Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 369 (2020) 113191
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Hierarchical matrix approximations for space-fractional diffusion equations

Wajih Boukaram[a], Marco Lucchesi[a], George Turkiyyah[b], Olivier Le Maître[c], Omar Knio[a,*], David Keyes[a]

[a] King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia
[b] Department of Computer Science, American University of Beirut, Beirut, Lebanon
[c] Centre de Mathématiques Appliquées, CNRS, Inria, Ecole Polytechnique, Palaiseau, France

Received 18 January 2020; received in revised form 27 May 2020; accepted 29 May 2020
Available online xxxx

Abstract

Space fractional diffusion models generally lead to dense discrete matrix operators, which lead to substantial computational challenges when the system size becomes large. For a state of size N, full representation of a fractional diffusion matrix would require $O(N^2)$ memory storage requirement, with a similar estimate for matrix–vector products. In this work, we present $\mathcal{H}^2$ matrix representation and algorithms that are amenable to efficient implementation on GPUs, and that can reduce the cost of storing these operators to $O(N)$ asymptotically. Matrix–vector multiplications can be performed in asymptotically linear time as well. Performance of the algorithms is assessed in light of 2D simulations of space fractional diffusion equation with constant diffusivity. Attention is focused on smooth particle approximation of the governing equations, which lead to discrete operators involving explicit radial kernels. The algorithms are first tested using the fundamental solution of the unforced space fractional diffusion equation in an unbounded domain, and then for the steady, forced, fractional diffusion equation in a bounded domain. Both matrix-inverse and pseudo-transient solution approaches are considered in the latter case. Our experiments show that the construction of the fractional diffusion matrix, the matrix–vector multiplication, and the generation of an approximate inverse pre-conditioner all perform very well on a single GPU on 2D problems with N in the range $10^5 - 10^6$. In addition, the tests also showed that, for the entire range of parameters and fractional orders considered, results obtained using the $\mathcal{H}^2$ approximations were in close agreement with results obtained using dense operators, and exhibited the same spatial order of convergence. Overall, the present experiences showed that the $\mathcal{H}^2$ matrix framework promises to provide practical means to handle large-scale space fractional diffusion models in several space dimensions, at a computational cost that is asymptotically similar to the cost of handling classical diffusion equations.

Keywords: Fractional diffusion; Smooth particle approximation; Hierarchical matrix; Linear complexity; GPU

1. Introduction

Nonlocal continuum models, expressed as fractional differential equations, have gained significant popularity in recent years, as they have shown great success in representing the behavior of a variety of systems in

* Corresponding author.
E-mail address: omar.knio@kaust.edu.sa (O. Knio).

https://doi.org/10.1016/j.cma.2020.113191
0045-7825/© 2020 Elsevier B.V. All rights reserved.

scientific and engineering application domains [1]. Nonlocal models allow the representation of solutions that exhibit more singular and anomalous behavior than is possible with local continuum PDE models, which have long been the mainstay of scientific and engineering modeling. One of the most successful fractional differential equations is perhaps the space-fractional diffusion equation, which uses a fractional Laplacian operator that generalizes its classical counterpart. This fractional diffusion equation appears as a canonical model for transport problems involving anomalous diffusion that is of great relevance to industrial and environmental applications [2]. Applications from many other disciplines including image processing, finance, and many others [1,3,4] have also come to rely on fractional Laplacian operators. In its simplest form, the equation assumes a uniform diffusivity coefficient in space and a constant fractional order derivative, however non-homogeneous coefficients in general geometries and varying fractional orders are also of practical interest.

Numerical approximations for the fractional derivatives that appear in the fractional diffusion equations have received much attention in the literature [5]. Many different equivalent definitions of the fractional Laplacian have been presented [6] and used as the starting point for appropriate discretizations of the fractional operator. Finite-difference and variational methods are often adopted to numerically discretize the equations [7–18]. Particle-based methods, using random walk approaches [19–24], have also been used to simulate the equations. Smoothed particle approximations have been recently proposed [25] and shown to be quite flexible in discretizing and simulating the fractional diffusion equations.

The practical discretization schemes proposed, however, share serious computational difficulties that limit their scalability. The resulting discrete operator they produce is fully dense due to the non local nature of the underlying equations and hence requires $O(N^2)$ memory and $O(N^2)$ operations for the core matrix vector multiplication operation. For simple cartesian geometries with constant coefficients, translation invariance characteristics can be exploited [26] to reduce the storage and operator application costs. However, the general setting inevitably leads to a quadratic growth in computational resource requirements. This must be tamed if the simulation of fractional diffusion, or other fractional differential equations, is to be done at scale. Hierarchical matrix approximations have been proposed as algebraic representations of the discretized fractional operators. In 1D, $\mathcal{H}$-matrix approximations [27] were shown to be effective and were used in a geometric multigrid solver. The $\mathcal{H}$-matrix approximability of the discrete operators in two spatial dimensions was studied in [28,29] starting from finite difference approximations of the fractional operator. Our work here compresses the fractional operator using the more memory-efficient $\mathcal{H}^2$ representation, and develops a performant GPU implementation to demonstrate it 2D. We also construct, algebraically, an approximate inverse, as a preconditioner for a Krylov solver.

Starting from a smoothed particle discretization of the fractional diffusion operator, we construct asymptotically optimal $\mathcal{H}^2$-matrix approximations of the resulting matrix operator. We show that the memory requirements of the discretized fractional operator grow only as $O(N)$ where N is the number of particles in the discretization, and that the cost of matrix–vector multiplication also grows at the same optimal linear rate. In addition, the constants that appear in these complexity estimates depend on the ranks of blocks in the matrix, which grow only very weakly with the desired accuracy of the matrix approximation, $k \sim \log |\epsilon|^c$ where k is a representative rank of low rank blocks of the $\mathcal{H}$ matrix, ϵ is the accuracy of the hierarchical matrix relative to the fully dense representation, and c is a constant that depends on the spatial dimension of the problem. Our numerical experiments, with different problem sizes and fractional orders, show the broad applicability of $\mathcal{H}^2$-matrix representations for fractional diffusion problems.

In practice, we show that these $\mathcal{H}^2$ representations are effective in multiple spatial dimensions, not only in their optimal asymptotic growth, but importantly in their practicality on modern workstations. The operators can be compressed by more than 3 orders of magnitude relative to a dense representation, and generated in under a couple of minutes, to reasonably high accuracy, on problems of size 2M on a standard scientific workstation. We also show that a higher order variant of Newton–Schulz iteration can be used to generate an approximate inverse of the discretized operator, which can be used as an effective preconditioner for the fractional elliptic Poisson problem.

Another major benefit of the $\mathcal{H}^2$ matrix approximations is that their small, and asymptotically optimal, storage requirements make them amenable to an effective GPU implementation. Modern scientific workstations generally feature manycore GPU accelerators, and algorithms that do not effectively take advantage of these architectures are unlikely to be competitive for scientific and financial applications. Modern GPU architectures feature decreasing ratios of memory bandwidth to processing power, smaller amounts of fast memory per processing core, and substantial latencies for accessing data in deep memory [30]. Competitive algorithms must therefore be able to orchestrate their computations for effective execution in this environment. Through the efficient compression of the discretized operator via its $\mathcal{H}^2$ representation, matrix data can be made to reside high on the memory hierarchy to allow substantial efficiencies to be realized, beyond the reduction in operations count to produce. Our numerical experiments demonstrate the effectiveness of the representation on GPUs. For example, on problems of size 2M, the operator application can be performed in under 40 ms on a modest NVIDIA P100 GPU.

The rest of this paper is organized as follows. Section 2 presents the particle approximation of the fractional diffusion equation. Section 3 describes the hierarchical $\mathcal{H}^2$ representation, the algorithm for generating an $\mathcal{H}^2$ representation of the operator, as well as the algorithms for matrix vector multiplication and construction of an approximate inverse. Section 4 shows the computational performance of these algorithms on various test problems, including memory footprint and processing times on CPUs and GPUs, to show scalability. The effect of desired accuracy on rank growth, and hence memory footprint, is shown as well. The effectiveness of different hierarchical approximate inverses, constructed to different accuracies, when used as preconditioners in a Krylov solver is also presented. Section 5 shows the results on two test applications. Transient, pseudo-transient, and steady state solutions are obtained and analyzed to show the approximation quality of the computed solution for various values of the fractional order of the operator, and to demonstrate the applicability of the hierarchical matrix representation across problem parameters. We conclude and outline future work in Section 6.

2. The fractional diffusion equation and its particle approximation

In this section, we briefly outline the formulation of the mathematical problem, and its particle approximation.

2.1. The fractional Laplacian

We shall focus on the simulation of the multi-dimensional fractional diffusion equation,

$$\frac{\partial u}{\partial t} = -\nabla \cdot \mathbf{Q}^{\beta}, \tag{1}$$

where

$$\mathbf{Q}^{\beta}(\mathbf{x},t) \equiv -\mathcal{D}\nabla^{\beta}u(\mathbf{x},t) = -\mathcal{D}\frac{2^{\alpha-1}\Gamma\left(\frac{d+\alpha}{2}\right)}{\pi^{\frac{d}{2}}\Gamma\left(\frac{2-\alpha}{2}\right)}\ \text{p.v.}\int_{\mathbb{R}^d}\frac{\mathbf{y}}{|\mathbf{y}|^{\alpha+d}}u(\mathbf{x}+\mathbf{y},t)\,\mathrm{d}V(\mathbf{y}), \tag{2}$$

is the fractional diffusion flux of order $\beta \in (0,1)$, $\alpha \equiv \beta + 1$, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ with $d \geq 1$, $\mathrm{d}V(\mathbf{y})$ is the measure in $\mathbb{R}^d$, $\mathcal{D}$ is the diffusivity, ∇^{β} denotes the fractional gradient operator of order β, and "p.v." indicates a principal value integral.

Assuming $\mathcal{D} \equiv 1$, (1) reduces to:

$$\frac{\partial u}{\partial t} = \Delta^{\alpha}u, \tag{3}$$

where $\Delta^{\alpha}u$ is the fractional Laplacian operator of order α [31]:

$$\Delta^{\alpha}u = \frac{\alpha 2^{\alpha-1}\Gamma\left(\frac{d+\alpha}{2}\right)}{\pi^{\frac{d}{2}}\Gamma\left(\frac{2-\alpha}{2}\right)}\ \text{p.v.}\int_{\mathbb{R}^d}\frac{u(\mathbf{y})-u(\mathbf{x})}{|\mathbf{y}-\mathbf{x}|^{d+\alpha}}\,\mathrm{d}V(\mathbf{y}). \tag{4}$$

Note that in the limit $\alpha \to 2$, the fractional Laplacian Δ^{α} defined in (4) coincides with the classical Laplacian. Also note that in some of the literature the fractional Laplacian is denoted as $\Delta^{\alpha/2}$, or as power of the Laplacian with an exponent falling between 0 and 1. Because we have started from the flux expression, involving a fractional operator of **order** $0 < \beta < 1$, we find it more convenient to denote the resulting diffusion in terms of a fractional operator of order $\alpha = \beta + 1$, with $1 < \alpha < 2$. This also provides a natural way to eventually extend the formulation to variable properties.

