Sparse Grid Discretizations based on a Discontinuous Galerkin Method

Alexander B. Atanasov[1,2] and Erik Schnetter[1,3,4]

[1] *Perimeter Institute for Theoretical Physics, Waterloo, ON, Canada*
[2] *Department of Physics, Yale University, New Haven, CT, USA*
[3] *Department of Physics, University of Guelph, Guelph, ON, Canada*
[4] *Center for Computation & Technology, Louisiana State University, LA, USA*

(Dated: 2017-10-04)

We examine and extend *Sparse Grids* as discretization method for partial differential equations (PDEs). Solving a PDE in D dimensions has a cost that grows as $O(N^D)$ with commonly used methods. Even for moderate D (e.g. $D = 3$), this quickly becomes prohibitively expensive for increasing problem size N. This effect is known as the *Curse of Dimensionality*. Sparse Grids offer an alternative discretization method with a much smaller cost of $O(N \log^{D-1} N)$. In this paper, we introduce Sparse Grids, and extend the method via a Discontinuous Galerkin approach. We then solve the scalar wave equation in $3 + 1$ and $5 + 1$ dimensions, comparing cost and accuracy between full and sparse grids. Sparse Grids perform far superior even in three dimensions. Our code is freely available as open source, and we encourage the reader to reproduce the results we show.

I. INTRODUCTION

It is very costly to accurately discretize functions living in four or more dimensions, and even more expensive to solve partial differential equations (PDE) in such spaces. The reason is clear – if one uses N grid points or basis functions for a one-dimensional discretization, then a straightforward tensor product ansatz leads to $P = N^D$ total grid points or basis functions in D dimensions. For example, doubling the resolution N in the time evolution of a three-dimensional system of PDEs increases the cost by $2^4 = 16$. As a result, such calculations usually require large supercomputers, often run for weeks or months, and one still cannot always reach required accuracies.

Higher-dimensional calculations up to $D = 7$ might be needed to discretize phase space e.g. to model radiation transport in astrophysical scenarios. These are currently only possible in an exploratory manner, i.e. while looking for qualitative results instead of achieving convergence. This unfortunate exponential scaling with the number of problem dimensions is known as the *Curse of Dimensionality* [1].

Sparse grids offer a conceptually elegant alternative. They were originally constructed in 1963 by Smolyak [2], and efficient computational algorithms were then developed in the 1990s by Bungartz, Griebel, Zenger, Zumbusch, and many collaborators. Sparse grids employ a discretization that is not based on tensor products, and can in the best case offer costs that grow only linearly (sic!) or log-linearly with the linear resolution N. At the same time, the accuracy is only slightly reduced by a factor logarithmic in N. See [3, 4] for very readable introductions and pointers to further literature. While we will give a basic introduction to these methods in our paper below, we refer the interested reader to these references for further details.

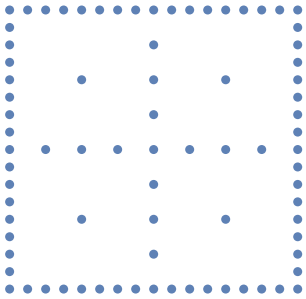

FIG. 1: The collocation points for a particular sparse grid with 4 levels in two dimensions. Note the typical structure which is dense along the edges and fractally sparse in the interior.

Sparse grid algorithms are very well developed. They exist e.g. for arbitrary dimensions, higher-order discretizations, and adaptive mesh refinement, and have been shown to work for various example PDEs, including wave-type and transport-type problems. The respective grid structure has a very characteristic look (it is sparse!) and is fractal in nature. See figure 1 for an example.

Surprisingly, sparse grids are not widely known in the physics community, and are not used in astrophysics or numerical relativity. We conjecture that this is largely due to the non-negligible complexity of the recursive D-dimensional algorithms, and likely also due to the lack of efficient open-source software libraries that allow experimenting with sparse grids. Here, we set out to resolve this:

1. We develop a novel sparse grid discretization method based on Discontinuous Galerkin Finite Elements (DGFE), based on work by Guo and Wang [5, 6]. Most existing sparse grid literature is instead based on finite differencing. DGFE methods are more local, which leads to superior computational efficiency on today's CPUs.

2. We solve the scalar wave equation in $3+1$ and $5+1$ dimensions, comparing resolution, accuracy, and cost to standard ("full grid") discretizations. We demonstrate that sparse grids are indeed vastly more efficient.

3. We introduce a efficient open source library for DGFE sparse grids. This library is implemented in the Julia programming language [7]. All examples and figures in this paper can be verified and modified by the reader.

II. CLASSICAL SPARSE GRIDS

The computational advantage of sparse grids comes from a clever choice of basis functions, which only include important degrees of freedom while ignoring less important ones. The meaning of "important" in the previous sentence is made precise via certain function norms, as described in [3, 4] and references therein. We will omit the motivation and proofs for error bounds here, and will only describe the particular basis functions used in sparse grids as well as the error bounds themselves.

Discretizing a function space $\mathbf{V}$ means choosing a particular set of basis functions $\phi_{l,i}$ for that space, and then truncating the space by using only a finite subset of these basis functions. We assume that the basis functions are ordered by a *level* ℓ, and each level

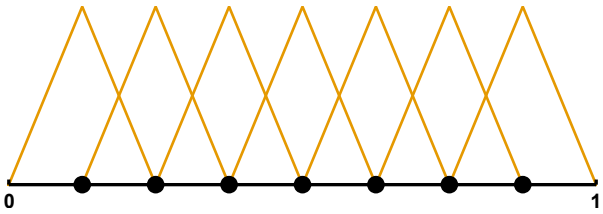

FIG. 2: Nodal basis of hat function with $N = 7$ basis functions.

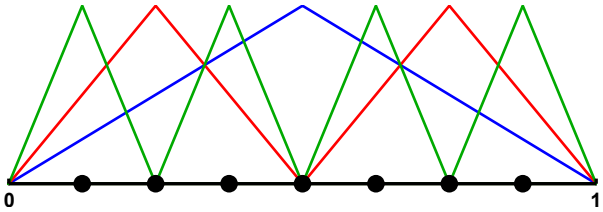

FIG. 3: A hierarchical basis for the same space as before, using levels $\ell = 0, 1, 2$.

contains a set of *modes*, here indexed by i. This *approximates* a function by only considering levels $0 \leq \ell \leq n$, i.e. by dropping all basis function with level $\ell > n$. For example, the real Fourier basis for analytic functions with 2π periodicity consists of $\phi_{k,0}(x) = \cos kx$ and $\phi_{k,1}(x) = \sin kx$. Here, each level k (except $k = 0$) has 2 modes. The notion of a level is important if one changes the accuracy of the approximation: one always includes or excludes all modes within a level. We will need the notion of levels and modes (or nodes) below.

A finite differencing approximation can be defined via a nodal basis. For example, a second-order accurate approximation where the discretization error scales as $O(N^{-2})$ for N basis functions (grid points) uses triangular (hat) functions, as shown in Figure 2, for a piecewise linear continuous ansatz. Level ℓ has 2^ℓ basis functions, so that a discretization with cut-off n has $N = 2^n - 1$ basis functions in total.

The examples given here in the introduction assume homogeneous boundaries for simplicity, i.e. they assume that the function value at the boundary is 0. This restriction can easily be lifted by adding two additional basis functions, as we note in section II A below.

The basis functions for sparse grids in multiple dimensions are defined via a one-dimensional *hierarchical basis*. This hierarchical basis is equivalent to the nodal basis above, in the sense that they span the same space when using the same number of levels n, and one can efficiently convert between a them with cost $O(N \log N)$ without loss of information. Figure 3 shows such a hierarchical basis with 3 levels.

A full set of nodal basis functions in multiple dimensions, corresponding to a grid as used in a finite differencing approximation, can be constructed as tensor product of one-dimensional nodal basis functions. A full set of hierarchical basis functions is likewise constructed as a tensor product.

A. The Hierarchical Sparse Basis

For an in-depth exposition on sparse grids, see [3, 4]. We will follow the conventions of the former. Historically, sparse grids were constructed from a basis of "hat" functions, defined

by taking appropriate shifts and scalings of an initial hat function ϕ:

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

$$\phi_{n,i}(x) = \phi\left(2^n \cdot \left(x - i \cdot 2^{-n}\right)\right). \tag{2}$$

to obtain a basis of functions supported on the unit interval $[0, 1]$.

Higher order discretizations and respective sparse grids are obtained via appropriate piecewise polynomial basis functions. For simplicity, we only discuss second-order accurate discretizations in this section. We will introduce higher order approximations in section III later.

For a fixed level n, we denote this finite-dimensional space in terms of the nodal basis by

$$\mathbf{V}_n := \text{span}\left\{\phi_{n,i},\ 1 \leq i \leq 2^n - 1\right\}. \tag{3}$$

Here n is a positive integer, termed the *level* of resolution, which determines the granularity of the approximation. Each element $\phi_{n,i}$ is localized to within a radius 2^{-n} neighbourhood around the point $i \cdot 2^{-n}$. Here we use the letter i, ranging from 1 to 2^n, to refer to the respective *node* at level n.

For a function $u(x)$ on $[0, 1]$, we can build its representation $\tilde{u}_n$ in this basis by

$$\tilde{u}_n(x) = \sum_{i=1}^{2^n - 1} u_{n,i}\ \phi_{n,i}(x). \tag{4}$$

Here $u_{n,i}$ are the *coefficients* of u in this basis representation on $\mathbf{V}_n$.

We find that $\mathbf{V}_n \subset \mathbf{V}_{n+1}$, and we can pick a decomposition

$$\mathbf{V}_n = \bigoplus_{\ell \leq n} \mathbf{W}_\ell. \tag{5}$$

so that the subspace $\mathbf{W}_\ell$ is complementary to $\mathbf{V}_{\ell-1}$ in $\mathbf{V}_\ell$, and $\mathbf{W}_1 = \mathbf{V}_1$. $\mathbf{W}_\ell$ can also be defined explicitly as

$$\mathbf{W}_\ell := \text{span}\left\{\phi_{\ell,i} : 1 \leq i \leq 2^\ell - 1,\ i \,\text{odd}\right\}. \tag{6}$$

(We write $\phi_{n,i}$ when referring to all nodes i, and $\phi_{\ell,i}$ when referring only to odd i, which are the ones defining level ℓ.)

When accounting for non-vanishing boundary conditions, we add in one more subspace denoted by $\mathbf{W}_0$ and defined to be $\text{span}\{x, 1 - x\}$. Note that these are the two only basis functions that don't vanish on the boundary. For the sake of simplicity, we will ignore this subspace in this section.

