A Framework for Solving Fractional Laplacian Problems using Eigenfunction Expansion

<Author 1>
<Institution 1>
<City 1>, <State 1>
<E-mail1>

<Author 2>
<Institution 1>
<City 1>, <State 1>
<E-mail2>

<Author 3>
<Institution 1>
<City 1>, <State 1>
<E-mail3>

ABSTRACT

In order to study the non-local diffusion operator known as the fractional laplacian, we have developed a framework to solve a simple fractional poisson problem using the spectral expansion method. This framework requires computation of a set of eigenpairs for the given problem geometry. For analytical purposes, we compute the entire eigenbasis in order to explore the impact of the non-local nature of the fractional laplacian on the convergence of the spectral expansion. Additionally, we employ the zfp compression library to reduce the amount of data required to store the full eigenbasis. Using this compression library, we explore strategies for balancing data reduction and performance impact in addition to leveraging any aspect of the non-local operator to improve either data reduction or performance.

ACM Reference Format:
<Author 1>, <Author 2>, and <Author 3>. 2019. A Framework for Solving Fractional Laplacian Problems using Eigenfunction Expansion. In *Proceedings of ICPP 2019 (ICPP'19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Anomalous diffusion is a type of diffusion that has a non-linear relationship with time and has been found to describe certain physical phenomenon better than typical diffusion. Add examples of anomalous diffusion in nature with citations.

Not only is anomalous diffusion interesting as a physical phenomenon, it also has a direct connection to the concept of fractional differential operators in mathematics. Unlike typical differential operators, fractional operators are inherently non-local. This property then has an impact on the type of solvers used for solving problems related to fractional operators. In this paper, we present a framework we have developed for computing approximate solutions to partial different equations that have an anomalous diffusion term. The specific operator that describes anomalous diffusion is known as the Fractional Laplacian and a more detailed discussion can be found in [1].

2 FRACTIONAL LAPLACIAN

Fractional operators are inherently non-local by definition and is partially why they are of interest. Because of this non-local property, typical approaches for solving fractional PDEs can be untenable or become more computationally expensive. For instance, one of the benefits of finite element methods for solving PDEs is that they result in sparse systems to be solved. Unfortunately, for fractional problems, FEM discretization results in a system becomes fully dense, making it a much more computationally expensive approach.

The model problem we use to verify the correctness of the eigenfunction expansion approach is the simple fractional poisson problem with zero dirichlet boundary conditions, as seen in equation (1). One of the benefits of this model problem is its simplicity and that it has some explicit solutions that can aid in the verification of our approach.

$$\begin{aligned} -\Delta \mathrm{u} &= \mathrm{f} && \in \Omega \\ \mathrm{u} &= 0 && \in \partial\Omega \end{aligned} \tag{1}$$

If we are going to have to solve a dense N by N system every time we want to solve a fractional poisson problem, then it is perhaps a better idea to avoid FEM and instead utilize an alternative approach. One such alternative approach is the method of eigenfunction expansion.

One description of this method, and further background details, can be found in [1]. For a bounded domain with zero dirichlet boundary condition, the fractional laplacian with respect to α is equivalent to the expansion seen in equation (2).

$$-(\Delta)^{\alpha/2}u(x) \approx \sum_{k=1}^{\infty} \lambda_k^{\alpha/2}(u, v_k)v_k \tag{2}$$

By expanding the function f in the model problem (1) and plugging in the expansion for $-(\Delta)^{\alpha/2}$, the solution u can then be obtained as seen in equation (3).

$$u \approx \sum_{k=1}^{\infty} \lambda_k^{-\alpha/2}(f, v_k)v_k \tag{3}$$

One of the benefits of this formulation of the solution is that the effect of the fractional operator's parameter α is limited to scaling λ_k. This means that the eigenfunction v_k and eigenvalue λ_k are simply those of the typical Laplace eigenvalue problem with zero dirichlet boundary condition, specifically those that solve equation (4).

$$\begin{aligned} -\Delta v_k &= \lambda_k v_k && \in \Omega \\ v_k &= 0 && \in \partial\Omega \end{aligned} \tag{4}$$

Since this is the typical Laplace eigenvalue problem, we can use existing techniques to produce a discretized system to be solved for as many eigenpairs as we need. Since the purpose of this framework is to be used to study fractional problems of all kinds, we specifically solve for the full set of discretized eigenpairs which comes with specific challenges for taking full advantage of available computing resources.