For an infinite domain with no internal boundaries, the fundamental solution of (4) is available in integral form [31] for the general d-dimensional case; see Appendix A.

2.2. Particle approximation

Smooth particle methods consist in discretizing the n-dimensional spatial domain using a finite set of particles. The ith particle is defined by its position $\mathbf{x}_i$, volume V_i, and strength u_i. The particle representation of a function, $u(\mathbf{x})$, is expressed as [32–36]:

$$u(\mathbf{x}) \approx u_\varsigma(\mathbf{x}) \approx \sum_{i\in\mathcal{I}} V_i u_i \eta_\varsigma(\mathbf{x}-\mathbf{x}_i), \qquad \eta_\varsigma(\mathbf{x}) = \frac{1}{\varsigma^d}\eta\left(\frac{\mathbf{x}}{\varsigma}\right), \tag{5}$$

where $\mathcal{I} \subseteq \mathbb{Z}$ is a finite index set, η is a radial kernel of unit mass, and ς is the so-called smoothing parameter. In this work, we adopt the second-order exponential kernel [37,38]:

$$\eta(\mathbf{x}) = \frac{1}{\pi^{\frac{d}{2}}} \exp\left(-|\mathbf{x}|^2\right). \tag{6}$$

Recently, Lucchesi et al. [25] exploited the regularized particle representation in (5) to construct different methods to approximate the fractional Laplacian (4) and the fractional diffusion flux (2). These included various variants of the so-called particle strength exchange (PSE) algorithms [39–43], a diffusion-velocity method [44], as well as a methodology based on direct (fractional) differentiation [36] of the particle representation in (5). Whereas the various approaches explored in [25] all led to consistent approximations, the resulting schemes exhibited different properties. In the present work, we focus exclusively on the direct differentiation (DD) approach, but note that developments outlined below readily extend to other approaches, provided diffusion is treating using a fixed particle grid.

In the DD approach, the fractional diffusion flux is estimated by estimating the fractional Laplacian of (5) analytically. (See Appendix B for details.) We next use a midpoint rule to estimate the evolution of the particle strength, leading to:

$$\frac{\partial u_i}{\partial t} = \frac{1}{\varsigma^\alpha}\sum_{j\in\mathcal{I}} V_j u_j G_\varsigma\left(\mathbf{x}_i-\mathbf{x}_j\right), \tag{7}$$

where

$$G_\varsigma(\mathbf{r}) = \frac{1}{\varsigma^d} G\left(\frac{\mathbf{r}}{\varsigma}\right), \quad G(\mathbf{r}) = -\frac{2^\alpha \Gamma\left(\frac{\alpha+d}{2}\right)}{\pi^{\frac{d}{2}}\Gamma\left(\frac{d}{2}\right)} {}_1F_1\left(\frac{\alpha+d}{2}, \frac{d}{2}, -|\mathbf{r}|^2\right), \tag{8}$$

and ${}_1F_1$ denotes the confluent hypergeometric function of the first kind [45]. For additional details regarding the construction, see [25].

Capitalizing on the indexing scheme used to enumerate the particles, the DD scheme can be written as a matrix system,

$$\dot{\mathbf{U}} = A\mathbf{U}, \tag{9}$$

where $\mathbf{U}$ is the vector concatenating the particle strengths, u_i, $\dot{\mathbf{U}}$ is its time derivative, and A is a matrix with elements

$$A_{i,j} = \frac{1}{\varsigma^\alpha} V_j G_\varsigma\left(\mathbf{x}_i-\mathbf{x}_j\right). \tag{10}$$

Note that the computation of each entry, $A_{i,j}$, requires the evaluation of a confluent hypergeometric function, which is computationally demanding [46]. However, if the particle grid is held fixed, evaluation of the diffusion matrix, A, may be performed once. Even if the diffusion is estimated using a fixed particle grid, however, a significant challenge arises because the matrix, A, is full. This generally leads to an $\mathrm{O}(N^2)$ storage requirement, where N is the total number of particles. This storage requirement can become prohibitively large, even for a relative modest computational grid. For instance, with particles uniformly distributed on a 2D grid with 301 cells in each direction, storage of the matrix in double precision would require about 61 GB of RAM. Even though this storage requirement could potentially be avoided by directly computing the local diffusion terms individually, such direct computation would necessitate an $\mathrm{O}(N^2)$ operation cost. Thus, one is generally faced with potentially prohibitive

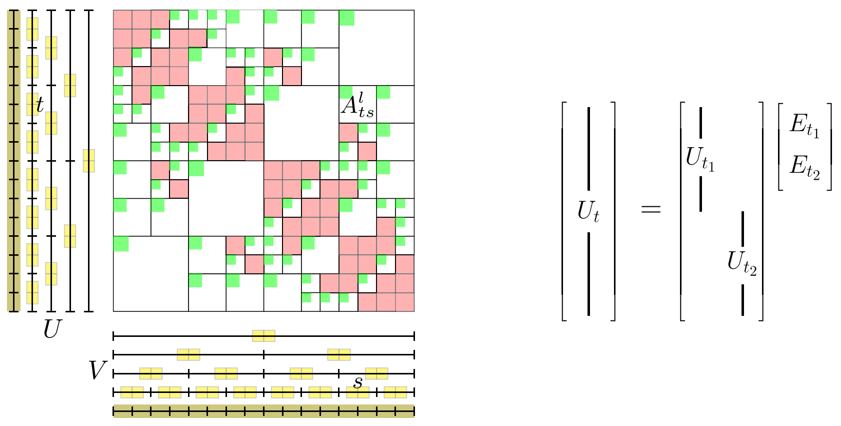


Fig. 1. Schematic representation of $\mathcal{H}^2$ approximation of a dense matrix operator.

memory and CPU requirements, unless the drawbacks of the direct matrix representation and source term evaluation are suitably addressed. The hierarchical methodology developed in the following section specifically focuses on this objective.

3. Hierarchical matrix methods for particle discretizations of FDE

3.1. Representation

The asymptotically smooth nature of the kernel G allows the discrete operator A to admit a hierarchical matrix representation that does not suffer from the polynomial growth in memory and computational complexity of dense matrix representations [47]. While some hierarchical matrix representations have log-linear complexity, the $\mathcal{H}^2$ representation we use in this work has an asymptotically optimal linear complexity both in its memory requirements and in the key matrix–vector multiplication operation that is at the core of the time evolution computations of the fractional PDE.

Fig. 1 shows a schematic of the $\mathcal{H}^2$ representation. Matrix blocks A_{ts} of size $n_b \times n_b$ are approximated by low rank factorizations of the form $A_{ts} = U_t S_{ts} V_s{}^T$ with rank $k_b \ll n_b$. These blocks are of different sizes representing different granularities. The matrix is stored as a quadtree, where coarser levels of the quadtree, closer to the root, include large blocks of size $n_b \approx N/2^l$ where $l \in [0, q]$ is the level number in a tree of depth q. At every level, l, of the quadtree some of the blocks are stored as low rank blocks while others are refined further. Leaves of the quadtree with size $n_b \leq n_{\min}$ are stored as dense blocks.

A geometric boolean “admissibility condition” determines whether a matrix block needs to be further subdivided or can be represented as a single low rank block. Since our matrix A is a discretization of a spatial operator, the admissibility condition encodes the intuition that a matrix block A_{ts} can be properly approximated by a low rank factorization to the desired accuracy ϵ if the set of particles t are sufficiently far from the set of particles s. Simple bounding box and distance computations can be performed to determine whether a matrix block is stored as a low rank block or not. The particular admissibility condition we use in the example simulations is described in Section 4.

In this representation, low rank matrix blocks may therefore appear in arbitrary locations in the matrix, as guided by the general admissibility condition. This allows for more flexibility than other fixed-structure representations, and allows the matrix structure to be adapted to the nature and patterns of the spatial discretization of the domain.

Another feature of the $\mathcal{H}^2$ representation, which allows it to reach the optimal asymptotic complexity $\mathcal{O}(N)$, is its use of a nested basis representation of the bases U and V. In a nested basis representation, a column basis for block row t is not stored explicitly but rather computed on-demand from the bases of the two children clusters t_1 and t_2 of t via small transfer matrices E of size corresponding to block rank rather than block size,

$$U_t = \begin{bmatrix} U_{t_1} & \\ & U_{t_2} \end{bmatrix} \begin{bmatrix} E_{t_1} \\ E_{t_2} \end{bmatrix}, \tag{11}$$

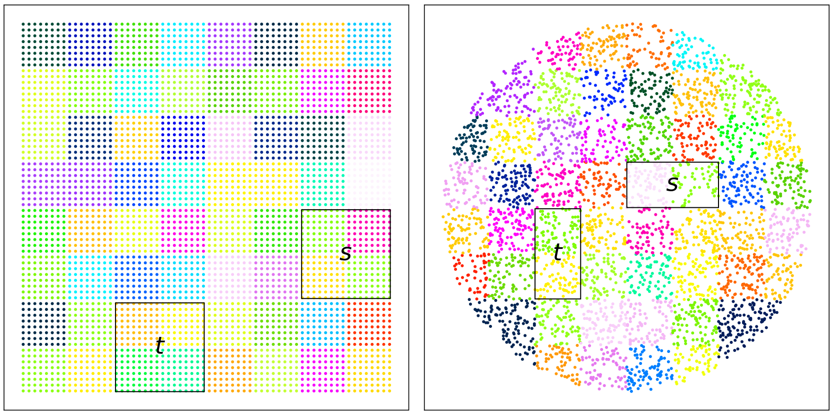


Fig. 2. Clustering produced by a binary k-d tree partitioning for a regular (left) and irregular (right) spatial discretization of square and circular regions. The labels t and s identify the clusters that affect a matrix block A_{ts}.

resulting in optimal complexity, as only the basis for leaf-level rows are explicitly computed and stored. A similarly nested basis is also used for the row basis V of block columns s. In the case of symmetric matrices, as in the case of pure fractional diffusion with no advection, V is identical to U and does need to be stored separately. GPU data structures and algorithms for operating on hierarchical matrices are described in [48].

The construction of the hierarchical approximation A is done in two phases. In a first phase, an initial approximation is generated via an interpolatory construction process which approximates the kernel function G by a high-order polynomial using Lagrangian interpolation. This is followed, in a second phase, by an algebraic compression process which essentially generalizes the SVD decomposition to hierarchical matrices, and truncates it to produce an optimal memory footprint to the desired accuracy ϵ in the approximation. These two phases are described next.

Related hybrid analytical-algebraic methods have been presented in the literature, including the HCA method [49] which overcomes the unreliability of the heuristic algebraic adaptive cross approximation (ACA) [50] method, by combining it with an interpolation-based separable approximation of the kernel. SMASH [51] also uses a hybrid compression method for the construction of $\mathcal{H}^2$ matrices using rank revealing QR factorizations. The use of linear algebra-based operations in hybrid methods generally results in better compression, smaller ranks, and more general applicability, than is possible with fast multipole methods [52].

