SIAM J. SCI. COMPUT.
Vol. 42, No. 4, pp. C169–C191

© 2020 Society for Industrial and Applied Mathematics

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

# COMPUTING WITH FUNCTIONS IN THE BALL*

NICOLAS BOULLÉ† AND ALEX TOWNSEND‡

**Abstract.** A collection of algorithms in object-oriented MATLAB is described for numerically computing with smooth functions defined on the unit ball in the Chebfun software. Functions are numerically and adaptively resolved to essentially machine precision by using a three-dimensional analogue of the double Fourier sphere method to form "Ballfun" objects. Operations such as function evaluation, differentiation, integration, fast rotation by an Euler angle, and a Helmholtz solver are designed. Our algorithms are particularly efficient for vector calculus operations, and we describe how to compute the poloidal-toroidal and Helmholtz–Hodge decompositions of a vector field defined on the ball.

**Key words.** functions, spherical, double Fourier sphere method, poloidal-toroidal decomposition, Helmholtz–Hodge decomposition

**AMS subject classification.** 65D05

**DOI.** 10.1137/19M1297063

## 1. Introduction.

Three-dimensional spherical geometries are common in computational science and engineering, arising in weather forecasting [18], geophysics [23, 45, 50, 61], hydrodynamics [36, 39, 64, 65], and computational fluid dynamics [38, 48]. In each of these applications, it is routine to derive models that are continuous, even though one immediately discretizes them to compute an approximate solution. Ballfun is a software system written in MATLAB that exploits object-oriented programming to allow users to compute with scalar- and vector-valued functions defined on the three-dimensional unit ball while being oblivious to our underlying discretizations. Ballfun is the first extension of Chebfun to three-dimensional spherical geometries [21] and follows the development of Spherefun [56] and Diskfun [60] for computing with functions in the sphere and the unit disk. Software systems in Dedalus [12] (written in Python) and Approxfun [43] (written in Julia) for computations on the ball may follow soon. Dedalus and Approxfun already have excellent functionality for computing on the 2-sphere and disks [43, 59].

For computations with functions defined on the unit ball, a standard approach is to employ spherical coordinates $(r, \lambda, \theta) \in [0, 1] \times [-\pi, \pi] \times [0, \pi]$, where $r$, $\lambda$, and $\theta$ denote the radial, azimuthal, and polar variables, respectively. Thus, computations on the unit ball can be conveniently related to analogous tasks involving functions defined on a cuboid, which allows for efficient algorithms based on tensor-product structure. Unfortunately, this simple coordinate transform comes with several significant disadvantages due to the artificial pole singularities introduced by the transform.

In this paper, we employ a technique known as the double Fourier sphere (DFS) method [25, 42, 45] in conjunction with tensor-product expansions of functions. More precisely, we use a three-dimensional analogue of the DFS method that extends ideas

*Submitted to the journal's Software and High-Performance Computing section November 1, 2019; accepted for publication (in revised form) June 29, 2020; published electronically August 17, 2020.
https://doi.org/10.1137/19M1297063
**Funding:** The work of the first author was supported by the EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling through grant EP/L015803/1. This work was supported by National Science Foundation grant 1818757.

†Mathematical Institute, University of Oxford, Oxford, OX2 6GG, UK (boulle@maths.ox.ac.uk).
‡Department of Mathematics, Cornell University, Ithaca, NY 14853 (townsend@cornell.edu).



Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

from the disk and sphere [56, 60] (implemented in Spherefun and Diskfun, which are part of Chebfun), while preserving the additional structure that is present in the three-dimensional ball (see Definition 2.1). The DFS method alleviates some of the computational difficulties with spherical coordinates while having approximants that have an underlying tensor-product structure for efficient algorithms and fast transforms based on the fast Fourier transform (FFT) [19]. We use DFS approximants to develop a collection of algorithms for performing everyday computational tasks on scalar- and vector-valued functions defined on the unit ball and thus provide a convenient computational environment for scientific explorations.

Our algorithms are designed, whenever possible, to compute each operation on a function to essentially machine precision by using data-driven techniques from approximation theory. Our codes are also designed to have no required user-defined algorithmic parameters and to be as intuitive as possible for MATLAB users. For example, `sum(v)` returns the sum of the entries of a vector `v` in MATLAB, while `sum3(f)` returns the integral of a function `f` over the unit ball. Moreover, `v.*w` performs entry-by-entry vector multiplication, while `f.*g` returns a function representing the multiplication of `f` and `g` in Ballfun. During the operation `f.*g`, our algorithm automatically selects the discretization of the output so that the result is approximated to essentially machine precision. We repeat this idea in the one hundred or so Ballfun commands by constantly expanding and pruning underlying discretizations to represent functions as efficiently as possible.

There are several existing approaches for computing with functions on the unit ball, and we seriously considered two other approaches:

**Spherical harmonic expansions:** Spherical harmonic expansions of a function are given by $f(r,\lambda,\theta) = \sum_{\ell=0}^{\infty}\sum_{m=-\ell}^{\ell} f_\ell^m r^\ell Y_\ell^m(\lambda,\theta)$, where $Y_\ell^m$ is a surface spherical harmonic. These expansions can be thought of as the ball analogue of trigonometric expansions for periodic functions. When truncated, they provide essentially uniform resolution of a function over the ball. They have major applications in geophysics [40] and the numerical solution of separable elliptic equations.

**Orthogonal polynomials on the ball:** Given an appropriate weight function on the ball, one can derive various families of orthogonal polynomials that are built from ultraspherical polynomials [22, sect. 5.1]. Expanding functions in any one of these bases provides excellent resolution properties, along with fast evaluation, differentiation, and integration of the expansions. Unlike spherical harmonic expansions, they are rarely employed in practice.

We require a representation for functions on the ball that can be adaptively computed, as we would like to achieve an accuracy close to machine precision. While there are optimal-complexity spherical harmonic transforms [49], it is highly desirable to have the most computationally efficient fast transform associated with an expansion. The Chebfun software project is currently not equipped with spherical harmonics transforms.

The DFS method offers a simple and computationally efficient fast transform based on the FFT (see subsection 2.2). In addition, the algorithms in Ballfun must interface with existing components of Chebfun such as Diskfun and Spherefun for computing with functions on the unit disk and sphere, which both use the DFS method. For example, a Spherefun object can be provided as boundary conditions for solving Helmholtz equations in the ball (see section 4). Unlike spherical harmonic and orthogonal polynomial expansions, the DFS method does not guarantee that an expansion is infinitely differentiable on the ball, even when the original function is.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

For this reason, our algorithms must strictly preserve a structure in the DFS expansions to ensure that they represent a smooth function on the ball (see Definition 2.1).

Using DFS approximants, we develop a variety of algorithmic tools to provide a convenient computational environment for integrating, differentiating, and solving partial differential equations (see section 4), as well as representing vector-valued functions. This allows us to develop a set of algorithms for performing vector calculus (see section 5), including computing the Helmholtz–Hodge and poloidal-toroidal decompositions.

The majority of Chebfun's multivariate approximation algorithms employs low-rank decompositions of functions. Here, we use standard tensor-product approximation together with the DFS method. Since the algorithms in Ballfun must preserve a symmetry structure, called block-mirror-centrosymmetric symmetries (see subsection 2.1), using low-rank decompositions would be quite technical on the ball (see (2.1)).

The paper is organized as follows. We briefly introduce the software that accompanies this paper in subsection 1.1. Then, in section 2, we explain the methods used to discretize smooth functions on the ball. Next, in section 3, we discuss some of the operations implemented in the software such as integration, differentiation, and a fast rotation algorithm. Following this, in section 4, we describe a fast and spectrally accurate Helmholtz solver for solving equations with Dirichlet or Neumann boundary conditions. Finally, section 5 consists of a description of the vector calculus algorithms, including the poloidal-toroidal and Helmholtz–Hodge decompositions.

**1.1. Software.** Ballfun is part of Chebfun [21], which is a software system for computing with functions and solving differential equations on an interval [5], rectangle [54], cuboid [34], disk [60], and the surface of a sphere [56]. Accompanying this paper is the publicly available MATLAB code in Chebfun [21] with two new classes called `ballfun` and `ballfunv`. We encourage the reader to explore this paper with the latest version of Chebfun downloaded and ready for interactive exploration. On the Chebfun website, we provide documentation in the form of a chapter of the Chebfun Guide [21] as well as several examples.[1] Functions on the ball can be easily constructed in the software by calling the appropriate command. For instance, `f = ballfun(@(x,y,z) sin(cos(y)))` defines the function $f(x, y, z) = \sin(\cos y)$. Underneath, Ballfun adaptively resolves the function to machine precision and represents it using the DFS method. For example,

```
f = ballfun(@(x,y,z) sin(cos(y))) % ballfun representing sin(cos(y))
ballfun object:
     domain           r    lambda    theta
    unit ball        21        45       41
```

where 21, 45, and 41 are discretization parameters that Ballfun automatically determined necessary to resolve $f$ to machine precision. The Ballfun software is highly adaptive and automatically truncates the expansion to resolve functions on the ball to machine precision after each operation. After its construction (see section 2), a function can be manipulated and analyzed through the nearly one hundred operations implemented in the package (see Table 1 and Table 2).

[1] The Ballfun examples are available at http://www.chebfun.org/examples/sphere/.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

TABLE 1
*A selection of Ballfun commands for scalar-valued functions.*

| Ballfun command | Operation |
|---|---|
| `+`, `-`, `.*`, `./` | basic arithmetic |
| `coeffs2vals`, `vals2coeffs` | fast transforms |
| `feval` | pointwise evaluation |
| `sum`, `sum2`, `sum3` | integration |
| `diff` | differentiation |
| `rotate` | rotate using Euler angles |
| `helmholtz` | Helmholtz solver |

TABLE 2
*A selection of Ballfun commands for vector-valued functions.*

| Ballfun command | Operation |
|---|---|
| `cross` | cross-product |
| `dot` | dot-product |
| `feval` | pointwise evaluation |
| `curl` | curl |
| `divergence` | divergence |
| `PTdecomposition` | poloidal-toroidal decomposition |
| `HelmholtzDecomposition` | Helmholtz–Hodge decomposition |

**2. The Ballfun constructor.** In this section, we explain how smooth functions, expressed in Cartesian or spherical coordinates, are discretized and constructed in our software. Smooth functions on the ball expressed in the spherical coordinate system $(r, \lambda, \theta) \in [0, 1] \times [-\pi, \pi] \times [0, \pi]$ can potentially introduce artificial boundaries at the origin or poles, as well as the loss of periodicity in the polar variable $\theta$. To overcome this issue, we first sample functions on a tensor-product grid in spherical coordinates. Then, we compute a Chebyshev–Fourier–Fourier (CFF) expansion that interpolates the samples, using the ball analogue for the DFS method.

**2.1. The DFS method in the ball.** The DFS method for the sphere was originally proposed by Merilees [42] and is used to construct Spherefun objects in Chebfun. It naturally extends to the three-dimensional settings and maps a function defined on a ball onto a three-dimensional cuboid so that the origin and poles of the ball are not treated as artificial boundaries and the polar variable can be represented in a Fourier series [10, 24, 45, 63]. The method can also be applied to disks, cylinders, and ellipsoids [56, 60].