ICPP'19, August 2019, Kyoto, Japan
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Another of the benefits of this approach is that instead of solving an N by N system, we can compute the solution by applying the discrete eigenbasis V to the vector of scaled coefficients λ. Additionally, since many of the coefficients will result in a contribution that is so small that it won't have any measurable impact, many of the columns of V can be ignored depending on the input.

3 EIGENBASIS COMPUTATION

In order to compute solutions to our fractional model problem, the method of eigenfunction expansion requires the computation of the discrete eigenbasis for a given geometry. Since scalability is the primary computational concern for this framework, we build our framework on top of PETSc [3] [4] and SLEPc [5]. PETSc is a parallel and scalable library of routines and data structures for scientific applications. PETSc handles how our matrices are stored in parallel and for operations like matrix multiplication.

Since our framework is built in the method of eigenfunction expansion, we build on top of the SLEPc library for solving the resulting eigenvalue problem. SLEPc has a number of routines available for solving eigenvalue problems and is also built on PETSc. Our approach requires computing the full set of eigenpairs for a system and therefore our framework uses a modified version of spectrum slicing with the independent eigenvalue problems being solved with SLEPc.

<Introduce Nektar++, expand nektar++ integration description, cite [2]>

3.1 Method

In order to construct solutions to the fractional poisson problem, we need to discretize the domain of interest. Using the finite element method, geometry can be converted into the symmetric, discrete system $Kv_k = \lambda_k M v_k$. For this, we have integerated our framework with Nektar++ which is used to convert a given mesh into two PETSc matrices K and M.

Equipped with these matrices, we can compute all of the eigenpairs in a given interval using SLEPc's Krylov-Schur solver. Krylov-Schur is an iterative solver that relies on applying K and M repeatedly and solving related systems. If we were to simply give K and M to SLEPc, these sparse matrices would be divided among the entire set of available processes. Using this naive approach, throwing more processes at the problem will just result in more communication between nodes and will hurt performance. Instead, we use an approach to split the full eigenbasis computation into independent subproblems that can be handled by a single node. This results in a much more scalable approach.

3.1.1 Spectrum Slicing. In order to compute the full set of eigenpairs using the available computing resources, we use the method of spectrum slicing. Because we know the eigenvalue problem is symmetric, all of the eigenvalues are real, and because the problem is semidefinite, all of the eigenvalues are contained in the interval $[0, U]$. Instead of solving for all eigenpairs within this interval using all processes, we break it into independent subintervals that can be solved in parallel. This is possible, again, due to the symmetry of the problem.

For a symmetric system, if any two eigenvectors have different eigenvalues, they are inherently orthogonal. Therefore any eigenvectors that are in separate subintervals can be solved for independently without any need for orthogonalization between subintervals.

Each subinterval is solved by a group of processes we refer to as an evaluator. Each evaluator is comprised of a number of nodes that is determined based on problem size and then each node is broken up into its component processes. This communication hierarchy ensures that each evaluator's system is spread out across as few nodes as possible, resulting in minimal communication when applying the discretized operators.

For this approach, we then needed some method for breaking the interval into subintervals that require equal amounts of work to solve. If we were to instead assume that the eigenvalues are equally distributed and then split the interval evenly among evaluators, very bad load imbalance occurs. The worst case of this load imbalance occurs using this naive splitting with two evaluators. For a simple cube domain, the lower half of the interval contains the vast majority of eigenvalues, and therefore the second evaluator has almost no work to do.

One potential way of making this assumption of uniformly distributed eigenvalues work as a splitting method is to split the interval into a large number of subintervals per evaluator and then distribute them in a round-robin fashion. This approach does reduce the load imbalance to some extent but isn't reliable. We did end up using some ideas from this, specifically having many subintervals per evaluator distributed in a round-robin fashion. This idea was helpful so that no single evaluator had all of their subintervals in a region that is very sparse.

3.1.2 Counting an Interval. In order to count how many eigenvalues are contained in an interval [a,b], there are two techniques that we explored. Both approaches are based on counting the number of eigenvalues greater than a and then the number greater than b and taking the difference to get the interval count.