A function u can now be represented on $\mathbf{V}_n$ in terms of the $\phi_{\ell,i}$ basis as

$$\tilde{u}_n(x) = \sum_{\ell=0}^{n} \sum_{\substack{i=1 \\ i \text{ odd}}}^{2^\ell - 1} u_{\ell,i}\ \phi_{\ell,i}(x) \tag{7}$$

where the $u_{\ell,i}$ are again the coefficients of $u(x)$ in the hierarchical representation.

This new choice of basis functions, $\phi_{\ell,i}$ with ℓ ranging from 1 to n and i odd, is called the *hierarchical* or *multi-resolution* basis for $\mathbf{V}_n$, because it includes terms from each resolution

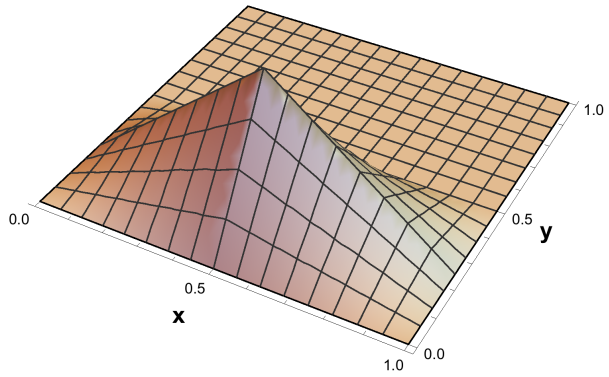

FIG. 4: A tensor product of two hat functions, namely the one at level $\ell = 0$ along x and one of the two at level $\ell = 1,\ i = 1$ along y. This corresponds to $\phi_{\mathbf{l}=(1,2),\,\mathbf{i}=(1,1)}$

level. This basis is especially convenient when one wants to increase the resolution (i.e. the number of levels n), as one does not have to recalculate existing coefficients: Adding new levels with all coefficients set to 0 does not change the represented function.

In higher dimensions, the basis functions are obtained through the usual tensor product construction [3]. Now, the multi-resolution basis for D dimensions consists of a set of basis functions $\phi_{\mathbf{l},\mathbf{i}}$ indexed by a D-tuple, $\mathbf{l}$, of levels, called a *multi-level*, and a second tuple of elements $\mathbf{i}$, called a *multi-node*. The tensor product construction defines

$$\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \prod_{d=1}^{D} \phi_{\mathbf{l}_d,\mathbf{i}_d}(\mathbf{x}_d), \tag{8}$$

$$\mathbf{W}_\mathbf{l} := \bigotimes_{d=1}^{D} \mathbf{W}_{\mathbf{l}_d} \tag{9}$$

$$\Rightarrow \mathbf{W}_\mathbf{l} = \operatorname{span}\left\{\phi_{\mathbf{l},\mathbf{i}} : 1 \le \mathbf{i}_j \le 2^{\mathbf{l}_j},\ \mathbf{i}_d \text{ odd for all } d\right\} \tag{10}$$

Figure 4 shows the two-dimensional basis function $\phi_{\mathbf{l}=(1,2),\mathbf{i}=(1,1)}$ as example. We write $\mathbf{k} \le \mathbf{l}$ if the inequality holds for all components, i.e. $\forall d\ \mathbf{k}_d \le \mathbf{l}_d$. Then,

$$\mathbf{V}_\mathbf{l} := \bigotimes_{d=1}^{D} \mathbf{V}_{\mathbf{l}_j} = \bigoplus_{\mathbf{k}\le\mathbf{l}} \mathbf{W}_\mathbf{k} \tag{11}$$

Often in numerical approximation, the maximum level along each axis is the same number n, i.e. $\mathbf{l} = (n, n, \ldots, n)$. In this case we write

$$\mathbf{V}_n^D := \mathbf{V}_{(n,n,\ldots,n)} = \bigoplus_{|\mathbf{k}|_\infty \le n} \mathbf{W}_\mathbf{k} \tag{12}$$

We then define the sparse subspace$\hat{\mathbf{V}}_n^D \subseteq \mathbf{V}_n^D$ in terms of the hierarchical subspaces $\mathbf{W}_\mathbf{k}$.

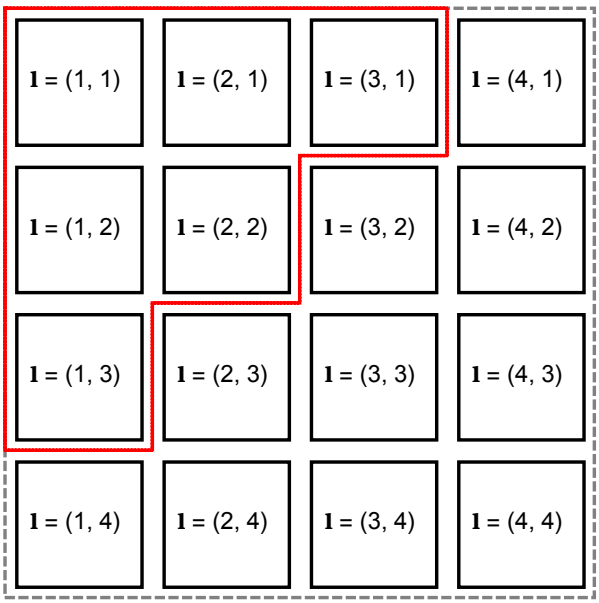

FIG. 5: Illustration of the "sparsification" of a set of full grid coefficients for $D = 2,\ n = 4$ by removing all multi-levels past a certain 1-norm. The dashed outline denotes the original multi-levels included in the full space, while the red line denotes only those included in the sparse space. Note that levels with higher indices contain exponentially more basis functions, so that the resulting reduction in dimensionality is significant, from $O(N^D)$ to $O(N \log^{D-1} N)$, as elucidated in the main text in section II B.

Rather than taking a direct sum over all the multi-levels $\mathbf{k}$, we define[1]

$$\hat{\mathbf{V}}_n^D := \bigoplus_{|\mathbf{k}|_1 \leq n} \mathbf{W}_\mathbf{k} \tag{13}$$

to be the sparse space. Note that we use here the 1-norm instead of the ∞-norm to decide which multi-level basis indices to include. This selects only $1/d!$ of the possible multi-levels. This subspace selection is illustrated in Figure 5 for $d = 2,\ n = 3$.

Because different levels have a different numbers of basis functions, and this process cuts off higher levels that have exponentially more basis functions than the lower ones, the resulting dimensionality reduction is significant. We discuss this in detail in the next subsection.

B. Comparing Costs of Classical Grids

To construct a representation of a function living in $\mathbf{V}_n^D$, 2^n basis coefficients must be determined along each axis. This number is often referred to as the number of collocation points or grid points along that dimension, and we denote it by N. Its inverse $h = 1/N$ is called the *resolution*.

[1] Note this is a slightly definition from that used in [3], namely $|\mathbf{k}|_1 \leq n + D - 1$. Our definition makes the direct sum considerably cleaner and translates more nicely to the DG case later.

In the full case, the dimension of $\mathbf{V}_n^D$ is the number of coefficients that must be stored for a representation up to level n along each axis, and it is straightforward to see that we can write this as

$$\dim \mathbf{V}_n^D = O(2^{Dn}) = O(N^D). \tag{14}$$

This exponential dependence of the size of this space on the number of spatial dimensions D is the manifestation of the curse of dimensionality. On the other hand, the dimension of the sparse space is

$$\dim \hat{\mathbf{V}}_n^D = \sum_{|\mathbf{k}|_1 \leq n} 2^{|\mathbf{k}|_1} = O(2^n\, n^{D-1}) = O(N \log^{D-1} N). \tag{15}$$

The sparse space consists of far fewer coefficients, and suppresses the exponential dependence on D to only the logarithmic term $\log N$.

Moreover, [3, 8] show that the two-norm of the error of an approximation $\tilde{u}_{\text{full}} \in \mathbf{V}_n^D$ of a function $u \in H_2([0,1]^D)$ in the Sobolev space scales as

$$||u(\mathbf{x}) - \tilde{u}_{\text{full}}(\mathbf{x})||_2 = O(N^{-2}) \tag{16}$$

while for $\tilde{u}_{\text{sparse}}$ in the corresponding sparse space, it is

$$||u(\mathbf{x}) - \tilde{u}_{\text{sparse}}(\mathbf{x})||_2 = O(N^{-2} \log^{D-1} N). \tag{17}$$

This means that the error increases only by a logarithmic factor, showing the significant advantage of the sparse basis over the full basis for all dimensions greater than one.

We define the *ε-complexity* [3, 9] of a scheme as the error bound ε that can be reached with a given number of coefficients P. A full grid has an ε-complexity in the L_2 norm of

$$\varepsilon_{L_2}^{\text{full}}(P) = O(P^{-2/D}) \tag{18}$$

which decreases quite slowly for larger dimensions. The sparse grid, on the other hand, does considerably better with

$$\varepsilon_{L_2}^{\text{sparse}}(P) = O(P^{-2} \log^{3(D-1)} P). \tag{19}$$

Though the logarithmic factor is significant enough that it cannot be ignored in practice, even in high dimensions this still represents a significant reduction in the number of coefficients required.

This can be generalized to higher-order finite differencing and corresponding hierarchical grids. In this case, the power P^{-2} improves to P^{-k} correspondingly for both full and sparse grids.

III. THE DISCONTINUOUS GALERKIN SPARSE BASIS

We review here the work of [5] and [6] on how one can generalize this sparse grid construction beyond hat functions or their higher-order finite-differencing (FD) equivalents. (Although we only discussed second-order accurate sparse grids in the previous section, these can be extended to arbitrary orders in a straightforward manner.) Then we introduce a scheme for defining a derivative operator in a manner similar to the Operator-Based Local Discontinuous Galerkin (OLDG) Method developed in [10].

The main advantage of DGFE sparse grids over classical, finite-differencing based sparse grids is the same as the advantage that regular DGFE discretizations possess over finite-difference discretization: (1) For high polynomial orders (say, $k > 8$), the collocation points within DGFE elements can be clustered near the element boundaries, leading to increased stability and accuracy near domain boundaries. (2) The derivative operators for DGFE discretizations exhibit a significant block structure, which is computationally more efficient than the band structure found in higher-order finite differencing operators. (3) Basis functions are localized in space, which allows *hp* adaptivity. Corresponding Adaptive Mesh Refinement (AMR) algorithms for FD discretization do not allow adapting the polynomial order. However, DGFE methods also have certain disadvantages. In particular, the Courant-Friedrichs-Lewy (CFL) condition is typically more strict, intuitively because the collocation points cluster near element boundaries.