The ball analogue of the DFS method is obtained by constructing a CFF expansion of a function defined on $[-1, 1] \times [-\pi, \pi] \times [-\pi, \pi]$ instead of $[0, 1] \times [-\pi, \pi] \times [0, \pi]$. A continuous function $f_{\text{cart}}(x, y, z)$ on the ball is first written in spherical coordinates as

$$f(r, \lambda, \theta) := f_{\text{cart}}(r \cos\lambda \sin\theta, r \sin\lambda \sin\theta, r \cos\theta), \quad (r, \lambda, \theta) \in [0, 1] \times [-\pi, \pi] \times [0, \pi].$$

The function $f(r, \lambda, \theta)$ is not periodic in $\theta$. Under the DFS mapping, it is recovered by "doubling up" the polar variable to $[-\pi, \pi]$ in the sense that $f$ is sampled twice. The radial variable is also doubled to remove the artificial boundary at $r = 0$. Using these ideas, we extend the function $f$ to a new function $\tilde{f}$, defined on $[-1, 1] \times [-\pi, \pi] \times [-\pi, \pi]$. The function $\tilde{f}$ can be expressed as

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

$$\tilde{f}(r,\lambda,\theta) = \begin{cases} g(r,\lambda+\pi,\theta), & (r,\lambda,\theta) \in [0,1]\times[-\pi,0]\times[0,\pi], \\ h(r,\lambda,\theta), & (r,\lambda,\theta) \in [0,1]\times[0,\pi]\times[0,\pi], \\ \\ g(-r,\lambda,\pi-\theta), & (r,\lambda,\theta) \in [-1,0]\times[0,\pi]\times[0,\pi], \\ h(-r,\lambda+\pi,\pi-\theta), & (r,\lambda,\theta) \in [-1,0]\times[-\pi,0]\times[0,\pi], \\ \\ h(r,\lambda+\pi,-\theta), & (r,\lambda,\theta) \in [0,1]\times[-\pi,0]\times[-\pi,0], \\ g(r,\lambda,-\theta), & (r,\lambda,\theta) \in [0,1]\times[0,\pi]\times[-\pi,0], \\ \\ h(-r,\lambda,\pi+\theta), & (r,\lambda,\theta) \in [-1,0]\times[0,\pi]\times[-\pi,0], \\ g(-r,\lambda+\pi,\pi+\theta), & (r,\lambda,\theta) \in [-1,0]\times[-\pi,0]\times[-\pi,0], \end{cases} \tag{2.1}$$

where

$$g(r,\lambda,\theta) = f(r,\lambda-\pi,\theta), \quad h(r,\lambda,\theta) = f(r,\lambda,\theta), \quad (r,\lambda,\theta) \in [0,1]\times[0,\pi]\times[0,\pi].$$

Functions that satisfy (2.1) are said to be block-mirror-centrosymmetric (BMC) [60]. A more intuitive description is given by the visualization

$$\tilde{f} = \left[ \begin{bmatrix} g & h \\ \texttt{flip1}(\texttt{flip3}(h)) & \texttt{flip1}(\texttt{flip3}(g)) \end{bmatrix}; \begin{bmatrix} \texttt{flip3}(h) & \texttt{flip3}(g) \\ \texttt{flip1}(g) & \texttt{flip1}(h) \end{bmatrix} \right], \tag{2.2}$$

where `flip1` (resp., `flip3`) refers to the MATLAB command that reverses the order of the first (resp., third) component of a tensor.

In addition to satisfying the BMC structure, $\tilde{f}$ must be constant at $r = 0$ as well as $\theta = 0$ and $\theta = \pi$, corresponding to the origin and the poles. We call these functions BMC-III functions. (BMC-I and BMC-II functions are defined in [56] and [60], respectively.)

DEFINITION 2.1 (BMC-III function). *A function $\tilde{f} : [-1,1]\times[-\pi,\pi]\times[-\pi,\pi] \to \mathbb{C}$ is a BMC-*III *(type-*III *BMC) function if it is a BMC function, $\tilde{f}(0,\cdot,\cdot) = \alpha$, and, for any $r \in [0,1]$, $\tilde{f}(r,\cdot,0) = \beta(r)$ and $\tilde{f}(r,\cdot,\pi) = \gamma(r)$, where $\beta$ and $\gamma$ only depend on $r$ such that $\beta(0) = \gamma(0) = \alpha$ for some constant $\alpha$.*

Figure 1 shows the DFS method applied to the earth and the type-III BMC structure.

There are two salient features of the DFS method that make it attractive for developing a package for computing with functions on the ball. First, tensor product expansions of Fourier and Chebyshev bases can be used to represent $\tilde{f}$. If $f_{\text{cart}}(x,y,z)$ is a function in Cartesian coordinates on the ball, then after applying the DFS method, we have a function $\tilde{f}(r,\lambda,\theta)$ defined on the cuboid $(r,\lambda,\theta) \in [-1,1]\times[-\pi,\pi]\times[-\pi,\pi]$ that can be approximated as

$$\tilde{f}(r,\lambda,\theta) \approx \sum_{i=0}^{m-1} \sum_{j=-n/2}^{n/2-1} \sum_{k=-p/2}^{p/2-1} \alpha_{ijk} T_i(r) e^{\mathrm{i}j\lambda} e^{\mathrm{i}k\theta}, \tag{2.3}$$

where $(r,\lambda,\theta)$ are spherical coordinates, $T_i$ denotes the Chebyshev polynomial of the first kind of degree $i$, and $m,n,p$ are integers determined by the adaptive procedure described in subsection 2.3. This representation allows us to use fast transforms as well as one- and two-dimensional algorithms for Chebyshev and trigonometric expansions. The second feature is that the DFS mapping of a function leads to a BMC structure (see Figure 1) that, if preserved, ensures smoothness of the solution throughout the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

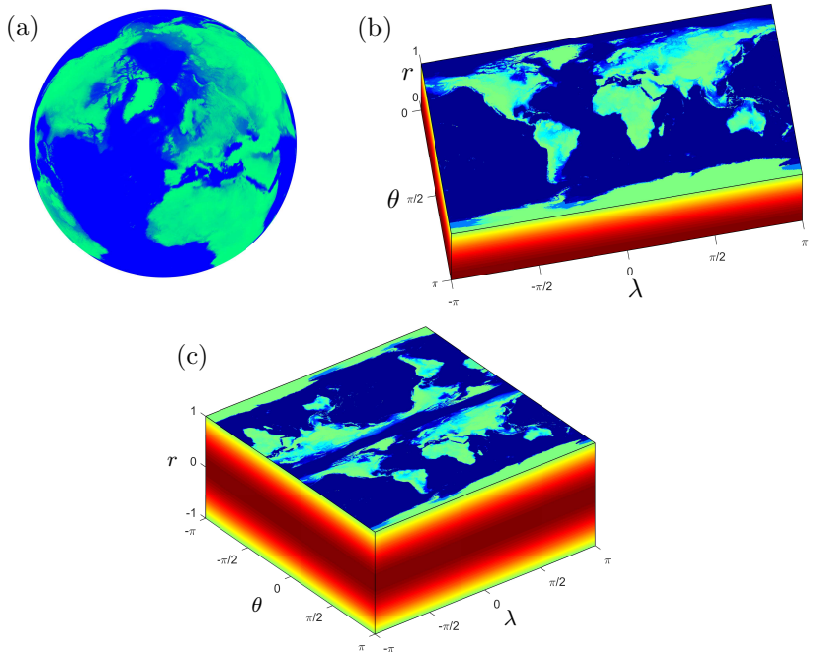


FIG. 1. *The DFS method applied to the globe.* (a) *The solid earth including the land masses.* (b) *The projection of the land masses using spherical coordinates.* (c) *Land masses after applying the DFS method. This is a BMC-*III *function that is periodic in* $\lambda$ *and* $\theta$ *and defined over* $(r, \lambda, \theta) \in [-1, 1] \times [-\pi, \pi] \times [-\pi, \pi]$.

entire domain [56, 60]. The BMC symmetry is imposed exactly by evaluating the function on $(r, \lambda, \theta) \in [0, 1] \times [-\pi, \pi] \times [0, \pi]$ and extending it to $[-1, 1] \times [-\pi, \pi] \times [-\pi, \pi]$ using (2.2). This ensures that the resulting function is smooth on the ball, i.e., at least continuous and differentiable. There are representations of functions on the ball that preserve full regularity [59]; however, they are less appropriate in our settings since they do not allow efficient FFT-based transforms.

**2.2. Computing the CFF coefficients.** Once the BMC-III function $\tilde{f}$ is found, it is approximated by a truncated CFF series [41, 57, 58]. For some even integers $m, n, p$, the CFF coefficients are stored as an $m \times n \times p$ tensor, and the entries are computed in $\mathcal{O}(mnp \log(mnp))$ operations, as follows:

1. The function $f$ is evaluated over $[0, 1] \times [-\pi, \pi] \times [0, \pi]$ at the tensor-product grid:

$$\left( \cos\left( \frac{\left(\frac{m}{2} - 1 - i\right)\pi}{m-1} \right), \frac{2j\pi}{n}, \frac{2k\pi}{p} \right), \quad 0 \le i \le \frac{m}{2} - 1, \quad -\frac{n}{2} \le j \le \frac{n}{2} - 1, \quad 0 \le k \le \frac{p}{2}. \tag{2.4}$$

2. The samples of $f$ are doubled-up (see (2.2)). This extends them to be samples of $\tilde{f}$ on $[-1, 1] \times [-\pi, \pi] \times [-\pi, \pi]$ at the tensor-product grid:

$$\left( \cos\left( \frac{i\pi}{m-1} \right), \frac{2j\pi}{n}, \frac{2k\pi}{p} \right), \quad 0 \le i \le m - 1, \quad -\frac{n}{2} \le j \le \frac{n}{2} - 1, \quad -\frac{p}{2} \le k \le \frac{p}{2} - 1$$

without any additional evaluations of $f$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

3. The CFF coefficients are computed using the discrete Chebyshev transform (DCT) [30, 31, 41] and the FFT [19].

There is also the inverse procedure, which evaluates $f$ at the grid in (2.4) in $\mathcal{O}(mnp\log(mnp))$ operations. This operation is particularly important in our plotting commands and is achieved by reversing steps 1–3, using the inverse DCT and FFT.

**2.3. Determination of the discretization size.** To construct a Ballfun object to represent a given function $f$, we first sample $\tilde{f}$ at a $17\times17\times17$ CFF grid and compute the corresponding CFF coefficients (see subsection 2.2). We then successively increase the grid size independently in each variable from 17 to 33 to 65, and so on, until we deem the function to be resolved in each variable. We use these samples to compute the CFF coefficients $A=(\alpha_{ijk})$ corresponding to an $m\times n\times p$ tensor, where $m,n,p=17,33,65,\ldots$, and then gauge the resolution in each variable by creating vectors of the absolute maximum of the coefficients along each variable, i.e.,

$$\text{Cols}_i=\max_{j,k}|\alpha_{ijk}|\,,\quad \text{Rows}_j=\max_{i,k}|\alpha_{ijk}|\,,\quad \text{Tubes}_k=\max_{i,j}|\alpha_{ijk}|\,.$$

One can now inspect these vectors to identify whether or not the function is resolved to machine precision in each variable, relative to the magnitude of $f$ on $[0,1]\times[-\pi,\pi]\times[0,\pi]$ [2, 34, 62]. One can identify a near-optimal discretization size in that variable by recording the last entry in each vector above machine precision, though the algorithm internally employed in Chebfun is more involved [2].