3.2. Interpolatory construction

We start by clustering the N particles into small regions with clusters of cardinality $\leq n_{\min}$ using a k-d tree binary partitioning procedure. The spatial partitioning procedure, which repeatedly divides the point set into two halves with roughly equal cardinality, uses axis-aligned hyperplanes with median split in the direction of maximum extent of a bounding box of the point set. Fig. 2 shows the resulting clusters for two sample particle discretizations of a square and a disk. The k-d tree partitioning procedure generates a hierarchy of clusters that define the block rows and block columns of the hierarchical matrix at different levels of granularity, ending at the leaf level clusters. The labels t and s of Fig. 2 refer to sample clusters in these hierarchies. Each such pair will generate the block A_{ts} of the matrix. The cluster tree together with an admissibility criterion define the matrix structure. A construction algorithm needs to generate: (1) the leaf-level bases U_t for all leaf clusters t, (2) the corresponding interlevel transfer matrices E_t for all clusters at all levels, and (3) the matrix blocks S_{ts} for all low rank blocks.

The basic element of the construction consists of tensor product grids of Chebyshev interpolation points of size m^d that are overlaid on axis-aligned bounding boxes Ω_t and Ω_s of clusters t and s, respectively. This construction is used to build Lagrange polynomials $p(x)$ of order dm over d-dimensional boxes. Chebyshev points have a well-known best approximation property for representing smooth functions. The core observations for constructing the

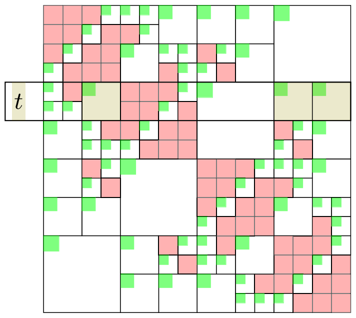


Fig. 3. Schematic illustration of a row having three blocks (shaded) at the same level of hierarchy.

nested basis hierarchical matrix are (1) that the kernel function can be written as a separable approximation:

$$G(x_t, x_s) \approx \tilde{G}(x_t, x_s) = \sum_i \sum_j S_{ij}\, p_{t,i}(x_t)\, p_{s,j}(x_s), \quad x_t \in \Omega_t, x_s \in \Omega_s, \text{and } i, j = 1 : m^d \tag{12}$$

where $p_{t,i}$ and $p_{s,j}$ are the ith and jth basis polynomials over Ω_t and Ω_s, and (2) that the m^d polynomials $p_{t,i}$ used in the approximation over a region Ω_t are expressed in terms of the approximating polynomials over the subregions of children clusters t_c $(t = \cup_c t_c)$

$$p_{t,i}(x) = \sum_k (E_{t_c})_{k,i}\, p_{t_c,k}(x), \quad x \in \Omega_{t_c}, \quad k = 1 : m^d, \text{and } c = 1, 2. \tag{13}$$

Based on these interpolation points, we can evaluate the elements of the hierarchical matrix approximation. The leaf-level bases U_t are of size $n_t \times k_t$, where n_t is the cardinality of cluster t and $k_t = m^d$ is the number of points in the interpolation grid. Their columns can be computed by evaluating the basis polynomials at the n_t points of t. The interlevel transfer matrices E_{t_c} for a child cluster t_c with parent cluster t can also be computed column by column by simply evaluating the basis polynomial of Ω_t at the interpolation points of Ω_{t_c}. Finally, the ij entries of the small S_{ts} matrices can be computed by evaluating the kernel function at the ith interpolation point of Ω_t and the jth interpolation point of Ω_s.

One question to address concerns the best values of m (which may vary from cluster to cluster as well from level to level) needed in order to reach a desired target approximation accuracy, ϵ, in the matrix. The order, m, used in the Lagrangian interpolation directly affects the error $\epsilon_G = \|G(x_t, x_s) - \tilde{G}(x_t, x_s)\|$ in the polynomial approximation of the kernel function. For smooth kernel functions the convergence of the approximating polynomial is exponential in m. It can also be shown that the error in the resulting hierarchical matrix approximation, measured either by the spectral or Frobenius norms, is also directly related to ϵ_G [53]. Unfortunately, the constants that appear in the matrix error estimates depend on the geometric sizes and shapes of the clusters and their relative position and orientations, and the bounds are not always tight. Therefore our strategy for constructing our matrix approximation is to use a conservative value, m, that first generates a tighter error than desired (and therefore uses more memory in the approximation of A), and then compress it algebraically to the desired tolerance, ϵ, to obtain the optimal approximation. We discuss the compression phase next.

3.3. Algebraic compression

The S_{ts} coupling blocks at different levels of the matrix constructed above are of size $k_t \times k_s$ where $k_t = k_s = m^d$ is the size of the interpolation grids over Ω_t and Ω_s. Algebraic compression seeks to reduce the memory footprints of these S_{ts} blocks. An obvious way to do so would be to compute an SVD decomposition of $S_{ts} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, and truncate the singular values at the desired approximation level, $\Sigma_{k+1} < \epsilon$. The matrix blocks are then compressed to rank k blocks as $A_{ts} = U'_{ts}\Sigma' V'^{\,T}_{st}$ where $U'_{ts} = U_t\tilde{U}_{:,1:k}$ and $V'_{ts} = V_t\tilde{V}_{:,1:k}$.

This compression works for a single block but will not generate the common and nested basis U'_t that is needed for optimal memory complexity of the whole matrix. In order to generate a common new column basis U'_t for the whole block row t, the SVD needs to include all blocks S_{ts_i} in it. Fig. 3 illustrates a block row t containing three blocks at the same level in the hierarchy. The common basis to use for compression of this block row can be obtained from the SVD of a matrix combining these blocks:

$$B_t = [S_{ts_a}\ \ S_{ts_b}\ \ S_{ts_c}]. \tag{14}$$

An efficient way to perform these computations is to factor B_t as $R_t Q_t$ using a QR decomposition of B_t^T and use $U_t R_t$ as the new common basis for block row t. An SVD of the small matrix R_t can be generated and truncated to the desired accuracy to yield a rank k block. If we denote the left singular vectors of R_t by $\tilde{U}_t$, its truncated approximation is $\bar{U}_t = \tilde{U}_t(1 : k)$. Assuming an orthogonal basis U_t, the new basis is now compressed to produce the desired $U_t' = U_t \bar{U}_t$.

In other words, the algebraic compression of the block row t produces the following approximation:

$$U_t\,[S_{ts_a}\ \ S_{ts_b}\ \ S_{ts_c}] \approx U_t'\,[\bar{S}_{ts_a}\ \ \bar{S}_{ts_b}\ \ \bar{S}_{ts_c}], \tag{15}$$

where $\bar{S}_{ts_j} = \bar{U}_t^T S_{ts_j}$ are the compressed coupling blocks expressed in the new basis U_t'. A similar compression can be done for the bases V_s for non-symmetric matrices where $V \neq U$. Once the new compressed U_t' and V_s' bases are obtained, the new coupling blocks S_{ts}' can be obtained by projecting S_{ts} on the new bases, $S_{ts}' = \tilde{U}_t^T S_{ts} \tilde{V}_s$. This yields new compressed representations $U_t' S_{ts}' V_s'^T$ for all matrix blocks.

This construction does not yet yield nested bases however. In order to produce the nested basis U_t' for all block rows t at all levels of granularity, the information at the coarse block levels must also be present at the finer levels in the cluster tree. This can be accomplished by propagating the relevant basis information from the root to the leaves in a downsweep pass through the basis and matrix trees. Every R_t will then have information from the coarser levels. Compression then proceeds in an upsweep pass through the basis tree. At the leaf level, this is done via an SVD of the leaf-level bases $U_t R_t$ of size $n \times k$ with $n < n_{\min}$. At higher levels of the hierarchy, this involves also the SVD of only small matrices involving the interlevel transfer operators E_t, of size comparable to the ranks, not the extent, of the blocks. These algorithms are described in [48]. Assuming a bounded number of blocks in each block row and block column of the various levels of the hierarchy, the compression can be done in linear complexity $O(N)$ and includes substantial data parallelism that is exploitable for efficient execution on GPUs and manycore architectures.

3.4. Matrix–vector multiplication

A core operation in time dependent fractional diffusion simulation is the product of the hierarchical matrix approximation of the discretized fractional Laplacian by a vector. This operation can be performed in optimal complexity $O(kN)$ where k is a bound on the block ranks for a problem of size N. The multiplication can be expressed as

$$Ax = A_d x + A_{lr} x = A_d x + \left(\sum_{l} \sum_{(t,s)\ \text{in level}\ l} U_t^l S_{ts}^l {V_s^l}^T \right) x. \tag{16}$$

The dense part of the matrix (A_d) is a block sparse matrix and its multiplication by a vector can be done by standard sparse matrix vector routines. The low rank (A_{lr}) part of the product is a generalization of the low rank dense matrix vector multiplication operation. When a matrix of the form USV^T is to be multiplied by a vector x, the computation is done in three phases: first the product $V^T x$ is formed, then S is multiplied by that intermediate vector, and finally the product of U is multiplied by this latter vector to generate the desired result. Generalizing this operation to the $\mathcal{H}^2$ representation, produces the following three steps:

- an upsweep pass through the basis tree V of the block columns produces the products ${V_s^l}^T x$ for all block rows s at all levels l;
- the products S_{ts}^l with the vector of the first phase are performed concurrently for all blocks at all levels;
- a downsweep pass through the basis tree U of the block rows produces and accumulates the products U_t^l with the vectors generated in the second phase. This result is added to the result of the dense phase to generate the product Ax.

The tree traversals for the upsweep and downsweep passes, and the products with the coupling blocks S, can all be done in linear complexity. In addition there is substantial data parallelism in all phases which allows for substantial performance improvements on GPUs, limited primarily by the bandwidth of the GPU memory. Performance results are shown in the next section.

3.5. General linear algebra operations on hierarchical matrices

The algebraic nature of hierarchical matrices allows the development of high performance algorithms that implement general linear algebra operations. Of particular interest is the low rank update operation

$$A = A + XY^T \tag{17}$$

which produces the sum of a hierarchical matrix A of size $N \times N$ and a globally low rank matrix whose X and Y factors are of size $N \times k$ with $k \ll N$. This operation can be efficiently implemented [30,54] by first adding the contributions of XY^T to the various blocks of A at all levels, and recompressing the resulting sum algebraically as described earlier. The low rank update operation is a key routine for an operation that generates an explicit hierarchical matrix representation of an operator accessible only via matrix vector products. Such an algorithm generalizes the popular randomized algorithms for generating low-rank approximations of large dense matrices [55] to the hierarchical case. The ability to sample a "black-box" matrix to produce a hierarchical matrix representation allows matrix multiplication and evaluation of general algebraic expressions of hierarchical matrices to be performed.

