CSRI SUMMER PROCEEDINGS 2021

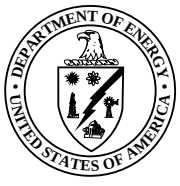
CSRI Summer Program
The Center for Computing Research at Sandia National Laboratories

Editors:
J.D. Smith and E. Galvan
Sandia National Laboratories

November 1, 2021

SAND#: SAND2021-15925 R

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.



Preface

The **Computer Science Research Institute (CSRI)** brings university faculty and students to Sandia National Laboratories for focused collaborative research on Department of Energy (DOE) computer and computational science problems. The institute provides an opportunity for university researches to learn about problems in computer and computational science at DOE laboratories, and help transfer results of their research to programs at the labs. Some specific CSRI research interest areas are: scalable solvers, optimization, algebraic preconditioners, graph-based, discrete, and combinatorial algorithms, uncertainty estimation, validation and verification methods, mesh generation, dynamic load-balancing, virus and other malicious-code defense, visualization, scalable cluster computers, beyond Moore's Law computing, exascale computing tools and application design, reduced order and multiscale modeling, parallel input/output, and theoretical computer science. The CSRI Summer Program is organized by CSRI and includes a weekly seminar series and the publication of a summer proceedings.

1. CSRI Summer Program 2021. In 2021, the CSRI summer program was executed completely virtually; all student interns worked from home due to the ongoing virus pandemic. It included students from 1400 – the Center for Computing Research (CCR), 8700 – the Center for Homeland Security & Defense Systems, 1300 – the Radiation & Electrical Science Center, and 1800 – the Material, Physical, and Chemical Science Center. This year's program included the traditional *Summer Seminar Series* and *Summer Proceedings* and continued the second annual *Virtual Poster Blitz*. As an additional opportunity, a tutorial on using Kokkos was offered. Additional details on each of these activities is provided below.

2. Seminar Series. The CSRI Summer Seminar Series is a quintessential part of the CSRI Summer Intern Program Experience. Students are exposed to a broad showcase of research from across Sandia, enriching their knowledge of the labs while providing introductions to novel subject areas. The theme for 2021 was **A Summer of HPC: To Exascale and Beyond**. These talks were focused on the history, mathematics, applications, and software of *high performance computing* (HPC) as well as new and emerging computing technologies. We extend our deepest thanks to the staff who spoke at the 2021 Seminar Series. These speakers and their talk titles are listed in Table 2.1.

Table 2.1: List of talks and speakers at the 2021 Seminar Series

Date	Name	Org	Title
6/15	James H. Laros III	1422	Those who fail to learn from history are condemned to NOT repeat it.
6/22	Mike Heroux	1400	The US Exascale Project Software Stack: Why it Matters to You.
6/29	Jerry Watkins	8754	Preparing for High-Fidelity Computational Fluid Dynamics on Exascale Systems.
7/6	Eric Cyr	1442	Perfection is the Enemy of the Fast: The Case for Mathematically Induced Parallelism.
7/13	Clayton Hughes	1422	HPC Architectures Beyond Exascale.
7/20	Corinne Teeter Denis Mamaluy Srideep Musuvathy	1421	Neuromorphic and Beyond Lightning Talk.
7/27	Hemanth Kolla	8753	Tensors for Statistical Analyses of Scientific Data: A Turbulent Combustion Perspective.
8/3	Patricia Crossno	1461	Slycat™ Ensemble Analytics.
8/10	Jon Berry Cindy Phillips	1461	Maintaining connected components for monitoring (infinite) cyber streams.
8/17	Mohan Sarovar Kevin Young Alicia Magann	8759	Quantum computing and quantum technologies: a motivation and introduction. Quantum computers: how they're different. Quantum computers: where we are now and where we are headed.
8/26	Brian Franke Shawn Pautz	1341	Algorithms and Applications of the SCEPTRE and ITS Radiation Transport Codes.
8/31	Jennifer Loe	1465	An Introduction to Trilinos.

3. Proceedings. All students and their mentors were strongly encouraged to contribute a technical article to the CSRI Proceedings. For many students, these proceedings are the first opportunity to write a research article. These proceedings serve both as documentation of summer research and also as research training, providing students the first draft of an article that could be submitted to a peer-reviewed journal. Each of these articles has been reviewed by a Sandia staff member knowledgeable in the technical area, with feedback provided to the authors.

Contributions to the 2021 CSRI Proceedings have been organized into three categories: *Computational & Applied Mathematics, Software & High Performance Computing* and *Applications*.

All participants and their mentors who have contributed their technical accomplishments to the proceedings should be proud of their work and we congratulate and thank them for participating. Additionally, we would like to thank those who reviewed articles for the proceedings. Their feedback is an extremely important part of the research training process and has significantly improved the proceedings quality. Our many thanks are extended to these reviewers: Erin Acquesta ✧ Patrick Blonigan ✧ Andrew Bradley ✧ Tiernan Casey ✧ Frances Chance ✧ Kenny Chowdhary ✧ Michael Crockatt ✧ Matthew Curry ✧ Mary Alice Cusentino ✧ Eric Cyr ✧ Danny Dunlavy ✧ Chris Eldred ✧ Brian Franke ✧ Christian

Glusa ✧ Oksana Guba ✧ Mamikon Gulian ✧ John Jakeman ✧ Reese Jones ✧ Eric Keiter ✧ Brian Kelley ✧ Ron Kensek ✧ Patrick Knapp ✧ Hemanth Kolla ✧ Rich Lehoucq ✧ Drew Lewis ✧ Cody Melton ✧ David Montes de Oca Zapiain ✧ Srideep Musuvathy ✧ Stephen Olivier ✧ Kevin Pedretti ✧ Gabe Popoola ✧ Teresa Portone ✧ Jaideep Ray ✧ Denis Ridzal ✧ Fred Rothganger ✧ Ahmad Rushdi ✧ Antonio Russo ✧ Andy Salinger ✧ Tom Seidl ✧ John Shadid ✧ Andrea Staid ✧ J. Adam Stephens ✧ Laura Swiler ✧ Mark Taylor ✧ Keita Teranishi ✧ Irina Tezaur ✧ Aidan Thompson ✧ Ahn Tran ✧ Ray Tuminaro ✧ Craig Ulmer ✧ Rick Vinyard ✧ Tom Voth ✧ Felix Wang ✧ Kat Ward ✧ Bekah White ✧ Mitch Wood.

4. Virtual Poster Blitz. To prep for the proceedinngs, a *virtual poster blitz* was held on 7/22/2021 where all interns hired under the CSRI intern posting were invited to participate. These students submitted a summary slide of their current results or intended research for their summer project. Other interns outside of CSRI were also invited to participate with their mentor's approval. This event provided an opportunity to formalize work directions, socialize their ideas, as well as offering a chance to network and interact with other interns and staff across the labs. The slides from the virtual poster blitz event are published under SAND number SAND2021-8956 PE.

5. Kokkos Tutorial. Interns were invited to attend a tutorial from the Kokkos Team on Kokkos Parallel Programming. This offered an opportunity to learn how the Kokkos EcoSystem provides a performance portability programming model, enabling science and engineering codes to leverage all major HPC platforms without rewriting code each time. We deeply thank Christian Trott and Nathan Ellingwood of 1465 – Scalable Algorithms – for presenting the two-part tutorial.

These two half-days included lectures and hands-on exercises teaching the basics of shared memory parallel programming and how to use Kokkos to write performance portable codes, allowing interns and attendees to get started with profiling and performance optimization. In addition to basic parallelism and data management, these lectures addressed memory access patterns, hierarchical parallelism, profiling with Kokkos-Tools and an introduction to KokkosKernels.