The constructor typically terminates when the magnitude of the coefficients of the vectors Cols, Rows, and Tubes exhibit a decay to machine precision, as shown in Figure 2 for $f(x,y,z)=\sin(\cos y)$. In particular, for $f(x,y,z)=\sin(\cos y)$, the Ballfun constructor selected a CFF series of size $21\times45\times41$ to represent $f$ to essentially machine precision over the ball. Once the function $\tilde{f}$ is represented in a

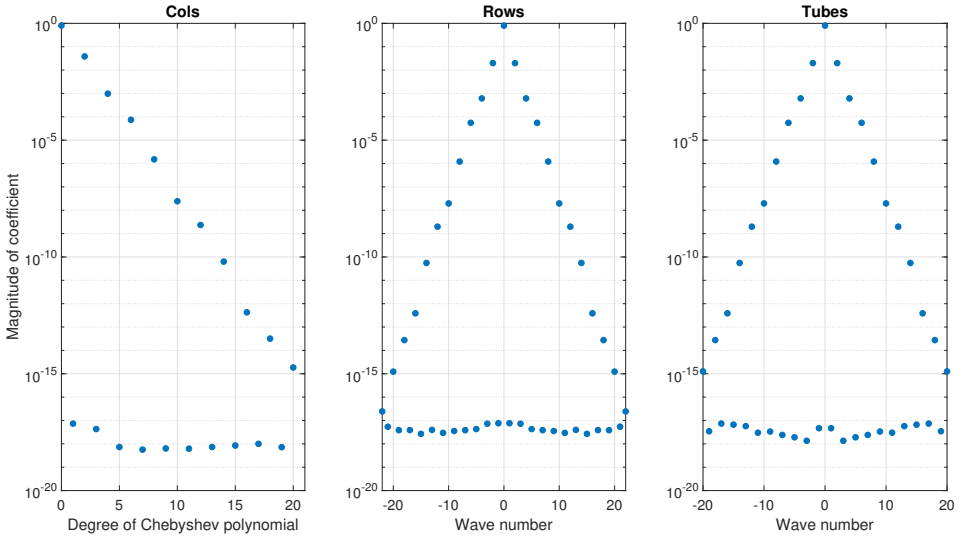


FIG. 2. *The absolute maximum Chebyshev and Fourier coefficients in the radial, azimuthal, and polar variables of* $f(x,y,z)=\sin(\cos y)$. *The Ballfun constructor selected a discretization size of* $21\times45\times41$ *to represent* $f$ *to essentially machine precision over the ball. One can visually see that the function is likely to be resolved as the entries of Cols, Rows, and Tubes decay to machine precision.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

CFF expansion, the approximant is stored as a `ballfun` object, ready for further computation.

**3. Algorithms for numerical computations with functions on the ball.** Once a `ballfun` object is computed, there are many operations that can be performed on it. In fact, many of the operations can be decomposed into a sequence of one-dimensional operations, which are particularly efficient for approximants of the form (2.3). This includes pointwise evaluation (see subsection 3.1), integration (see subsection 3.2), differentiation (see subsection 3.3), and fast rotation (see subsection 3.4).

**3.1. Pointwise evaluation.** The evaluation of a function $f(r,\lambda,\theta)$ at a point $(r^*,\lambda^*,\theta^*)$ in the ball can be computed in $\mathcal{O}(mnp)$ operations. It follows from the CFF approximation of $\tilde{f}$ in (2.3) that one has

$$\begin{aligned}\tilde{f}(r^*,\lambda^*,\theta^*) &= \sum_{i=0}^{m-1}\sum_{j=-n/2}^{n/2-1}\sum_{k=-p/2}^{p/2-1}\alpha_{ijk}T_i(r^*)e^{\mathbf{i}j\lambda^*}e^{\mathbf{i}k\theta^*}\\ &= \sum_{k=-p/2}^{p/2-1}\left(\sum_{j=-n/2}^{n/2-1}\left(\sum_{i=0}^{m-1}\alpha_{ijk}T_i(r^*)\right)e^{\mathbf{i}j\lambda^*}\right)e^{\mathbf{i}k\theta^*}.\end{aligned}$$

Therefore, $\tilde{f}(r^*,\lambda^*,\theta^*)$ can be computed by first evaluating $\sum_{i=0}^{m-1}\alpha_{ijk}T_i(r^*)$ using Clenshaw's algorithm [17], which returns an $n\times p$ matrix of values. Then, one can compute the summand over the $j$ index using Horner's scheme [62], which returns an $p$-vector, before finally computing the summand over the $k$ index using Horner's scheme. This algorithm is implemented in the `feval` command and returns a scalar for $\tilde{f}(r^*,\lambda^*,\theta^*)$. It is also possible to evaluate functions in Cartesian coordinates, and Ballfun does a change of variables to spherical coordinates in this case.

**3.2. Integration.** The triple definite integral of a function $\tilde{f}(r,\lambda,\theta)$ on the unit ball can be written as follows:

$$\begin{aligned}\int_{B(0,1)}\tilde{f}(r,\lambda,\theta)\,dV &= \int_0^1\int_0^\pi\int_{-\pi}^{\pi}\tilde{f}(r,\lambda,\theta)r^2\sin\theta\,d\lambda\,d\theta\,dr\\ &= \sum_{i=0}^{m-1}\sum_{j=-n/2}^{n/2-1}\sum_{k=-p/2}^{p/2-1}\alpha_{ijk}\int_0^1 r^2T_i(r)\,dr\int_{-\pi}^{\pi}e^{\mathbf{i}j\lambda}\,d\lambda\int_0^\pi\sin\theta e^{\mathbf{i}k\theta}d\theta,\\ &= 2\pi\sum_{i=0}^{m-1}\sum_{k=-p/2}^{p/2-1}\alpha_{i0k}\left(\int_0^1 r^2T_i(r)\,dr\right)\left(\int_0^\pi\sin\theta e^{\mathbf{i}k\theta}\,d\theta\right)\\ &= 2\pi\sum_{i=0}^{m-1}\sum_{k=-p/2}^{p/2-1}\alpha_{i0k}\nu_i\omega_k,\end{aligned}$$

where $\nu_i$ and $\omega_k$ are defined by

$$\nu_i=\begin{cases}\frac{3-i^2}{(i^2-1)(i^2-9)}, & i \text{ even},\\ \frac{3-2i(-1)^{\frac{i-1}{2}}-i^2}{(i^2-1)(i^2-9)}, & i \text{ odd},\end{cases}\qquad \omega_k=\begin{cases}\frac{1+e^{\mathbf{i}\pi k}}{1-k^2}, & k\neq\pm1,\\ 0, & k=\pm1.\end{cases}$$

Moreover, $\nu_i$ has removable singularities for $i=1,3$ and $\nu_1=1/4$, $\nu_3=-1/12$. Here, the last equality in the computation of the integral of $f$ on the unit ball follows by

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

calculating the integrals in $\theta$ (see, for example, [56, eq. (4.3)]) and in $r$ explicitly. Therefore, the integral of $\tilde{f}$ reduces to a basic task in linear algebra and can be computed in $\mathcal{O}(mp)$ operations. This is implemented in Ballfun in the `sum3` command. For example, the function $f(x,y,z) = x^2$ has an integral of $4\pi/15$ over the ball and can be computed in Ballfun by

```
f = ballfun(@(x,y,z) x.^2); % ballfun representing x^2
sum3(f)                     % Integrate over the ball
ans =
    0.837758040957278
```

The absolute error is computed as `abs(sum3(f) - 4*pi/15)` and is given by $1.102 \times 10^{-16}$.

**3.3. Differentiation.** Differentiation of a function on the ball with respect to spherical coordinates in $r$, $\lambda$, and $\theta$ may introduce singularities at the poles and origin. For instance, consider the smooth function $f(r,\lambda,\theta) = r\cos\theta$. The derivative of $f$ with respect to $\theta$ is $-r\sin\theta$, which is not smooth at the poles because it does not satisfy the BMC structure described in subsection 2.1. However, we are interested in computing derivatives that arise in vector calculus, such as the gradient, the divergence, the curl, or the Laplacian. All these operations can be expressed as partial derivatives in the Cartesian coordinates system. Therefore, our default is to allow for Ballfun objects to be differentiated in the Cartesian coordinate system.

We follow the same approach as Spherefun [56] and express the partial derivatives in $x$, $y$, and $z$ in terms of the spherical coordinates $r$, $\lambda$, and $\theta$ as follows:

$$\frac{\partial}{\partial x} = \cos\lambda\sin\theta\frac{\partial}{\partial r} - \frac{\sin\lambda}{r\sin\theta}\frac{\partial}{\partial\lambda} + \frac{\cos\lambda\cos\theta}{r}\frac{\partial}{\partial\theta}, \tag{3.1}$$

$$\frac{\partial}{\partial y} = \sin\lambda\sin\theta\frac{\partial}{\partial r} + \frac{\cos\lambda}{r\sin\theta}\frac{\partial}{\partial\lambda} + \frac{\sin\lambda\cos\theta}{r}\frac{\partial}{\partial\theta}, \tag{3.2}$$

$$\frac{\partial}{\partial z} = \cos\theta\frac{\partial}{\partial r} + \frac{\sin\theta}{r}\frac{\partial}{\partial\theta}. \tag{3.3}$$

Then, (3.1), (3.2), and (3.3) involve $\mathcal{O}(mnp)$ operations on the tensor of CFF coefficients representing $\tilde{f}$. For example, the derivative of $\tilde{f}$ with respect to $\lambda$ can be expressed as

$$\frac{\partial\tilde{f}}{\partial\lambda} = \sum_{i=0}^{m-1}\sum_{j=-n/2}^{n/2-1}\sum_{k=-p/2}^{p/2-1}\alpha_{ijk}\mathbf{i}jT_i(r)e^{\mathbf{i}j\lambda}e^{\mathbf{i}k\theta}.$$

Multiplications and divisions by $\sin\lambda$, $\cos\lambda$, $\sin\theta$, and $\cos\theta$ in (3.1)–(3.3) are computed by multiplying the tensor of CFF coefficients $A = (\alpha_{ijk})$ by the corresponding matrices of linear operators, expressed in the Fourier basis. For example, we write $\tilde{f}(r,\lambda,\theta)/\sin\theta \approx \sum_{i=0}^{m-1}\sum_{j=-n/2}^{n/2-1}\sum_{k=-p/2}^{p/2-1} b_{ijk}T_i(r)e^{\mathbf{i}j\lambda}e^{\mathbf{i}k\theta}$, where $B = (b_{ijk})$ satisfies

$$B(:,j,:) = A(:,j,:)M_{\sin}^{-\top}, \quad -\frac{n}{2} \le j \le \frac{n}{2}-1, \quad M_{\sin} = \frac{\mathbf{i}}{2}\begin{bmatrix} 0 & 1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & -1 & 0 \end{bmatrix}.$$

Here, $M_{\sin}$ is the matrix of multiplication by $\sin\theta$ in the Fourier basis. It is nonsingular if we choose $p$ to be even (in this case the eigenvalues are $\cos(\pi l/(p+1))$, $1 \le l \le p$).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Moreover, we write $\tilde{f}(r,\lambda,\theta)/r \approx \sum_{i=0}^{m-1}\sum_{j=-n/2}^{n/2-1}\sum_{k=-p/2}^{p/2-1} b_{ijk}T_i(r)e^{\mathbf{i}j\lambda}e^{\mathbf{i}k\theta}$, where $B = (b_{ijk})$ satisfies

$$B(:,j,:) = M_r^{-1}A(:,j,:), \quad -\frac{n}{2} \le j \le \frac{n}{2}-1, \quad M_r = \begin{bmatrix} 0 & \frac{1}{2} & & & & \\ 1 & 0 & \frac{1}{2} & & & \\ & \frac{1}{2} & \ddots & \ddots & & \\ & & \ddots & \ddots & \frac{1}{2} & \\ & & & \frac{1}{2} & 0 & \frac{1}{2} \\ & & & & \frac{1}{2} & 0 \end{bmatrix}.$$

Here, $M_r$ stands for the matrix of multiplication by $r$ in the Chebyshev basis. This matrix is invertible for even $m$ since its determinant is equal to $-(-1/4)^{m/2-1}/2$.