A. Constructing Galerkin Bases

The space $\mathbf{V}_n$ of hat-functions in one dimension is equivalent to the span of 2^n hat functions defined on sub-intervals of length 2^{-n} that subdivide $[0, 1]$ into 2^n intervals of the form $I_{i,n} := [(i-1)\cdot 2^{-n},\, i\cdot 2^{-n}]$. A natural generalization of this is to consider the linear sum of spaces $P_k(I_{i,n})$ of functions that, when restricted to $I_{i,n}$ become polynomials of degree less than k, and are zero everywhere outside.

This leads to a space of basis functions on $[0, 1]$ called $\mathbf{V}_{n,k}$, defined as:

$$\mathbf{V}_{n,k} := \bigoplus_{i=1}^{2^n} P_k(I_{i,n}). \tag{20}$$

Note that $\dim \mathbf{V}_{n,k} = k\cdot 2^n$. The analogue of the modal basis above is the following collection of basis vectors

$$\{h_{n,i,m} : 1 \le i \le 2^n,\ 1 \le m \le k\} \tag{21}$$

So that for a given i, each $h_{n,i,m}$ is supported only on the interval $I_{i,n}$ from before, and together they constitute a basis of k orthonormal polynomials of degree less than k on that interval. In this case, $h_{n,i,m}$ can be identified as an appropriate shift and scale of the m-th Legendre polynomial with support only on $[-1, 1]$. We then obtain the orthonormal basis:

$$h_{n,i,m}(x) := \begin{cases} 2^{(n+1)/2}\, P_m\left(2^n \cdot (x - i\cdot 2^{-n})\right) & \text{if } x \in I_{i,n} \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

As before, $\mathbf{V}_{n-1,k} \subset \mathbf{V}_{n,k}$, and so we can again construct a hierarchical decomposition by choosing a complement to $\mathbf{V}_{n-1,k}$ in $\mathbf{V}_{n,k}$. In fact, we can obtain this decomposition canonically by using an inner product on the space of piecewise polynomial functions. For f, g in this space, define:

$$\langle f, g\rangle := \int_{[0,1]} f(x)\, g(x)\, dx. \tag{23}$$

Then letting $\mathbf{W}_{n,k}$ be the orthogonal complement to $\mathbf{V}_{n-1,k}$ in $\mathbf{V}_{n,k}$, we obtain

$$\mathbf{V}_{n,k} = \mathbf{V}_{n-1,k} \oplus \mathbf{W}_{n,k}, \qquad \mathbf{W}_{n,k} \perp \mathbf{V}_{n-1,k}. \tag{24}$$

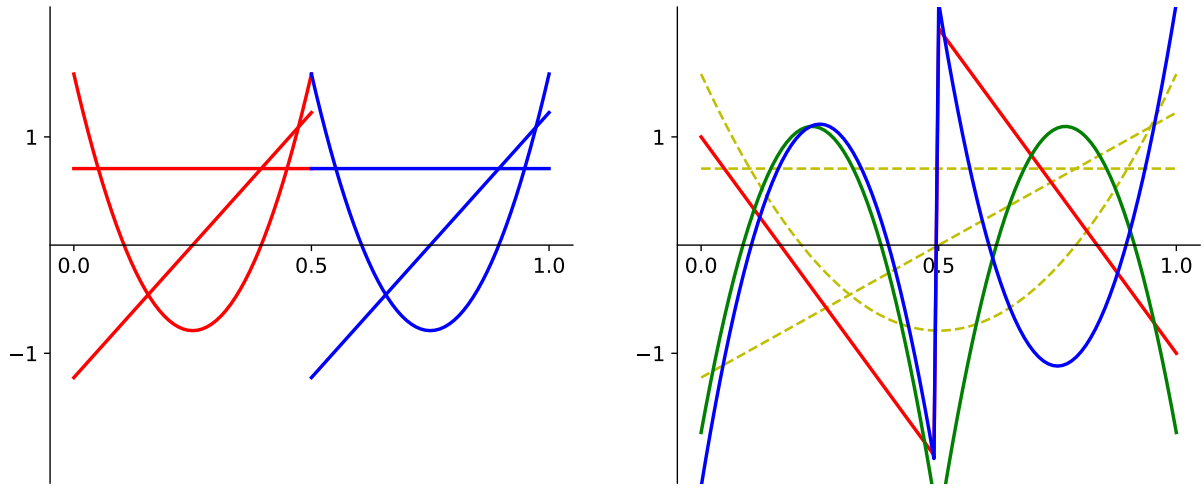

FIG. 6: Illustration of two different orthogonal bases for resolving the two half intervals of $[0, 1]$ (corresponding to resolution level one). On the left are piecewise Legendre polynomials for the original discontinuous Galerkin (DG) basis. On the right are the zeroth and first levels of the multi-resolution basis as $k = 3$. Level zero modes consist of Legendre polynomials, shown as dashed yellow lines. Higher levels are shifts/scalings to the appropriate subintervals of the three level-one basis modes drawn in red, green, and blue. This construction is comparable to h-refinement in Adaptive Mesh Refinemend (AMR) schemes.

From this we can construct a hierarchical orthonormal basis of functions for the entire space $\mathbf{V}_{n,k}$:

$$v_{\ell,i,m}(x) \qquad 0 \leq \ell \leq n,\ 1 \leq i \leq 2^l,\ 1 \leq m \leq k. \tag{25}$$

Here ℓ is the level and i is the element at level l. m is called the *mode* and indexes the k orthogonal polynomials on the subinterval corresponding to (l, i). For a more explicit construction of this basis, see [6, 11].[2]

Figure 6 illustrates resulting polynomials for $k = 3$. Unlike in the FD basis before, we now allow for non-vanishing boundary values, and the level now ranges from 0 to n rather than 1 to n. (Level 0 in the FD basis corresponds to basis functions that are non-vanishing on the boundary. Since these form a special case, we do not describe them here for simplicity.)

In the D-dimensional case, we progress as before to apply the tensor product construction with the conventions of the previous Section II A:

$$v_{\mathbf{l},\mathbf{i},\mathbf{m}}(\mathbf{x}) = \prod_{d=1}^{D} v_{\mathbf{l}_d,\mathbf{i}_d,\mathbf{m}_d}(\mathbf{x}_d), \tag{26}$$

$$\mathbf{W}_{\mathbf{l}}^{k} = \bigotimes_{d=1}^{D} \mathbf{W}_{\mathbf{l}_d}^{k}. \tag{27}$$

The basis $v_{\mathbf{l},\mathbf{i},\mathbf{j}}$ spans the full discontinuous Galerkin grid space in D dimensions. As before, $\mathbf{l}, \mathbf{i}$ are D-tuples of levels and elements called the multi-level and multi-element. $\mathbf{m}$ is a D-tuple

[2] It is unfortunate that the notation used in the DGFE community and the notation used in [6, 11] differ. Here, we follow the notation of [6, 11]. In DGFE notation, elements would be labelled k, modes labelled i, and the maximum polynomial order would be labelled p.

of modes called a *multi-mode*. We can write this space in terms of its level decomposition as:

$$\mathbf{V}_{n,k}^{D} = \bigoplus_{|\mathbf{l}|_\infty \leq n} \mathbf{W}_{\mathbf{l}}^{k}. \tag{28}$$

The corresponding sparse space is then defined as before [5, 6] by using the 1-norm to select only a subset of basis functions:

$$\hat{\mathbf{V}}_{n,k}^{D} = \bigoplus_{|\mathbf{l}|_1 \leq n} \mathbf{W}_{\mathbf{l}}^{k}. \tag{29}$$

We will use this space in our tests and examples to efficiently construct solutions for hyperbolic PDEs in the next section.

B. Comparing Costs of Galerkin Grids

Analogous to Equations (14) and (15) in section II B, we describe here the cost (number of basis functions) and errors associated with full and sparse DG spaces.

$$\dim \mathbf{V}_{n,k}^{D} = O(2^{Dn}\, k^{D}) = O(N^{D}\, k^{D}) \tag{30}$$

$$\dim \hat{\mathbf{V}}_{n,k}^{D} = O(2^{n}\, k^{D}\, n^{D-1}) = O(N\, k^{D}\, \log^{D-1} N). \tag{31}$$

This shows that a sparse grid reduces the cost significantly as the resolution N is increased, while our construction has no effect on the way the cost depends on the polynomial order k. These cost functions omit large constant factors that depend on the number of dimensions D.

Further, from [12] we obtain the DGFE analogues of Equations (16) and (17) for the L^2-norm of the approximation error:

$$||u(\mathbf{x}) - \tilde{u}_{\text{full}}(\mathbf{x})||_2 = O(N^{-k}) \tag{32}$$

$$||u(\mathbf{x}) - \tilde{u}_{\text{sparse}}(\mathbf{x})||_2 = O(N^{-k}\, \log^{D-1} N). \tag{33}$$

which is equivalent to (16) and (17), except that we now achieve a higher convergence order k instead of just 2: The error found in sparse grids is only slightly (by a logarithmic factor) larger than that in full grids.

This yields the resulting ε-complexities:

$$\varepsilon_{L^2}^{\text{full DG}}(P) = O\left(P^{-k/D}\right) \tag{34}$$

$$\varepsilon_{L^2}^{\text{sparse DG}}(P) = O\left(P^{-k}\, k^{Dk}\, \log^{(k+1)(D-1)} P\right) \tag{35}$$

This is consistent with an analogous estimate in [3]. Again, the sparse grid DG case provides a significant improvement over the full grid DG case. We compare these scalings graphically in figure 7 for $D = 3$.