Approximate hierarchical matrix inverses can be generated via an iterative Newton–Schulz method or its higher order variants. Newton–Schulz allows matrix inverses to be generated by iterating on the evaluation of

$$X_{p+1} = (2I - X_p A)X_p, \tag{18}$$

involving two matrix multiplication operations per iteration, which can be performed directly in the hierarchical matrix format. For positive definite matrices, the iterations converge globally to A^{-1} starting from the scaled identity $X_0 = I/\|A\|_\infty$. In practice, we do not need to converge to a tight tolerance, since the inverse is meant to be used as a preconditioner in a Krylov solver. The termination condition $\|AX_p - I\|_2 < \epsilon$ can be used to produce approximate inverses that can be practical and effective preconditioners, as we demonstrate in the numerical results in the following section. Hierarchical inverse preconditioners for non-fractional operators have been studied in [56,57].

Newton–Schulz iterations can be generalized to hyperpower iterative methods that improve the second order convergence to higher order. An order v hyperpower iteration is defined as [58]:

$$X_{p+1} = X_p \left(I + R_p + \cdots + R_p^{l-1}\right) = X_p \sum_{i=0}^{v-1} R_p^i, \tag{19}$$

where $R_p = I - AX_p$, involving l matrix products. Setting $v = 2$ gives the standard Newton–Schulz iteration. In our hierarchical matrix computation framework, the computation of X_{p+1} requires sampling the right-hand side of Eq. (19), i.e., multiplying the polynomial expression by sampling vectors. This can be performed efficiently using Horner's method for evaluating polynomials. The resulting products are used in the level by level construction of the hierarchical matrix X_{p+1} [54]. We use methods of order 16 to construct the approximate inverse preconditioners used in the results shown below.

4. Performance of hierarchical matrix operations

In this section, we demonstrate the efficiency of the hierarchical matrix operations used in this work on both CPU and GPU machines. For these experiments we use a set of N particles placed at the nodes of a regular $n \times n$ grid in order to assess the scalability, in both time and memory, of the hierarchical matrix computations with problem size. In all experiments, the matrix structure is constructed by hierarchically clustering the N points using a k-d tree data structure. The k-d tree partitioning algorithm subdivides the point set until a leaf size of $n_{\min} = 64$ is reached. A pair of clusters t and s at level l of the hierarchy generates a low rank matrix block A^l_{ts} if the inter-center distance between the point sets is large relative to their average size, $\|\mathbf{x}_t - \mathbf{x}_s\| > \eta(D_t + D_s)/2$. In this algebraic admissibility condition, $\mathbf{x}$ represents the location of the center of the cluster, D the diameter of a ball enclosing it, and η is a

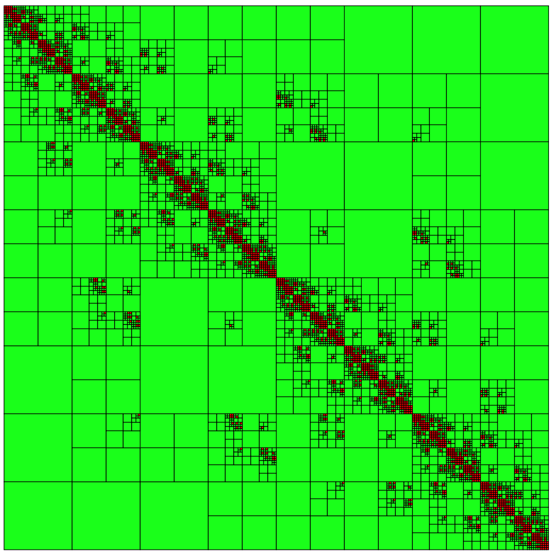
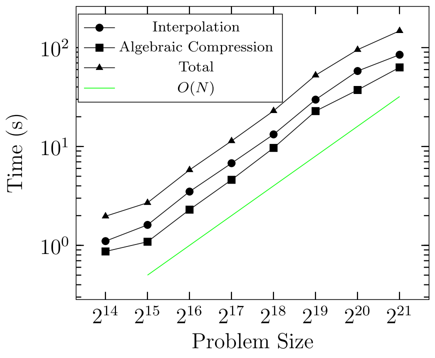
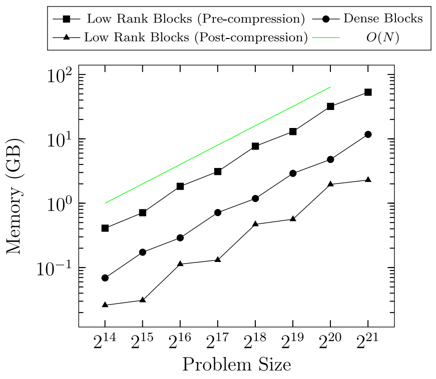


Fig. 4. Hierarchical Matrix Structure produced by admissibility parameter $\eta = 0.8$ for a regular 2D grid.



(a) CPU time for the two phases of the construction.



(b) Memory consumption and effect of algebraic compression on low rank blocks memory.

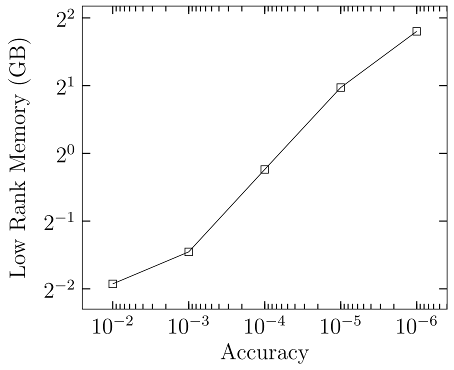
Fig. 5. Performance of matrix construction to an error threshold of 10^{-5} on a range of problem sizes: (a) CPU time for interpolation and algebraic compression phases; (b) Memory footprint of the dense and low rank blocks of the resulting matrix (before and after algebraic compression).

tuning parameter of the condition that eventually has the effect of controlling the block refinement of the resulting hierarchical matrix. We fixed $\eta = 0.8$ for these experiments, and used a fractional order $\alpha = 1.5$. The resulting matrix structure for a problem size $N = 16{,}384$ is shown in Fig. 4.

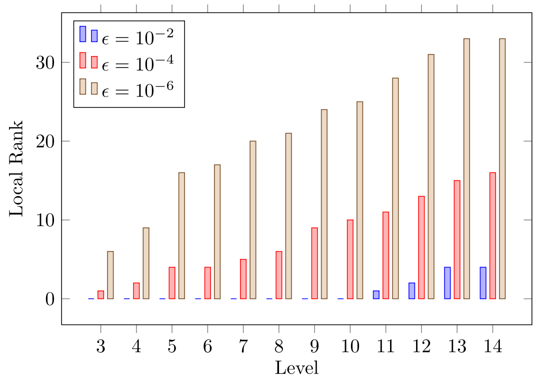
Fig. 5(a) shows the performance of the two phases of the construction of the discrete operator as well as the overall time needed to produce the final compressed matrix. For the first phase, as described in Section 3.2, we use a constant rank of 100 for the interpolation to achieve an overall accuracy of about 10^{-6} which we verify by sampling 5% of the entries of the output of the full dense matrix with a random vector and comparing against the output of the hierarchical matrix vector product. The resulting storage requirement of the low rank portion of the matrix grows linearly with the problem size, but it is still relatively high. The second phase, as described in Section 3.3, performs algebraic compression to an accuracy of 10^{-5} to greatly reduce the overall memory footprint. Both phases have linear asymptotic complexity which is seen in Fig. 5(a). The memory consumption of the dense (A_d) and low rank (A_{lr}) portions of the matrix are shown in Fig. 5(b). The reduction in memory footprint achieved

Table 1
The memory consumption in GB and memory savings of using a hierarchical representation instead of a dense one.

N	$\mathcal{H}^2$ memory	Dense memory	Memory ratio
16384	0.095	2	21
32768	0.204	8	39
65536	0.403	32	79
131072	0.846	128	151
262144	1.653	512	310
524288	3.471	2048	590
1048576	6.743	8192	1215
2097152	14.01	32768	2339



(a) Effect of the approximation accuracy on low rank storage.



(b) Maximum block ranks for each level of the matrix for varying compression thresholds.

Fig. 6. Impact of the choice of the compression threshold on the memory footprint and maximum block rank of a fractional diffusion hierarchical matrix of size 2^{20}.

by the algebraic compression can be seen by comparing the two low rank curves in Fig. 5(b). A factor of more than 20× reduction in storage for the low rank blocks can be achieved on the larger problem sizes.

Table 1 also shows the total savings in memory of the hierarchical matrix (dense blocks plus low rank blocks) computed to an overall accuracy of $\epsilon = 10^{-5}$ when compared with a full dense matrix representation. The linear growth in memory footprint of the $\mathcal{H}^2$ hierarchical representation stands in sharp contrast to the quadratic growth of the dense representation. This is, principally, why these hierarchical matrices have the potential for being the basic computational engine of fractional diffusion simulations at scale.

Fig. 6(a) shows the effect of increasing the compression threshold ϵ on the low rank memory consumption. As expected there is a weak, logarithmic, dependence of the memory footprint on the accuracy of the approximation, and we observe a growth of about $O(|\log \epsilon|^2)$. This translates to an almost tenfold increase in low rank block storage for an increase of 4 digits of accuracy from an initial 2 digits when compressing a matrix of size 2^{20}. The memory footprint of the dense blocks is obviously not affected. Fig. 6(b) shows the maximum block rank for every level of the hierarchy in the matrix. The corresponding increase in the maximum ranks of each level of the hierarchical matrix for compression thresholds of 10^{-2}, 10^{-4}, and 10^{-6} also shows a weak logarithmic dependence on desired approximation accuracy, of about $O(|\log \epsilon|^2)$.

Once the matrix has been compressed, we can efficiently perform the necessary matrix vector products on both the CPU and the GPU. These experiments are performed on a 12-core workstation and a P100 NVIDIA Pascal GPU, respectively. Though the GPU memory is limited to 16 GB, this does not pose any issues for problem sizes up to about two million, due to the greatly reduced storage requirements of the compressed matrix. Fig. 7 shows the performance of the `hgemv` operation averaged over 10 runs, clearly demonstrating the linear complexity of the algorithm, and the high efficiency of the GPU in particular, which is able to perform the matvec operation in under 40 ms on a problem of size 2M.

Using the hierarchical matrix compression and matrix vector product operations, we construct an approximate inverse and evaluate its effectiveness as a preconditioner for the conjugate gradient method. The inverse of a

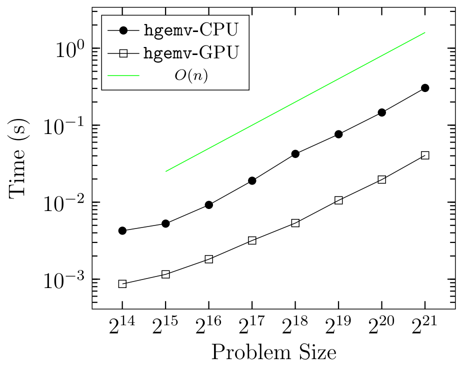
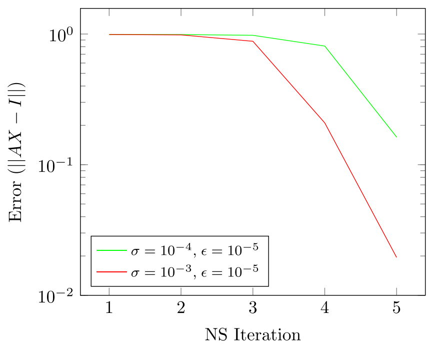
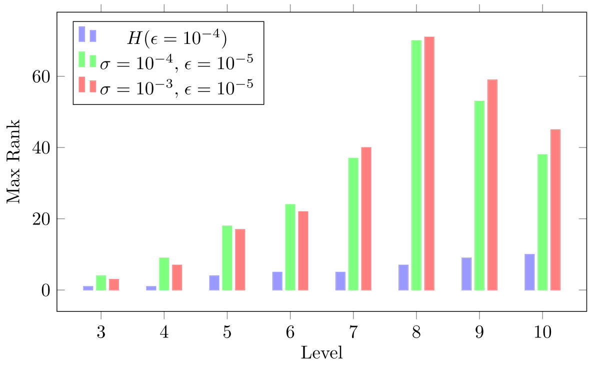


Fig. 7. Performance of the matrix–vector multiplication.