Working directly on coefficients allows us to circumvent potential singularity issues at $r = 0$ and the poles, while the standard technique uses a "shifted grid" procedure in the physical space [13, 24, 35]. This procedure shifts the grid of sampled points in the latitude and radial directions, which avoids evaluation at the poles and the origin but can be numerically inaccurate near these points.

These operations are implemented in Ballfun in the `diff` command. For example, the derivative of $f(x,y,z) = \cos(xy)$ with respect to $x$ is also represented as a Ballfun object and can be computed as

```
f = ballfun(@(x,y,z) cos(x.*y));% ballfun representing cos(xy)
diff(f, 1)                      % Compute ballfun representing df/dx
ans =
 ballfun object
     domain          r    lambda    theta
     unit ball      24        43       40
```

Ballfun calls the constructor after each operation to readjust the grid sizes (see subsection 2.3). Here, a discretization size of $24 \times 43 \times 40$ was determined necessary to resolve $\partial f/\partial x$ while $f$ is represented by a $21 \times 41 \times 37$ CFF series.

**3.4. Fast rotation algorithm using a nonuniform Fourier transform.** Rotating functions defined on the ball has applications in many fields, including quantum mechanics, inverse scattering, and geophysics. Ballfun has a `rotate` command to efficiently perform rigid-body rotations of functions. Every rigid-body rotation can be specified by an Euler angle $(\alpha,\beta,\gamma)$ in the Z-X-Z convention [1], which corresponds to rotating first by $\alpha$ around the $z$-axis then rotating by $\beta$ around the (original) $x$-axis and then, finally, rotating by $\gamma$ around the new $z$-axis. All the angles are given in radians. The algorithm to achieve this rotation requires a nontrivial computation because the rotated function must be represented by an approximant in the original coordinate system.

The classical algorithm for computing the rotation of a function $f$ on the ball is to first express $f$ in terms of a spherical harmonic expansion and then to use the fact that the spherical harmonics form a basis of $SO(3)$ [29]. Since Ballfun does not represent functions using spherical harmonic expansions, we use an algorithm based on the DFS method and the two-dimensional nonuniform FFT [47]. We do this by taking the CFF grid in (2.4) and rotating it by Euler angle $(\alpha,\beta,\gamma)$. Then, we evaluate the function at this rotated grid and call the Ballfun constructor. Since the rotated grid is almost always nonuniform in the $\theta$ and $\lambda$ variables of the doubled-up spherical coordinates and a Chebyshev grid in $r$, the evaluation is done in $\mathcal{O}(mnp\log(mnp))$ operations with a two-dimensional nonuniform FFT in $\theta$ and $\lambda$ and a DCT in $r$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

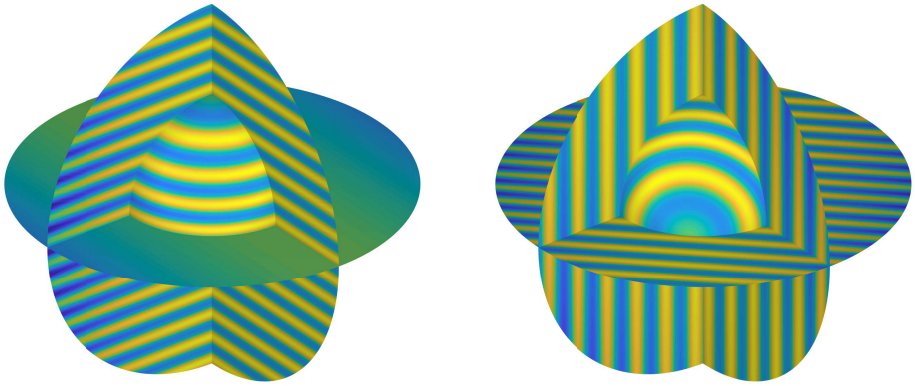

FIG. 3. *The function* $f(x,y,z) = \sin(50z) - x^2$ *(left) and its rotation by the Euler angles* $\phi = -\pi/4$, $\theta = \pi/2$, *and* $\psi = \pi/8$ *(right). The rotation of the function is computed with the* `rotate` *command and the functions are visualized with the* `plot` *command.*

For example, the following code snippet calculates the rotation of $f(x,y,z) = \sin(50z) - x^2$ by Euler angle $(-\pi/4, \pi/2, \pi/8)$ (see Figure 3).

```
f = ballfun(@(x,y,z) sin(50*z) - x.^2) % ballfun for sin(50z)-x^2
f =
   ballfun object
       domain          r    lambda    theta
       unit ball      90         5      179
g = rotate(f, -pi/4, pi/2, pi/8)       % Rotate by (-pi/4,pi/2,pi/8)
g =
   ballfun object
       domain          r    lambda    theta
       unit ball      91       180      182
```

As one can see, the rotate command is also adaptive and selects the appropriate discretization to resolve the rotated function.

**4. Fast spectral method for solving the Helmholtz equation.** In this section, we describe a fast algorithm for solving the Helmholtz equation on the ball with Neumann boundary conditions. An optimal-complexity algorithm for solving the Helmholtz equation with Dirichlet conditions on the boundary of the ball is described in [26], though it cannot immediately be generalized to the situation with Neumann conditions. Helmholtz solvers are useful in computational fluid dynamics as well as the computation of vector decompositions such as the poloidal-toroidal and Helmholtz–Hodge decompositions [3, 7].

**4.1. Discretization of the Helmholtz equation.** Consider the Helmholtz equation on the ball, i.e., $u_{xx} + u_{yy} + u_{zz} + K^2 u = f$ with Neumann boundary conditions $g(x,y,z)$ on the sphere $x^2 + y^2 + z^2 = 1$ and a real wave number $K$. The change of variables given by $(x,y,z) = (r\cos\lambda\sin\theta, r\sin\lambda\sin\theta, r\cos\theta)$, where $r \in [0,1]$, $\lambda \in [-\pi,\pi]$, and $\theta \in [0,\pi]$, transforms the equation into

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial u}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial u}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2 u}{\partial\lambda^2} + K^2 u = f. \tag{4.1}$$

One can multiply (4.1) by $r^2\sin^2\theta$ to remove the singularities in the variable coefficients at the origin and at the poles of the ball. We then use the DFS method

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

(see subsection 2.1) to represent $u$ over the domain $(r,\lambda,\theta) \in [-1,1]\times[-\pi,\pi]\times[-\pi,\pi]$. This allows us to solve (4.1) by seeking a tensor of CFF coefficients for $u$ (see (2.3)).

Let $U = (u_{ijk})$ and $F = (f_{ijk})$ be $m \times n \times p$ tensors of CFF coefficients of $u$ and $(r^2\sin^2\theta)f$, respectively. Since (4.1) decouples in the azimuthal variable $\lambda$, the following equation holds for $-n/2 \le j \le n/2-1$:

$$\sin^2\theta\frac{\partial}{\partial r}\left(r^2\frac{\partial u_j}{\partial r}\right) + \sin\theta\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial u_j}{\partial\theta}\right) + \left(K^2r^2\sin^2\theta - j^2\right)u_j = f_j, \tag{4.2}$$

where the functions $u_j$ and $f_j$ are defined by

$$u_j(r,\theta) = \sum_{i=0}^{m-1}\sum_{k=-p/2}^{p/2-1} u_{ijk}T_i(r)e^{\mathbf{i}k\theta}, \qquad f_j(r,\theta) = \sum_{i=0}^{m-1}\sum_{k=-p/2}^{p/2-1} f_{ijk}T_i(r)e^{\mathbf{i}k\theta}.$$

We discretize (4.2) in the radial variable using the ultraspherical spectral method [44], and in the polar variable using the Fourier spectral method. Partial derivatives in the polar variable $\theta$ and multiplication by $\sin\theta$ are represented by sparse and banded matrices in the Fourier basis. The ultraspherical spectral method results in sparse and banded matrices of operators (such as differentiation or multiplication by $r$) between Chebyshev and ultraspherical polynomials. This allows us to write (4.2) in the form of a generalized Sylvester equation [28] in the unknown matrix $U(:,j,:)$:

$$L_rU(:,j,:)M_{\sin^2}^\top + S_{02}U(:,j,:)L_\theta^{j\top} = S_{02}F(:,j,:), \tag{4.3}$$

where $L_r$ is the matrix representing the operator $u \mapsto \frac{\partial u}{\partial r}(r^2\frac{\partial u}{\partial r}) + K^2r^2u$ from the Chebyshev basis $T$ to the ultraspherical basis $C^{(2)}$ and $S_{02}$ is the conversion matrix between these bases [44]. The matrices $M_{\sin^2}$ and $L_\theta^j$ represent the multiplication by $\sin^2\theta$ and the operator $u \mapsto \sin\theta\frac{\partial u}{\partial\theta}(\sin\theta\frac{\partial u}{\partial\theta}) - j^2u$ in the Fourier basis, respectively.