Once more, the intern program was held virtually. The technical glitches of telework are challenging and it is no easy task to undertake a virtual internship. The program found success primarily due to the hard work of our students and the dedication of their staff mentors. We would like to thank all students and mentors for their extradorinary patience and dedication. We would also like to thank the program managers for the CSRI Summer Intern Program, Michael Wolf (1465) and Jerry McNeish (8754). Their support has been critical throughout the organization of the seminars and the editing of these proceedings. Furthermore, the CSRI Summer Intern Program would not be possible without the administrative support of Becky March, Hailey Poole, Sandra Portlock, and Cookie Santamaria.

J.D. Smith
E. Galvan

November 1, 2021

Table of Contents

Preface
J.D. Smith and E. Galvan . . . iii
Articles . . . 1
I. Computational & Applied Mathematics
J.D. Smith and E. Galvan . . . 1
A Parallel-in-Time Approach to Solving the 1D Advection Equation Using the Euler-Lagrange Equations
N.J. Christensen and E.J. Parish . . . 2
Partitioned solution of a coupled Reduced order model - Finite element model (ROM-FEM model) for a transmission problem
A. de Castro and P. Kuberry and P. Bochev . . . 10
A Survey of the Koopman operator and Moment Propagation
T.Z. Dean and E. Galvan . . . 24
Majorize-Minimize Algorithms for Streaming Generalized Canonical Polyadic Tensor Decompositions
K. Gilman and E. Phipps . . . 47
Study of the recovery discontinuous Galerkin method and it's application to EMPIRE
M.F. McCracken and S. Miller . . . 63
Data-driven Model Reduction for Physics-Constrained Optimization
S. A. McQuarrie, J. Hart, B. van Bloemen Waanders, and K. Willcox . . . 74
Comparison of Tempered and Truncated Fractional Models
H.A. Olson, M. D'Elia, M. Foss, M. Gulian, and P. Radu . . . 85
A reduced-space formulation for nonlinear programming with index-1 differential algebraic equation systems
R. Parker, B. Nicholson, J. Siirola, C. Laird, and L. Biegler . . . 93
Multifidelity data fusion in convolutional encoder/decoder assembly networks for computational fluid dynamics applications
L. Partin and G. Geraci and A.A. Rushdi and M.S. Eldred and D.E. Schiavazzi 102
Algebraic Multigrid based on Low-order Systems
A. Voronin and R. Tuminaro . . . 120
II. Software & High Performance Computing
J.D. Smith and E. Galvan . . . 132
Accelerating Electronic Structure Calculation with Atom-Decomposed Neural Modeling
J. Fox, N.A. Modine, and S. Rajamanickam . . . 133
Fine-Grained Parallel Graph Partition Refinement
M. S. Gilbert, K. Madduri, E. Boman, and S. Rajamanickam . . . 143
An Exploration of MiniMod Overhead and Granularity
D.A. Kruse, W.P. Marts, and M.G.F. Dosanjh . . . 158
Efficient Computation of Higher Order Joint Moment Tensor
Z. Li and H. Kolla . . . 166
pMEMCPY: an efficient I/O library for PMEM
L.M. Logan and G.F. Lofstead . . . 177
Geo-spatial Visualizations
M.R. Low and A.T. Wilson . . . 188
Periodic Loss Function for Grid Cell Encodings
S. Luca and F. Wang . . . 198

Integrating PGAS and MPI-Based Graph Analysis
T.M. McCrary, K.D. Devine, and A.J. Younge 208
Towards Viable Use of Machined Learned Models in Physical Simulation
C. Pereyra and M. A. Wood 223
Verification and Performance Testing of the Advanced Tri-lab Software Environment
C. Woods and M.L. Curry 235
III. Applications
J.D. Smith and E. Galvan 241
Expected Information Gain Estimates and Bayesian Optimal Experimental Design
T.A. Alsup and T.A. Catanach 242
Importance Sampling in Bayesian OED for Sensor Placement
J.P. Callahan and T.A. Catanach 256
A Variance deconvolution approach to sampling uncertainty quantification for Monte Carlo radiation transport solvers
K.B. Clements, G. Geraci, and A.J. Olson 266
Benchmarking neural network ansätze for quantum chemistry applications
C. Frink, Q. Campbell, C. Smith, A.D. Baczewski, and T. Albash 281
Air-Sea Light
M.J. Gaiewski, K.C. Sockwell, J.C Connors, and P.B. Bochev 294
Precipitation Model Aggregation using Optimal Transport
K.E. Gerot and K.L. DiPietro 308
The Schwarz alternating method for multiscale contact mechanics
J. Hoy, I. Tezaur, and A. Mota 320
Learning Transferable Neural Network Surrogates for Kohn-Sham Density Functional Theory
K.R. Lennon and S. Rajamanickam 339
Improved Vertical Remapping Accuracy in the NH-HOMME Atmosphere Dynamical Core
J.L. Torchinsky and M.A. Taylor 352
Machine-Learning for Single Particle Motion in Plasmas
S.M. Valaitis and K.A. Maupin 365

Articles

I. Computational & Applied Mathematics

Computational & Applied Mathematics are concerned with the design, analysis, and implementation of algorithms to solve mathematical, scientific, or engineering problems. Articles in this section describe methods to design new neural network architectures, discretize and solve partial differential equations, couple multiphysics systems of equations, and analyze sensitivity & quantify uncertainty in complex systems.

1. *Christensen* and *Parish* leverage the least-squares principles and the Euler-Lagrange equations to develop a parallel-in-time approach to solving the **1D Advection Equation**.
2. *de Castro, Kuberry,* and *Bochev* develop a partition solution method for a coupled **Reduced Order Model-Finite Element Model** using a proper orthogonal decomposition and a Lagrange multiplier. This approach is tested on a transmission problem.
3. *Dean* and *Galvan* survey techniques and results employing the **Koopman Operator** to dynamical systems, providing applications to uncertainty quantification.
4. *Gilman* and *Phipps* utilize majorize-minimize algorithms to compute the **Generalized Canonical Polyadic Tensor Decomposition** from streaming data.
5. *McCracken* and *Miller* implement a **Recovery Discontinuous Galerkin Method** for diffusive fluxes in the Navier-Stokes equation set.
6. *Mcquarrie, Hart, van Bloemen Waanders,* and *Willcox* utilize operator inference as dynamical constraints for **Data-Driven Model Reduction** applied to two control problems.
7. *Olson, D'Elia, Foss, Gulian,* and *Radu* analyze a truncated variation of **Tempered Fractional Operators**, comparing results to the non-truncated more computationally intensive operator.
8. *Parker, Nicholson, Siirola, Laird,* and *Biegler* describe a reduced-space formulation for optimization of index-1 **Differential Algebraic Equation Systems** and implement the formulation in Pyomo. This formulation is then used to solve dynamic optimization problems.
9. *Partin, Geraci, Rushdi, Eldred,* and *Schiavazzi* apply convolutional neural networks to **Fluid Dynamics** problems, providing predictive uncertainty to a partial differential equation model trained on both high- and low-fidelity data.
10. *Voronin, Tuminaro, Olson,* and *MacLachlan* develop **Monlithic Algebraic Multigrid Preconditioners** for mixed finite-element discretizations of coupled partial differential equations. This method is applied to investigate higher-order discretizations of the Stokes equations.

J.D. Smith
E. Galvan

November 1, 2021

A Parallel-in-Time Approach to Solving the 1D Advection Equation Using the Euler-Lagrange Equations

NICHOLAS J. CHRISTENSEN* AND ERIC J. PARISH†

Abstract. Parallel-in-time solvers for partial differential equations allow for increased parallelism at the cost of extra work. However, parallel-in-time solvers have been difficult to apply to hyperbolic systems. We propose to use the least-squares principles and the Euler-Lagrange equations to convert a semi-discrete system of ODEs into a large two-point space–time boundary value problem that may be more amenable to parallel-in-time solution algorithms. We test this approach using finite differences on the 1D advection problem.