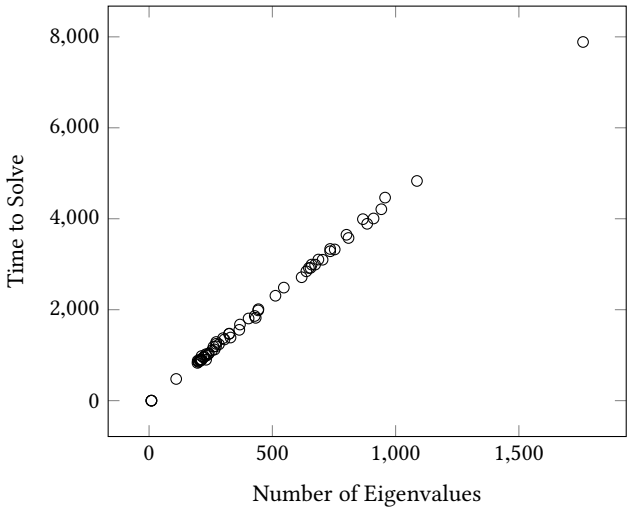
Given some matrix A that can be factored such that $A = LDL^T$, where L is a lower triangular matrix and D is a diagonal matrix, the Sylvester inertia theorem states that the inertia of D and A are the same. This means that the number of entries of D that are greater than 0 is also the number of eigenvalues of A that are greater than 0. Therefore if we let A be the shifted eigenvalue problem centered around a, specifically that $A = K - aM$, we can compute this factorization to get the number of eigenvalues greater than some given value of a.

While this method results in very accurate eigenvalue counts, it becomes prohibitively expensive as the size of the problem increases. Therefore an approximate counting technique was required to find a balance between accuracy and scaling complexity. For this, we use a method called polynomial expansion filtering.

This approach, which is outline in full detail in [6], relies on two separate approximations that are composed together. The first bit of machinery is that of approximating the trace of a matrix using n_v random vectors. Specifically, this approximation can be seen in equation (5).

$$\mathrm{tr}(A) \approx \frac{1}{n_v} \sum_{k=1}^{n_v} v_k^T A v_k \tag{5}$$

Figure 1: Comparison of number of eigenvalues within a subinterval and the amount of time taken to solve for all eigenpairs within the subinterval. Observe the clear linear relationship between the two values.



The approximate count of eigenvalues in an interval is computed by approximating the trace of the eigenprojector of A, which is referred to as P. The second bit of machinery is that of approximating this eigenproject P by polynomial expansion filtering as seen in equation (6). By plugging the matrix A into the chebyshev expansion of degree p of the heaviside function centered at 0, an approximate form of P is constructed.

$$P \approx \phi_p(A) = \sum_{j=0}^{p} \gamma_j T_j(A) \tag{6}$$

For the generalized eigenvalue problem we are attempting to solve, the count of eigenvalues in an interval $[a, b]$ is then the difference of the counts for the two shifted systems $K - aM$ and $K - bM$. Specifically, this can be seen in equation (7).

$$\mu_{[a,b]} = \mu_a - \mu_b \approx \mathrm{tr}[\phi_p(K - aM) - \phi_p(K - bM)] \tag{7}$$

3.2 Results

One of the main assumptions that determine the load balancing of the spectrum slicing approach is that if two independent subintervals have an equal number of eigenvalues, that they take equal amounts of time to solve. Some of the early experiments with the approximate eigenvalue counting technique resulted in very uneven distribution of eigenvalues per subinterval. However, using the data from these experiments, we plotted each subinterval's solve time and eigenvalue count and can be seen in figure 1. From this it can be seen there is a very distinct linear relationship between the eigenvalue count and solve time.

In order to verify that our method results in near-optimal load balancing, we ran the eigenbasis computation for a range of problem sizes and number of evaluators. These experiments were run on the <Institution 1>'s <Cluster 1> computing cluster with 28 processes per node and a single node per evaluator. The results of this scaling experiment can be seen in figure 2.

Figure 2: The time it takes to compute the full eigenbasis for various problem sizes vs the number of nodes used.

