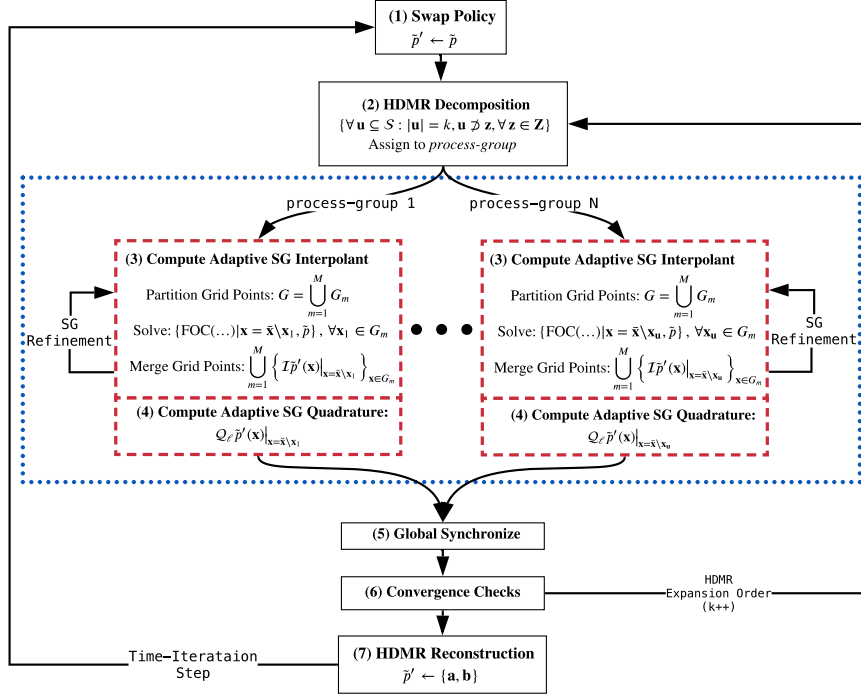


FIG. 5. The primary layer of parallelization (dashed blue lines) occurs in computing the HDMR component functions from steps (2–5). The secondary layer of parallelism (dotted red line) takes place while solving the system of the nonlinear equations at each SG grid point and carrying out quadrature operations, as shown in steps (3) and (4), respectively.

mary layer, compared to the SG component, the secondary layer of the algorithm. As highlighted in [12], this is due to the unutilized computing resources in the SG algorithm at low refinement levels and also because of the repeated synchronization on increasing refinement levels. For example, in a one-dimensional SG with a maximum refinement level of $\ell = 4$, we would have 1,2,4, and 8 grid points at each refinement level. Thus allocating 8 cores for this computation would only achieve full utilization on the last refinement level. Furthermore, at each refinement level, we would require synchronization.

6. Results. This section demonstrates the capabilities of the parallelized DDSG time iteration framework introduced in Section 5. We begin in Section 6.1 with a set of basic performance tests for grid point reduction, vectorization performance, scalability, and speedup. In Section 6.2, we utilize DDSG as a tool to analyze both variates of the IRBC model described in Section 3.1. Using conclusions drawn from this analysis, in Section 6.3, we deploy the parallel DDSG time iteration framework to solve a set of large-scale smooth and non-smooth IRBC models. We introduce the following naming standard for the parameterization of the DDSG routine:

$$\text{DD}_{\mathcal{K}}^{\epsilon_{\eta}} \text{SG}_{\ell}^{\epsilon_{\gamma}}.$$

It should be assumed that $\epsilon_{\rho} = \epsilon_{\eta}$ (see Section 4.3 for definitions of ϵ_{ρ} and ϵ_{η}) unless otherwise noted. If a nonadaptive variate of the DDSG method is used, the values of ϵ_{η} and ϵ_{γ} are omitted. To measure our scheme’s convergence, we follow the previous

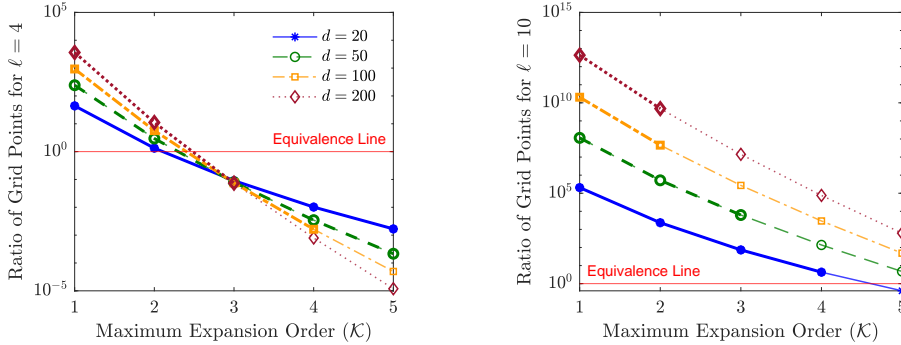


FIG. 6. A plot of the ratio of SG to DDSG grid points for a maximum refinement level $\ell = 4$ (left), and 10 (right) with respect to the maximum expansion order \mathcal{K} at varying dimension. In both figures, the thicker lines represent data points for which the number of DDSG grid points is $< 10^9$.