1. Introduction. PDE solvers often parallelize in the spatial dimensions to solve problems faster. However, the high number of processors on modern machines often means spatial parallelism becomes saturated on a fraction of the machine leaving compute resources underused. Parallel-in-time (PinT) solvers attempt to address this by extending parallel computing to the time dimension. This increases the number of floating point operations and memory accesses, but allows for the use of more processors which may compensate for these costs. Numerous PinT solvers have been developed throughout the years; popular examples include parareal [3] and multigrid reduction in time (MGRIT) [2]. While these and other PinT solvers have seen success in accelerating the time-to-solution for parabolic systems, extensions to hyperbolic systems have suffered stability and convergence problems [1, 5].

We propose a parallel-in-time approach based on a least-squares formulation of a semi-discrete dynamical system. In this approach, we convert the initial condition problem corresponding to the dynamical system of interest into a two-point boundary value problem via the Euler–Lagrange equations associated with the least-squares principle. We believe that the system emerging from this process is more amenable to PinT solution strategies — particularly for dynamical systems emerging from the spatial discretization of hyperbolic PDEs — due to its parabolic-like structure. This work comprises an initial investigation into the proposed approach. In the following, we derive the approach for linear problems and show results for preliminary numerical experiments with the 1D advection equation using finite differences.

2. Derivation from the Euler-Lagrange Equations. Let the vector function $\mathbf{u}(t) \in \mathbb{R}^{N_x}$ denote values of the scalar function $u(x,t)$ for N_x discrete spatial points $x \in [a,b]$ and

$$\dot{\mathbf{u}}(t) - \mathbf{f}(t, \mathbf{u}(t)) = \mathbf{0} \tag{2.1}$$

with $t \in [0,T]$ and $\mathbf{u}(0) = \mathbf{u}_0$. A typical finite difference code discretizes the system in space and time and solves for the value of each time step sequentially starting from the initial condition. We instead seek a function $\mathbf{u}(t)$ to minimize the objective functional

$$J[\mathbf{u}(t)] = \int_0^T \|\dot{\mathbf{u}}(t) - \mathbf{f}(t, \mathbf{u}(t))\|_2^2 \, dt \tag{2.2}$$

subject to boundary conditions with Lagrangian

*University of Illinois at Urbana-Champaign, njchris2@illinois.edu
†Sandia National Laboratories, ejparis@sandia.gov

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number *DE-NA0003963*.

$$L(t, \mathbf{u}(t), \dot{\mathbf{u}}(t)) = \|\dot{\mathbf{u}}(t) - \mathbf{f}(t, \mathbf{u}(t))\|_2^2 = \dot{\mathbf{u}}(t)^T \dot{\mathbf{u}}(t) + \mathbf{f}^T \mathbf{f} - 2\mathbf{f}^T \dot{\mathbf{u}}(t). \tag{2.3}$$

The functional J is minimized for critical functions $\mathbf{u}(t)$ that satisfy[1]

$$\nabla J[\mathbf{u}] = \frac{\partial L}{\partial \mathbf{u}}(t, \mathbf{u}(t), \dot{\mathbf{u}}(t)) - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{u}}(t)}(t, \mathbf{u}(t), \dot{\mathbf{u}}(t))\right) = \mathbf{0}. \tag{2.4}$$

Substituting our expression for $L(t, \mathbf{u}(t), \dot{\mathbf{u}}(t))$ into (2.4) we obtain

$$\left[\frac{\partial \mathbf{f}^T}{\partial \mathbf{u}}\right](\dot{\mathbf{u}} - \mathbf{f}) + \ddot{\mathbf{u}} - \frac{d\mathbf{f}}{dt} = \mathbf{0}. \tag{2.5}$$

If $\mathbf{f}$ is a linear function $\mathbf{f} : \mathbf{u} \mapsto \mathcal{A}\mathbf{u}$, then we have

$$\mathcal{A}^T(\dot{\mathbf{u}} - \mathcal{A}\mathbf{u}) + \ddot{\mathbf{u}} - \mathcal{A}\dot{\mathbf{u}} = \mathbf{0}. \tag{2.6}$$

To solve (2.6) using finite differences we replace $\mathbf{u}$ with $\hat{\mathbf{u}} = [\hat{u}(t_0, x_0), \dots \hat{u}(t_0, x_{N_x-1}), \hat{u}(t_1, x_0), \dots \hat{u}(t_{N_t-1}, x_{N_x-1})]^T \in \mathbb{R}^{N_t \times N_x}$ representing the approximate solution at $N_t \times N_x$ discrete space-time points. We next replace the derivative operators with first and second order finite difference operators in time, which we denote with B_t and C_t respectively. Finally, for the 1D advection problem we replace $\mathcal{A}$ with A_s, a first order finite difference operator in space.

This leaves us with a linear system to solve.

$$(A_s^T(B_t - A_s) + C_t - A_s B_t))\hat{\mathbf{u}}_{interior} = \mathbf{0}. \tag{2.7}$$

2.1. Boundary conditions. Equation (2.7) only holds for the interior time steps. Time steps on and near the boundary obey different equations. Initial conditions in the sequential formulation become Dirichlet boundary conditions for $\hat{\mathbf{u}}$ at and near $t = 0$ in this parallel-in-time formulation. The boundary at $t = T$ is governed by a natural boundary condition,

$$\dot{\mathbf{u}}(T) - \mathbf{f}(T) = \mathbf{0} \tag{2.8}$$

with a corresponding finite dimensional approximation

$$(B_t - A_s)\hat{\mathbf{u}}_{N_t-1} = \mathbf{0}. \tag{2.9}$$

where $\hat{\mathbf{u}}_{N_t-1} = [\hat{u}(t_{N_t-1}, x_0), \dots, \hat{u}(t_{N_t-1}, x_{N_x-1})]^T$.

2.2. Amenability to PinT approaches. We believe that the system 2.5 is more amenable for PinT solvers than the standard formulation (2.1) — particularly when the system (2.1) emerges from the spatial discretization of a hyperbolic set of equations — as it results in a two-point boundary value problem that is parabolic-like in structure; we refer the interested reader to Appendix 1 for more details on this concept within the context of the 1D advection equation. As highly effective parallel iterative methods (such as multigrid) exist to solve these types of systems, we believe that the system (2.5) is a promising starting point for PinT solvers.

[1] For more details on the calculus of variations see Peter Olver's excellent introduction to the topic.[4]

3. Implementation. While the above derivation is general for linear functions $\mathbf{f}(t, \mathbf{u}(t))$, we focus the implementation and numerical experiments on the 1D advection problem whereby $\mathbf{f}(t, \mathbf{u}(t)) = \frac{\partial \mathbf{u}(t)}{\partial x}$.