C. Constructing the Derivative Operator

To solve hyperbolic PDEs, we must define a derivative operator on the multi-dimensional sparse basis. To do this, we proceed in steps:

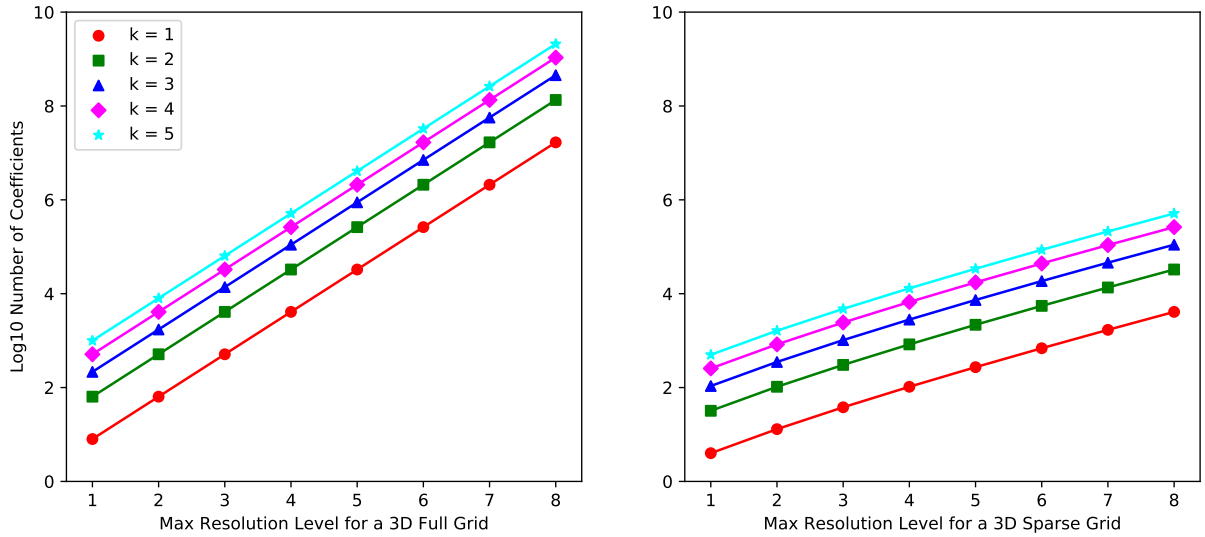

FIG. 7: The number of coefficients vs. the maximum level, comparing full (left) and sparse (right) grids. This shows the storage (memory) cost, on a log scale, vs. the maximum level of resolution n of a degree k Galerkin scheme in three dimensions. Shown for both the full and sparse spaces, i.e. $\dim \mathbf{V}_{n,k}$ and $\dim \hat{\mathbf{V}}_n^k$ for $D = 3$. The reduction in cost is evident. The slight downward curvature in the right figure is caused by the logarithmic term, which remains relevant in practice for the shown number of levels.

1. Define the derivative on the one-dimensional Galerkin space $\mathbf{V}_{n,k}$ in terms of the modal basis of Legendre polynomials.

2. Express this operator in the one-dimensional hierarchical basis.

3. Using the tensor-product construction, define the derivative matrix elements in the full multi-dimensional basis $\mathbf{V}_{n,k}^D$.

4. Restrict this operator to the sparse subspace, i.e. only calculate matrix elements between basis functions in $\hat{\mathbf{V}}_{n,k}^D$, discarding those matrix elements that lead out of this subspace. This step introduces an additional approximation into our scheme.

To define the derivative in the standard Galerkin basis in one dimension, we follow the methods outlined in a prior paper [10]. For a given maximum number of levels n and degree k, let $h_{n,i,m}$ be as before.

The derivative operator is then defined in the weak sense:

$$\begin{aligned}\int_{[0,1]} h_{n,i,m}(x)\,\frac{df}{dx}(x)\,dx &= \int_{I_{i,n}} h_{n,i,m}(x)\,\frac{df}{dx}(x)\,dx \\ &= \Big[h_{n,i,m}(x)\,f(x)\Big]_{(i-1)2^{-n}}^{i2^{-n}} - \int_{I_{i,n}} f(x)\,\frac{dh_{n,i,m}}{dx}(x)\,dx\end{aligned} \tag{36}$$

Because the $h_{n,i,m}$ are discontinuous at the two boundary points, where they jump from zero to nonzero values, we define $h_{n,i,m}((i-1)2^{-n})$ and $h_{n,i,m}(i2^{-n})$ to be the average of the two, namely half the endpoint value, following [13, 14]. [15] contains also a very good description of the weak formulation for derivatives, although it considers only elliptic systems.

This choice of the average corresponds to a central flux. The same choice was made in [10], where the authors argue that this is a convenient "black box" choice that works for many systems of strongly hyperbolic PDEs, and is not restricted to manifestly flux-conservative formulations. Another reason for this choice here is that it results in a linear operator, which allows formulating and applying the derivative operator in modal space (i.e. on individual modes) instead of having to reconstruct the function values at element interfaces. The latter would be quite complex to define, as finer levels have element interfaces at locations in the interior of coarser levels (see the right subfigure in figure 6), and – different from standard DG methods – in a hierarchical basis all levels are "active" are contribute to the representation simultaneously.

The boundary term then becomes:

$$\frac{1}{2}\lim_{\epsilon\to 0^+}\left[f\left(i2^{-n}-\epsilon\right)\,h_{n,i,m}\left(i2^{-n}-\epsilon\right)-f\left((i-1)2^{-n}+\epsilon\right)\,h_{n,i,m}\left((i-1)2^{-n}+\epsilon\right)\right] \tag{37}$$

while in the second integral, $h_{n,i,m}$ is treated exactly as a polynomial function. Its derivative is then exactly computable. By knowing the basis representation of f in terms of $h_{n,i,m}$, we can obtain an explicit matrix form of the derivative operator via

$$\mathcal{D}_{i,m;\,i',m'} := \int_{[0,1]} h_{n,i,m}(x)\,\frac{dh_{n,i',m'}}{dx}(x)\,dx. \tag{38}$$

This matrix will be a sum of two matrices, as above: One term corresponding to the discontinuity terms at the boundary, and the second term corresponding to the regular derivative operator on the smooth polynomial functions in the interior.

By orthogonally transforming $h_{n,i,j}$ into the hierarchical Galerkin basis $v_{\ell,i,j}$ through a transformation Q, we can conjugate this derivative operator into its hierarchical form $\mathcal{D}^{\text{hier}} = QDQ^T$. We generalize this operator to higher dimensions through the standard tensor product construction. The partial derivative operator ∂_a, $1 \le a \le d$, is represented by the derivative operator $\mathcal{D}_a^{\text{full}}$ acting on the basis $v_{\mathbf{l},\mathbf{i},\mathbf{j}}$ as

$$\mathcal{D}_a^{\text{full}}(v_{\mathbf{l},\mathbf{i},\mathbf{j}}) = \left[\mathcal{D}^{\text{hier}} v_{\mathbf{l}_a,\mathbf{i}_a,\mathbf{j}_a}\right]\prod_{m\neq a} v_{\mathbf{l}_m,\mathbf{i}_m,\mathbf{j}_m}. \tag{39}$$

The last step is to restrict this full derivative operator to the sparse space. We obtain $\mathcal{D}_a^{\text{sparse}}$ from $\mathcal{D}_a^{\text{full}}$ by restricting the latter to the sparse basis space. Note, however, that $\mathcal{D}_a^{\text{full}}$ does not preserve the sparse space, as it has nonzero matrix entries mapping sparse to non-sparse basis elements. (This is also the case for the 1D derivative operator $\mathcal{D}^{\text{hier}}$, which doesn't stabilize $\mathbf{V}_{n,k} \subset \mathbf{V}_{n+1,k}$. When restricting the derivative matrix to $\mathbf{V}_{n,k}$, we ignore those components of the operator that map outside of $\mathbf{V}_{n,k}$.) We shall do the same here, and define

$$\mathcal{D}_a^{\text{sparse}} = \text{Proj}_{\hat{\mathbf{V}}_{n,k}^d}\,\mathcal{D}_a^{\text{full}}. \tag{40}$$

This will be the working definition for our derivative operator in the sparse basis.

Further, the matrix representing this derivative operator is itself sparse. This is clear upon noting that $\mathcal{D}_a^{\text{full}}$ only acts on the a-indexed components of the basis $v_{\mathbf{l},\mathbf{i},\mathbf{j}}$. Thus $\mathcal{D}_a^{\text{sparse}}$ has a nonzero matrix entry exactly when $\mathcal{D}^{\text{hier}}$ does on the one-dimensional basis along the ath axis: $v_{\mathbf{l}_a,\mathbf{i}_a,\mathbf{j}_a}$. Looking at $\mathcal{D}_a^{\text{hier}}$ in one dimension, since a given element at (ℓ, i) will only have nonzero derivative entry against basis vectors supported either on itself, its

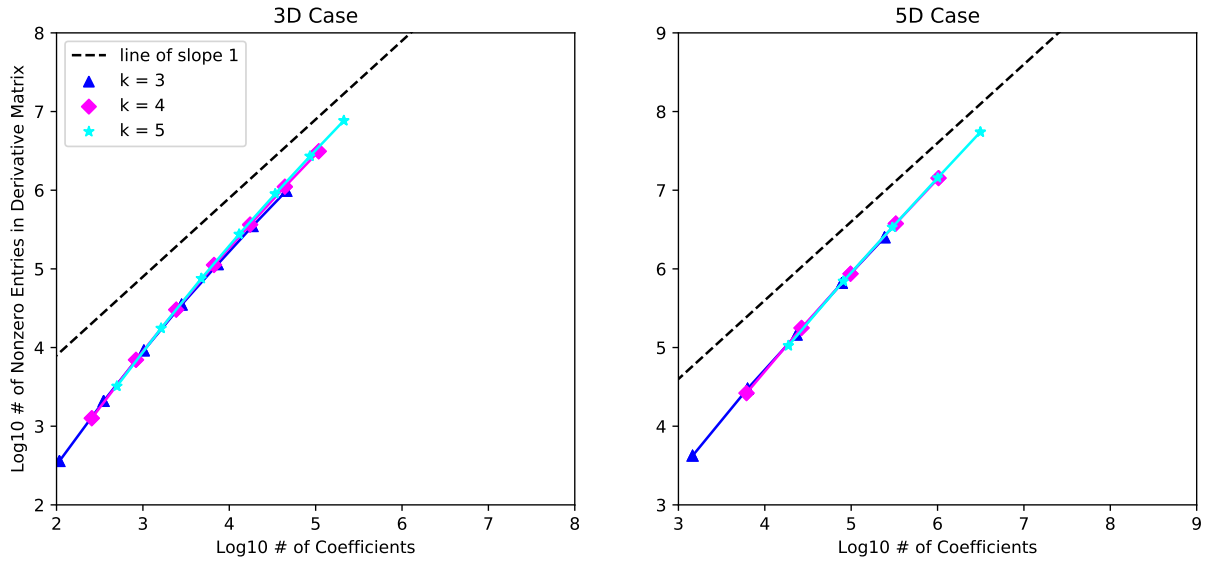

FIG. 8: This figure illustrates that the number of nonzero coefficients in the sparse derivative matrix scales as $P \log P$ where P is the total number of coefficients stored at a given resolution and order. Note that the slope of the graphs in this log-log plot tends to 1 as the number of coefficient grows, indicating a scaling that is much closer to $O(N)$ than $O(N^2)$. This indicates that the derivative matrix is sparse, confirming our $O(N \log N)$ estimate (see main text).

neighbours, or elements that contain it, this gives a total of $O(N \log N)$ nonzero entries for the derivative matrix. This factor is amplified to $O(P \log P)$ in the D-dimensional case, and this log-linearity is numerically verified in Figure 8. This sparse structure is crucially important when solving PDEs, as it determines the run-time cost of the algorithm.