**4.2. Imposing Neumann boundary conditions when $\mathbf{K \neq 0}$.** It is essential to slightly modify (4.3) to impose Neumann boundary conditions on $u$, i.e., $\partial_r u|_{r=1} = g(\lambda,\theta)$. The first step is to double-up the smooth function $g(\lambda,\theta)$ in the $\theta$ variable using the DFS method [56] and define its Fourier–Fourier matrix of coefficients $G^+ = (g^+_{jk})$. Since the radial variable $r$ of (4.1) has been doubled-up, we need to impose a Neumann condition at $r = 1$ and $r = -1$. The matrix of coefficients of the boundary condition at $r = -1$, $G^- = (g^-_{jk})$, can be deduced from $G^+$ (see subsection 2.1) and takes the form

$$G^-(j,:) = (-1)^jG^+(j,:), \quad -\frac{n}{2} \le j \le \frac{n}{2}-1.$$

The Neumann operators $u \mapsto \frac{\partial u}{\partial r}\big|_{r=1}$ and $u \mapsto \frac{\partial u}{\partial r}\big|_{r=-1}$ are represented in the Chebyshev basis by the $1 \times m$ matrices

$$B^+(i) = i^2, \quad B^-(i) = (-1)^{i+1}i^2, \quad 0 \le i \le m-1,$$

respectively. The Neumann conditions also decouple in the variable $\lambda$ and can be written as

$$\begin{pmatrix} B^+ \\ B^- \end{pmatrix} U(:,j,:) = \begin{pmatrix} G^+(j,:) \\ G^-(j,:) \end{pmatrix}, \quad -\frac{n}{2} \le j \le \frac{n}{2}. \tag{4.4}$$

Therefore, a Helmholtz's solver (4.1) with Neumann boundary conditions is realized by solving the following $m \times p$ generalized Sylvester equation with linear constraints:

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

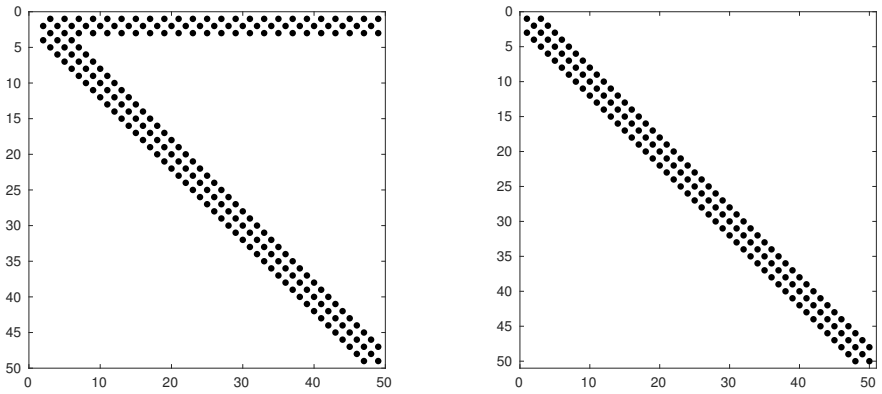

FIG. 4. *Sparsity structure of the matrix $\tilde{L}_r$ (left) and sparsity structure of the banded matrix $M_{\sin^2}$ (right) for $m = p = 50$.*

$$L_r U(:,j,:) M_{\sin^2}^\top + S_{02} U(:,j,:) L_\theta^{j\top} = S_{02} F(:,j,:), \tag{4.5}$$

$$\begin{pmatrix} B^+ \\ B^- \end{pmatrix} U(:,j,:) = \begin{pmatrix} G^+(j,:) \\ G^-(j,:) \end{pmatrix}, \tag{4.6}$$

where $-n/2 \le j \le n/2 - 1$. The constraints (4.6) can be used to remove degrees of freedom in $U(:,j,:)$ and transform (4.5) into a generalized Sylvester equation with a unique solution without constraints [53], i.e.,

$$\tilde{L}_r X_j M_{\sin^2}^\top + \tilde{S}_{02} X_j L_\theta^{j\top} = \tilde{S}_{02} \tilde{F}(:,j,:), \quad -\frac{n}{2} \le j \le \frac{n}{2} - 1. \tag{4.7}$$

Figure 4 shows the sparsity structure of the matrices $\tilde{L}_r$ and $M_{\sin^2}$ in (4.7). We obtain $n$ decoupled Sylvester matrix equations, where each one can be solved in $\mathcal{O}(m^3 + p^3)$ operations using the Bartels–Stewart algorithm [4, 28]. Once $X_j$ has been computed, the matrix of coefficients $U(:,j,:)$ can be recovered using the linear constraints. Thus, the total complexity is $\mathcal{O}((m^3 + p^3)n)$ operations.

**4.3. Imposing Neumann boundary conditions when K = 0.** We consider the zeroth Fourier mode $j = 0$ of (4.3) with $K = 0$ (Poisson equation). The solution to this equation with Neumann boundary conditions is unique only up to a constant. However, this additional constraint cannot be imposed on a Sylvester matrix equation. Therefore, we transform (4.3) into the Chebyshev–Legendre basis to decouple this Sylvester equation in the polar variable $\theta$. The function $u_0(r,\theta)$, defined in subsection 4.1, satisfies the following equation:

$$\frac{\partial}{\partial r}\left(r^2 \frac{\partial u_0}{\partial r}\right) + \frac{1}{\sin\theta} \frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial u_0}{\partial \theta}\right) = r^2 f_0. \tag{4.8}$$

The functions $u_0$ and $r^2 f_0$ can be expressed in the Chebyshev–Legendre basis as

$$u_0(r,\theta) = \sum_{i=0}^{m-1} \sum_{k=0}^{p-1} \tilde{u}_{i0k} T_i(r) P_k(\cos\theta), \quad r^2 f_0(r,\theta) = \sum_{i=0}^{m-1} \sum_{k=0}^{p-1} \tilde{f}_{i0k} T_i(r) P_k(\cos\theta).$$

The zeroth Fourier modes $j = 0$ of the Neumann boundary conditions at $r = 1$, $g_0^+(\theta)$, and at $r = -1$, $g_0^-(\theta)$, can also be written as Legendre series

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

$$g_0^+(\theta) = \sum_{k=0}^{p-1} \tilde{g}_{0k}^+ P_k(\cos\theta), \quad g_0^-(\theta) = \sum_{k=0}^{p-1} \tilde{g}_{0k}^- P_k(\cos\theta).$$

The orthogonality of the Legendre basis allows us to decouple (4.8) in the polar variable $\theta$ as $p$ ordinary differential equations with Neumann boundary conditions:

$$\frac{\partial}{\partial r}\left(r^2 \frac{\partial \tilde{u}_{0k}}{\partial r}\right) - k(k+1)\tilde{u}_{0k} = \tilde{f}_{0k}, \tag{4.9}$$

$$\left.\frac{\partial \tilde{u}_{0k}}{\partial r}\right|_{r=1} = \tilde{g}_{0k}^+, \quad \left.\frac{\partial \tilde{u}_{0k}}{\partial r}\right|_{r=-1} = \tilde{g}_{0k}^- \tag{4.10}$$

for $0 \le k \le p-1$. The functions $\tilde{u}_{0k}$ and $\tilde{f}_{0k}$ are defined by

$$\tilde{u}_{0k}(r) = \sum_{i=0}^{m-1} \tilde{u}_{i0k} T_i(r), \qquad \tilde{f}_{0k}(r) = \sum_{i=0}^{m-1} \tilde{f}_{i0k} T_i(r), \quad r \in [-1,1].$$

For each $0 < k \le p-1$, (4.9) and (4.10) can be solved in $\mathcal{O}(m)$ operations using the ultraspherical spectral method [44]. The case $k = 0$ is solved by the same technique with the additional linear constraint $\tilde{u}_{000} = 0$ to impose uniqueness of the global solution $u$.

Once the matrix of Chebyshev–Legendre coefficients $\tilde{U}_0 = (\tilde{u}_{i0k})$ has been computed, it can be converted to the Chebyshev–Fourier basis in $\mathcal{O}(mp\log^2 p)$ operations using the Legendre–Chebyshev tranform [55].

**4.4. Numerical examples.** In Figure 5(a) we plot a solution to the Helmholtz equation

$$\nabla^2 u + 20u = -80\sin(10x) \tag{4.11}$$

with Neumann boundary conditions $g(x,y,z) = 10x\cos(10x)$. The error between the computed and the exact solution to (4.11) is shown in Figure 5(c) and confirms the spectral convergence of our method. The computed solution is resolved to machine precision for $n \ge 50$. Our Helmholtz solver on the ball can be invoked in Ballfun via the `helmholtz` command.

```
rhs = ballfun(@(x,y,z)-80*sin(10*x));    % Right-hand side
bc = @(x,y,z)10*x.*cos(10*x);            % Boundary conditions
K = sqrt(20);                            % Wave number
n = 50;                                  % Spectral discretization
u = helmholtz(rhs, K, bc, n, 'neumann'); % Helmholtz solver
```

The execution times[2] to solve (4.11) for different discretization sizes $n$ are displayed in Figure 5(d). We can then solve Helmholtz's equation on a ball with one million degrees of freedom in a few seconds on a standard CPU.

As a second example we consider the advection-diffusion equation on the unit ball

$$\frac{\partial c}{\partial t} = D\nabla^2 c - v \cdot \nabla c, \tag{4.12}$$

where $D$ is the diffusion coefficient and $v$ is a divergence-free vector field. We choose $D = 1/5000$ and $v = \nabla \times [ze^{-5(x^2+y^2+z^2)}(x,y,z)]$ to satisfy the no-slip condition. The

[2]Timings were performed on a 3.3 GHz Intel Core i5 CPU using MATLAB 2018a without explicit parallelization.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

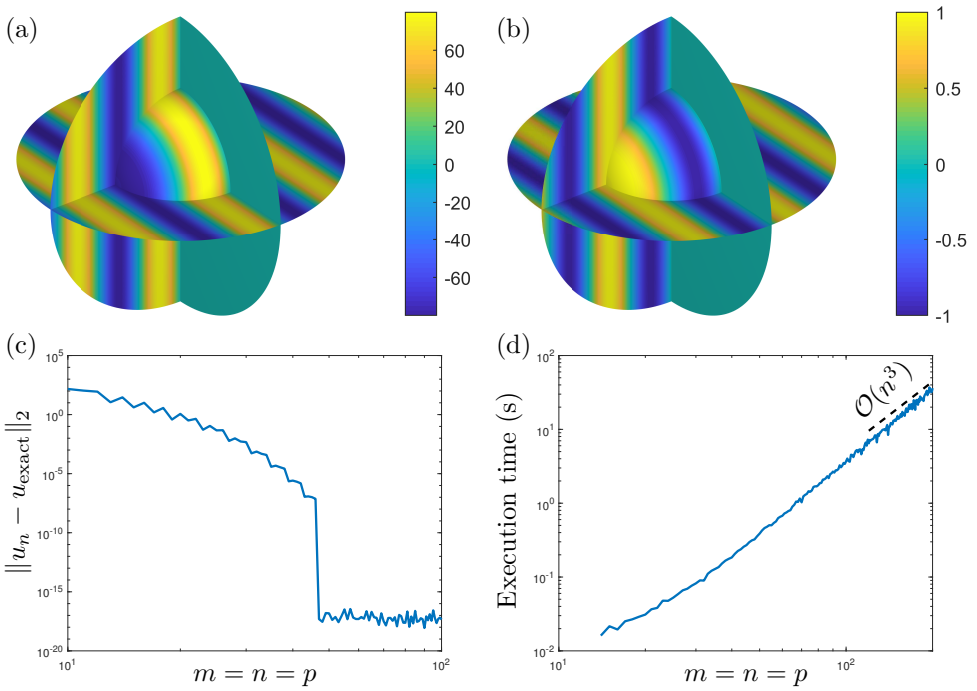


FIG. 5. *Function* $f(x, y, z) = -80\sin(10x)$ (a) *and solution* $u$ *to the Helmholtz equation* $\Delta u + 20u = f(x, y, z)$ *with Neumann boundary conditions* $g(x, y, z) = 10x\cos(10x)$ (b). *The solution* $u$ *is computed with a spectral discretization of* $m = n = p = 50$. (c) *Error in the* 2*-norm between the exact solution* $u_{exact} = \sin(10x)$ *and the computed solution obtained with the* `helmholtz` *command.* (d) *The computational timings.*

no-flux condition for $c$ reduces to $\partial c/\partial \vec{n} = 0$ at the boundary. We impose the initial condition $c^0(x, y, z) = -xe^{-5(x^2+y^2+z^2)}$ and solve (4.12) by using the implicit-explicit backward differentiation of order one (IMEX-BDF1) scheme. This yields the following Helmholtz equation:

$$\nabla^2 c^{n+1} + K^2 c^{n+1} = K^2 c^n + \frac{1}{D} v \cdot \nabla c^n, \quad \left.\frac{\partial c}{\partial \vec{n}}\right|_{\partial B(0,1)} = 0,$$

where $c^n$ denotes the solution at time $t = n\Delta t$, $\Delta t = 5 \times 10^{-2}$ is the time step, and $K^2 = -1/(D\Delta t)$. The solution $c$ to (4.12) at different times is illustrated in Figure 6.