3.1. Global and local system. The global system for the advection equation consists of $(A_s^T(B_t - A)s) + C_t - A_s B_t))$ expanded to incorporate boundary conditions. Let us denote the global system matrix with M. In general, M is block banded and with boundary conditions we solve a system of the form

$$M\hat{\mathbf{u}} = \begin{bmatrix} & & & & & E_{boundary} & & & & \\ E_0 & \dots & E_{k-1} & D & F_0 & \dots & F_{j-1} & 0 & \dots & 0 \\ 0 & E_0 & \dots & E_{k-1} & D & F_0 & \dots & F_{j-1} & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & E_0 & \dots & E_{k-1} & D & F_0 & \dots & F_{j-1} \\ & & & & & F_{boundary} & & & & \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}_0 \\ \hat{\mathbf{u}}_1 \\ \vdots \\ \hat{\mathbf{u}}_{l-k} \\ \vdots \\ \hat{\mathbf{u}}_{l-1} \\ \hat{\mathbf{u}}_l \\ \hat{\mathbf{u}}_{l+1} \\ \vdots \\ \hat{\mathbf{u}}_{l+j} \\ \vdots \\ \hat{\mathbf{u}}_{N_t-1} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{b}}_0 \\ \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{l-k} \\ \vdots \\ \hat{\mathbf{b}}_{l-1} \\ \hat{\mathbf{b}}_l \\ \hat{\mathbf{b}}_{l+1} \\ \vdots \\ \hat{\mathbf{b}}_{l+j} \\ \vdots \\ \hat{\mathbf{b}}_{N_t-1} \end{bmatrix} \tag{3.1}$$

where the subscripts on $\hat{\mathbf{u}}$ and $\hat{\mathbf{b}}$ index the N_t discrete times. The specific forms of D, E, and F depend on the temporal and spatial stencils as these submatrices arise from sums of Kronecker products of temporal and spatial finite difference matrices. Here, D is a square submatrix of size $N_x \times N_x$ where N_x is the spatial grid size, submatrix $E = [E_0 \dots E_{k-1}]$ is of size $N_x \times kN_x$ with k determined by the choice of temporal stencil, and $F = [F_0 \dots F_{j-1}]$ is an $N_x \times jN_x$ submatrix with j also determined by the choice of temporal stencil. The $kN_x \times N_t N_x$ submatrix $E_{boundary}$ enforces the initial condition. For the advection problem it is simply the identity matrix. Submatrix $F_{boundary}$ of size $jN_x \times N_t N_x$ accounts for the possible use of different temporal stencils near time T and enforces the Neumann boundary condition at time T.

3.2. Block-Jacobi solver. We used a relatively simple block Jacobi solver for our initial investigation of this parallel-in-time formulation. Block Jacobi comprises an iterative method which leverages the matrix splitting

$$M = L_G + D_G + U_G$$

where D_G comprises the block-diagonal of M and L_G and U_G are strictly lower-block and upper-block matrices, respectively. The block Jacobi iteration then takes the form

$$D_G \hat{\mathbf{u}}^p = \hat{\mathbf{b}} - (L_G + U_G)\hat{\mathbf{u}}^{p-1}.$$

In the present context, the block Jacobi iteration reduces to performing the following iteration in p at each time step l

$$D\hat{\mathbf{u}}_l^p = \hat{\mathbf{b}}_l - E \begin{bmatrix} \hat{\mathbf{u}}_{l-k} \\ \vdots \\ \hat{\mathbf{u}}_{l-1} \end{bmatrix}^{p-1} - F \begin{bmatrix} \hat{\mathbf{u}}_{l+1} \\ \vdots \\ \hat{\mathbf{u}}_{l+j} \end{bmatrix}^{p-1}$$

with $E = [E_0 \dots E_{k-1}]$ and $F = [F_0 \dots F_{j-1}]$. This system can be solved using SciPy's *splu* function among other methods.

This decomposition allows the solver to be easily parallized. We assign each processor to solve a contiguous subset of time steps of size ceil(N_t/n_{ranks}) with the final process given a smaller block if N_t is not perfectly divisible by n_{ranks}. With time steps divided in this way, the local solution for most time steps can be solved using only local data. Only at the boundaries of a rank's local set of timesteps does information need to be exchanged. A loop over all n_t local time steps generates a local solution $\hat{\mathbf{u}}^p_{local} = [\hat{\mathbf{u}}^p_l, \dots, \hat{\mathbf{u}}^p_{l+n_t-1}]^T$.

To solve the global system, we incorporate the local sparse solve inside a Jacobi loop that also performs a halo exchange and checks the solution for convergence. While the inner for-loop in Algorithm 1 is presented with matrix-vector (BLAS2) operations for clarity, better performance is achieved by implementing the equivalent operation using matrix-matrix (BLAS3) operations. Since the D matrix is invariant, it can be prefactored outside of the loop for an additional performance improvement. Our implementation uses both of these optimizations.

Algorithm 1 Parallel global solve

procedure GLOBAL JACOBI ITERATION
  initialize $\hat{\mathbf{u}}^0$
  initialize *max_diff* to large value
  $p \leftarrow 1$
  while *max_diff* $>$ *tol* **do**
    for local timesteps l **do**
      $\mathbf{r}^p_l \leftarrow \hat{\mathbf{b}}_l - E\hat{\mathbf{u}}^{p-1}_{l-k:l-1} - F\hat{\mathbf{u}}^{p-1}_{l+1:l+j}$
      $\hat{\mathbf{u}}^p_l \leftarrow solve(D, \mathbf{r}^p_l)$
    diff $\leftarrow \|\hat{\mathbf{u}}^p_{local} - \hat{\mathbf{u}}^{p-1}_{local}\| / \|\hat{\mathbf{u}}^p_{local}\|$
    max_diff $\leftarrow$ *global_max*(*diff*)
    $\hat{\mathbf{u}}^p_{local} \leftarrow exchange\_boundary\_data(\hat{\mathbf{u}}^p_{local})$
    $p \leftarrow p + 1$
  return $\hat{\mathbf{u}}^p_{local}$

3.3. Finite difference stencils. The above method is detached from any particular finite difference stencils. For the purposes of this paper, we construct A_s with a second order central difference stencil and B_t with a second order Crank-Nicolson upwind stencil. We use a modified second order upwind stencil for the second temporal derivative. The derivation for this stencil follows.

Let $u_t = \frac{du}{dt}$, $u_{tt} = \frac{d^2u}{dt^2}$, and so forth for higher derivatives. We begin with Taylor expansions around t.

$$u(t-h,x) = u(t,x) - u_t(t,x)h + \frac{u_{tt}(t,x)h^2}{2} - \frac{u_{ttt}(t,x)h^3}{6} + \frac{u_{tttt}(t,x)h^4}{24} + ...$$

$$u(t-2h,x) = u(t,x) - u_t(t,x)2h + \frac{u_{tt}(t,x)4h^2}{2} - \frac{u_{ttt}(t,x)8h^3}{6} + \frac{u_{tttt}(t,x)16h^4}{24} + ...$$

We multiply the first equation by 8, subtract the second equation, and solve for $u_{tt}(t,x)$ to obtain

$$u_{tt}(t,x) = \frac{8u(t-h,x) - u(t-2h,x) - 7u(t,x) + 6hu_t(t,x)}{2h^2} + \frac{u_{tttt}(t,x)h^2}{6} + ...$$

For the advection problem, $u_t(t,x) = -\frac{\partial u(t,x)}{\partial x}$ so we can substitute the spatial derivative for the temporal derivative. After simplification this gives

$$u_{tt}(t,x) = \underbrace{\frac{8u(t-h,x) - u(t-2h,x) - 7u(t,x)}{2h^2} - \frac{3}{h}\frac{\partial u(t,x)}{\partial x}}_{\text{retained terms}} + \frac{u_{tttt}(t,x)h^2}{6} + ...$$

If we approximate $\frac{\partial u(t,x)}{\partial x}$ with second order finite differences in space (central differences for instance), then the second derivative approximation is $O(\Delta x^2) + O(h^2)$. Since the first derivative stencils are second order we expect the entire solver to be second order convergent in space and time. This formulation reduces the required number of known initial time steps from three in the typical second order upwinding formulation to two in this modified formulation.