(a) Convergence rate of the order 16 hyperpower iterates for the regularized matrix.



(b) Maximum block ranks at each level of the hierarchy for the final iterates, when compressed to a threshold of 10^{-5}.

Fig. 8. Performance of the order 16 hyperpower iterative inversion.

regularized version of the discretized operator ($\tilde{A} = A + \sigma I$) is constructed and used. We use an order 16 hyperpower iterative method to compute the approximate inverse as described in Section 3.5 and an approximation accuracy ϵ for the constructed iterates.

Fig. 8(a) shows the convergence of the hyperpower iterates for a problem size of 2^{16} for two values of the regularization parameter $\sigma = \left(10^{-3}, 10^{-4}\right)$, where each iterate is constructed to a threshold of $\epsilon = 10^{-5}$. The runtime per iteration depends heavily on the ranks of the iterates which tend to increase for the intermediate iterates. For example, the first iteration completes in about 56 s, whereas the final iteration takes 763 s, taking about 1771 s in total for the five iterations performed. Fig. 8(b) shows the maximum block ranks for each level of the hierarchy for the approximate inverses reached after five iterations. There is a noticeable growth in the block ranks of the approximate inverse compared to the ranks of the original matrix.

Employing the resulting matrices as preconditioners for the solution of the system $Ax = b$ using a conjugate gradient method, we obtain the iteration counts of Fig. 9. The right-hand side vector used in the figure is a vector of all ones, $b = \mathbf{1}$, with the solution started at $x_0 = \mathbf{0}$, although comparable results were obtained for values of b randomly sampled from the uniform [0, 1] distribution. We note that the diagonal preconditioner cannot produce a reliable solution, while the two approximate inverses obtained after five iterations of the iterative scheme (19) are able to produce a solution robustly to high accuracy. Not unexpectedly, the approximate inverse with the smaller regularization term is more effective than the one with the larger regularization term when it comes to CG iteration count. In both cases, we note that a coarse approximation of the regularized inverse is sufficient for robust convergence.

In order to assess the impact of finer spatial discretizations on the effectiveness of this preconditioner, we consider a set of increasingly finer grids for a 1D problem with $\alpha = 1.5$ in the spatial region $[-10, 10]$, ranging from

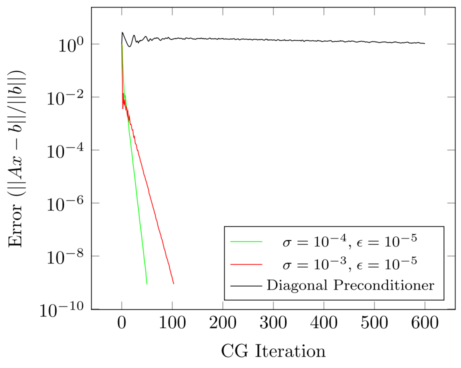


Fig. 9. The convergence behavior of the preconditioned conjugate gradient method using a diagonal preconditioner and the two approximate (regularized) inverse preconditioners.

Table 2
Number of CG iterations for increasingly finer discretizations, using two different preconditioners. Only mild growth in number of iterations is experienced using the first one–an approximate inverse. The second preconditioner, using a pre-converged Newton–Schulz iterate, is less robust.

Problem size	1K	2K	4K	8K	16K	32K	64K	128K
Preconditioner 1	3	3	3	4	5	7	10	11
Preconditioner 2	5	5	5	5	8	12	20	33

$N_x = 1024$ to $N_x = 131072$, corresponding to a mesh spacing ranging from about $h = 2 \times 10^{-2}$ to $h = 2 \times 10^{-4}$. We construct the $\mathcal{H}^2$ discrete fractional operator A as above with an approximation accuracy of $\epsilon = 10^{-8}$ and construct the approximate inverse of the regularized operator $\tilde{A} = A + \sigma I$ with $\sigma = 10^{-4}$ using an 8th order hyperpower Newton–Schulz iteration cold-started with a scaled identity as X^0. We first construct an inverse X^k by iterating until the 2-norm of $(\tilde{A}X^k - I)$ is less than 10^{-2}, which took about 8 high order NS iterations on the larger problem sizes. Construction cost of the preconditioner grows log-linearly with problem size and takes less than 2s per high-order NS iteration for the 128K problem on an NVIDIA P100 GPU. For contrast, we also constructed a lower-quality preconditioner by simply iterating a fixed number of iterations in the pre-convergence regime of NS (5 iterations on the larger problem sizes). Table 2 shows the number of CG iterations it took to converge to a relative tolerance $\|Ax - b\|/\|b\|$ of 10^{-9} starting from a zero-vector, using these two preconditioners. There is only very mild growth in the number of CG iterations when using the first preconditioner. However, as expected, the second preconditioner X^5 experiences slightly larger growth in the resulting number of iterates with problem size, as it has not yet converged to an inverse with reasonable accuracy. With fewer NS iterations, the preconditioner's efficacy will naturally deteriorate further. With a simple diagonal preconditioner, we get very little decrease in the residual even after several hundred CG iterations.

5. Applications

In this section, we analyze the performance of the hierarchical solution algorithm in the context of two applications. In the first application (Case 1), we focus on the solution of the fractional diffusion equation in an unbounded 2D domain, and rely on the available analytical solution to test the approximation quality of the computed solution. This enables us to assess the impact of approximating the full diffusion matrix by its hierarchical counterpart, and also to analyze the computational gains achieved through this approximation. In the second application (Case 2), we consider the solution of the forced fractional diffusion equation in a bounded domain, and carry out the integration for a sufficiently long interval so as to effectively reach a steady-state solution. In this second setting to explore the potential of applying the hierarchical matrix algorithms as means to obtain a (pseudo-transient) solution of a fractional elliptic problem. We also compare the pseudo-transient solution to that obtained by the conjugate gradient method, preconditioned by a hierarchical approximate inverse, for solving directly the corresponding steady state fractional Poisson problem.

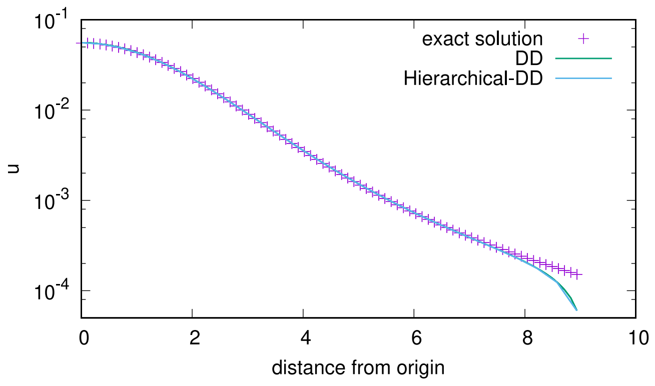


Fig. 10. Fundamental solution at $t_f = 1.5$ for $\alpha = 1.5$. Also plotted are profiles obtained using the full diffusion matrix and its hierarchical approximation; in both cases $N_x = 161$.

5.1. Case 1

In this section, we assess the performance of the hierarchical decomposition algorithm based on the transient solution of the fractional diffusion equation (3) in $\mathbb{R}^2$. We use as initial condition the fundamental solution at $t_0 = 0.5$ and carry out the integration until a final time $t_f = 1.5$. The availability of the analytical solution A and of numerical solutions computed with direct particle–particle interactions [25] provide a suitable setup to assess both performance, as well as errors associated with hierarchical approximation of the full diffusion matrix.

To provide a suitable spatial discretization throughout the period of integration, particles are distributed over a uniform grid covering a square domain of side $2D$, where

$$D = 4t_f^{1/\alpha} R_\alpha, \tag{20}$$

and R_α is the characteristic length defined in (A.4). The grid size (inter-particle distance) is $h = 2D/(N_x - 1)$, where N_x is the number of points in the x and y directions.

5.1.1. Accuracy

Fig. 10 contrasts the analytical and numerical solutions at $t_f = 1.5$ for the case $\alpha = 1.5$. The numerical solutions are obtained using the full diffusion matrix with $N_x = 161$ and $\varsigma = 2h$, and its hierarchical approximation. A good agreement with the exact solution is observed, though small but noticeable errors are observed near the boundaries of the truncated domain. As discussed in [25] the observed discrepancies are dominated by domain truncation effects. Fig. 10 also shows that solutions obtained using the full and hierarchical diffusion matrices are in close agreement throughout the domain. This suggests that the hierarchical matrix representation does not appreciably affect the quality of the computed solution.

To further quantify prediction accuracy, as in [25] we estimate the l_1 error defined according:

$$\varepsilon_{D_\varepsilon} = \frac{\sum\limits_{i \in \mathcal{I}_\epsilon} V_i |u_i(t_f) - \mathcal{G}_\alpha(\mathbf{x}_i, t_f)|}{\displaystyle\int_{D_\varepsilon} \mathcal{G}_\alpha(\mathbf{x}, t_f)\,\mathrm{d}V(\mathbf{x})}, \tag{21}$$

where D_ε denotes the subregion $[-5, 5] \times [-5, 5]$. Fig. 11 shows estimates of $\varepsilon_{D_\varepsilon}$ for different orders ($\alpha = 1.1$, 1.5, and 1.9) and grid resolutions ($N_x = 41$, 81, 161, 321, and 641). In all cases, the regularization parameter $\varsigma = 2h$. The results indicate that the hierarchical approximation of the diffusion matrix has insignificant impact on the computed errors. In particular, the error in the case of the hierarchical algorithm also exhibits second-order behavior. Note that as h decreases, estimates of $\varepsilon_{D_\varepsilon}$ tend to level off. As discussed in [25] this phenomenon occurs as discretization errors drop and approach domain truncation errors.

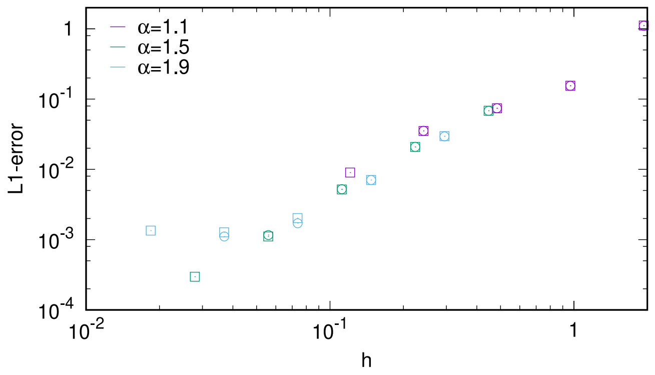


Fig. 11. L1-error $\varepsilon_{D_\varepsilon}$ for different values of α and h, as indicated. Plotted are results obtained using the full diffusion matrix (circles) and its hierarchical approximation (squares).