For PDEs with periodic boundary conditions, the 1D derivative operator, from which the full and sparse derivatives are constructed, will have the boundary terms in the integration by parts (36) overlap at the opposite endpoints 0 and 1. This turns the interval $[0, 1]$ into the torus T^1.

IV. EVOLVING A SCALAR WAVE IN THE DGFE BASIS

Using the sparse derivative operator that we have constructed, we turn to solving the scalar wave equation (in a flat spacetime) as example and test case. This equation can be reduced to first order in time as

$$\begin{aligned} \dot{\varphi}(\mathbf{x}, t) &= \psi(\mathbf{x}, t) \\ \dot{\psi}(\mathbf{x}, t) &= \nabla^2 \varphi(\mathbf{x}, t) \end{aligned} \quad \Rightarrow \quad \frac{d}{dt} \begin{pmatrix} \varphi \\ \psi \end{pmatrix} = \begin{pmatrix} 0 & \mathbf{1} \\ \nabla^2 & 0 \end{pmatrix} \begin{pmatrix} \varphi \\ \psi \end{pmatrix}. \tag{41}$$

We then use the DGFE basis to represent this as

$$\varphi(\mathbf{x}, t) = \sum_{\mathbf{l},\mathbf{i},\mathbf{j}} \varphi_{\mathbf{l},\mathbf{i},\mathbf{j}}(t)\, v_{\mathbf{l},\mathbf{i},\mathbf{j}}(\mathbf{x}), \qquad \psi(\mathbf{x}, t) = \sum_{\mathbf{l},\mathbf{i},\mathbf{j}} \psi_{\mathbf{l},\mathbf{i},\mathbf{j}}(t)\, v_{\mathbf{l},\mathbf{i},\mathbf{j}}(\mathbf{x}), \tag{42}$$

$$\nabla^2 \to \sum_a \mathcal{D}_a^{\text{sparse}} \cdot \mathcal{D}_a^{\text{sparse}}. \tag{43}$$

As written, this representation is exact – it only corresponds to a particular basis choice. We then introduce a cut-off for the number of levels n, which introduces an approximation error.

Here, $\varphi_{\mathbf{l},\mathbf{i},\mathbf{j}}$ and $\psi_{\mathbf{l},\mathbf{i},\mathbf{j}}$ are the coefficients representing φ, ψ. The problem is now framed as a set of coupled first-order ordinary differential equations (ODEs). Such ODEs can be solved in a straightforward manner via a wide range of ODE integrators, e.g. of the Runge-Kutta family.

Since the derivative and Laplacian operators are sparse matrices, and since the maximum allowable time step scales with the resolution h and thus with the linear problem size N due to the Courant-Friedrichs-Lewy (CFL) criterion, the overall cost to calculate a solution scales as $O(N^2 \log^{D-1} N)$, which is significantly more efficient than the $O(N^{D+1})$ scaling for a full grid. Here D is the number of spatial dimensions. In principle it is possible to employ a sparse space-time discretisation as well, reducing the cost further to $O(N \log^D N)$. However, this will require a more complex time evolution algorithm instead of the ODE solver we employ, and we thus do not investigate this approach in this paper.

Initial conditions for this time evolution are given by the coefficients of the initial position and velocity of the wave φ_0, ψ_0. This requires projecting given functions into the sparse space. (Later, time evolution proceeds entirely in coefficient space.) Accurately projecting onto the basis functions requires (in practice) a numerical integration for each coefficient. If one is satisfied with the accuracy and convergence properties of sparse grids, then one can replace this numerical integration with sampling the given function at the collocation points of the DG basis functions, as is often done. However, we do not want to make such assumptions in this paper, and thus use a (more accurate, and more expensive) numerical integration to properly project the initial conditions onto the basis functions.

It is advantageous (the calculation is simpler) if the initial conditions can be cast as sums of tensor products of 1D functions. By projecting these individual components of the solution in 1D, and then applying the tensor construction to those 1D coefficients, we can recover the coefficients of the full D-dimensional function. This yields a considerable speedup of the initial data construction, as all integrations only need to be performed in 1D.

Such a construction can in particular be applied to all sine waves of the form $\sin(\mathbf{k}\cdot\mathbf{x}-\phi)$ by expressing them as sums of products of sines and cosines in one variable. In the general case, a sparse integrator (e.g. based on sparse grids!) could be used to avoid concerns with the cost of high-dimensional integration. As mentioned above, out of caution we avoid this since we do not want to use sparse methods to test our sparse discretization – we want to test via non-sparse methods only.

We present results in section VI below, after briefly introducing the Julia package we developed to perform these calculations.

V. OVERVIEW OF THE OPEN-SOURCE PACKAGE

The Julia code for performing interpolation and wave evolution using the sparse Galerkin bases outlined in this paper is publicly available as open source under the "MIT" licence at

https://github.com/ABAtanasov/GalerkinSparseGrids.jl

To obtain the basis coefficients resulting from projecting a function `f` in `D` dimensions at degree `k` and level `n`, we use the command `coeffs_DG` as follows:

```
using GalerkinSparseGrids
D = 2; k = 3; n = 5;
f = x -> sin(2*pi*x[1])*sin(2*pi*x[2])
full_coeffs   = coeffs_DG(D, k, n, f; scheme="full"  )
sparse_coeffs = coeffs_DG(D, k, n, f; scheme="sparse")
```

Obviously, calling `full_coeffs` is much more expensive than `sparse_coeffs`, and should only be used to compare cost and accuracies.

This creates a dictionary-like data structure for efficient access to coefficients at the various levels and elements. The coefficients at each level/element are then stored as dense array for efficiency. Given such a dictionary of coefficients, we can evaluate the function at a `D`-dimensional point `x` using `reconstruct_DG`.

```
full_val   = reconstruct_DG(full_coeffs  , x)
sparse_val = reconstruct_DG(sparse_coeffs, x)
```

The explicit function's representation can then be written (given access to the `sparse_coeffs`) as:

```
f_rep = x -> reconstruct_DG(sparse_coeffs, [x...])
```

We also include a Monte-Carlo error function `mcerr` for a quick calculation of the error between the exact and the computed function

```
mcerr(f::Function, g::Function, D::Int; count = 1000)
```

This provides a good estimate the L^2 norm of the error between f and g. The data points in this paper representing errors were calculated using this method. For example, to generate one of the data points in Figure 9, one uses

```
sparse_coeffs = coeffs_DG(D, k, n, f)
f_rep = x -> reconstruct_DG(sparse_coeffs, [x...])
err = mcerr(f, f_rep, D)
```

For algorithms such as time evolution and differentiation, the dictionary-like coefficient tables are converted into vectors (one-dimensional arrays), so that the derivative operators act as multiplication by a sparse matrix, and all operations can be performed efficiently. To convert a dictionary `dict` of coefficient tables to a single coefficient vector `vcoeffs` (or vice versa) we use

```
vcoeffs = D2V(D, k, n, dict; scheme="sparse")
dict    = V2D(D, k, n, vcoeffs; scheme="sparse")
```

Given a vector of coefficients, one can easily compute the L^2 norm of the representation (since the basis is orthonormal) by using the built-in Julia function

```
L2_norm = norm(vcoeffs)
```

Explicitly, one can obtain the derivative operator matrix $\mathcal{D}_a$ as:

```
D_matrix(D::Int, a::Int, k::Int, n::Int; scheme="sparse")
```

The full list of $\mathcal{D}_a, a \in [1, D]$ matrices can be computed through:

```
grad_matrix(D::Int, k::Int, n::Int; scheme="sparse")
```

The corresponding Laplacian operator is the sum of the squares of this list, and is given by

```
laplacian_matrix(D::Int, k::Int, n::Int; scheme="sparse")
```

All of these operators are meant to act on the appropriate vectors of coefficients.

To solve the wave equation given initial conditions, we provide the following function:

```
wave_evolve(D::Int, k::Int, n::Int,
                f0::Function, v0::Function,
                time0::Real, time1::Real;
                order="45", scheme="sparse")
```

Using the representation of the initial data and Laplacian operator given in Equations (42) and (43), this represents the D-dimensional wave equation with initial conditions `f0` for φ and `v0` for ψ by a set of linearly coupled first order ODEs. It is straightforward to modify this code to solve different systems of equations. To support non-linear systems, one needs to introduce collocation points for the DG modes and perform the calculation in the respective nodal basis. While this is not yet implemented, it is well-known how to do this, as is e.g. done in [10].

This system can then be evolved by any integrator in the various Julia packages for ordinary differential equations. In the present package, we chose to use `ode45` and `ode78` from the `ODE.jl` package [16].

For example, to solve a standing wave in 2D from time `t_0` to time `t_1` with polynomials of degree less than `k` on each element, using a sparse depth `n`, and with the `ode78` ODE integrator:

```
D = 2; k = 5; n = 5;
f0 = x->sin(2*pi*x[1])*sin(2*pi*x[2])
v0 = 0
soln = wave_evolve(D, k, n, f0, v0, 0, 0.25; order="78")
```

The array structure of `soln` is like any solution given by the ODE.jl package. `soln` will have three components. The first is an array of times at which the timestep was performed. The second is an array of coefficients corresponding to the evolution of the initial data `f0` at those time points. The third is an array of coefficients corresponding to the evolution of the initial data `v0`.

For example, to calculate the final error vs. the exact solution for time-evolving the standing sine wave from before, we write

```
f_exact = x -> cos(2*pi*x[1])*sin(2*pi*x[2])
f_rep = x -> reconstruct_DG(soln[2][end], [x...])
err = mcerr(f_exact, f_rep, D)
```

In the case that the user wants to evolve specifically a travelling wave, we provide the function

```
travelling_wave_solver(k::Int, n::Int, m::Array{Int,1},
                           time0::Real, time1::Real;
                           scheme="sparse", phase=0.0, A=1.0, order="45")
```

to evolve a travelling wave with wave number `2*pi*m`, amplitude `A`, and phase `phase` from time `t0` to time `t1`. This function was used to generate several of the plots of wave evolution error in this paper. It makes use of being able to write any sine-wave as a sum of products of one-dimensional sine waves along each axis. Such a construction can be done explicitly using the method

```
tensor_construct(D::Int, k::Int, n::Int, coeffArray; scheme="sparse")
```

which takes D vectors coefficient and constructs the coefficient of their corresponding D-dimensional tensor product function.