3.4. Forming the local system. The local system may be formed straightforwardly from basic (i.e. 1D) finite difference matrices using Kronecker products.

$$A_s = I_t \otimes \hat{A}_s$$

$$B_t = \hat{B}_t \otimes I_s$$

$$C_t = \hat{C}_t \otimes I_s + \frac{3}{dt} A_s$$

where $\hat{A}_s$ is the finite difference matrix for the first derivative in space, $\hat{B}_t$ is the finite difference matrix for the first derivative in time, and $\hat{C}_t$ is the finite difference matrix for the second derivative in time corresponding to the stencil $\frac{8u(t-h,x)-u(t-2h,x)-7u(t,x)}{2h^2}$. Local temporal boundary conditions may be enforced by masking out matrix entries on the boundary and adding a matrix $G = \hat{G} \otimes I_s$ that enforces the boundary conditions such that

$$M_{local} = \text{mask}(A_s^T(B_t - A_s) + C_t - A_s B_t)) + G$$

4. Numerical experiments. We implemented the parallel-in-time solver in Python with use of SciPy's sparse matrix library. Parallelism and data exchange was implemented with MPI for Python.

We tested the parallel-in-time solver on a 1D advection problem with an initial condition (i.e. a temporal Dirichlet boundary) of $\hat{u}(x,0) = sin(\pi x)^{10}$ with $x \in [0,1]$ and a periodic boundary condition in space. For the temporal domain we used $t \in [0,T]$ with $T = 1$ (i.e. one period). We used a central difference stencil for the first derivative in space, a second order upwind stencil for the first derivative in time and the modified second order upwind stencil for the second derivative in time. The solver tolerance for these tests was 10^{-6}. This scheme is theoretically second order in space and time and the results shown in 4.1 verify this is the case.

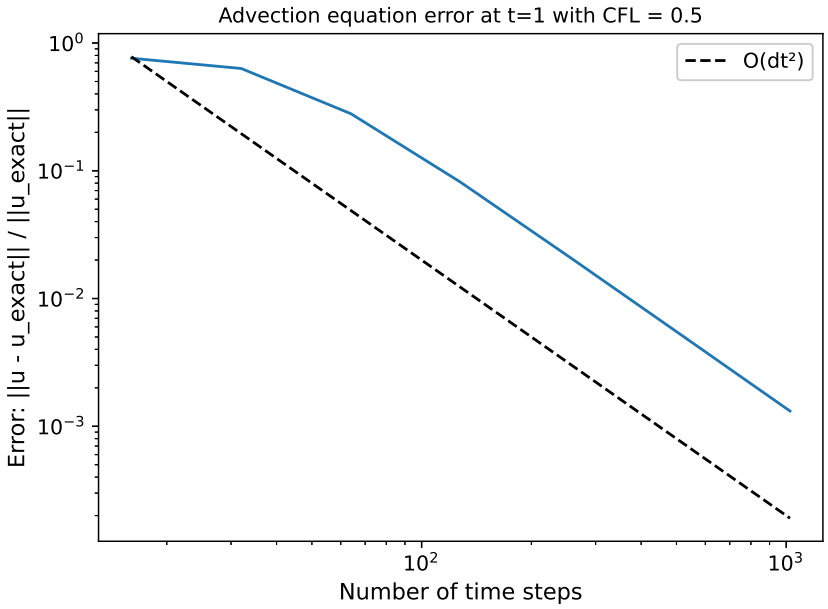


Fig. 4.1: Parallel-in-time solver convergence experiment using the block Jacobi solver. Error here is measured in the two-norm. The black dotted line shows a reference slope for $O(dt^2)$ error scaling.

Figure 4.2 shows the results of a block-Jacobi strong-scaling experiment with a varying number of MPI ranks on one Lassen node. A Lassen node comprises two sockets, each with twenty-two 3.8 GHz IBM Power9 cores, and 256 GB of CPU memory with a peak bandwidth of 170 GB/s. Times in this chart are the average over all MPI ranks. The number of of global solve iterations was invariant of the number of processors, requiring 1024 iterations to reach the specified tolerance. Excluding the setup time, most of the time in the global solve is spent within the local sparse solve (SciPy's *splu* which uses the SuperLU library) with an increasing fraction of time spent in inter-rank communication as the decreasing amount of work per rank allows the latency cost to become relatively large. Figure 4.3 shows very good strong scaling is retained up to eight MPI ranks with decreasing efficiency thereafter. While communication accounts for some of this loss of efficiency, the amount of time spent in the sparse solve also ceases to halve as the number of MPI ranks doubles above eight ranks and the amount of work per rank becomes small. The cause of the relatively high communication cost for two ranks and four ranks is unclear.

The above result suggests this parallel-in-time formulation has potential for strong scale. However, with a Block-Jacobi solver the parallel-in-time strategy is not competive with time sequential solvers. A serial Crank-Nicolson solver computing the solution for 512 spacial grid points and 1024 time steps on a single Lassen core completes in about 0.06 s, about two orders of magnitude smaller than the fastest time measured in the parallel-in-time experiments above. This is expected theoretically. If we denote the sparse solve time for a

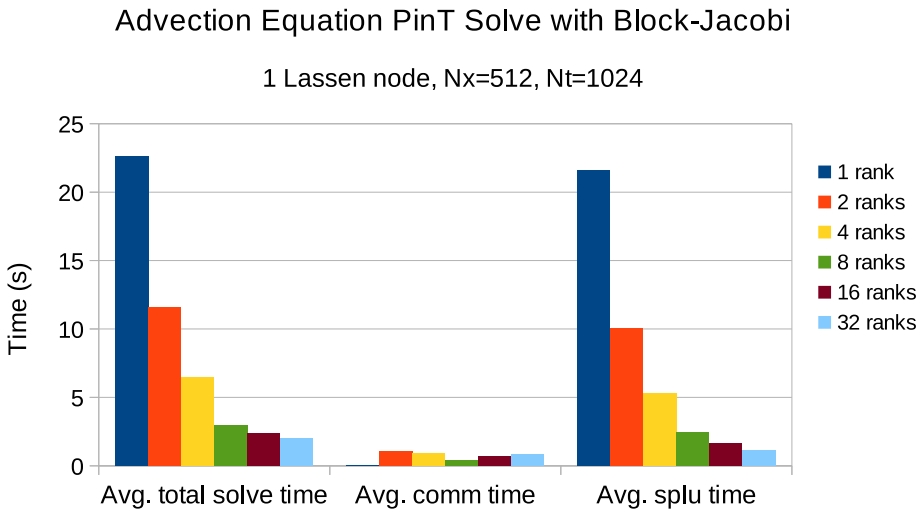


Fig. 4.2: A strong scaling experiment using the block Jacobi solver on one Lassen node.

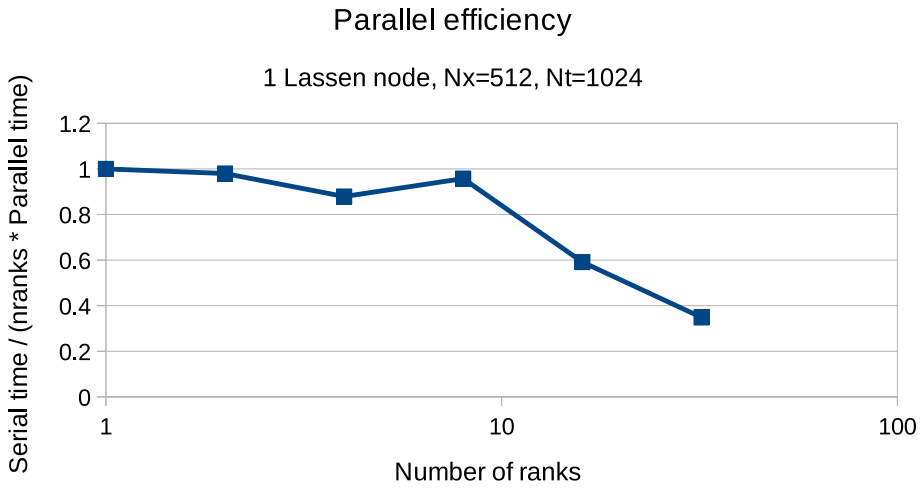


Fig. 4.3: Parallel efficiency plot for the same strong scaling experiment.

single time step as $t_{solve}(N_x)$ then the expected speedup (ignoring communication costs) is

$$\text{Speedup} = \frac{T_{serial}}{T_{parallel}} = \Theta\left(\frac{N_t t_{solve}(N_x)}{N_{iter} t_{solve}(N_x)\frac{N_t}{p}}\right) = \Theta\left(\frac{p}{N_{iter}}\right).$$

This tells us speedup greater than one is generally only possible when the number of global solver iterations is less than the number of processors. Block-Jacobi, requiring one iteration

per time step, is not ideal for this problem. A multigrid-based solver may reduce the number of iterations and make this approach competitive against a serial code.