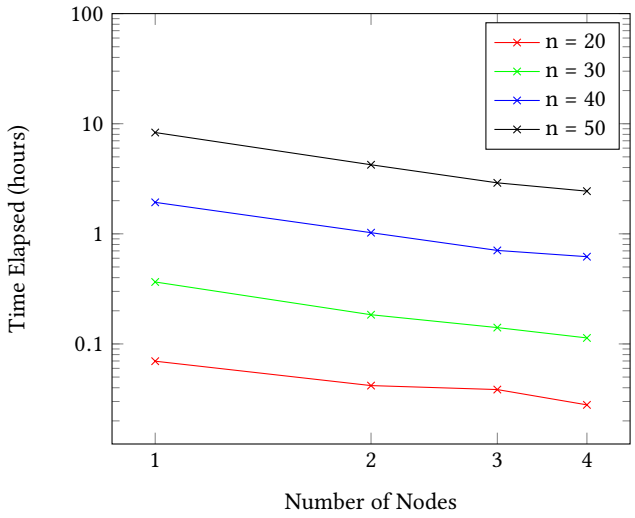
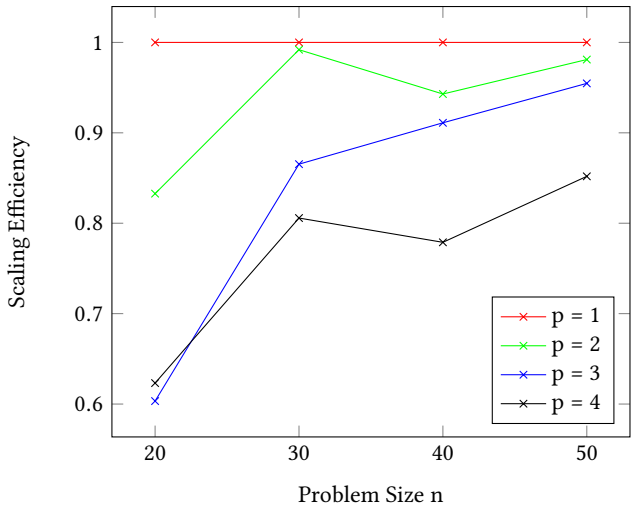


Figure 3: Scaling efficiency vs problem size with respect to number of nodes.



Since the total eigenbasis computation is divided into smaller independent tasks, one way to determine load balancing is to observe the scaling efficiency per number of evaluators. For p evaluators, the scaling efficiency is then equal to $t_1/(p \cdot t_p)$, where t_1 is the time elapsed for a single evaluator and t_p is the maximum time elapsed for all of the p evaluators. The closer this value is to 1, the closer the work is divided perfectly evenly among evaluators. Ideally, this value should be 1 or close to it. The resulting scaling efficiency can then be seen in figure 3.

<Comparison of performance and accuracy of exact and approximate interval counting>

3.3 Potential Improvements

Our framework results in close to optimal load balancing for the simple domains we have tested, however, it isn't necessarily guaranteed. We could make better guarantees of the load balancing by designating a single process to maintain a queue of subintervals to be processed. Using this approach, as each evaluator finishes a subinterval, it can report to the task manager and request the next subinterval. This way, the only time any evaluator is starved for work is when there is no more work to be done.

Since the newest versions of PETSc have incorporated GPU acceleration, we could then set up GPU evaluators alongside the previously mentioned CPU evaluators. In this case, a task queue would be essential to properly manage the work among the different types of evaluators. Even if the GPU evaluators are not faster than the CPU evaluators, they would be able to function in parallel, thus resulting in an effective evaluator count greater than the actual number of machines available.

4 EIGENBASIS STORAGE AND COMPRESSION

For our N by N sparse system, we are computing the entire set of N eigenvectors and eigenvalues. The resulting eigenbasis is a dense N by N matrix V. The amount of memory required to store V can be excessively large and scales rapidly with respect to the number of elements in the discretization. Therefore some method needed to be employed to reduce the storage requirements and potentially also reduce the amount of work necessary to apply the eigenbasis. In order to compress the data produced by our framework, we utilized Lawrence Livermore's zfp compression library [7] to explore different strategies.

Since V is a PETSc matrix that is distributed among all of the available processes, the first compression strategy we explored was to have each process use zfp to compress its local block. This approach doesn't help reduce the amount of work to apply V but it gives us an idea of how much compression is possible using zfp.

The second approach is to compress each eigenvector independently. For any given input function f for the fractional poisson problem, there may be many eigenvectors (if not the majority) that will have scaling coefficients smaller than machine precision. By using this compression strategy, we could only decompress the specific eigenvectors we need for a given f and skip any that will have no effect on the solution. By only keeping the m most recently used eigenvectors in the decompressed state, we could also minimize the amount of time compressing and decompressing eigenvectors.