More documentation is available in the Github repository and as part of the source code.

VI. RESULTS

The Galerkin bases and derivative operators constructed above allow representing arbitrary functions, calculating their derivatives, calculating norms, solving PDEs, and evaluating the solution at arbitrary points, as is the case with any other discretization method.

We focus here on the scalar wave equation as simple example application. Using the constructions developed the prior sections, we use the Galerkin sparse grid package to calculate efficient and accurate representations in 3D and 5D, as well evolving the wave equation in time in 3+1 and 5+1 dimensions. At this accuracy, this would have been prohibitively expensive using full grid methods.

We note that, in general, spectral methods can offer superior accuracy, but they also suffer from the curse of dimensionality, and representing non-smooth functions requires high orders since exponential convergence is lost in this case. If only a single spectral level is used, then DG sparse grids are equivalent to a spectral method. DG sparse grids are thus a true superset of spectral discretizations.

In the following discussion, all errors have been calculated via a Monte Carlo integration to evaluate integrals of the form

$$\int_{[0,1]^D} |u^*(\mathbf{x}) - \tilde{u}(\mathbf{x})|^2 \, d^D\mathbf{x} \tag{44}$$

for u^* the analytically known solution and $\tilde{u}$ the DGFE approximation to the solution. The L^2 error is output as the square root of the above quantity. We use approximately 1000 points, enough to guarantee convergence to satisfactory precision for error measurement in our case. We tested that using more sampling points did not noticeably change the results.

A. Approximation Error for Representing Given Functions

We begin by evaluating the approximation error made when representing a given function with a sparse basis at a particular cut-off level. We choose as function a sine wave of the form

$$f(\mathbf{x}) \;=\; A\cos(\mathbf{k}\cdot\mathbf{x} + \phi) \tag{45}$$

with $A = 1.3,\ \phi = 0.4$, using $\mathbf{k} = 2\pi\cdot[1, 2, -1]$ in 3D and $\mathbf{k} = 2\pi\cdot[1, 0, -1, 2, 1]$ in 5D. We also employ periodic boundary conditions. These first tests do not yet involve derivatives or time integration.

Figure 9 compares the accuracy achieved with a particular number of coefficients for full and sparse grids. As expected, sparse grids are more efficient (in the sense of the accuracy-to-cost ratio), and this efficiency increases for higher resolutions. As also expected, this effect is even more pronounced in the $D = 5$ case.

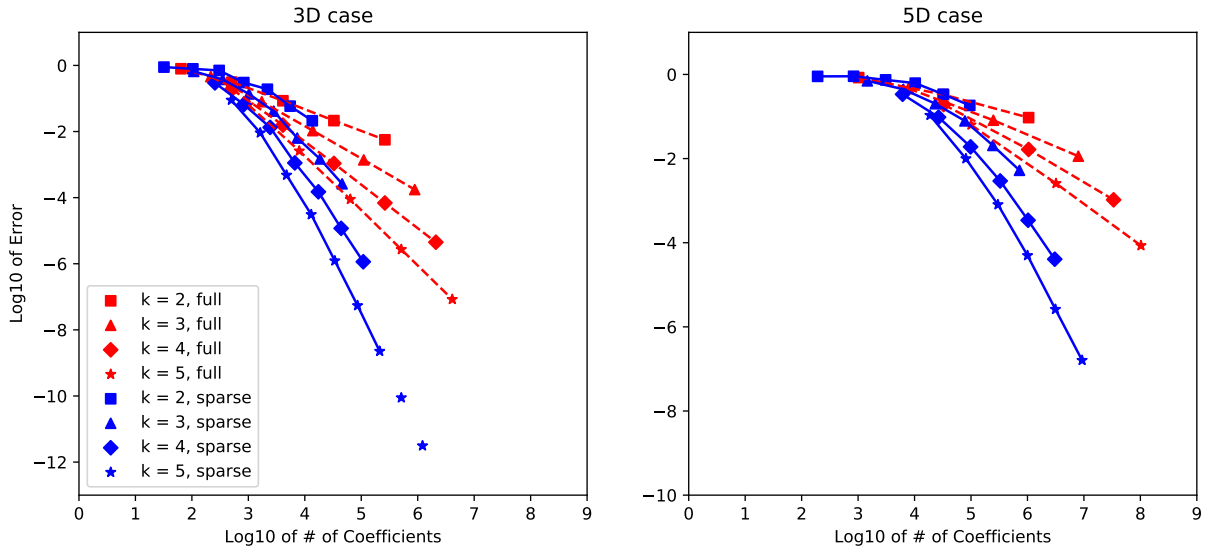

FIG. 9: Comparing approximation errors as a function of the number of coefficients for $D = 3$ (left) and $D = 5$ (right) for values of the polynomial order k from 1 to 5. Even at moderate scales (10^5 coefficients), sparse grids for $k = 5$ exhibit errors as much as $1,000$ times smaller than their full counterparts. Particularly noteworthy here is the fact that the accuracy gain from higher polynomial order k in the sparse scheme is far more significant than in the full scheme. This shows significant promise for higher-order sparse Galerkin methods. See main text for more details.

	sparse grid			full grid		
n	Avg. time	Mem. usage	Error	Avg. time	Mem. usage	Error
1	0.80 ms	7.8 MB	$1.1 \cdot 10^{-1}$	9.7 ms	68 MB	$6.3 \cdot 10^{-2}$
2	7.6 ms	54 MB	$9.9 \cdot 10^{-3}$	1.1 s	5.0 GB	$2.6 \cdot 10^{-3}$
3	40 ms	270 MB	$8.0 \cdot 10^{-4}$	?	?	$8.5 \cdot 10^{-5}$
4	220 ms	1.3 GB	$5.0 \cdot 10^{-5}$	n/a	n/a	n/a
5	1.0 s	5.5 GB	$2.6 \cdot 10^{-6}$			
6	4.5 s	23 GB	$1.6 \cdot 10^{-7}$			
7	21 s	97 GB	?			

TABLE I: Comparing run times and memory usage for sparse and full grids. This table shows average time of applying the Laplacian operator to a $k = 5$ representation of a waveform in $D = 5$ dimensions, for various levels n, as well as the memory needed to store the Laplacian matrix and coefficient vectors. Levels $n = 3$ and above required more than 200 GByte of memory for a full grid **TODO: ES: can you confirm this?** and could not be run on a single node. (Our implementation does not yet support distributed-memory systems.) As described by equations (32) and (32), sparse grids have a slightly lower accuracy than full grids for the same number of levels (by a factor logarithmic in n).

TODO: ES: Can you check the error estimates? Why do we have benchmarks but no errors or errors but no benchmarks in some cases?

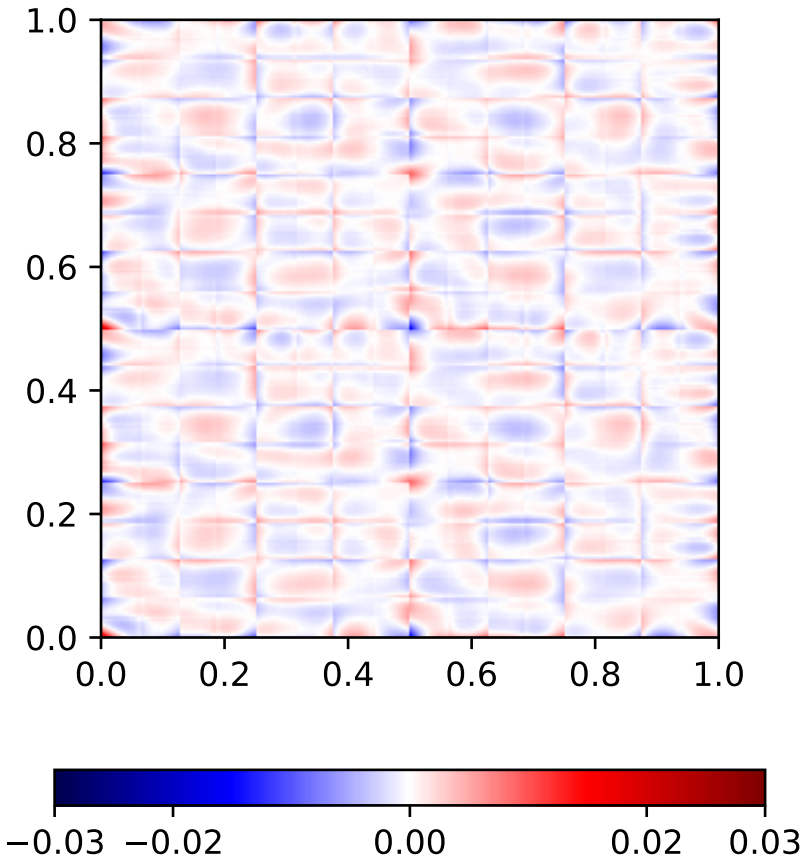

FIG. 10: Approximation error (difference to exact value) of a sine wave $\varphi(\mathbf{x}) = A\cos(\mathbf{k}\cdot\mathbf{x}+\phi)$ on $[0,1]\times[0,1]$ with $A = 1.3$, $\phi = 0.4$, and $\mathbf{k} = 2\pi\cdot[1,2]$ in 2D for $k = 3$, $n = 5$. The small discontinuities typically introduced at element boundaries and corners by DGFE methods are clearly visible. Contrary to the naive assumption, the error is *not* the largest in the interior (away from the boundaries), where the sparse grid structure has removed most collocation points.

TODO: ES: idea: maybe we can average the solution the same way jonah did in our OLDG paper? this should reduce the error at element boundaries. AA: I will have to look at this. It could reduce error by a lot (from the error plots I've seen in 1D, it is exactly as in figure 18 of the OLDG paper)

Even in two dimensions, the sparse Galerkin basis already provides a considerable advantage in efficiency. For the sake of visual intuition, we show the approximation error of representing a sine wave in 2D by Galerkin sparse grids of in Figure 10, for polynomial order $k = 3$ and sparse level $n = 5$.

Table I compares run times and memory usage between sparse and full grids, run on a single high-memory node of the *Grace* cluster at Yale.[3] It is evident that sparse grids require far fewer resources than full grids for the same accuracy.