5. Conclusions. We have shown the feasibility of solving the advection equation using a parallel-in-time solver derived from the Euler-Lagrange equations. The approach may be a viable method of strong scaling for higher dimensional problems. In future work we plan to explore the performance of the solver in higher space dimensions, on problems other than the advection equation, coupled with other methods like multigrid, and on GPUs. We also plan to explore this approach within a Galerkin context rather than finite differences and to investigate the effects of this formulation on the conditioning of the problem.

Appendix A. Parabolic form of the normal equations for the 1D advection equation. In this section we demonstrate that the normal equations associated with the 1D advection equation comprise a parabolic system. For the 1D advection equation, the system differential operator is given by

$$\mathcal{L} = \frac{\partial}{\partial t} + c\frac{\partial}{\partial x}.$$

The standard adjoint of the differential operator is given as

$$\mathcal{L}^* = -\frac{\partial}{\partial t} - c\frac{\partial}{\partial x}.$$

The differential operator associated with the normal equations is given by

$$\mathcal{L}^*\mathcal{L} = -\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right)\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right).$$

The above simplifies to

$$\mathcal{L}^*\mathcal{L} = -\frac{\partial^2}{\partial t^2} - 2c\frac{\partial^2}{\partial x \partial t} - c^2\frac{\partial^2}{\partial x^2}.$$

Following standard terminology we write the above as

$$\mathcal{L}^*\mathcal{L} = A\frac{\partial^2}{\partial t^2} - B\frac{\partial^2}{\partial x \partial t} - C\frac{\partial^2}{\partial x^2}$$

with $A = -1$, $B = -2c$, and $C = -c^2$. The above describes an elliptic operator if $B^2 - 4AC < 0$, a hyperbolic operator if $B^2 - 4AC > 0$, and a parabolic operator if $B^2 - 4AC = 0$. In the present context we get

$$B^2 - 4AC = 4c^2 - 4(-1)(-c^2) = 0$$

and thus $\mathcal{L}^*\mathcal{L}$ is parabolic.

References.

[1] H. De Sterck, S. Friedhoff, A. J. M. Howse, and S. P. MacLachlan, *Convergence analysis for parallel-in-time solution of hyperbolic systems*, Numerical Linear Algebra with Applications, 27 (2020), p. e2271. e2271 nla.2271.

[2] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C635–C661.

[3] J.-L. Lions, Y. Maday, and G. Turinici, *A "parareal" in time discretization of PDE's*, Comptes Rendus de l'Académie des Sciences - Series I - Mathematics, 332 (2001), pp. 661–668.

[4] P. J. Olver, *The calculus of variations*, 2021.

[5] A. Schmitt, *Numerical Investigation of Parallel-in-Time Methods for Dominantly Hyperbolic Equations*, PhD thesis, Technische Universität, Darmstadt, July 2018.

PARTITIONED SOLUTION OF A COUPLED REDUCED ORDER MODEL - FINITE ELEMENT MODEL (ROM-FEM MODEL) FOR A TRANSMISSION PROBLEM

AMY DE CASTRO[†], PAUL KUBERRY[‡], AND PAVEL BOCHEV[§]

Abstract. Application of reduced order modeling (ROM) on select subdomains can help to increase the computational efficiency of multiphysics simulations. We develop a partitioned scheme for a model interface problem which couples a ROM with a conventional finite element method. The proper orthogonal decomposition (POD) approach is implemented to construct a low-dimensional reduced basis on half the domain and solve the subdomain problem in terms of this basis using POD/Galerkin projection. The ROM solution is then coupled to the FEM solution using a Lagrange multiplier representing the interface flux. The multiplier at the current time step can be expressed as an implicit function of the state solutions through a Schur complement. As a result, application of an explicit time integration scheme decouples the subdomain problems, allowing their independent solution for the next time step.

1. Introduction. In this report, we formulate a new explicit partitioned scheme for a transmission problem that combines a Reduced Order Model (ROM) with the conventional Finite Element Model. The scheme extends the approaches in [2] and [3] which start from a monolithic formulation of the transmission problem and then use a Schur complement to obtain an approximation of the interface flux that serves as a Neumann boundary condition for each subdomain problem. In other approaches for ROM couplings, such as [1], domain decomposition is used as a tool to accelerate or improve the generation of ROMs. The method presented in this paper is a coupling method for multiphysics problems in which different parts of the domain are discretized by different schemes, such as a ROM.

2. Interface Problem. We consider a bounded region $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ with a Lipschitz-continuous boundary Γ. We assume that Ω is divided into two non-overlapping subdomains Ω_1, Ω_2. Let γ denote the interface shared between the two subdomains, and let $\Gamma_i = \partial\Omega_i \backslash \gamma$ for $i = 1, 2$, as illustrated in Figure 2.1.

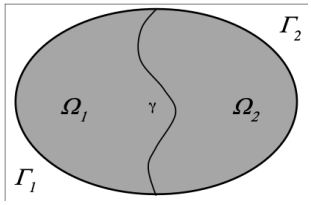


Fig. 2.1: Non-overlapping subdomains

We take $\mathbf{n}_\gamma$ to be the unit normal on the interface pointing toward Ω_2.
We consider a model transmission problem given by the advection-diffusion equation:

$$\begin{aligned} \dot{\varphi}_i - \nabla \cdot F_i(\varphi_i) &= f_i \qquad \text{on } \Omega_i \times [0, T] \\ \varphi_i &= g_i \qquad \text{on } \Gamma_i \times [0, T], \quad i = 1, 2 \end{aligned} \tag{2.1}$$

[†]Clemson University, agmurda@clemson.edu
[‡]Sandia National Laboratories, pakuber@sandia.gov
[§]Sandia National Laboratories, pbboche@sandia.gov

where the unknown φ_i is a scalar field, $F_i(\varphi_i) = \epsilon_i \nabla \varphi_i - \mathbf{u}\varphi_i$ is the total flux function, $\epsilon_i > 0$ is the diffusion coefficient in Ω_i, and $\mathbf{u}$ the velocity field. We augment (2.1) with initial conditions:

$$\varphi_i(\mathbf{x}, \mathbf{0}) = \varphi_{i,0}(\mathbf{x}) \qquad \text{in } \Omega_i, \quad i = 1, 2\,. \tag{2.2}$$

Along the interface, we enforce continuity of the states and continuity of the total flux, i.e., our interface conditions are:

$$\varphi_1(\mathbf{x}, t) - \varphi_2(\mathbf{x}, t) = 0 \quad \text{and } F_1(\mathbf{x}, t) \cdot \mathbf{n}_\gamma = F_2(\mathbf{x}, t) \cdot \mathbf{n}_\gamma \quad \text{on } \gamma \times [0, T] \tag{2.3}$$

We note that one also has the option to enforce only equilibrium of the diffusive flux exchanged between the two subdomains. We do not consider this option here as the resulting partitioned scheme will be similar to that obtained by enforcing continuity of the total flux.