literature and report the *Euler Equation Errors* (see [6], Appendix C for details). A total of 10,000 error samples are gathered for which the maximum (Maximum Euler Error) and average (Average Euler Error) are reported in \log_{10} scale. In all test cases, the reported targeted Average and Maximum Euler errors in our approximate solutions align with the current literature. All experiments were conducted on the Piz Daint supercomputer at CSCS (Cray XC50 with 12-cores and 64GB of memory per node).

6.1. Unit Tests. We now outline a set of unit test results to highlight the reduction in the number of grid points, the performance of the vectorized interpolation, the parallel scalability, and the speedup of the framework with respect to the state-of-the-art SG framework.

6.1.1. Grid point reduction. In Figure 6, we show the SG to DDSG grid point ratios with respect to the maximum expansion order \mathcal{K} for various function dimensionalities. We refer the reader to Section 6 for further details on the adopted notation standard $DD_{\mathcal{K}}^{\epsilon_{\eta}}SG_{\ell}^{\epsilon_{\gamma}}$. The red line in each graph denotes a ratio of one—that is to say, where the number of grid points for DDSG and SG are equivalent. The plot lines not marked with thick lines are where using DDSG would result in a number of grid points $> 10^9$. In these conditions, memory issues would render DDSG, or even SG, inoperable. The two plots presented correspond to two scenarios, one where the underlying function exhibits relatively smooth dynamics (lower maximum refinement levels) and the other where one wishes to resolve non-smooth futures (higher maximum refinement levels). For a lower refinement level $\ell = 4$, shown in the left panel, DDSG provides a reduction in grid points up to a maximum expansion order $\mathcal{K} = 2$. At a maximum expansion order of $\mathcal{K} = 1$, we can see orders of magnitude in grid point reduction. This trend is further exaggerated when we require higher SG refinement levels. We show the same test in the right panel but with $\ell = 10$. Here there is a reduction in grid points up to an expansion order of $\mathcal{K} = 5$. However, at such high expansion orders and refinement levels, the overall number of grid points is much too high for practical usage.

6.1.2. Function call performance. The solution time for the first-order conditions of the IRBC models outlined in Sections 3.1.1 and 3.1.2 is heavily dependent on both the number of optimizer function invocations per grid point and the time required for the function invocation. In Figure 7, we present the results for one step

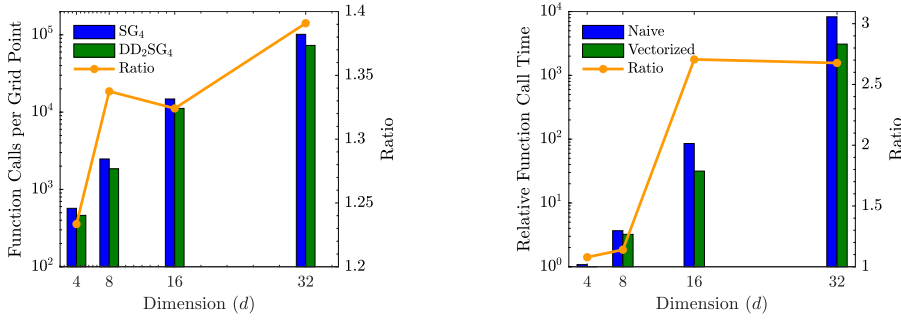


FIG. 7. A plot of the number of function calls per grid point for SG_4 and DD_2SG_4 (left), and relative function call time of the DDSG naive and vectorized DDSG interpolation (right), with respect to varying dimensions for the smooth IRBC model.