4.1 Results

Using the already computed eigenbasis V for $N = 30^3$, we ran experiments for both compression strategies to determine the time it takes to compress and the amount of data reduction achieved. For the first strategy, we compressed each local block using two evaluators, each with 28 processes. In figures 4 and 5, the results of this experiment can be seen.

Then the compression experiment is repeated for strategy 2, where instead of computing the local data blocks, each column of the matrix V is compressed independently. The results of this compression experiment can then be seen in figures 6 and 7.

Figure 4: The amount of data reduction achieved by compressing each process's local block.

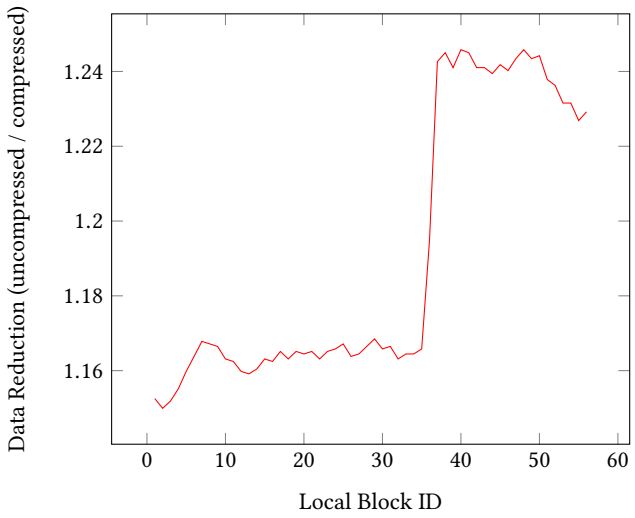
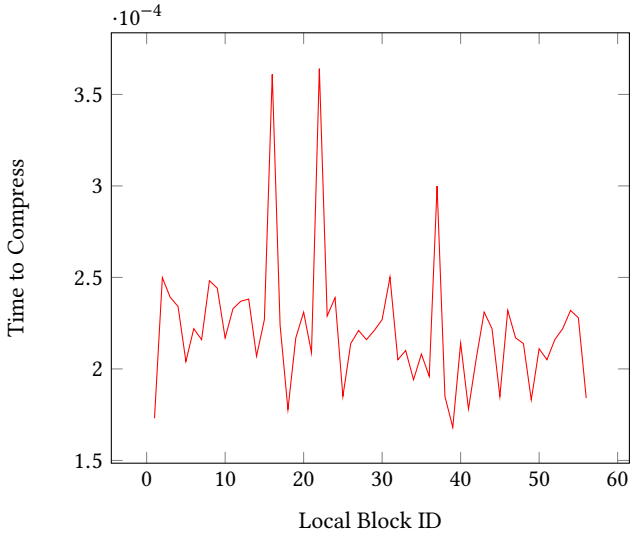


Figure 5: The amount of time it took to compress each process's local block.



An important thing to note is that zfp is designed for compressing uniformly sampled data sets. Since we are compressing eigenvectors for arbitrary complex geometry, the data that is being compressed is not uniformly sampled and therefore the data reduction is fairly modest. Since there is also no correlation between eigenvectors, zfp's strengths are not being fully taken advantage of. A more fitting approach would be to use something that utilizes the geometry of the problem to compress the eigenvectors.

Figure 6: The amount of data reduction achieved by compressing each individual eigenvector, sorted in ascending order by eigenvalue.