[3] The respective compute nodes of Grace had Intel Broadwell CPUs with 28 cores running at 2.0 GHz, and with 250 GByte of memory. The cluster setup is described at http://research.computing.yale.edu/support/hpc/clusters/grace.

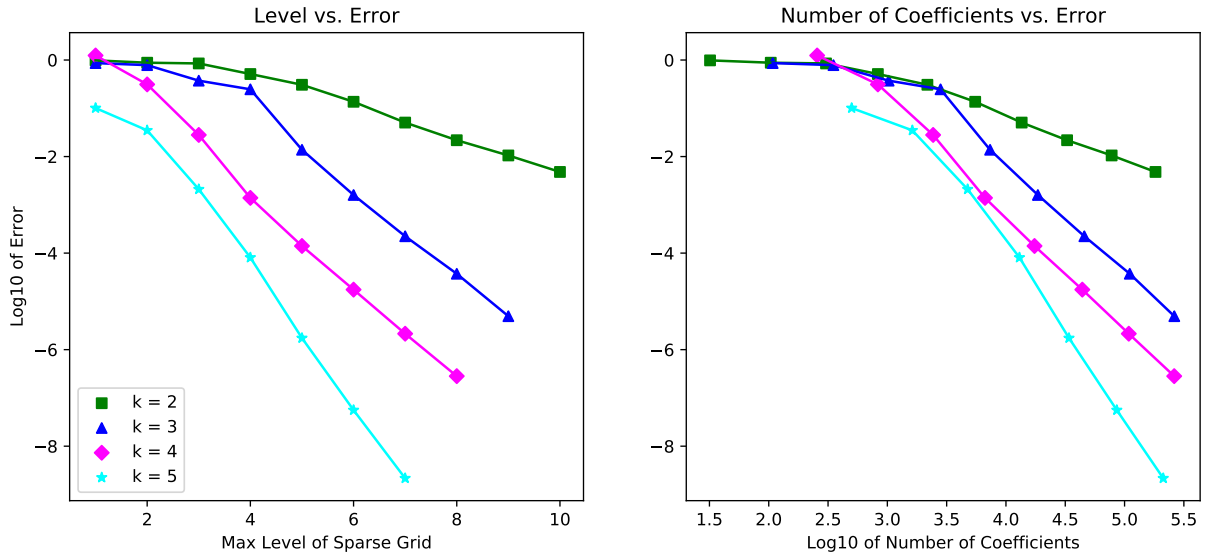

FIG. 11: Error of travelling wave evolution after 1.32 crossing times in 3D, using $\mathbf{k} = 2\pi \cdot [1, 2, -1]$ from $t = 0$ to $t = 0.54$ at various levels and polynomial degrees, with periodic boundary conditions.

TODO: ES: can you show full grid results for comparison? Done, see Figure 13. I think we should replace this figure with that one because it is more illustrative of the utility of sparse grids.

B. Approximation Error in Time Evolution

Beyond just representing given functions, Figures 11 and 12 demonstrate the accuracy of evolving a scalar wave using DGFE sparse grids. After projecting the initial position and velocity of a travelling wave in 3 and 5 dimensions onto the sparse basis, we evolve the system in time using the derivative matrices $\mathcal{D}_a^{\text{sparse}}$ and the wave evolution scheme described in Sections III C and IV, respectively.

Initial conditions for the evolution are

$$\varphi(\mathbf{x}, 0) = \cos(\mathbf{k} \cdot \mathbf{x} + \phi), \;\; \psi(\mathbf{x}, 0) = -\omega \sin(\mathbf{k} \cdot x + \phi)$$

where here $\mathbf{k}$ is a wave number vector compatible with the periodic boundary conditions and $\omega = |\mathbf{k}|$.

In summary:

TODO: ES: Can you take the 5D solution for the wave equation evolution and extract a 2D plot similar to figure 10? you could e.g. set $x_2 = x_3 = x_4 = 0$ and choose both $t = 0$ and $t = 1.32$ crossing times. I've done this in 4D, see Figure 14

VII. CONCLUSION

We review Sparse Grids as discretization method that breaks the curse of dimensionality. We develop a novel sparse grid discretization method based on Operator-based Discontinuous Galerkin Finite Elements (OLDG), based on work by Guo and Wang [5, 6]. Particularly for higher-order methods, DGFE methods exhibit more locality than finite-differencing (FD) based methods: derivative operator matrices have a block-structured sparsity structure that

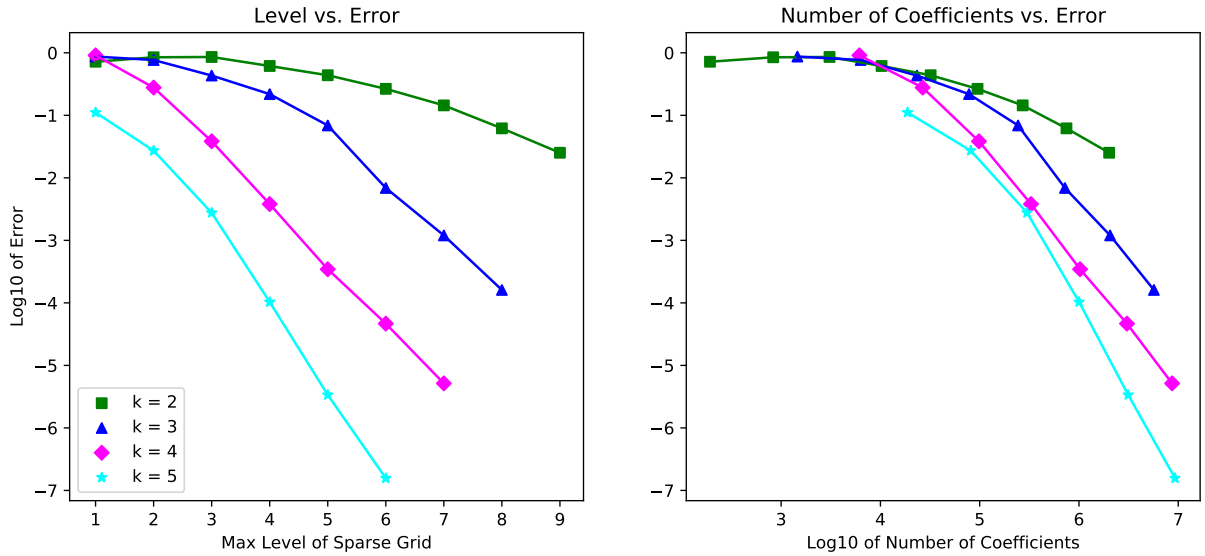

FIG. 12: Error of travelling wave evolution after 1.43 crossing times in 5D, using $\mathbf{k} = 2\pi \cdot [1, 0, -1, 2, 1]$ from $t = 0$ to $t = 0.54$ at various levels and polynomial degrees, with periodic boundary conditions.

is more efficient (cache friendly) on current computing architectures than the banded structure one finds in FD methods.

We compare theoretical estimates and actual measurements for numerical accuracy vs. storage and run-time cost, and find that sparse grids are far superior in all cases we examine, including evolving the scalar wave equation in $3 + 1$ and $5 + 1$ dimensions. We are able to perform these calculation on a single workstation. On a full grid, the state vector for the highest-resolution 5-D wave equation ($n = 6$, $k = 5$; see figure 12) would consume 55 TeraByte of storage **TODO: ES: update these numbers if we add a result with a higher resolution** that would require a supercomputer.

We are optimistic that these accuracy improvements and cost reductions will survive contact with real-world applications that include turbulence, shock waves, or highly anisotropic scenarios. In how far this is indeed the case will be the topic of further research. Well-known numerical methods such as adaptive mesh refinement have been successfully applied to Sparse Grids [3, 17–19], and should help mitigate issues encountered in real-world scenarios.

Another avenue that is described in the literature, but which we have not explored yet, is to choose a slightly different norm for our function space. A good example of this is by using the Sobolev space energy norm for H_0^1 as in [3, 20]. This selects basis functions in a different order, and further reduces the cost from $O(N \log^{D-1} N)$ to $O(N)$. While this is already supported by our software, we have not tested it yet. It induces a certain increase in complexity in managing basis functions, and the practical benefit of this method for relativistic astrophysics still needs to be demonstrated.

We anticipate that Sparse Grids will allow novel higher-dimensional studies in astrophysics and numerical relativity that are currently impossible to perform, or will at least significantly reduce the cost of such studies. This might be particularly relevant e.g. to radiative transfer in supernova calculations, to numerical investigations of black holes higher-dimensional ($D > 3 + 1$) spacetimes, or to simply reduce the cost of high-resolution gravitational wave-form calculations in "standard" numerical relativity.

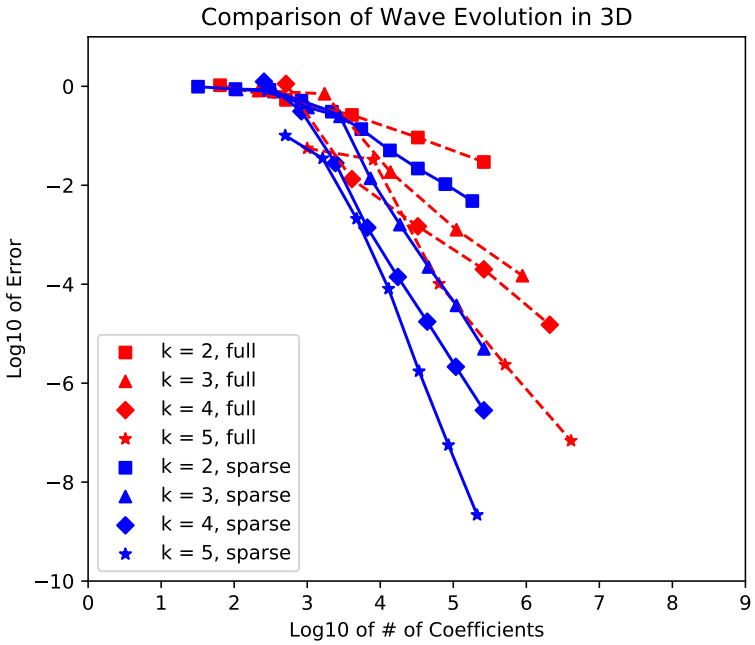

FIG. 13: Comparison of full vs. sparse traveling wave evolution for 1.32 crossing times in 3D, using $\mathbf{k} = 2\pi \cdot [1, 2, -1]$ from $t = 0$ to $t = 0.54$ at various levels and polynomial degrees, with periodic boundary conditions.