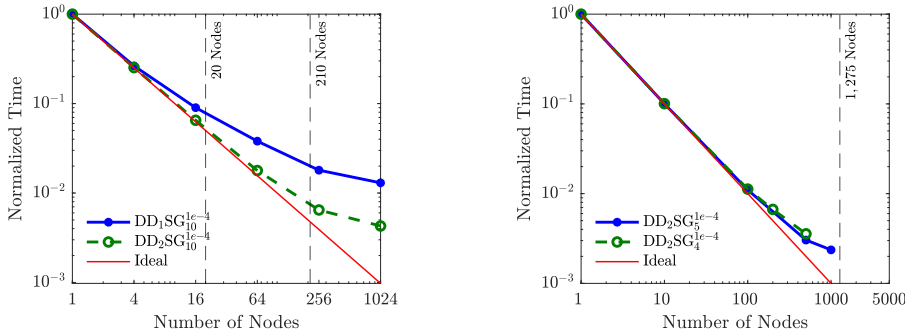


FIG. 8. Normalized compute time of a 20- (left), and a 50-dimensional model (right) taking one time iteration step using DDSG with expansion orders 1 and 2, and maximum refinement levels 4 and 5, respectively.

of the time iteration using SG_4 and DD_2SG_4 . In all tests, the optimizer is set with a termination tolerance of 10^{-4} . In the left panel, we can see that DDSG provides roughly 30% reduction in the number of function calls compared to SG. In the right panel, we see the relative compute times of the naive and vectorized approach (see Section 5.1) for DDSG interpolation. Here the naive approach is the evaluation of (4.16) directly. We can see that the vectorized approach provides a speedup of roughly 2.8 times that of the naive implementation.

6.1.3. Scalability. In Figure 8, we show the normalized compute time for one step of the time iteration for a 20 and 50-dimensional IRBC model. As noted in 5.2 and described in detail in [12], we expect the primary layer of parallelization, the DD component of the DDSG algorithm, to have better parallel efficacy than the SG component, that is, the secondary layer. The left panel shows numerical results for a 20-dimensional model with a fixed maximum refinement level of 10 and a varying maximum expansion order 1 and 2. Here, we have 20 and 210 component functions for the respective maximum expansion orders. We can see almost ideal strong scaling up to a number of nodes equal to the number of component functions for both tests. After this point, additional parallelism is taken from the secondary, less efficient layer of parallelism in the SG algorithm. In the right panel, we use a 50-dimensional model with a fixed maximum expansion order of 2 and varying maximum refinement levels of 4 and 5. In contrast to previous test cases, we have 1,275 component functions for

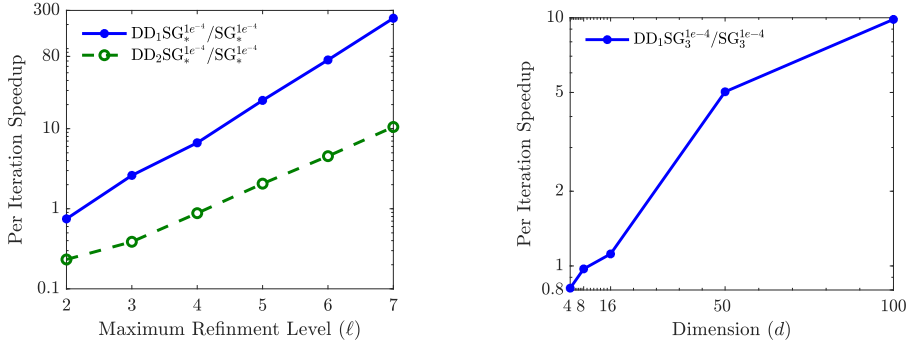


FIG. 9. The plot of the speedup of DDSG over SG for a 8-dimensional smooth and non-smooth IRBC model with respect to maximum refinement levels (left), and varying dimensionality (right). The non-smooth IRBC model represents the green dashed line for which we used $K = 2$.

both tests, which is larger than the maximum number of nodes. Here we can observe almost ideal strong scaling up to 1,000 nodes, which is the same for both tests. Furthermore, we see that the difference in the SG maximum refinement level slightly affects the parallel performance, with higher maximum refinement levels providing a marginal advantage in scalability.

6.1.4. Speedup. In Figure 9, we display the speedup of the parallelized DDSG time iteration framework with respect to an SG version. We show a single time iteration step of an 8-dimensional model for DDSG at maximum expansion order 1 and 2 at varying refinement levels in the left panel. This test is done on a single node using shared-memory parallelism, and both DDSG and SG approximation methods use the same adaptive coefficient. At refinement level 7, for the respective tests, we can see that the DDSG approach provides 240 and 10 times faster runtimes than the SG approach. In the right panel, we show the time-to-solution for a single DDSG time iteration step at a maximum expansion order of 1, maximum refinement level 3, and varying dimensions for the smooth IRBC model. The tests are deployed on a number of nodes equal to that of the dimensionality of the model. The DDSG routine provides a speedup of up to 10 times over SG implementation. The reason for this speedup is primarily due to DDSG operating on 100 one-dimensional SGs while the SG time iteration framework operates on a 100-dimensional SG.