**5. Vector-valued functions on the ball.** Ballfun is also designed to work with vector-valued functions defined on the unit ball as well as scalar-valued ones. Expressing vector-valued functions in spherical coordinates is inconvenient since the unit vectors in this coordinate system are singular at the poles of the ball [51]. Therefore, we express vector-valued functions in Cartesian coordinates as the components of the vector field are then themselves smooth functions. After using this convention, vector-valued functions introduce few complications from the point of view of approximation as each component is represented as an independent scalar function. A vector-valued function can be constructed in Ballfun as follows:

```
V = ballfunv(@(x,y,z) sin(x), @(x,y,z) x.*y, @(x,y,z) cos(z))
ballfunv object containing
ballfun object:
```

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

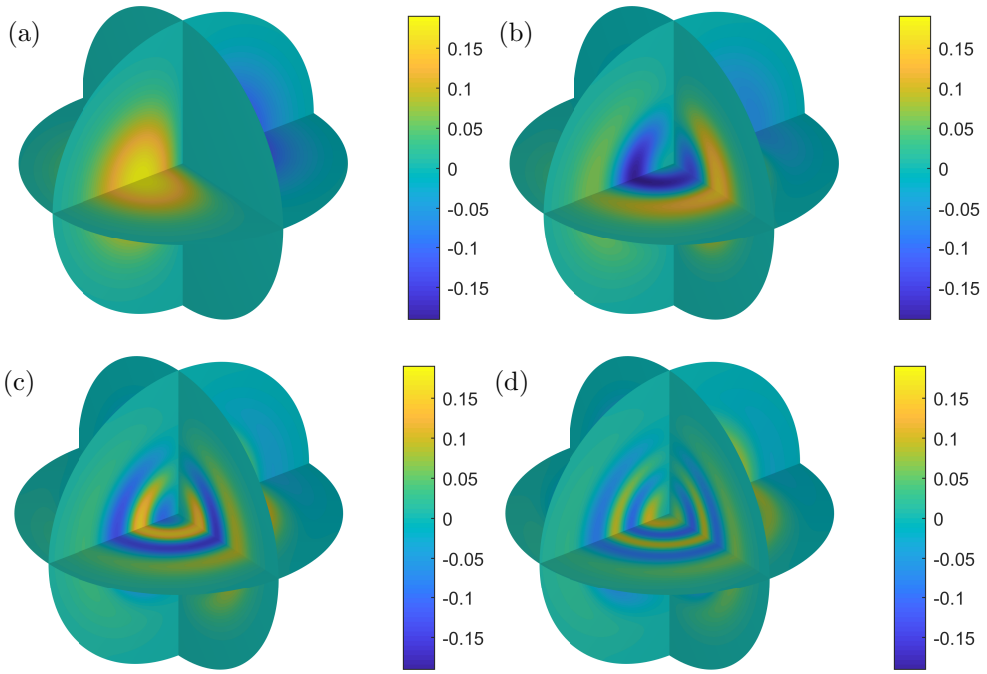


FIG. 6. *Solutions to the advection-diffusion equation at* $t = 0$ (a), $t = 5$ (b), $t = 10$ (c), *and* $t = 15$ (d).

```
    domain          r    lambda    theta
    unit ball      14        27       27
ballfun object:
    domain          r    lambda    theta
    unit ball       3         5        5
ballfun object:
    domain          r    lambda    theta
    unit ball      15         1       29
```

It can be seen that a vector field is stored as a `ballfunv` object, which consists of three `ballfun` objects corresponding to the three components in Cartesian coordinates. Each `ballfun` object has its own discretization in $r$, $\lambda$, and $\theta$.

**5.1. Vector calculus on the ball.** The more interesting side of vector-valued functions in Ballfun is the set of operations that can be implemented, which are potentially useful for applications. The standard operations for vector calculus such as the curl, the gradient, or the divergence are implemented in Ballfun in the `curl`, `gradient`, and `divergence` commands, respectively. Due to the way we represent vector-valued functions, we compute these operations in the Cartesian coordinate system. For example, the curl of a vector-valued function $\mathbf{V}$ can be written as

$$\nabla \times \mathbf{V} = \left[ \frac{\partial \mathbf{V}_z}{\partial y} - \frac{\partial \mathbf{V}_y}{\partial z}, \frac{\partial \mathbf{V}_x}{\partial z} - \frac{\partial \mathbf{V}_z}{\partial x}, \frac{\partial \mathbf{V}_y}{\partial x} - \frac{\partial \mathbf{V}_x}{\partial y} \right]^T.$$

The curl of $\mathbf{V}(x, y, z) = (\sin x, xz, \cos z)$ can be computed and displayed using the `quiver` command (see Figure 7):

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

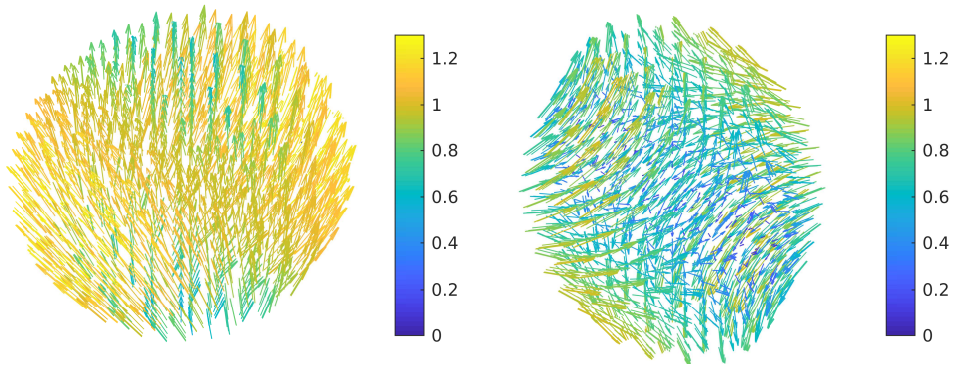


FIG. 7. *The vector-valued function* $V(x, y, z) = (\sin(x), xz, \cos(z))$ *(left) and its curl (right), plotted using the* `quiver` *command.*

```
W = curl( V ); % Compute curl of V
quiver( W )    % Plot vector field W using quiver
```

One can also verify basic vector calculus identities. For example, the divergence theorem asserts that the vector-valued function $\mathbf{V}$ satisfies

$$\iiint_{B(0,1)} (\nabla \cdot \mathbf{V})\, dV = \oiint_{\partial B} \mathbf{V} \cdot \vec{n}\, dS,$$

where $\vec{n}$ denotes the unit normal vector to $B(0,1)$. This theorem can be verified numerically in Ballfun by executing the following code:

```
lhs = sum3(divergence(V));  % Compute volume-integral of div( V )
nhat = spherefunv.unormal;  % Unit normal vector to surface of ball
Vbc = V(1,:,:, 'spherical');% Restrict V to the bdy of ball
rhs = sum2(dot(Vbc, nhat)); % Compute dot-product & surface-integral
```

The absolute error between `lhs` and `rhs` is $2.2204 \times 10^{-15}$.

**5.2. Poloidal-toroidal decomposition.** The poloidal-toroidal (PT) decomposition of a smooth divergence-free vector field expresses the field as the sum of two orthogonal fields. The PT decomposition is a well-known tool in fluid dynamics [37] and magnetohydrodynamics [6, 8, 9] simulations to analytically impose an incompressibility condition on flows in cylindrical and spherical geometries. In this section, we describe an algorithm for computing the PT decomposition of a smooth vector field in the ball.

Given a smooth divergence-free vector field, $\mathbf{V}$, defined on the ball, the PT decomposition [3] writes $\mathbf{V}$ as an orthogonal sum of a poloidal and toroidal field, i.e.,

$$\mathbf{V} = P + T, \qquad \int_{B(0,1)} P \cdot T\, dV = 0.$$

Here, there exist two poloidal and toroidal scalar-valued functions $\Phi$ and $\Psi$ such that $P = \nabla \times \nabla \times (r\Phi \mathbf{e}^r)$ and $T = \nabla \times (r\Psi \mathbf{e}^r)$, where $\mathbf{e}^r$ is the unit radial vector. It can be shown that $\Phi$ and $\Psi$ are unique up to the addition of an arbitrary function of $r$ [3].

A vector field $\mathbf{V}$ whose components are expressed in the Cartesian coordinate system $(\mathbf{V}_x, \mathbf{V}_y, \mathbf{V}_z)$ can be converted in spherical coordinates $(\mathbf{V}_r, \mathbf{V}_\lambda, \mathbf{V}_\theta)$ using the following identities:

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

$$\begin{aligned}\mathbf{V}_r &= \sin\theta(\cos\lambda\mathbf{V}_x + \sin\lambda\mathbf{V}_y) + \cos\theta\mathbf{V}_z,\\ \mathbf{V}_\lambda &= -\sin\lambda\mathbf{V}_x + \cos\lambda\mathbf{V}_y,\\ \mathbf{V}_\theta &= \cos\theta(\cos\lambda\mathbf{V}_x + \sin\lambda\mathbf{V}_y) - \sin\theta\mathbf{V}_z.\end{aligned}$$

Then, any poloidal and toroidal scalars for $\mathbf{V} = (\mathbf{V}_r, \mathbf{V}_\lambda, \mathbf{V}_\theta)$ satisfy the following relations [3]:

$$\nabla_1^2\Phi = -r\mathbf{V}_r, \tag{5.1}$$

$$\nabla_1^2\Psi = \frac{1}{\sin\theta}\left[\frac{\partial}{\partial\theta}(\mathbf{V}_\lambda\sin\theta) - \frac{\partial}{\partial\lambda}\mathbf{V}_\theta\right], \tag{5.2}$$

where $\nabla_1^2$ stands for the dimensionless surface Laplacian defined in the spherical coordinate system $(r, \lambda, \theta)$ by

$$\nabla_1^2 = \frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\lambda^2}.$$

After multiplying by $\sin^2\theta$ to remove the singularities, (5.1) and (5.2) become

$$\sin\theta\cos\theta\frac{\partial\Phi}{\partial\theta} + \sin^2\theta\frac{\partial^2\Phi}{\partial\theta^2} + \frac{\partial^2\Phi}{\partial\lambda^2} = -r\sin^2\theta\mathbf{V}_r, \tag{5.3}$$

$$\sin\theta\cos\theta\frac{\partial\Psi}{\partial\theta} + \sin^2\theta\frac{\partial^2\Psi}{\partial\theta^2} + \frac{\partial^2\Psi}{\partial\lambda^2} = -\sin\theta\left[\partial_\theta(\mathbf{V}_\lambda\sin\theta) - \partial_\lambda\mathbf{V}_\theta\right]. \tag{5.4}$$

Moreover, any smooth function $u$ on the unit ball has a unique interpolant $\tilde{u}$ (section 1) of the form

$$\tilde{u}(r,\lambda,\theta) \approx \sum_{i=0}^{m-1}\sum_{j=-n/2}^{n/2-1}\sum_{k=-p/2}^{p/2-1}\alpha_{ijk}T_i(r)e^{\mathbf{i}j\lambda}e^{\mathbf{i}k\theta}.$$

Thus, (5.3) and (5.4) decouple in $\lambda$ and $r$ with this basis. However, these equations are not well defined since $P$ and $T$ are unique up to addition of arbitrary functions of $r$. Then, (5.3) and (5.4) are solved numerically with the condition that the zeroth Fourier mode in $\lambda$, and $\theta$ is equal to zero. This is equivalent to $\alpha_{i00} = 0$ for all $0 \le i \le m-1$.