In contrast to conventional partitioned schemes our approach starts from a well-posed monolithic formulation. To obtain this formulation we use a Lagrange multiplier to enforce continuity of states, i.e., the first condition in (2.3). Specifically, the monolithic problem is given by the following weak formulation of the transmission problem: *seek* $\{\varphi_1, \varphi_2, \lambda\} \in V := H^1_\Gamma(\Omega_1) \times H^1_\Gamma(\Omega_2) \times H^{-1/2}_\Gamma(\gamma)$, *such that*

$$\begin{aligned}
(\dot{\varphi_1}, \nu)_{\Omega_1} + (\epsilon_1 \nabla \varphi_1, \nabla \nu)_{\Omega_1} - (\mathbf{u}\varphi_1, \nabla \nu)_{\Omega_1} + (\lambda, \nu)_\gamma &= (f_1, \nu)_{\Omega_1} \qquad \forall \nu \in H^1_\Gamma(\Omega_1) \\
(\dot{\varphi_2}, \eta)_{\Omega_2} + (\epsilon_2 \nabla \varphi_2, \nabla \eta)_{\Omega_2} - (\mathbf{u}\varphi_2, \nabla \eta)_{\Omega_2} - (\lambda, \eta)_\gamma &= (f_2, \eta)_{\Omega_2} \qquad \forall \eta \in H^1_\Gamma(\Omega_2) \\
(\varphi_1, \mu)_\gamma - (\varphi_2, \mu)_\gamma &= 0 \qquad \forall \mu \in H^{-1/2}(\gamma)
\end{aligned} \tag{2.4}$$

It is easy to see that the Lagrange multiplier λ is the flux exchanged through the interface, i.e., $\lambda = F_1 \cdot \mathbf{n}_\gamma = F_2 \cdot \mathbf{n}_\gamma$. This observation is at the core of our partitioned method formulation. Indeed, if we could somehow determine λ, then each subdomain problem becomes a well-posed mixed boundary value problem with a Neumann condition on γ provided by λ:

$$\begin{aligned}
\dot{\varphi_i} - \nabla \cdot F_i(\varphi_i) &= f_i \qquad \text{on } \Omega_i \times [0, T] \\
\varphi_i &= g_i \qquad \text{on } \Gamma_i \times [0, T] \qquad , \quad i = 1, 2\,. \\
F_i(\varphi_i) \cdot \mathbf{n}_i &= (-1)^i \lambda \qquad \text{on } \gamma \times [0, T]
\end{aligned} \tag{2.5}$$

In other words, knowing λ allows us to decouple the subdomain equations and solve them independently. Of course, this cannot be done within the framework of (2.4), which is a fully coupled problem in terms of the states ϕ_i *and* the Lagrange multiplier λ. However, an independent estimation of λ may be possible in the context of a discretized version of this coupled problem.

2.1. A semi-discrete monolithic formulation. Let $V^h \subset V$ be a conforming finite element space spanned by a basis $\{\nu_i, \eta_j, \mu_k\}$; $i = 1, \ldots, N_1$; $j = 1, \ldots, N_2$; $k = 1, \ldots, N_\gamma$. A finite element discretization of (2.4) yields the following Differential Algebraic Equation (DAE) system:

$$\begin{aligned}
M_1 \dot{\mathbf{\Phi}}_1 + G_1^T \boldsymbol{\lambda} &= \bar{\mathbf{f}}_1(\mathbf{\Phi}_1) \\
M_2 \dot{\mathbf{\Phi}}_2 - G_2^T \boldsymbol{\lambda} &= \bar{\mathbf{f}}_2(\mathbf{\Phi}_2) \\
G_1 \mathbf{\Phi}_1 - G_2 \mathbf{\Phi}_2 &= 0
\end{aligned} \tag{2.6}$$

where for $r = 1, 2$, $\mathbf{\Phi}_r$ are the coefficient vectors corresponding to φ_r, M_r are the mass matrices, the right hand side vector $\bar{\mathbf{f}}_r(\mathbf{\Phi}_r) = \mathbf{f}_r - (D_r + A_r)\mathbf{\Phi}_r$ with D_r, A_r corresponding

to the diffusive and advective flux terms, respectively, and G_r are the matrices enforcing the (weak) continuity of the states. Assembly of these matrices is standard, for example, $(M_1)_{ij} = (\nu_j, \nu_i)_{\Omega_1}$, $(D_2)_{ij} = \epsilon_2(\nabla\eta_j, \nabla\eta_i)_{\Omega_2}$; $(G_1)_{i,j} = (\nu_j, \mu_i)_\gamma$; $(G_2)_{i,j} = (\eta_j, \mu_i)_\gamma$, and so on. We note here that the space for the Lagrange multiplier λ can be taken to be the trace of the finite element space on either of Ω_1 or Ω_2; either choice will be stable. In practice using the coarser of the two interface spaces for the Lagrange multiplier space improves accuracy; see [2] and [3] for details and discussion.

Our approach is predicated on the ability to express $\boldsymbol{\lambda}$ as an implicit function of the subdomain states. This, however is not possible for (2.6) because it is an Index-2 Hessenberg DAE. Following [2] we reduce the index of (2.6) by differentiating the constraint equation in time to obtain the following Index-1 Hessenberg DAE:

$$\begin{aligned} M_1\dot{\boldsymbol{\Phi}}_1 + G_1^T\boldsymbol{\lambda} &= \overline{\mathbf{f}}_1(\boldsymbol{\Phi}_1) \\ M_2\dot{\boldsymbol{\Phi}}_2 - G_2^T\boldsymbol{\lambda} &= \overline{\mathbf{f}}_2(\boldsymbol{\Phi}_2) \\ G_1\dot{\boldsymbol{\Phi}}_1 - G_2\dot{\boldsymbol{\Phi}}_2 &= 0 \end{aligned} \tag{2.7}$$

Assuming the initial data are continuous across γ, the new constraint $(\dot{\varphi}_1, \mu)_\gamma - (\dot{\varphi}_2, \mu)_\gamma = 0$ is equivalent to the original one, i.e., (2.7) is equivalent to the original monolithic problem (2.6). In what follows we refer to (2.7) as the FEM-FEM model.

3. Explicit partitioned scheme for FEM. The coupled system (2.7) defining the FEM-FEM model can be written in matrix form as:

$$\begin{bmatrix} M_1 & 0 & G_1^T \\ 0 & M_2 & -G_2^T \\ G_1 & -G_2 & 0 \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\Phi}}_1 \\ \dot{\boldsymbol{\Phi}}_2 \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{f}}_1(\boldsymbol{\Phi}_1) \\ \overline{\mathbf{f}}_2(\boldsymbol{\Phi}_2) \\ 0 \end{bmatrix} \tag{3.1}$$

A partitioned scheme for this problem can be realized by solving a Schur complement system for the Lagrange multiplier and then applying an explicit time integration scheme [2], or alternatively by applying a time integration scheme before solving the Schur complement system [3]. We briefly review both methods here; for more details and stability analysis, see [2] and [3]. Then, in Section 5 we extend the first scheme to include a ROM on one of the subdomains.

3.1. "Eliminate then discretize" partitioned scheme. Following [2] we refer to this scheme as the Implicit Value Recovery (IVR) method. To explain IVR it is convenient to write (3.1) in the canonical DAE form:

$$\begin{aligned} \dot{y} &= f(t, y, z) \\ 0 &= g(t, y, z) \end{aligned} \tag{3.2}$$

where $y = (\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2)$ is the differential variable, $z = \boldsymbol{\lambda}$ is the algebraic variable,

$$f(t, y, z) = \begin{pmatrix} M_1^{-1}\left(\overline{\mathbf{f}}_1(\boldsymbol{\Phi}_1) - G_1^T\boldsymbol{\lambda}\right) \\ M_2^{-1}\left((\overline{\mathbf{f}}_2(\boldsymbol{\Phi}_2) + G_2^T\boldsymbol{\lambda}\right) \end{pmatrix} \tag{3.3}$$

and

$$g(t, y, z) = S\boldsymbol{\lambda} - G_1M_1^{-1}\overline{\mathbf{f}}_1(\boldsymbol{\Phi}_1) + G_2M_2^{-1}(\overline{\mathbf{f}}_2(\boldsymbol{\Phi}_2)\,. \tag{3.4}$$

The matrix $S = G_1M_1^{-1}G_1^T + G_2M_2^{-1}G_2^T$ in (3.4) is the Schur complement of the upper left 2×2 block submatrix of the matrix in (3.1).