Table 3
Computational cost and memory requirements of the full-matrix and the hierarchical-matrix computations. All the simulations were conducted on the same workstation, comprising 28 physical cores (56 in hyperthreading) on Intel CPU's, and 127 GB of RAM. The codes were designed to run in shared memory parallelism, and compiled using the same compiler.

N_x	DD	Hierarchical DD		
		$\alpha = 1.1$	$\alpha = 1.5$	$\alpha = 1.9$
Time to form the matrix (s)				
41	1	17	18	18
81	1	17	18	18
161	4	24	25	25
321	30	49	49	50
641	-	124	122	121
Time to integrate the system (s)				
41	0.2	4	4	4.1
81	5.6	4.3	4	3.9
161	96	18	16	15
321	1700	76	70	66
641	-	606	569	560
Required workspace (GB)				
41	0.021	0.014	0.014	0.013
81	0.32	0.074	0.072	0.066
161	5	0.341	0.329	0.297
321	79.1	1.457	1.406	1.26
641	-	6.041	5.82	5.205

5.1.2. Performance

The performance gains achieved by the hierarchical algorithm are assessed in terms of the wall time required to construct the hierarchical matrix representation, the time required to integrate the governing equation, and the memory requirements. Note that the wall time is affected by other system loads, and so its measurement may not provide a precise estimate of the required CPU time. Nonetheless, it still provides a systematic means of comparing and scaling the performance of the full and hierarchical matrix computations.

Table 3 reports computational times and workspace requirement estimates for full and hierarchical matrix computations. Provided are results obtained for different values of N_x and α. It is clear when the system size is small ($N_x = 41$ and 81), application of the hierarchical representation algorithm is not attractive. This is the case because of the overheads incurred in forming the hierarchical representation, which in this parameter range dominate the relatively small gains achieved during the integration. However, as the system size increases, $N_x \geq 161$, the advantages of the hierarchical algorithms become substantial. Specifically, despite the effort required to form the hierarchical matrix, the hierarchical algorithm results in net savings. The overall savings are about a factor of 2 at $N_x = 161$; they amount to a factor of 14 approximately for $N_x = 321$. Note that for $N_x = 641$, we are no longer able to form the full diffusion matrix, as storage of the full matrix would require an estimated 1250 GB of RAM. In contrast, the workspace required by the hierarchical algorithm remains at a reasonable level, as it is seen to grow linearly with system size (N_x^2). The efficiency gains can also be appreciated by noting that the hierarchical simulation at $N_x = 641$ required a fraction of the time needed to perform the full matrix simulation at the coarser resolution level ($N_x = 321$). In light of the present experiences, the hierarchical representation can be viewed as an enabling feature for extending the present fractional diffusion algorithms to large systems in multiple dimensions.

5.2. Case 2

In this section, we apply the hierarchical matrix representation to compute the evolution solution of forced, fractional diffusion equation:

$$\frac{\partial u}{\partial t} = \Delta^\alpha u - q(\mathbf{x}), \tag{22}$$

in the square domain $\mathcal{D} = (-D, D) \times (-D, D)$, with initial condition $u(\mathbf{x}, 0) = 0$, and boundary condition $u(\mathbf{x}, t) = 0$ for $\mathbf{x} \notin \mathcal{D}$. Here $q(\mathbf{x})$ is a given steady source term.

We also use a conjugate gradient method, preconditioned by an approximate hierarchical inverse, to solve directly the fractional elliptic problem:

$$\Delta^\alpha u = q(\mathbf{x}). \tag{23}$$

5.2.1. Transient solution

We first carry out the simulations to approximate the steady solution of (22), and thus apply the present setting as a pseudo-transient approach to simulate the solution of (23).

We rely on a uniform particle grid with size $h = 2D/(N_x - 1)$. We set $D = 4$, and q to be a Gaussian pulse with zero mean and unit variance.

Let $\mathbf{U}$ be the vector of particle strengths, and let $\mathbf{q}$ denote the corresponding source term vector. Applying a first-order explicit time discretization of (22) leads to the following evolution equation for the particle strengths

$$\frac{\mathbf{U}^{s+1} - \mathbf{U}^s}{\Delta t} = A\mathbf{U}^s - \mathbf{q}, \tag{24}$$

where the superscript denotes the time level.

We consider three values of the fractional order, namely $\alpha = 1.1$, 1.5, and 1.9, and carry out the computations up to $t_f = 64$. For all considered values α, this final time is sufficiently large for the solution to become essentially steady. To verify that this is the case, we estimate the relative change in time of the solution at the origin

$$\delta u^{s+1} = \frac{u^{s+1}(x = 0, y = 0)}{u^s(x = 0, y = 0)} - 1. \tag{25}$$

Results are plotted in Fig. 12. At early times, the evolution of the solution near the origin is dominated by the source term, and so the curves for different α are almost identical. When the diffusion becomes significant, the evolution exhibits substantial dependence of α. As we approach t_f, in all cases the relative change becomes very small, showing that the solution can be considered as essentially steady.

Fig. 13 illustrates the computed solutions at the final for all considered values of α. The simulations are conducted using a grid with $N_x = 161$, i.e. with $N = 25{,}921$ particles. The time integration uses a step size $\Delta t = 2.5 \cdot 10^{-3}$, selected within the stability range of the explicit integration scheme [25]. Despite the large number of integration steps (25,600), implementation of the hierarchical scheme enables us to estimate the steady solution efficiently. Examination of the results (not shown) indicates that in all cases the solution remains radially symmetric, as expected. Consequently, the depicted axial profiles provide a full characterization of the predictions. In all cases, a smooth solution is obtained, with a peak at $\mathbf{x} = 0$.

To examine the dependence of the computed solution on the resolution, we repeat the same experiment for a higher number of grid points, $N_x = 321$ and 641. For brevity, we only report in Fig. 14 results obtained for $\alpha = 1.5$.

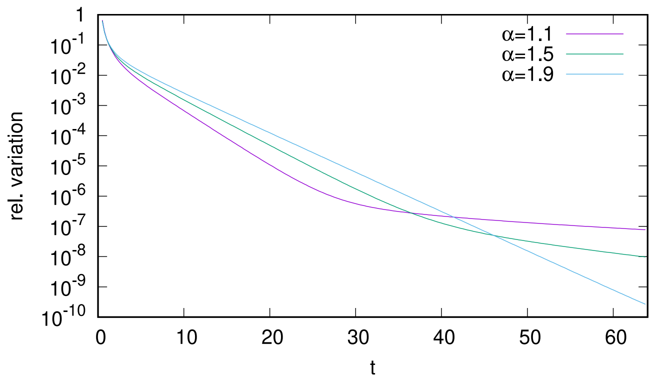


Fig. 12. Relative variation in time of the solution at $\mathbf{x} = 0$ and different values of α as indicated. The simulations are performed using $N_x = 161$ and $\Delta t = 2.5 \cdot 10^{-3}$.

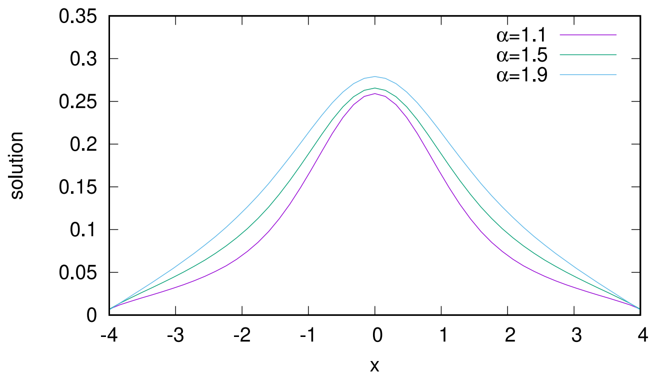


Fig. 13. Profile of the computed solution at $t = 64$ along $y = 0$. Plotted are curves generated for different values of α, as indicated.

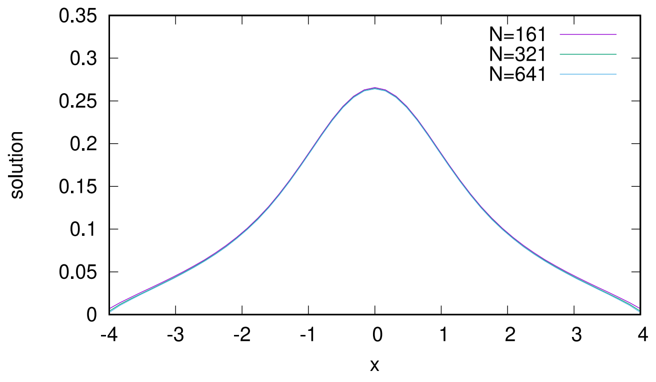


Fig. 14. Profile of the computed solution at $t = 64$ along $y = 0$. Plotted are curves generated $\alpha = 1.5$, and different values of N_x, as indicated.

The plot indicates that for all values of N_x, the solutions are nearly identical over most of the computational domain, whereas small differences are discernible near the boundaries. Note that the boundary condition is not exactly satisfied, which is in fact anticipated because a simplified treatment of boundary condition is adopted, specifically based on setting to zero the strength of particles falling outside the domain. Because a smoothing function is used, the boundary condition is not exactly achieved. However, as the discretization is refined, the predictions at the boundary are seen to drop towards zero. The rate of convergence of the solution is estimated to be approximately 1. Thus, in the present case, though it is evidently accurate in the interior, the scheme is effectively behaving as first

Table 4
Cost of A and its approximate inverse, and CG iterations, for different values of α.

α	A_d mem (GB)	A_{lr} mem (GB)	Approx inverse LR mem (GB)	Inversion time (s)	CG iterations
1.1	0.173	0.078	0.181	529.4	60
1.5	0.173	0.071	0.134	478.2	39
1.9	0.173	0.052	0.149	421.3	26

order. This is also expected because in a bounded domain, solutions of the fractional elliptic equations are generally not smooth at the boundary (e.g. [59–62]). Evidently, the loss of regularity at the boundary does not impact the performance of the algorithm in the present pseudo-transient approach. As reflected in the results below, this is also the case for the inverse approach.

5.2.2. *Steady state solution*

Finally, we consider the behavior of a CG method for solving the steady state equations (23) for the same fractional order values $\alpha = 1.1, 1.5$, and 1.9, $N_x = 161$, and the same problem parameters D and q used above. The hierarchical discrete operator A is generated to an accuracy of 10^{-6} for all values of α as above. We construct an approximate inverse preconditioner $(A+\sigma I)^{-1}$ with $\sigma = 10^{-4}$ by using five iterations of an order 16 hyperpower iterative inversion method. The final iterate is compressed to an accuracy of $\epsilon = 10^{-5}$ for the cases of $\alpha = 1.1$ and 1.3, but required a tighter accuracy of $\epsilon = 10^{-6}$ for the $\alpha = 1.9$ case to be an effective preconditioner. The CG solution was computed to a relative accuracy $\|AU - q\|/\|q\|$ of 10^{-10} using these preconditioners.