6.2. Model analysis. We now use the active dimension selection criterion outlined in Section 4.3 as an analysis tool to assess the significance of the component functions of the policy function. In Figure 10, we show the aggregate minimum, average, and maximum values of $\eta_{\mathbf{u}}$ for the 8-dimensional smooth and non-smooth models, at varying expansion orders. For a given expansion order, an increase in variability between the maximum and minimum values of $\eta_{\mathbf{u}}$ signifies that active dimension selection could be effective in selecting only a subset of the component functions. In contrast, if both the maximum and minimum values are equivalent to the average, then we would conclude that no component function could be considered to be more significant than the other. In such scenarios, active dimension selection would not be an effective way to reduce the computational cost of approximating the respective policy functions. In the left panel of Figure 10, we can see that the smooth model’s policy function only significantly impacts up to the third expansion order in the left panel. The second and third-order terms are approximately 100 times less significant compared to the first-order component functions. Our experiments show that com-

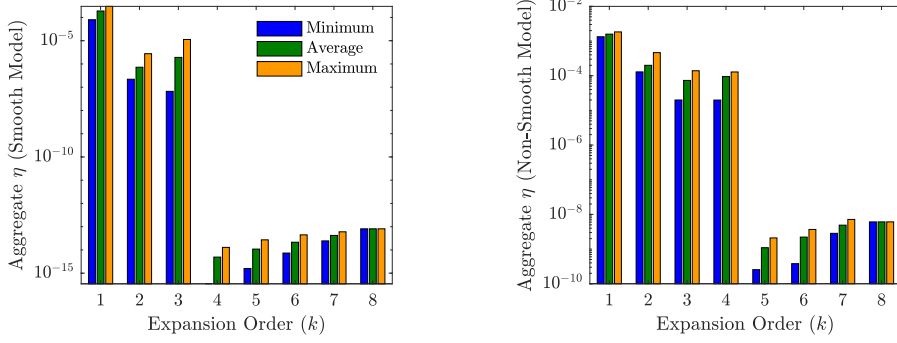


FIG. 10. Aggregate minimum, average, and maximum values of the active dimensional selection criterion for the smooth (left), and the non-smooth (right) 8-dimensional IRBC models.

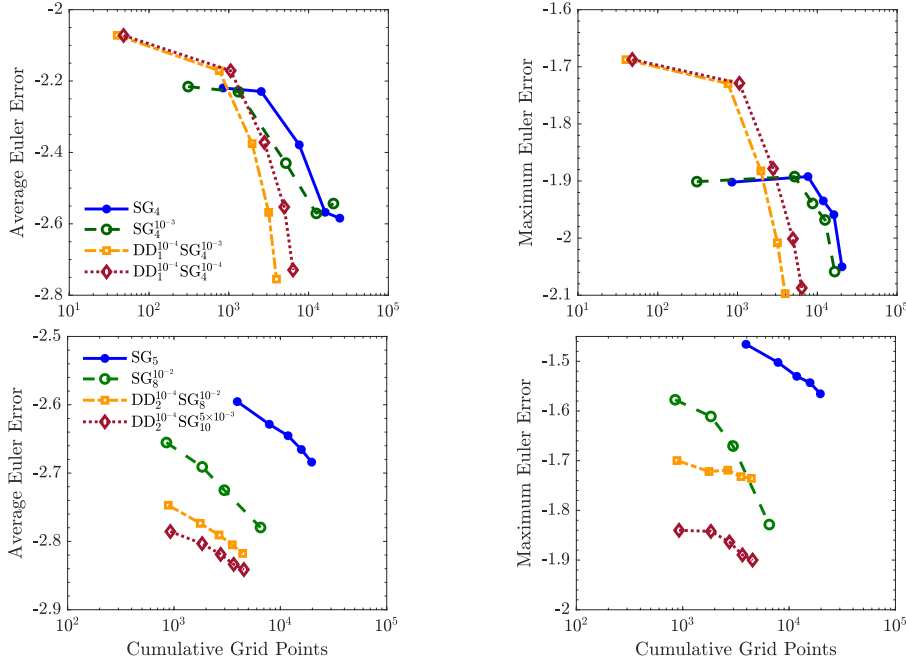


FIG. 11. Average and maximum Euler error with respect to the cumulative number of grid points for an 8-dimensional smooth (top panel) and non-smooth (bottom panel) IRBC model. Notice that each data point represents a time iteration step.

ponent functions with $\eta_{\mathbf{u}} < 10^{-4}$ do not play a significant role in the approximation of the policy function. Thus, they can be eliminated without significantly degrading the overall approximation quality. Notice that using $\epsilon_{\eta} = \epsilon_{\rho} = 10^{-4}$, the active dimensional selection criterion, and the expansion criterion would truncate the expansion at the first DDSG expansion order. In the right panel of Figure 10, we see the same analysis for the non-smooth IRBC model, whereas in this case, the component functions show significance up to the fourth-order expansion terms. Here we can see that both the first and second-order component functions are required while the majority of the third and fourth-order component functions fall below the 10^{-4} threshold. With this analysis, we proceed with our experiments using $\mathcal{K} = 1$ and 2 for the smooth and non-smooth IRBC models, respectively, and $\epsilon_{\eta} = \epsilon_{\rho} = 10^{-4}$ for both cases.