Let $M_{\sin\cos}$ and $M_{\sin^2}$ be the multiplication matrices for $\sin\theta\cos\theta$ and $\sin^2\theta$ in the Fourier basis, $D_p$ the matrix of differentiation with respect to $\theta$, $F$ the tensor of CFF coefficients of $-r\sin^2\theta v_r$ and $G$ the tensor of CFF coefficients of $-\sin\theta[\partial_\theta(v_\lambda\sin\theta) - \partial_\lambda v_\theta]$. For example,

$$D_p = \operatorname{diag}\left(\left[-\frac{p}{2}\mathbf{i}, \ldots, -\mathbf{i}, 0, \mathbf{i}, \ldots, \frac{p}{2}\mathbf{i}\right]\right).$$

Equations (5.3) and (5.4) are discretized into

$$\begin{aligned}\left(M_{\sin\cos}D_p + M_{\sin^2}D_p^2 - j^2I\right)P(i,j,:) &= F(i,j,:),\\ \left(M_{\sin\cos}D_p + M_{\sin^2}D_p^2 - j^2I\right)T(i,j,:) &= G(i,j,:)\end{aligned}$$

for $0 \le i \le m-1$ and $-n/2 \le j \le n/2-1$. $P(i,j,:)$ denotes the vector of Fourier coefficients in $\theta$. These equations are of the form

$$AX(i,j,:) = B(i,j,:),$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

where $A$ is a sparse banded matrix with bandwidth $b = 4$, which is the Fourier number of $\sin\theta\cos\theta$ and $\sin^2\theta$. Then, by band Gaussian elimination [32], it can be solved in $\mathcal{O}(b^2p)$ operations. Thus, the complexity of the PT algorithm is $\mathcal{O}(mnp)$, which is linear in the number of interpolation points over the ball.

As an example we consider the induction equation [20, Chap. 2], which is one of the equations arising in magnetohydrodynamics and resulting from Maxwell's equations,

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{V} \times \mathbf{B}) + D\nabla^2\mathbf{B}, \tag{5.5}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{5.6}$$

where $\mathbf{B}$ denotes the magnetic field, $\mathbf{V} = \nabla \times [e^{-5(x^2+y^2+z^2)}(x^2, y^2, xz)]$ is the velocity of particles, and $D = 1/3000$ is the diffusion constant. We use the PT decomposition to ensure that the divergence-free condition on $\mathbf{B}$ is satisfied and decouple (5.5) into the following equations on the poloidal and toroidal scalars $\Phi_{\mathbf{B}}$ and $\Psi_{\mathbf{B}}$:

$$\frac{\partial \Phi_{\mathbf{B}}}{\partial t} = \Phi_{\nabla\times(\mathbf{V}\times\mathbf{B})} + D\nabla^2\Phi_{\mathbf{B}}, \tag{5.7}$$

$$\frac{\partial \Psi_{\mathbf{B}}}{\partial t} = \Psi_{\nabla\times(\mathbf{V}\times\mathbf{B})} + D\nabla^2\Psi_{\mathbf{B}}. \tag{5.8}$$

The two equations above are decoupled and independent, and therefore the use of the PT decomposition does not induce any additional error in the solution. Then, according to the IMEX-BDF1 time-stepping scheme (see subsection 4.4), at each time step we compute the PT decomposition of the nonlinear term $\nabla \times (\mathbf{V} \times \mathbf{B})$ and solve two Helmholtz equations:

$$\nabla^2\Phi_{\mathbf{B}}^{n+1} + K^2\Phi_{\mathbf{B}}^{n+1} = K^2\Phi_{\mathbf{B}}^{n} + \frac{1}{D}\Phi^n_{\nabla\times(\mathbf{V}\times\mathbf{B})},$$

$$\nabla^2\Psi_{\mathbf{B}}^{n+1} + K^2\Psi_{\mathbf{B}}^{n+1} = K^2\Psi_{\mathbf{B}}^{n} + \frac{1}{D}\Psi^n_{\nabla\times(\mathbf{V}\times\mathbf{B})},$$

where $\Phi^n_{\mathbf{B}}$ (resp. $\Psi^n_{\mathbf{B}}$) denotes the magnetic poloidal (resp., toroidal) scalar at time $t = n\Delta t$, $\Delta t = 5 \times 10^{-2}$ is the time step, and $K^2 = -1/(D\Delta t)$. We choose the initial magnetic field $\mathbf{B} = \nabla \times [ze^{-5(x^2+y^2+z^2)}(x, y, z)]$ and impose homogeneous Dirichlet boundary conditions on the poloidal and toroidal scalars, which are computed at each time step using the following code snippet:

```
B = ballfunv.PT2ballfunv(Phi_B, Psi_B);% Compute B from P and T
N = curl(cross(V, B));                 % Nonlinear term
[Phi_N, Psi_N] = PTdecomposition(N);   % PT decomposition of N
% Solve the  toroidal and poloidal equation
Phi_B = helmholtz(K^2*Phi_B+Phi_N/D, K, @(x,y,z)0, 100);
Psi_B = helmholtz(K^2*Psi_B+Psi_N/D, K, @(x,y,z)0, 100);
```

The magnetic field $\mathbf{B}$ is illustrated at different time steps in Figure 8.

**5.3. Helmholtz–Hodge decomposition.** The Helmholtz–Hodge decomposition has been an important tool in computational fluid dynamics since the introduction of projection methods by Chorin [14, 15, 16] to solve the Navier–Stokes equations for incompressible fluids. The decomposition is then used to preserve the divergence-free property of the velocity field during the computation of the solution. Applications of the Helmholtz–Hodge decomposition also arise in computer graphics and visualization of incompressible fluids such as water [11, 46, 52]. The Helmholtz–Hodge decomposition has also been exploited in the field of computer vision and robotics to analyze

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

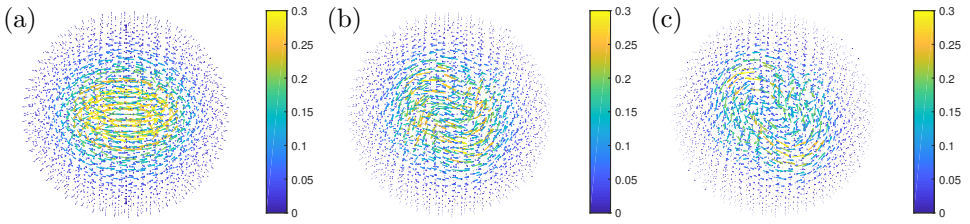


FIG. 8. *Solution to the induction equation* (5.5)–(5.6) *at time* $t = 0$ (a), $t = 1$ (b), *and* $t = 2$ (c).

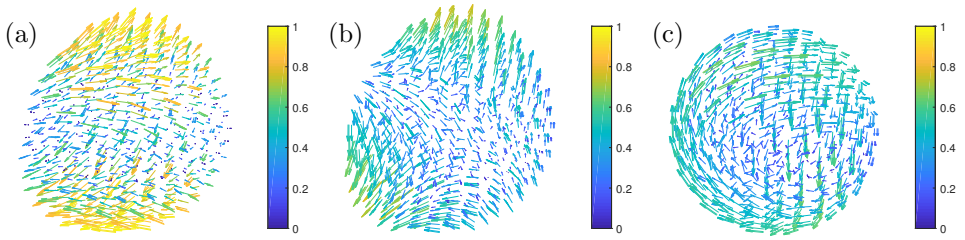


FIG. 9. *The vector field* $\mathbf{V}(x, y, z) = (\cos(xy)z, \sin(xz), yz)$ (a), *together with its Helmholtz–Hodge decomposition* $\nabla f$ (b) *and* $\Psi$ (c). *The decomposition is computed using* `HelmholtzDecomposition` *and plotted with the* `quiver` *command.*

cardiac videos [33] and find singularities in fingerprints images [27]. A survey of applications is available in [7]. Ballfun has a `HelmholtzDecomposition` command, which computes and returns the Helmholtz–Hodge decomposition of a smooth vector field.

The Helmholtz–Hodge decomposition [7] states that any smooth vector field $v$ on the unit ball can be decomposed into a sum of a solenoidal and irrotational fields

$$\mathbf{V} = \nabla f + \psi, \tag{5.9}$$

where $\psi$ is a divergence-free vector field. Moreover, this decomposition can be made unique by imposing that the incompressible component, $\psi$, is tangent to the boundary. This condition is equivalent to a Neumann boundary condition on the scalar function $f$. That is, $\vec{n} \cdot \nabla f = \vec{n} \cdot \mathbf{V}$ [7]. The first step of the algorithm implemented in Ballfun consists of taking the divergence of $\mathbf{V}$ in (5.9) to obtain the Poisson equation

$$\nabla^2 f = \nabla \cdot \mathbf{V}$$

with the Neumann boundary conditions given by

$$\vec{n} \cdot \nabla f = \left. \frac{\partial f}{\partial r} \right|_{\partial B} = \vec{n} \cdot \mathbf{V}.$$

We then obtain the incompressible component using the following equality:

$$\psi := \mathbf{V} - \nabla f.$$

Finally, the poloidal and toroidal scalars of $\psi$ are computed using the algorithm described in subsection 5.2, and the command `HelmholtzDecomposition` returns the scalar function $f$ together with the poloidal and toroidal scalars of $\psi$.

Figure 9 shows the Helmholtz–Hodge decomposition of the vector-valued function $\mathbf{V}(x, y, z) = (\cos(xy)z, \sin(xz), yz)$ computed by Ballfun using the following code:

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

```
% Define the vector field V
V = ballfunv(@(x,y,z)cos(x.*y).*z, @(x,y,z)sin(x.*z), @(x,y,z)y.*z);
% Compute the Helmholtz-Hodge decomposition of V
[f, Ppsi, Tpsi] = HelmholtzDecomposition( V );
% Recover Psi from it poloidal and toroidal scalars
Psi = ballfunv.PT2ballfunv(Ppsi,Tpsi);
```

**6. Conclusions.** The analogue of the DFS method for the ball is exploited to impose BMC structure on functions and represent them by CFF series. A collection of fast and spectrally accurate algorithms is developed for differentiation, rotation, solving the Helmholtz equation, vector calculus, PT decomposition, and Helmholtz–Hodge decomposition. These ideas have been implemented in Ballfun, which is part of the freely available software Chebfun.