TODO: ES: i see that the full grid graphs extend further to the right (have more coefficients) than the sparse grid graphs. this looks strange. can you make the blue lines longer?

Acknowledgments

We thank Jonah Miller for his contributions to the early stages of this project, as well as for valuable discussions and comments. We acknowledge support from the Perimeter Institute for Theoretical Physics where the majority of this research took place. Perimeter is supported by the Government of Canada through the Department of Innovation, Science and Economic Development Canada and by the Province of Ontario through the Ministry of Research, Innovation and Science. AA acknowledges support from Yale University Department of Physics for sponsoring travel during the winter to work towards completing this project **TODO: ES: alex, describe what they funded and/or what particular funding source at Yale you used – "winter travel". done**. ES acknowledges support from NSERC for Discovery Grant 1505 (2012). We used open-source software tools, in particular Julia and Python. We used computing services of the Grace cluster at Yale University. We are using hosting services by Github to distribute our open source package.

[1] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[2] Sergey A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSSR*, 4:240–243, 1963.

[3] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta numerica*, 13:147–269, 2004.

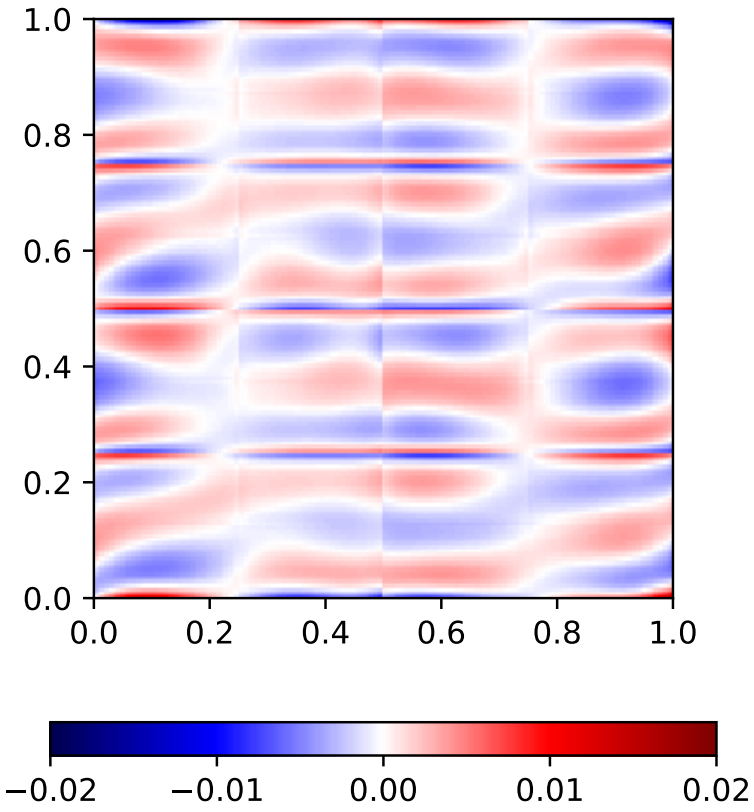

FIG. 14: Residual error of wave evolution for 1.43 crossing times in 4D plotted on the plane $x_2 = 0.1, x_3 = 0.2$, using $\mathbf{k} = 2\pi \cdot [1, -1, 1, 2]$ from $t = 0$ to $t = 0.54$ at $k = 5$ and $n = 3$.

TODO: ES: an error of 0.02 seems large. can you do this for a larger n, e.g. $n = 5$? but please keep the original figure as well as i want to see which looks better.

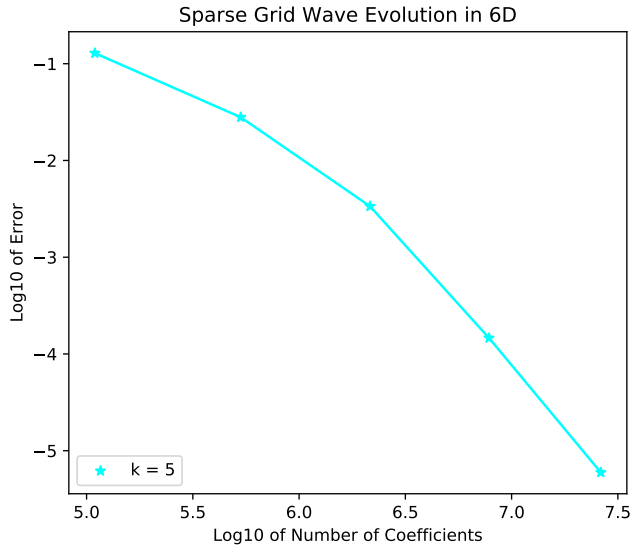

FIG. 15: Error of wave evolution for 1.52 crossing times in 6D using $\mathbf{k} = 2\pi \cdot [1, 0, -1, 2, 1, -1]$ from $t = 0$ to $t = 0.54$ at $k = 5$ from $n = 1$ to 5,

Symbol	Meaning
$\mathbf{V}$	Function space to be discretized
N	total number of subintervals in the discretization
D	total dimension
ϕ	Hat function on $[-1, 1]$
ℓ	Denoting a level in 1-D
n	Max level cutoff (all dimensions)
i	Cell number at a given level, referred to as the *node*
$\phi_{\ell,i}$	Shifted and scaled hat function ϕ to ith cell of level ℓ
$\mathbf{V}_n$	Discretization of function space on 1D unit interval $[0, 1]$ using hat functions up to level n
$\mathbf{W}_\ell$	Complement of $\mathbf{V}_{\ell-1}$ in $\mathbf{V}_\ell$ in terms of hat function basis
$\mathbf{l}$	D-tuple of levels (a multi-level)
$\phi_{\mathbf{l}}$	Tensor product of functions $\phi_{\mathbf{l}_i}(\mathbf{x}_i)$
$\mathbf{V}_{\mathbf{l}}$	Tensor product of one-dimensional spaces $\mathbf{V}_{\mathbf{l}_i}$
$\mathbf{W}_{\mathbf{l}}$	Tensor product of one-dimensional spaces $\mathbf{W}_{\mathbf{l}_i}$
$\mathbf{V}_n^D$	Shorthand for $\mathbf{V}_{(n,n,\ldots,n)}$: full grid space of hat functions up to level n in D dimensions
$\hat{\mathbf{V}}_n^D$	Sparse grid space of hat functions up to level n in D dimensions
P	Dimension of discretized space, AKA number of coefficients for a function representation
k	Dimension of space of polynomials on each cell of a DG basis.
m	Mode number in a given cell, running from 1 to k
$h_{n,i,m}$	Basis function of the mth mode in the ith cell at level n
$v_{\ell,i,m}$	Hierarchical basis of DG functions
$\mathbf{V}_{n,k}$	Discretization of function space on 1D unit interval $[0, 1]$ using a DG basis up to level n and polynomial order $< k$
$\mathbf{W}_{n,k}$	Orthogonal complement of $\mathbf{V}_{n-1,k}$ in $\mathbf{V}_{n,k}$
$\mathbf{W}_{\mathbf{l},k}$	Tensor product of one-dimensional spaces $\mathbf{W}_{\mathbf{l}_i}$
$\mathbf{V}_{n,k}^D$	Shorthand for $\mathbf{W}_{(n,n,\ldots,n),k}$: full grid space of DG functions up to level n, order $< k$ in D dimensions
$\hat{\mathbf{V}}_{n,k}^D$	Sparse grid space of DG functions up to level n, order $< k$ in D dimensions
$\mathcal{D}_{i,m;i',m'}$	Matrix element of 1-D derivative operator in position basis
$\mathcal{D}^{\text{hier}}$	Derivative operator in 1-D hierarchical basis
$\mathcal{D}_a^{\text{full}}$	Derivative operator in full basis along the ath axis
$\mathcal{D}_a^{\text{sparse}}$	Derivative operator in sparse basis along the ath axis

TABLE II: Notations used within this paper

[4] Thomas Gerstner and Michael Griebel. *Sparse Grids*. John Wiley & Sons, Ltd, 2010.

[5] Wei Guo and Yingda Cheng. A sparse grid discontinuous galerkin method for high-dimensional transport equations and its application to kinetic simulations. *SIAM Journal on Scientific Computing*, 38(6):A3381–A3409, 2016.

[6] Zixuan Wang, Qi Tang, Wei Guo, and Yingda Cheng. Sparse grid discontinuous galerkin methods for high-dimensional elliptic equations. *J. Comput. Phys.*, 314(C):244–263, June 2016.

[7] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[8] Christoph Zenger. *Sparse Grids*, volume 31. Friedrich Vieweg & Sohn Verlagsgesellschaft, 1991.

[9] Joseph Traub and Henryk Wozniakowski. *A General Theory of Optimal Algorithms*. Academic Press, January 1980.

[10] Jonah M Miller and Erik Schnetter. An operator-based local discontinuous galerkin method compatible with the bssn formulation of the einstein equations. *Classical and Quantum Gravity*, 34(1):015003, 2017.

[11] Bradley K. Alpert. A class of bases in l2 for the sparse representations of integral operators. *SIAM J. Math. Anal.*, 24(1):246–262, January 1993.

[12] Schwab, Christoph, Sli, Endre, and Todor, Radu Alexandru. Sparse finite element approximation of high-dimensional transport-dominated diffusion problems. *ESAIM: M2AN*, 42(5):777–819, 2008.

[13] Gary Cohen and Sebastien Pernet. *Finite Element and Discontinuous Galerkin Methods for Transient Wave Equations*. 2017.

[14] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[15] Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and Donatella Marini. *Discontinuous Galerkin Methods for Elliptic Problems*, pages 89–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[16] ODE.jl. 2017-09-23. 0.7.0. https://github.com/JuliaDiffEq/ODE.jl.

[17] M. Griebel. Adaptive sparse grid multilevel methods for elliptic pdes based on finite differences. *Computing*, 61(2):151–179, October 1998.

[18] M. Hegland. Adaptive sparse grids. In *ANZIAM J*, pages 335–353, 2001.

[19] Dirk Pflger, Benjamin Peherstorfer, and Hans-Joachim Bungartz. Spatially adaptive sparse grids for high-dimensional data-driven problems. *Journal of Complexity*, 26(5):508 – 522, 2010. SI: HDA 2009.

[20] Glenn Byrenheid, Dinh Dng, Winfried Sickel, and Tino Ullrich. Sampling on energy-norm based sparse grids for the optimal recovery of sobolev type functions in h. *Journal of Approximation Theory*, 207(Supplement C):207 – 231, 2016.