We now look at the effect of these parameters on the convergence trajectories. In the top panel of Figure 11, we show the average and maximum Euler errors for the smooth IRBC model using SG, adaptive SG, and the DDSG time iteration algorithm. The horizontal axis corresponds to the cumulative number of grid points evaluated for several time iteration steps. For DDSG to be a superior approximation method than SG and adaptive SG, we should attain smaller Euler errors for the same number of grid points. We test our DDSG implementation with $\mathcal{K} = 1$ at $\ell = 4$, using $\epsilon_\gamma = 10^{-3}$ and 10^{-4} . The observed average and maximum Euler errors begin relatively high in both configurations but quickly decrease beyond classical and adaptive SG. Notice that DDSG with $\epsilon_\gamma = 10^{-4}$ does not improve the convergence rate in comparison to DDSG with $\epsilon_\gamma = 10^{-3}$, but there is a reduction in the number of grid points. With this said, DDSG requires roughly 10 times fewer grid points to attain equivalent Euler adaptive SG errors. In the bottom panel of Figure 11, we show the average and maximum errors for the non-smooth IRBC model using SG, adaptive SG, and the DDSG method at varying SG adaptivity coefficients. Unlike the previous case, the non-smooth IRBC model requires significantly higher SG refinement levels to represent its non-smooth policy function adequately. Two tests were conducted using the DDSG method with $\mathcal{K} = 2$, one using $\ell = 8$ with and $\epsilon_\gamma = 10^{-2}$, and the other with $\ell = 10$ and 5×10^{-3} . Notice that in high-dimensions, at such high SG refinement levels, the number of grid points would almost surely render a model uncomputable for adaptive SGs, even if run on modern supercomputer facilities. For example, using an SG at $\ell = 10$, a 20-dimensional model would consist of about 5 billion grid points. As observed in the right bottom panel, DDSG requires higher refinement levels to sufficiently decrease the maximum Euler error. Even at this refinement level, in comparison to adaptive SG at $\ell = 8$ and $\epsilon_\gamma = 10^{-2}$, the DDSG method allows for a significant reduction in the Euler error with half the number of grid points of adaptive SG.

6.3. Large-scale models. Based on the analysis conducted in the previous section, we proceed here by adopting the DDSG parameters noted above as a baseline for solving a set of large-scale IRBC models with up to 300 and 60 dimensions for the smooth and non-smooth IRBC models, respectively. At such dimensions, adaptive SGs would not be a fitting numerical approach due to the sheer number of grid points. The effect of this massive increase in the number of grid points is proportional to an increase in the computation time, which quickly surpasses a day of runtime on a high-performance computer. In Table 2, we show the results for the two IRBC model variants. Firstly, we achieve low average and maximum Euler errors for the smooth model for all test cases. For the 300-dimensional model, we have used a lower SG refinement level of 3, compared to 4 for the 100- and 200-dimensional models. To compensate for this lower refinement level, we use $\epsilon_\gamma = 10^{-6}$. With this said, we can see that even for the 300-dimensional case, the proposed framework is sufficient to achieve -2.89 and -1.78 for the average and maximum Euler errors, respectively. For comparative purposes, we show the number of SG grid points at refinement level 4. There is roughly a four-orders-of-magnitude difference between the required number of grid points between SG and the DDSG. Using adaptive SGs will undoubtedly decrease the number of grid points. However, dimensions > 100 remain uncomputable using adaptive SG. The per iteration runtimes of the smooth IRBC model tests are 0.5, 1.6 and 4.2 hours using 100, 200 and 300 nodes, for model dimensions 100, 200 and 300, respectively. For the non-smooth IRBC models, we look at model dimensions of 20, 40, and 60. Compared to the smooth model, the number of grid points required is significantly larger, as we need a much denser grid to capture the strong nonlinearities

TABLE 2
Large-scale results for the smooth and non-smooth IRBC model.