Table 4 shows in columns 2–4 the memory footprint of the dense and low rank blocks of the forward operator A and of its approximate inverse used as preconditioner. The fifth column shows the time needed to generate the preconditioner on the P100 Pascal GPU. The final column of the table shows the number of CG iterations for convergence with the Gaussian pulse input used above, starting from an initial all-zeros guess. Except for using a compression accuracy of 10^{-6} for $\alpha = 1.9$ and a looser accuracy of 10^{-5} for $\alpha = 1.1, 1.5$, the hierarchical preconditioners are all quite robust across the range of the order of the fractional operator. We note that increasing α leads to slightly lower inversion time primarily due to slightly lower ranks in the matrix and its inverse.

The approximate inverse requires more memory than the forward operator, as the ranks of the low rank blocks increase, however the overall savings remain substantial across the values of α. Whereas a fully dense representation of the operator or its dense inverse would require 5GB, the hierarchical inverse is quite compressible with a 12–15$\times$ reduction in overall memory use even for this relatively small problem with $N = 25{,}921$. We also note that the solution time, once the preconditioner is built, is always a fraction of a second on the GPU.

6. Conclusions

Fractional diffusion operators, generalizing the venerable Laplacian operator, have received substantial interest in recent years because of their usefulness in modeling a variety of problems in application domains such as financial option pricing, image processing, anomalous diffusion, and many others. However, the non local nature of fractional diffusion equations produces discretizations that generally involve dense matrices. As simulations scale up in size, the computational costs, both in memory and processing time, go up super-linearly and the resulting computations become far too expensive to be feasible even on high performance machines.

In this work, we presented hierarchical matrix representations and algorithms, specifically $\mathcal{H}^2$ representations, that can effectively handle discretizations of fractional diffusion operators. $\mathcal{H}^2$ matrices reduce the computational cost of storing these operators to an asymptotically optimal $O(N)$, with a constant that depends on the ranks of the matrix blocks and directly controlled by the desired accuracy of the approximation. Matrix–vector multiplication operations, the core of time-evolution simulations, can be also be performed in asymptotically linear time $O(N)$. The validity of the hierarchical representations and algorithms was tested in the context of 2D space-fractional diffusion equations with constant diffusivity. The latter were discretized using smooth particle approximations using fixed uniform grids, which led to dense discrete operators involving explicit radial kernels. The numerical applications first focused on the simulation of the fundamental solution of the unforced space fractional in an unbounded domain. Using the analytical solution of the problem, the tests showed that, for the entire range of parameters and fractional orders considered, results obtained using the hierarchical approximations were in close agreement with results obtained using dense operators, and exhibited the same spatial order of convergence. The implementations also considered simulation of the steady, forced, space-fractional diffusion equation in compact domain with Dirichlet boundary conditions. This setup was specifically used to demonstrate the possibility of practically constructing a hierarchical inverse of the system matrix via iterative methods. In particular, we showed that few iterations of a high order Newton–Schulz method can produce high quality and robust preconditioners for various values of the fractional order α. The validity of the resulting approximations was analyzed using the original dense operators, as well as solutions obtained using a pseudo-transient approaches relying on either dense operators or their hierarchical representations.

An important feature of the algorithms is that they are also amenable to efficient implementation on GPUs. Our experiments show that the construction of the fractional diffusion matrix, the matrix–vector multiplication, and the generation of an approximate inverse preconditioner all perform very well on two-dimensional problems of size in the 10^5–10^6 range with a single GPU. The generation of the matrix and its approximate inverse can be done in minutes, and the core matrix–vector multiplication can be done in milliseconds, to useful accuracy, on problems of this size. We expect that a multiGPU version of these algorithms will scale near linearly to larger sizes. Overall, the $\mathcal{H}^2$ matrix framework promises to provide the practical ability to handle multidimensional large scale fractional diffusion simulations at a computational cost that is asymptotically similar to the cost of handling classical diffusion equations. In future work, we plan to take advantage of various integral approximations of the fractional diffusion flux and source term, which provide a natural framework for accommodating generalized boundary conditions. In particular, we plan to capitalize on strength exchange type approximations [25], which appear to be ideally suited to address generalized Neumann type conditions. Because they involve smooth positive kernels, having similar asymptotic properties as those explored in the present work, one would expect that the $\mathcal{H}^2$ matrix framework would also prove effective in such settings. Finally, we also plan to explore extensions of this framework to large-scale 3D problems, and to models involving variable coefficients and/or variable order fractional operators.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Research reported in this publication was supported by research funding from King Abdullah University of Science and Technology (KAUST).

Appendix A. The fundamental solution

The fundamental solution of the fractional diffusion equation in $\mathbb{R}^d$ can be expressed in integral form as [31]:

$$\mathcal{G}_\alpha(\mathbf{x}, t) = \frac{1}{t^{d/\alpha}} \Phi_{d,\alpha}\left(\frac{1}{t^{1/\alpha}}\mathbf{x}\right) \tag{A.1}$$

where the (dimensionless) shape function $\Phi_{d,\alpha}$ is given by:

$$\Phi_{d,\alpha}(\mathbf{r}) = \frac{1}{(2\pi)^d}\int_{\mathbb{R}^d} e^{-|\mathbf{w}|^\alpha - i\mathbf{r}\cdot\mathbf{w}}\, \mathrm{d}V(\mathbf{w}). \tag{A.2}$$

It can be verified that as $\alpha \to 2$, $\mathcal{G}_\alpha$ becomes the fundamental solution of the Fickian diffusion equation [31].

In this work we relied on the two-dimensional fundamental solution, containing the function [31]

$$\Phi_{2,\alpha}(r) = \frac{1}{2\pi}\int_0^\infty \omega e^{-\omega^\alpha} \mathrm{J}_0(r\omega)\, \mathrm{d}\omega, \tag{A.3}$$

where $\mathrm{J}_0(x)$ is the Bessel function of the first kind [45].

A characteristic length R_α can be obtained as first moment of $\Phi_{1,\alpha}$:

$$R_\alpha \equiv \frac{\displaystyle\int_{-\infty}^{\infty} |x|\, \Phi_{1,\alpha}(x)\mathrm{d}x}{\displaystyle\int_{-\infty}^{\infty} \Phi_{1,\alpha}(x)\mathrm{d}x}. \tag{A.4}$$

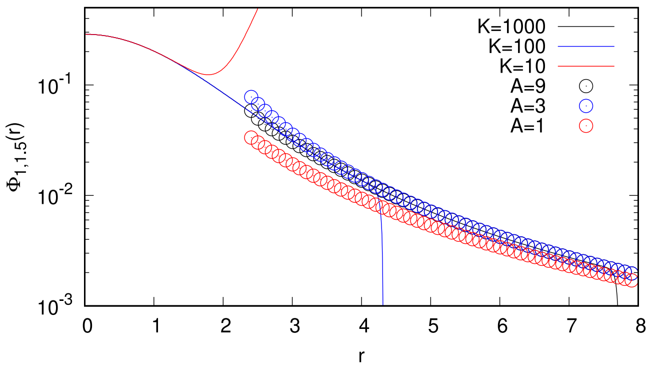


Fig. A.15. $\Phi_{1,1.5}(r)$, given by Eq. (A.5) (solid lines) and by Eq. (A.6) (symbols), represented with K and A terms respectively. Note that R_α rapidly decays with increasing α.

The function $\Phi_{1,\alpha}$ is an even, bell-shaped function, having the series representation [63]:

$$\Phi_{1,\alpha}(x) = \frac{1}{x\pi}\sum_{m=1}^{\infty}(-x)^m \frac{\Gamma(1+m/\alpha)}{m!}\sin\left[\frac{-m\pi}{2}\right]. \tag{A.5}$$

For $x \to +\infty$, it admits the following asymptotic expansion [63]:

$$\Phi_{1,\alpha}(x) \sim \frac{-1}{x\pi}\sum_{m=1}^{\infty}(-x^{-\alpha})^m \frac{\Gamma(1+\alpha m)}{m!}\sin\left[\alpha m\frac{\pi}{2}\right]. \tag{A.6}$$

$\Phi_{1,1.5}$ is illustrated in Fig. A.15, showing the convergence of Eq. (A.5) and (A.6) with respect to the number of terms in the series.

In our simulations, we used the values $R_{1.1} = 6.688$, $R_{1.5} = 1.705$, $R_{1.9} = 1.19$.

Appendix B. Derivation of the direct differentiation method

In this appendix, we provide a brief outline of the construction of the DD method, as originally proposed in [25]. We aim to solve the fractional diffusion equation (3), which features the fractional Laplacian (4). The Fourier transform of the fractional Laplacian is:

$$\mathcal{F}\left[\Delta^\alpha\right] = -|2\pi f|^\alpha. \tag{B.1}$$

where f is the frequency. Using the basic properties of the Fourier transform, the fractional Laplacian of u_ς (5) is:

$$\mathcal{F}[\Delta^\alpha u_\varsigma] = -|2\pi f|^\alpha \mathcal{F}[\eta_\varsigma]\mathcal{F}[u], \tag{B.2}$$

where $\mathcal{F}[\eta_\varsigma]$ denotes the Fourier transform of the regularization kernel. For the second-order exponential kernel in (6), the Fourier transform is:

$$\mathcal{F}\left[\frac{\exp\left(-\frac{\mathbf{x}^2}{\varsigma^2}\right)}{\varsigma^d \pi^{\frac{d}{2}}}\right] = \exp\left(-\pi^2\varsigma^2 f^2\right). \tag{B.3}$$

One readily observes that the fractional Laplacian of u_ς can be expressed as

$$\Delta^\alpha u_\varsigma = \varsigma^{-\alpha} u * G_\varsigma(\mathbf{x}), \tag{B.4}$$

where $G_\varsigma(\mathbf{x})$ is a function defined via its Fourier transform:

$$\mathcal{F}[\varsigma^{-\alpha} G_\varsigma(\mathbf{x})] = -|2\pi f|^\alpha \mathcal{F}[\eta_\varsigma]. \tag{B.5}$$

Making use of the result

$$\int_{\mathbb{R}^d} |\mathbf{y}|^{\mu} \exp\left(-a|\mathbf{y}|^2 + ib\mathbf{x}\cdot\mathbf{y}\right) \mathrm{d}V(\mathbf{y}) = \pi^{\frac{d}{2}} a^{-\frac{\mu+d}{2}} \frac{\Gamma\left(\frac{\mu+d}{2}\right)}{\Gamma\left(\frac{d}{2}\right)} {}_1F_1\left(\frac{\mu+d}{2}, \frac{d}{2}, -\frac{b^2}{4a}|\mathbf{x}|^2\right), \tag{B.6}$$

we obtain the kernel G_ς given in (8). The discrete evolution equation (7) is obtained by applying the midpoint rule to (B.4). The same result is obtained if one differentiates term by term the particle approximation of u_ς in (5).