**Acknowledgments.** We thank Vassilios Dallas and Jonasz Słomka for discussions on the poloidal-toroidal and Helmholtz–Hodge decomposition. We also thank Nick Trefethen, Heather Wilber, and Grady Wright for comments on the Ballfun software and the paper. We thank the referees for their time and expert reviews.

REFERENCES

[1] G. B. Arfken, *Mathematical Methods for Physicists*, 3rd ed., Academic Press, Orlando, FL, 1985.
[2] J. L. Aurentz and L. N. Trefethen, *Chopping a Chebyshev series*, ACM Trans. Math. Software, 43 (2017), pp. 33:1–33:21.
[3] G. Backus, *Poloidal and toroidal fields in geomagnetic field modeling*, Rev. Geophys., 24 (1986), pp. 75–109.
[4] R. H. Bartels and G. W. Stewart, *Solution of the matrix equation $AX+XB=C$*, Commun. ACM, 15 (1972), pp. 820–826.
[5] Z. Battles and L. N. Trefethen, *An extension of MATLAB to continuous functions and operators*, SIAM J. Sci. Comput., 25 (2004), pp. 1743–1770.
[6] E. R. Benton and K. A. Whaler, *Rapid diffusion of the poloidal geomagnetic field through the weakly conducting mantle: A perturbation solution*, Geophys. J. Internat., 75 (1983), pp. 77–100.
[7] H. Bhatia, G. Norgard, V. Pascucci, and P.-T. Bremer, *The Helmholtz–Hodge decomposition–A survey*, IEEE Trans. Visual. Comput. Graph., 19 (2013), pp. 1386–1404.
[8] P. Boronski and L. S. Tuckerman, *Poloidal–toroidal decomposition in a finite cylinder. I: Influence matrices for the magnetohydrodynamic equations*, J. Comput. Phys., 227 (2007), pp. 1523–1543.
[9] P. Boronski and L. S. Tuckerman, *Poloidal–toroidal decomposition in a finite cylinder:* II. *Discretization, regularization and validation*, J. Comput. Phys., 227 (2007), pp. 1544–1566.
[10] J. P. Boyd, *The choice of spectral functions on a sphere for boundary and eigenvalue problems: A comparison of Chebyshev, Fourier and associated Legendre expansions*, Monthly Weather Rev., 106 (1978), pp. 1184–1191.
[11] R. Bridson, *Fluid Simulation for Computer Graphics*, AK Peters/CRC Press, Boca Raton, FL, 2015.
[12] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. P. Brown, *Dedalus: A flexible framework for numerical simulations with spectral methods*, Phys. Rev. Res., 2 (2020), 023068.
[13] H.-B. Cheong, *Application of double Fourier series to the shallow-water equations on a sphere*, J. Comput. Phys., 165 (2000), pp. 261–287.
[14] A. J. Chorin, *A numerical method for solving incompressible viscous flow problems*, J. Comput. Phys., 2 (1967), pp. 12–26.
[15] A. J. Chorin, *Numerical solution of the Navier–Stokes equations*, Math. Comp., 22 (1968), pp. 745–762.
[16] A. J. Chorin, *On the convergence of discrete approximations to the Navier–Stokes equations*, Math. Comp., 23 (1969), pp. 341–353.
[17] C. Clenshaw, *A note on the summation of Chebyshev series*, Math. Comp., 9 (1955), pp. 118–120.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

[18] J. Coiffier, *Fundamentals of Numerical Weather Prediction*, Cambridge University Press, Cambridge, UK, 2011.
[19] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.
[20] P. A. Davidson, *An Introduction to Magnetohydrodynamics*, Cambridge University Press, Cambridge, UK, 2002.
[21] T. A. Driscoll, N. Hale, and L. N. Trefethen, *Chebfun Guide*, Pafnuty Publications, Oxford, UK, 2014, http://www.chebfun.org/docs/guide/.
[22] C. F. Dunkl and Y. Xu, *Orthogonal Polynomials of Several Variables*, Cambridge University Press, Cambridge, UK, 2014.
[23] N. Flyer and G. B. Wright, *A radial basis function method for the shallow water equations on a sphere*, Proc. A, 471 (2009), pp. 1–28.
[24] B. Fornberg, *A pseudospectral approach for polar and spherical geometries*, SIAM J. Sci. Comput., 16 (1995), pp. 1071–1081.
[25] B. Fornberg and D. Merrill, *Comparison of finite difference-and pseudospectral methods for convective flow over a sphere*, Geophys. Res. Lett., 24 (1997), pp. 3245–3248.
[26] D. Fortunato and A. Townsend, *Fast Poisson solvers for spectral methods*, IMA J. Numer. Anal., (2019).
[27] H. Gao, M. K. Mandal, G. Guo, and J. Wan, *Singular point detection using discrete Hodge Helmholtz decomposition in fingerprint images*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2010, pp. 1094–1097.
[28] J. D. Gardiner, A. J. Laub, J. J. Amato, and C. B. Moler, *Solution of the Sylvester matrix equation* $AXB^T + CXD^T = E$, ACM Trans. Math. Software, 18 (1992), pp. 223–231.
[29] I. M. Gelfand, R. A. Minlos, and Z. Y. Shapiro, *Representations of the Rotation and Lorentz Groups and their Applications*, Courier Dover Publications, New York, 2018.
[30] W. M. Gentleman, *Implementing Clenshaw–Curtis quadrature,* I. *Methodology and experience*, Commun. ACM, 15 (1972), pp. 337–342.
[31] W. M. Gentleman, *Implementing Clenshaw–Curtis quadrature,* II. *Computing the cosine transformation*, Commun. ACM, 15 (1972), pp. 343–346.
[32] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 2012.
[33] Q. Guo, M. K. Mandal, G. Liu, and K. M. Kavanagh, *Cardiac video analysis using Hodge–Helmholtz field decomposition*, Comput. Biol. Med., 36 (2006), pp. 1–20.
[34] B. Hashemi and L. N. Trefethen, *Chebfun in three dimensions*, SIAM J. Sci. Comput., 39 (2017), pp. C341–C363.
[35] W. Heinrichs, *Spectral collocation schemes on the unit disc*, J. Comput. Phys., 199 (2004), pp. 66–86.
[36] R. Hollerbach, *Magnetohydrodynamic Ekman and Stewartson layers in a rotating spherical shell*, Proc. A, 444 (1994), pp. 333–346.
[37] S. Horn and O. Shishkina, *Toroidal and poloidal energy in rotating Rayleigh–Bénard convection*, J. Fluid Mech., 762 (2015), pp. 232–255.
[38] R. Kerswell, *Recent progress in understanding the transition to turbulence in a pipe*, Nonlinearity, 18 (2005), R17.
[39] W. Kuang and J. Bloxham, *Numerical modeling of magnetohydrodynamic convection in a rapidly rotating spherical shell: Weak and strong field dynamo action*, J. Comput. Phys., 153 (1999), pp. 51–81.
[40] W. Lowrie, *Fundamentals of Geophysics*, Cambridge University Press, Cambridge, UK, 2007.
[41] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*, CRC Press, Boca Raton, FL, 2002.
[42] P. E. Merilees, *The pseudospectral approximation applied to the shallow water equations on a sphere*, Atmosphere, 11 (1973), pp. 13–20.
[43] S. Olver, G. Goretkin, R. M. Slevinsky, and A. Townsend, *Approxfun*, 2019, https://github.com/JuliaApproximation/ApproxFun.jl.
[44] S. Olver and A. Townsend, *A fast and well-conditioned spectral method*, SIAM Rev., 55 (2013), pp. 462–489.
[45] S. A. Orszag, *Fourier series on spheres*, Monthly Weather Rev., 102 (1974), pp. 56–75.
[46] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, Morgan Kaufmann, Cambridge, MA, 2016.
[47] D. Ruiz-Antolin and A. Townsend, *A nonuniform fast Fourier transform based on low rank approximation*, SIAM J. Sci. Comput., 40 (2018), pp. A529–A547.
[48] E. Serre and J. Pulicani, *A three-dimensional pseudospectral method for rotating flows in a cylinder*, Comput. Fluids, 30 (2001), pp. 491–519.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[49] R. M. Slevinsky, *Fast and backward stable transforms between spherical harmonic expansions and bivariate Fourier series*, Appl. Comput. Harmon. Anal., 47 (2019), pp. 585–606.
[50] W. F. Spotz, M. A. Taylor, and P. N. Swarztrauber, *Fast shallow-water equation solvers in latitude-longitude coordinates*, J. Comput. Phys., 145 (1998), pp. 432–444.
[51] P. N. Swarztrauber, *The approximation of vector functions and their derivatives on the sphere*, SIAM J. Numer. Anal., 18 (1981), pp. 191–210.
[52] J. Tan and X. Yang, *Physically-based fluid animation: A survey*, Sci. China Inf. Sci., 52 (2009), pp. 723–740.
[53] A. Townsend and S. Olver, *The automatic solution of partial differential equations using a global spectral method*, J. Comput. Phys., 299 (2015), pp. 106–123.
[54] A. Townsend and L. N. Trefethen, *An extension of Chebfun to two dimensions*, SIAM J. Sci. Comput., 35 (2013), pp. C495–C518.
[55] A. Townsend, M. Webb, and S. Olver, *Fast polynomial transforms based on Toeplitz and Hankel matrices*, Math. Comput., 87 (2018), pp. 1913–1934.
[56] A. Townsend, H. Wilber, and G. B. Wright, *Computing with functions in spherical and polar geometries* I. *The sphere*, SIAM J. Sci. Comput., 38 (2016), pp. C403–C425.
[57] L. N. Trefethen, *Spectral Methods in MATLAB*, SIAM, Philadelphia, PA, 2000.
[58] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, PA, 2013.
[59] G. M. Vasil, D. Lecoanet, K. J. Burns, J. S. Oishi, and B. P. Brown, *Tensor calculus in spherical coordinates using Jacobi polynomials. Part-*I*: Mathematical analysis and derivations*, J. Comput. Phys., 3 (2019), 100013.
[60] H. Wilber, A. Townsend, and G. B. Wright, *Computing with functions in spherical and polar geometries* II. *The disk*, SIAM J. Sci. Comput., 39 (2017), pp. C238–C262.
[61] G. B. Wright, N. Flyer, and D. A. Yuen, *A hybrid radial basis function–pseudospectral method for thermal convection in a* 3*-D spherical shell*, Geochem. Geophys. Geosyst., 11 (2010), Q07003.
[62] G. B. Wright, M. Javed, H. Montanelli, and L. N. Trefethen, *Extension of Chebfun to periodic functions*, SIAM J. Sci. Comput., 37 (2015), pp. C554–C573.
[63] S. Y. K. Yee, *Studies on Fourier series on spheres*, Monthly Weather Rev., 108 (1980), pp. 676–678.
[64] K. Zhang and G. Schubert, *Magnetohydrodynamics in rapidly rotating spherical systems*, Ann. Rev. Fluid Mech., 32 (2000), pp. 409–443.
[65] K.-K. Zhang and F. Busse, *Convection driven magnetohydrodynamic dynamos in rotating spherical shells*, Geophys. Astrophys. Fluid Dynam., 49 (1989), pp. 97–116.

Downloaded 10/27/22 to 128.84.126.210 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.