Model Type	d	DDSG Parameters				Grid Points		Euler Error	
		\mathcal{K}	$\epsilon_\eta = \epsilon_\rho$	ℓ	ϵ_γ	DDSG	SG*	Avg.	Max.
smooth	100	1	10^{-4}	4	10^{-3}	8.1×10^2	1.4×10^6	-3.35	-2.21
	200	1	10^{-4}	4	10^{-3}	1.6×10^3	1.1×10^7	-2.95	-2.15
	300	1	10^{-4}	3	10^{-6}	1.5×10^3	3.6×10^7	-2.89	-1.78
non-smooth	20	2	10^{-4}	10	5×10^{-3}	4.3×10^3	1.4×10^9	-2.79	-1.92
	40	2	10^{-4}	10	5×10^{-3}	1.7×10^4	$\gg 10^{10}$	-2.71	-1.98
	60	2	10^{-4}	10	5×10^{-3}	3.1×10^4	$\gg 10^{10}$	-2.84	-1.96

*Average and maximum Euler errors for smooth IRBC models using the DDSG time iteration method. Note that no SG tests could be conducted at such high dimensions. *The reported SG grid points are for comparison purposes and are reported using $\ell = 4$ and 10 for the smooth and non-smooth model, respectively.*

in the policy functions. We can efficiently alleviate this problem by using DDSG with a refinement level of 10. The shortfall of a pure SG becomes apparent when we look at a comparative number of SG grid points at the same refinement level, surpassing trillions of grid points in a 60-dimensional model with level 10. Using adaptive SGs can provide some degree of efficiency in lower-dimensions, for example, [6] show that a 20-dimensional model can be solved using roughly 10^4 grid points. With this said, the DDSG approximation method required more than half the grid points to achieve the same error metrics. Furthermore, in a higher dimension, the number of grid points for both SG and adaptive SG will increase significantly, rendering the model uncomputable on contemporary supercomputers. The per-iteration-runtimes of the kink model are 1.8, 9.9, and 16.4 hours using 38, 156, and 354 nodes, for model dimensions 20, 40 and 60, respectively.

7. Conclusions. We introduced a computational framework to solve large-scale dynamic stochastic economic models on practical time scales. As a secondary benefit, it can serve as an a priori analysis tool to shed light on the model’s complexity. At the core of the methodology is the DDSG function approximation that combines an HDMR technique with adaptive SGs. We parallelized the DDSG method by leveraging the intrinsic separability in the computation, embedded it in a time iteration algorithm, and deployed it on the Cray XC50 system installed at CSCS. Our numerical experiments—that is, solving a set of smooth and non-smooth IRBC models, showed a speedup of 10 times in comparison to state-of-the-art adaptive SGs in cases of mid-scale models, where both methods were applicable. In addition, we showed that even for a relatively small 50-dimensional model, the proposed framework provides excellent strong scaling up 1,000 compute nodes. Furthermore, we demonstrated that we can compute global solutions to IRBC models with at least 300 continuous dimensions in only a few hours. This is a substantial improvement over the previous literature and opens new possibilities for a richer set of model specifications. Finally, note that the scope of the presented method is not restricted to models that are recursively formulated via first-order conditions, but more broadly to high-dimensional models that can be characterized in the functional equation (1.1). The latter also nests the common characterizations of recursive equilibria in discrete time, where p is the value function, and the operator \mathcal{H} captures the Bellman equation it has to satisfy—or the Hamilton-Jacobi-Bellman equation in continuous time [14]. However, tackling such models with the proposed method is subject to further research.

Acknowledgments. We thank Lorenzo Bretscher, Johannes Brumm, Felix Kübler, Philipp Renner, Olaf Schenk, Karl Schmedders, Tony Smith, Fabio Trojani, and seminar participants at the University of Geneva, the University of Lausanne, Università della Svizzera italiana, the Russian Presidential Academy of Science, Stanford University, the University of Zurich for their extremely valuable comments.