References

[1] Q. Du, Nonlocal Modeling, Analysis, and Computation, SIAM, 2019.

[2] J. Klafter, I.M. Sokolov, Anomalous diffusion spreads its wings, Phys. World 18 (8) (2005) 29.

[3] P. Gatto, J.S. Hesthaven, Numerical approximation of the fractional laplacian via *hp*-finite elements, with an application to image denoising, J. Sci. Comput. 65 (1) (2015) 249–270.

[4] S.G. Kou, A jump-diffusion model for option pricing, Manage. Sci. 48 (8) (2002) 1086–1101.

[5] C. Li, M. Cai, Theory and Numerical Approximations of Fractional Integrals and Derivatives, SIAM, 2020.

[6] M. Kwasnicki, Ten equivalent definitions of the fractional Laplace operator, Fract. Calc. Appl. Anal. 20 (1) (2017) 7–51.

[7] V.E. Lynch, B.A. Carreras, D. del Castillo-Negrete, K. Ferreira-Mejias, H. Hicks, Numerical methods for the solution of partial differential equations of fractional order, J. Comput. Phys. 192 (2) (2003) 406–421.

[8] C. Tadjeran, M.M. Meerschaert, H.-P. Scheffler, A second-order accurate numerical approximation for the fractional diffusion equation, J. Comput. Phys. 213 (1) (2006) 205–213.

[9] H. Zhang, F. Liu, V. Anh, Galerkin finite element approximation of symmetric space-fractional partial differential equations, Appl. Math. Comput. 217 (6) (2010) 2534–2545.

[10] X. Li, C. Xu, Existence and uniqueness of the weak solution of the space–time fractional diffusion equation and a spectral method approximation, Commun. Comput. Phys. 8 (5) (2010) 1016.

[11] K. Mustapha, An implicit finite-difference time-stepping method for a sub-diffusion equation, with spatial discretization by finite elements, IMA J. Numer. Anal. 31 (2) (2010) 719–739.

[12] C. Çelik, M. Duman, Crank–Nicolson method for the fractional diffusion equation with the Riesz fractional derivative, J. Comput. Phys. 231 (4) (2012) 1743–1750.

[13] K. Mustapha, W. McLean, Superconvergence of a discontinuous Galerkin method for fractional diffusion and wave equations, SIAM J. Numer. Anal. 51 (1) (2013) 491–515.

[14] W. Deng, J.S. Hesthaven, Local discontinuous Galerkin methods for fractional diffusion equations, ESAIM Math. Model. Numer.Anal. 47 (6) (2013) 1845–1864.

[15] K. Mustapha, B. Abdallah, K. Furati, A discontinuous Petrov–Galerkin method for time-fractional diffusion equations, SIAM J. Numer. Anal. 52 (5) (2014) 2512–2529.

[16] H. Zhou, W. Tian, W. Deng, Quasi-compact finite difference schemes for space fractional diffusion equations, J. Sci. Comp. 56 (1) (2013) 45–66.

[17] K. Mustapha, Time-stepping discontinuous Galerkin methods for fractional diffusion problems, Numer. Math. 130 (3) (2015) 497–516.

[18] Y. Liu, Y. Yan, M. Khan, Discontinuous Galerkin time stepping method for solving linear space fractional partial differential equations, Appl. Numer. Math. 115 (2017) 200–213.

[19] R. Gorenflo, F. Mainardi, D. Moretti, G. Pagnini, P. Paradisi, Discrete random walk models for space–time fractional diffusion, Chem. Phys. 284 (1) (2002) 521–541.

[20] R. Gorenflo, A. Vivoli, F. Mainardi, Discrete and continuous random walk models for space–time fractional diffusion, Nonlinear Dynam. 38 (1) (2004) 101–116.

[21] R. Gorenflo, F. Mainardi, A. Vivoli, Continuous-time random walk and parametric subordination in fractional diffusion, Chaos Solitons Fractals 34 (1) (2007) 87–103.

[22] H. Zhang, F. Liu, V. Anh, Numerical approximation of Lévy–Feller diffusion equation and its probability interpretation, J. Comput. Appl. Math. 206 (2) (2007) 1098–1115.

[23] L. Shi, Z. Yu, Z. Mao, A. Xiao, H. Huang, Space–time fractional diffusion equations and asymptotic behaviors of a coupled continuous time random walk model, Physica A 392 (23) (2013) 5801–5807.

[24] Y. Luchko, A new fractional calculus model for the two-dimensional anomalous diffusion and its analysis, Math. Model. Nat. Phenom. 11 (3) (2016) 1–17.

[25] M. Lucchesi, S. Allouch, O. Le Maître, K. Mustapha, O. Knio, Particle simulation of space-fractional diffusion equations, Comput. Part. Mech. 7 (2020) 491–507.

[26] V. Minden, L. Ying, A simple solver for the fractional Laplacian in multiple dimensions, 2018, arXiv:1802.03770.

[27] X. Zhao, X. Hu, W. Cai, G.E. Karniadakis, Adaptive finite element method for fractional differential equations using hierarchical matrices, Comput. Methods Appl. Mech. Engrg. 325 (2017) 56–76.

[28] M. Karkulik, J.M. Melenk, $\mathcal{H}$-matrix approximability of inverses of discretizations of the fractional Laplacian, Adv. Comput. Math. 46 (2019).

[29] K. Xu, E. Darve, Efficient numerical method for models driven by Lévy process via hierarchical matrices, 2018, arXiv:1812.08324.

[30] D. Keyes, H. Ltaief, G. Turkiyyah, Hierarchical algorithms on hierarchical architectures, Philos. Trans. R. Soc. A 378 (2019) 201900, http://dx.doi.org/10.1098/rsta.2019.0055.

[31] C. Pozrikidis, The Fractional Laplacian, Chapman and Hall/CRC, 2016.

[32] A.J. Chorin, Numerical study of slightly viscous flow, J. Fluid Mech. 57 (4) (1973) 785–796.

[33] A. Leonard, Vortex methods for flow simulation, J. Comput. Phys. 37 (3) (1980) 289–335.

[34] A. Leonard, Computing three-dimensional incompressible flows with vortex elements, Annu. Rev. Fluid Mech. 17 (1) (1985) 523–559.

[35] G.-H. Cottet, P.D. Koumoutsakos, Vortex Methods: Theory and Practice, Cambridge University Press, 2000.

[36] D. Fishelov, A new vortex scheme for viscous flows, J. Comput. Phys. 86 (1) (1990) 211–224.

[37] J.T. Beale, A. Majda, High order accurate vortex methods with explicit velocity kernels, J. Comput. Phys. 58 (2) (1985) 188–208.

[38] J.D. Eldredge, A. Leonard, T. Colonius, A general deterministic treatment of derivatives in particle methods, J. Comput. Phys. 180 (2) (2002) 686–709.

[39] P. Degond, S. Mas-Gallic, The weighted particle method for convection–diffusion equations. I. The case of an isotropic viscosity, Math. Comp. 53 (188) (1989) 485–507.

[40] P.-A. Raviart, An analysis of particle methods, in: Numerical Methods in Fluid Dynamics, Springer, 1985, pp. 243–324..

[41] P. Degond, F.-J. Mustieles, A deterministic approximation of diffusion equations using particles, SIAM J. Sci. Stat. Comput. 11 (2) (1990) 293–310.

[42] O.M. Knio, A.F. Ghoniem, Vortex simulation of a three-dimensional reacting shear layer with infinite-rate kinetics, AIAA J. 30 (1) (1992) 105–116.

[43] J. Choquin, S. Huberson, Particles simulation of viscous flow, Comput. Fluid 17 (2) (1989) 397–410.

[44] P. Mycek, G. Pinon, G. Germain, E. Rivoalen, Formulation and analysis of a diffusion-velocity particle model for transport-dispersion equations, Comput. Appl. Math. 35 (2) (2016) 447–473.

[45] K.B. Oldham, J. Myland, J. Spanier, An Atlas of Functions: With Equator, the Atlas Function Calculator, Springer Science & Business Media, 2010.

[46] J.W. Pearson, Computation of Hypergeometric Functions (Ph.D. thesis), University of Oxford, 2009.

[47] W. Hackbusch, Hierarchical Matrices: Algorithms and Analysis, Springer, 2015.

[48] W. Boukaram, G. Turkiyyah, D. Keyes, Hierarchical matrix operations on GPUs: Matrix–vector multiplication and compression, ACM Trans. Math. Software 45 (1) (2019) 3:1–3:28.

[49] S. Börm, L. Grasedyck, Hybrid cross approximation of integral operators, Numer. Math. 101 (2) (2005) 221–249.

[50] M. Bebendorf, S. Rjasanow, Adaptive low-rank approximation of collocation matrices, Computing 70 (1) (2003) 1–24.

[51] D. Cai, E. Chow, L. Erlandson, Y. Saad, Y. Xi, Smash: Structured matrix approximation by separation and hierarchy, Numer. Linear Algebra Appl. 25 (6) (2018) e2204.

[52] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the laplace equation in three dimensions, Acta Numer. 6 (1997) 229–269.

[53] S. Börm, Efficient Numerical Methods for Non-Local Operators: $\mathcal{H}^2$-Matrix Compression, Algorithms and Analysis, vol. 14, European Mathematical Society, 2010.

[54] W. Boukaram, G. Turkiyyah, D. Keyes, Randomized GPU algorithms for the construction of hierarchical matrices from matrix–vector operations, SIAM J. Sci. Comput. 41 (4) (2019) C339–C366.

[55] N. Halko, P. Martinsson, J.A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288.

[56] R. Kriemann, S.L. Borne, $\mathcal{H}$-FAINV: hierarchically factored approximate inverse preconditioners, Comput. Vis. Sci. 17 (3) (2015) 135–150.

[57] G. Chavez, G.M. Turkiyyah, S. Zampini, D.E. Keyes, Parallel accelerated cyclic reduction preconditioner for three-dimensional elliptic pdes with variable coefficients, J. Comput. Appl. Math. 344 (2018) 760–781.

[58] V. Pan, F. Soleymani, L. Zhao, An efficient computation of generalized inverse of a matrix, Appl. Math. Comput. 316 (2018) 89–101.

[59] W. McLean, K. Mustapha, A second-order accurate numerical method for a fractional wave equation, Numer. Math. 105 (2007) 481–510.

[60] K. Mustapha, An implicit finite difference time-stepping method for a sub-diffusion equation with spatial discretization by finite elements, IMA J. Numer. Anal. 31 (2011) 719–739.

[61] R.H. Nochetto, E. Otárola, A.J. Salgado, A PDE approach to fractional diffusion in general domains: A priori error analysis, Found. Comput. Math. 15 (2015) 733–791, http://dx.doi.org/10.1007/s10208-014-9208-x.

[62] V.J. Ervin, N. Heuer, J.P. Roop, Regularity of the solution to 1-d fractional order diffusion equations, Math. Comp. 87 (2018) 2273–2294, http://dx.doi.org/10.1090/mcom/3295.

[63] F. Mainardi, Y. Luchko, G. Pagnini, The fundamental solution of the space–time fractional diffusion equation, Fract. Calc. Appl. Anal. 4 (2001) 153–192.