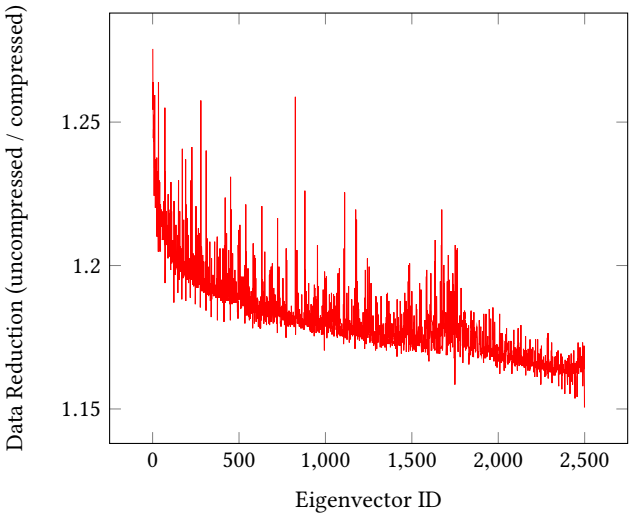
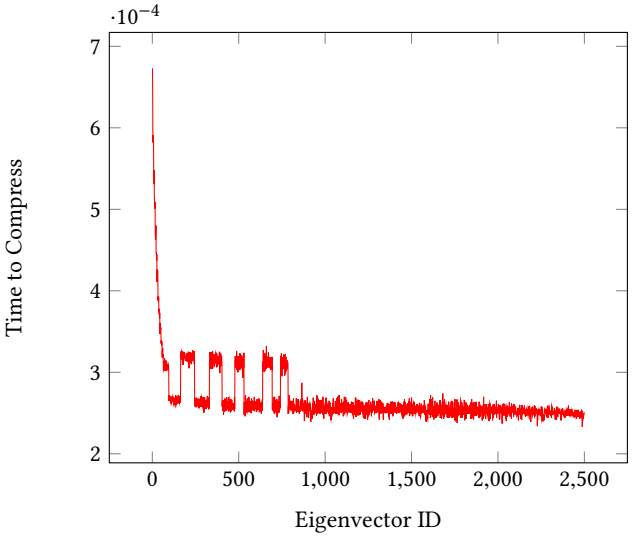


Figure 7: The amount of time it took to compress each individual eigenvector, sorted in ascending order by eigenvalue.



Additionally, since these experiments were run using a simple cube domain, for at least one dimension of the data, there is correlation between neighboring entries. Even with this benefit, the data reduction is fairly modest. We then repeated the compression experiment using a sphere geometry where zfp would see no benefit and the results can be seen in figures ? and ?.

<Data Reduction: Sphere, Strategy 1>

<Data Reduction: Sphere, Strategy 2>

5 RESULTS

with our framework, we now have access to the eigenbasis necessary to construct approximate solutions to our fractional model problem. In order to determine the correctness, we compare our approximate eigenfunction expansion solution with an exact solution for disc and sphere geometry.

<Equations and description of exact solution>

<Plot scaled coefficients for f = 1, disc>

<Show convergence of spectral expansion for f = 1, disc>

<Example solution that targets unusual or non-contiguous portion of the spectrum, coefficient plot and solution plot>

<Compare computed eigenvalues with analytical eigenvalues, [cube, sphere surface]>

6 CONCLUSION

In conclusion, we have developed an early version of a scalable framework for computing approximate solutions to fractional PDEs and presented some results for some simple geometries. This is the first step towards more interesting solvers based on the method of eigenfunction expansion.

In addition to computing the eigenbasis in a scalable, efficient manner, we have laid the groundwork for a compression scheme that will both reduce the amount of data required for such a method as well as reducing the amount of work for applying the eigenbasis. While zfp didn't give quite the results we would prefer, the next step would be to explore geometry-aware compression schemes along the lines of FMM. One of the benefits of such a scheme would be that the geometry doesn't change on an eigenvector by eigenvector basis, any of the geometry-related data structures would only need to be computed once for the mesh. In addition to this, a data structure for compressing and decompressing eigenvectors as needed would need to be developed and tested.

While the eigenbasis computation is in a scalable and efficient state, without GPU acceleration and a task manager, we are leaving computing resources on the table so to speak. Further development on this framework will need to include these features in order to achieve peak efficiency on heterogeneous computing nodes.

Looking ahead, we hope to explore more complicated fractional PDEs such as space-fractional Diffusion-Reaction or Cahn-Hiliard. This framework will act as the foundation for these explorations and should allow for very large-scale experiments to be done to verify existing and future theoretical work in this field.

REFERENCES

[1] Anna Lischke, Guofei Pang, Mamikon Gulian, Fangying Song, Christian Glusa, Xiaoning Zheng, Zhiping Mao, Wei Cai, Mark M. Meerschaert, Mark Ainsworth, and George Em Karniadakis. What Is the Fractional Laplacian? *arXiv e-prints*, page arXiv:1801.09767, Jan 2018.

[2] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205 – 219, 2015.

[3] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[4] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F.

Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.11, Argonne National Laboratory, 2019.

[5] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[6] Edoardo Di Napoli, Eric Polizzi, and Yousef Saad. Efficient estimation of eigenvalue counts in an interval. *arXiv e-prints*, page arXiv:1308.4275, Aug 2013.

[7] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, Dec 2014.