REFERENCES

- [1] M. AZINOVIC, L. GAEGAUF, AND S. SCHEIDEGGER, *Deep equilibrium nets*, (2019), <http://dx.doi.org/10.2139/ssrn.3393482>.
- [2] R. E. BELLMAN, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.
- [3] J. BENGUI, E. G. MENDOZA, AND V. QUADRINI, *Capital mobility and international sharing of cyclical risk*, *Journal of Monetary Economics*, 60 (2013), pp. 42–62.
- [4] J. BRUMM, C. KRAUSE, A. SCHAAB, AND S. SCHEIDEGGER, *Sparse grids for dynamic economic models*, (2021), <https://ssrn.com/abstract=3979412>.
- [5] J. BRUMM, D. MIKUSHIN, S. SCHEIDEGGER, AND O. SCHENK, *Scalable high-dimensional dynamic stochastic economic modeling*, *Journal of Computational Science*, 11 (2015), pp. 12 – 25.
- [6] J. BRUMM AND S. SCHEIDEGGER, *Using adaptive sparse grids to solve high-dimensional dynamic models*, *Econometrica*, 85 (2017), pp. 1575–1612.
- [7] H. J. BUNGARTZ AND S. DIRNSTORFER, *Multivariate Quadrature on Adaptive Sparse Grids*, *Computing (Vienna/New York)*, 71 (2003), pp. 89–114.
- [8] H.-J. BUNGARTZ AND M. GRIEBEL, *Sparse grids*, *Acta Numerica*, 13 (2004), pp. 1–123.
- [9] W. J. COLEMAN, *Solving the stochastic growth model by policy-function iteration*, *Journal of Business & Economic Statistics*, 8 (1990), pp. 27–29.
- [10] W. J. DEN HAAN, K. L. JUDD, AND M. JUILLARD, *Computational suite of models with heterogeneous agents ii: Multi-country real business cycle models*, *Journal of Economic Dynamics and Control*, 35 (2011), pp. 175–177.
- [11] W. J. DEN HAAN AND A. MARCET, *Solving the stochastic growth model by parameterizing expectations*, *Journal of Business & Economic Statistics*, 8 (1990), pp. 31–34.
- [12] A. EFTEKHARI, S. SCHEIDEGGER, AND O. SCHENK, *Parallelized dimensional decomposition for large-scale dynamic stochastic economic models*, in *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '17*, New York, NY, USA, 2017, ACM, pp. 9:1–9:11.
- [13] J. FERNÁNDEZ-VILLAVERDE, S. HURTADO, AND G. NUNO, *Financial frictions and the wealth distribution*. Working paper, 2019.
- [14] J. FERNÁNDEZ-VILLAVERDE, J. RUBIO-RAMÍREZ, AND F. SCHORFHEIDE, *Chapter 9 - solution and estimation methods for dsge models*, vol. 2 of *Handbook of Macroeconomics*, Elsevier, 2016, pp. 527–724.
- [15] Z. GAO AND J. S. HESTHAVEN, *On anova expansions and strategies for choosing the anchor point*, *Applied Mathematics and Computation*, 217 (2010), pp. 3274–3285.
- [16] M. HOLTZ, *Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance*, *Lecture Notes in Computational Science and Engineering*, 77 (2010), pp. 1–192.
- [17] G. HOOKER, *Generalized functional anova diagnostics for high-dimensional functions of dependent variables*, *Journal of Computational and Graphical Statistics*, 16 (2007), pp. 709–732.
- [18] G. HOOKER, *Generalized Functional ANOVA Diagnostics for High-Dimensional Functions of Dependent Variables*, *Journal of Computational and Graphical Statistics*, 16 (2007), pp. 709–732.
- [19] S. HSIANG, D. ALLEN, S. ANNAN-PHAN, K. BELL, I. BOLLIGER, T. C. S. CHONG, H. DRUCKENMILLER, L. HUANG, A. HULTGREN, E. KRASOVICH, P. LAU, J. LEE, E. ROLF, J. TSENG, AND T. WU, *The effect of large-scale anti-contagion policies on the covid-19 pandemic*, *Nature*, 585 (2020).
- [20] K. JUDD, *Numerical Methods in Economics*, Scientific and Engineering, MIT Press, 1998.
- [21] K. L. JUDD, *Numerical methods in economics*, vol. 1, The MIT press, 1998.
- [22] M. JUILLARD AND S. VILLEMOT, *Multi-country real business cycle models: Accuracy tests and test bench*, *Journal of Economic Dynamics and Control*, 35 (2011), pp. 178 – 185.
- [23] G. KAPLAN, B. MOLL, AND G. L. VIOLANTE, *Monetary policy according to hank*, *American Economic Review*, 108 (2018), pp. 697–743.
- [24] R. KOLLMANN, S. MALIAR, B. A. MALIN, AND P. PICHLER, *Comparison of solutions to the multi-country Real Business Cycle model*, *Journal of Economic Dynamics and Control*, 35 (2011), pp. 186–202.

- [25] L. KOTLIKOFF, F. KUBLER, A. POLBIN, AND S. SCHEIDEGGER, *Pareto-improving carbon-risk taxation*, *Economic Policy*, 36 (2021), pp. 551–589.
- [26] D. KRUEGER AND F. KUBLER, *Computing equilibrium in OLG models with stochastic production*, *Journal of Economic Dynamics and Control*, 28 (2004), pp. 1411 – 1436.
- [27] D. KRUEGER, K. MITMAN, AND F. PERRI, *Chapter 11 - macroeconomics and household heterogeneity*, vol. 2 of *Handbook of Macroeconomics*, Elsevier, 2016, pp. 843 – 921.
- [28] F. KUBLER AND S. SCHEIDEGGER, *Self-justified equilibria: Existence and computation*, (2019), <https://ssrn.com/abstract=3494876>.
- [29] F. KUBLER AND K. SCHMEDDERS, *Stationary equilibria in asset-pricing models with incomplete markets and collateral*, *Econometrica*, 71 (2003), pp. 1767–1793.
- [30] F. Y. KUO, I. H. SLOAN, G. W. WASILKOWSKI, AND H. WOŹNIAKOWSKI, *On decompositions of multivariate functions*, *Mathematics of Computation*, 79 (2009), pp. 953–966.
- [31] G. LI AND H. RABITZ, *General formulation of HDMR component functions with independent and correlated variables*, *Journal of Mathematical Chemistry*, 50 (2012), pp. 99–130.
- [32] G. LI, C. ROSENTHAL, AND H. RABITZ, *High dimensional model representations*, *The Journal of Physical Chemistry A*, 105 (2001), pp. 7765–7777.
- [33] L. LJUNGQVIST AND T. J. SARGENT, *Recursive macroeconomic theory*, Mit Press, 2004.
- [34] X. MA AND N. ZABARAS, *An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations*, *J. Comput. Phys.*, 228 (2009), pp. 3084–3113.
- [35] X. MA AND N. ZABARAS, *An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations*, *Journal of Computational Physics*, 229 (2010), pp. 3884–3915.
- [36] L. MALIAR AND S. MALIAR, *Chapter 7 - numerical methods for large-scale dynamic economic models*, in *Handbook of Computational Economics Vol. 3*, K. Schmedders and K. L. Judd, eds., vol. 3 of *Handbook of Computational Economics*, Elsevier, 2014, pp. 325 – 477.
- [37] L. MALIAR, S. MALIAR, AND P. WINANT, *Deep learning for solving dynamic economic models.*, *Journal of Monetary Economics*, 122 (2021), pp. 76–101.
- [38] D. MICHIE, “*memo*”*functions and machine learning*, *Nature*, 218 (1968), pp. 19–22.
- [39] V. MINH NGUYEN-THANH, L. TRONG KHIEM NGUYEN, T. RABCZUK, AND X. ZHUANG, *A surrogate model for computational homogenization of elastostatics at finite strain using high-dimensional model representation-based neural network*, *International Journal for Numerical Methods in Engineering*, (2020).
- [40] A. MURARASU, J. WEIDENDORFER, G. BUSE, D. BUTNARU, AND D. PFLÜGER, *Compact data structure and parallel algorithms for the sparse grid technique*, in *16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2011.
- [41] D. PFLÜGER, *Spatially Adaptive Sparse Grids for High-Dimensional Problems*, PhD thesis, München, Aug. 2010.
- [42] D. PFLÜGER, B. PEHERSTORFER, AND H.-J. BUNGARTZ, *Spatially adaptive sparse grids for high-dimensional data-driven problems*, *Journal of Complexity*, 26 (2010), pp. 508 – 522.
- [43] H. RABITZ, Ö. F. ALIŞ, J. SHORTER, AND K. SHIM, *Efficient input-output model representations*, *Computer Physics Communications*, 117 (1999), pp. 11–20.
- [44] H. RABITZ AND Ö. F. ALIŞ, *General foundations of high-dimensional model representations*, *Journal of Mathematical Chemistry*, 25 (1999), pp. 197–233.
- [45] P. RENNER AND S. SCHEIDEGGER, *Machine learning for dynamic incentive problems*, (2018).
- [46] S. SCHEIDEGGER AND I. BILIONIS, *Machine learning for high-dimensional dynamic stochastic economies*, *Journal of Computational Science*, 33 (2019), pp. 68 – 82.
- [47] S. SCHEIDEGGER, D. MIKUSHIN, F. KUBLER, AND O. SCHENK, *Rethinking large-scale economic modeling for efficiency: Optimizations for gpu and xeon phi clusters*, in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2018, pp. 610–619.
- [48] S. SCHEIDEGGER AND A. TRECCANI, *Pricing american options under high-dimensional models with recursive adaptive sparse expectations*, *Journal of Financial Econometrics*, (2018).
- [49] I. M. SOBOL, *Theorems and examples on high dimensional model representation*, *Reliability Engineering & System Safety*, 79 (2003), pp. 187–193.
- [50] N. L. STOKEY, R. E. LUCAS, AND E. C. PRESCOTT, *Recursive Methods in Economic Dynamics*, Harvard University Press, 1989.
- [51] A. WÄCHTER AND L. T. BIEGLER, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, *Math. Program., Ser. A*, (2005).
- [52] X. WANG, *On the approximation error in high dimensional model representation*, *Proceedings of the 2008 Winter Simulation Conference*, (2008), pp. 453–462.
- [53] X. YANG, M. CHOI, G. LIN, AND G. E. KARNIADAKIS, *Adaptive ANOVA decomposition of stochastic incompressible and compressible flows*, *Journal of Computational Physics*, 231 (2012), pp. 1587–1614.