It can be shown that the Schur complement S is nonsingular; see Proposition 4.1 in [2]. This implies that the Jacobian $\partial_z g = S$ is also nonsingular for all t. As a result, the equation $g(t, y, z) = 0$ defines z as an implicit function of the differential variable. After solving this equation for the algebraic variable and inserting the solution into (3.1) we obtain a coupled system of ODEs in terms of the states:

$$\begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{\Phi}}_1 \\ \dot{\mathbf{\Phi}}_2 \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{f}}_1(\mathbf{\Phi}_1) - G_1^T \boldsymbol{\lambda} \\ \overline{\mathbf{f}}_2(\mathbf{\Phi}_2) + G_2^T \boldsymbol{\lambda} \end{bmatrix} \tag{3.5}$$

The IVR scheme is based on the observation that application of an explicit time integration scheme to discretize (3.5) in time effectively decouples the equations and makes it possible to solve them independently.

The IVR algorithm for solving the coupled system is now as follows. Let $D_t^n(\mathbf{\Phi})$ be a forward time differencing operator such as the Forward Euler operator $D_t^n(\mathbf{\Phi}) = (\mathbf{\Phi}^{n+1} - \mathbf{\Phi}^n)/\Delta t$. For each time step t^n:

1. *Compute modified forces*: For $i = 1, 2$ use $\mathbf{\Phi}_i^n$ to compute the vector
$$\tilde{\boldsymbol{f}}_i^n := \overline{\mathbf{f}}_i(\mathbf{\Phi}_i^n) = \mathbf{f}_i - (D_i + A_i)\mathbf{\Phi}_i^n$$
2. *Compute the Lagrange multiplier*: Solve the Schur complement system
$$\left(G_1 M_1^{-1} G_1^T + G_2 M_2^{-1} G_2^T\right)\boldsymbol{\lambda}^n = G_1 M_1^{-1} \tilde{\boldsymbol{f}}_1^n - G_2 M_2^{-1} \tilde{\boldsymbol{f}}_2^n$$
for $\boldsymbol{\lambda}^n$. Compute $G_1^T\boldsymbol{\lambda}^n$ and $G_2^T\boldsymbol{\lambda}^n$.
3. *Update the state variables:* For $i = 1, 2$, solve the systems
$$M_i D_t^n(\mathbf{\Phi}_i) = \tilde{\boldsymbol{f}}_i^n + (-1^i) G_i^T \boldsymbol{\lambda}^n \tag{3.6}$$

3.2. "Discretize then eliminate". We will now describe an alternative method in which we discretize in time first before forming and solving the Schur complement system. Following [3] we refer to this scheme as the Interface Flux Recovery (IFR) method. The advantage of the IFR scheme is that it does not require an index reduction step and an explicit time integration to decouple the subdomain equations. Thus, the starting point for the development of the IFR scheme is the original Index-2 DAE system (2.6).

To derive the IFR scheme we first move the advective and diffusive terms from the right hand side vector $\overline{\boldsymbol{f}}_i$ into the left hand side of (2.6) and write this DAE as:

$$\begin{aligned} M_1\dot{\mathbf{\Phi}}_1 + G_1^T\boldsymbol{\lambda} + K_1\mathbf{\Phi}_1 &= \mathbf{f}_1(t) \\ M_2\dot{\mathbf{\Phi}}_2 - G_2^T\boldsymbol{\lambda} + K_2\mathbf{\Phi}_2 &= \mathbf{f}_2(t) \\ G_1\mathbf{\Phi}_1 - G_2\mathbf{\Phi}_2 &= 0 \end{aligned} \tag{3.7}$$

where $K_r = D_r + A_r$, $r = 1, 2$. This system is now discretized using the θ-method for $\theta_i \in [0, 1]$ as follows:

$$\begin{aligned} M_1(\mathbf{\Phi}_1^{n+1} - \mathbf{\Phi}_1^n)/\Delta t + G_1^T\boldsymbol{\lambda}^{n+1} + K_1(\theta_1\mathbf{\Phi}_1^{n+1} + (1-\theta_1)\mathbf{\Phi}_1^n) &= \theta_1\mathbf{f}_1(t^{n+1}) + (1-\theta_1)\mathbf{f}_1(t^n) \\ M_2(\mathbf{\Phi}_2^{n+1} - \mathbf{\Phi}_2^n)/\Delta t - G_2^T\boldsymbol{\lambda}^{n+1} + K_2(\theta_2\mathbf{\Phi}_2^{n+1} + (1-\theta_2)\mathbf{\Phi}_2^n) &= \theta_2\mathbf{f}_2(t^{n+1}) + (1-\theta_2)\mathbf{f}_2(t^n) \\ G_1\mathbf{\Phi}_1^{n+1} - G_2\mathbf{\Phi}_2^{n+1} &= 0 \end{aligned} \tag{3.8}$$

Following the approach in [3], define the matrix $W_i = M_i + \Delta t\theta_i K_i$, and define a new right hand side vector as:

$$\mathbf{g}_i(\mathbf{\Phi}_i^n) = \Delta t(\theta_i\mathbf{f}_i(t^{n+1}) + (1-\theta_i)(\mathbf{f}_i(t^n) - K_i\mathbf{\Phi}_i^n)) + M_i\mathbf{\Phi}_i^n\,.$$

Then, the fully discrete problem (3.8) is equivalent to the system of equations:

$$\begin{bmatrix} W_1 & 0 & \Delta t G_1^T \\ 0 & W_2 & -\Delta t G_2^T \\ G_1 & -G_2 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_1^{n+1} \\ \mathbf{\Phi}_2^{n+1} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1(\mathbf{\Phi}_1^n) \\ \mathbf{g}_2(\mathbf{\Phi}_2^n) \\ 0 \end{bmatrix} \tag{3.9}$$

Note that if we discretize (3.7) in time using the θ-method with $\theta_i = 0$ on both subdomains then the system in (3.9) reduces to

$$\begin{bmatrix} M_1 & 0 & \Delta t G_1^T \\ 0 & M_2 & -\Delta t G_2^T \\ G_1 & -G_2 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_1^{n+1} \\ \mathbf{\Phi}_2^{n+1} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1(\mathbf{\Phi}_1^n) \\ \mathbf{g}_2(\mathbf{\Phi}_2^n) \\ 0 \end{bmatrix} \tag{3.10}$$

In the case of a consistent mass matrix, the Schur complement system is closely related to the system from the IVR scheme.

$$\Delta t\Big(G_1 M_1^{-1} G_1^T + G_2 M_2^{-1} G_2^T\Big)\boldsymbol{\lambda} = G_1 M_1^{-1}\mathbf{g}_1(\mathbf{\Phi}_1^n) - G_2 M_2^{-1}\mathbf{g}_2(\mathbf{\Phi}_2^n) \tag{3.11}$$

For now, we assume explicit time integration schemes; the procedure here can be easily extended to implicit schemes. Assuming that $\mathbf{\Phi}_1^n, \mathbf{\Phi}_2^n$ are known at time t^n, compute $\mathbf{\Phi}_1^{n+1}, \mathbf{\Phi}_2^{n+1}$ as follows:

1. Set $\theta_i = 0$ for $i = 1, 2$, and compute the vector

$$\tilde{\mathbf{g}}_i^n := \mathbf{g}_i(\mathbf{\Phi}_i^n) = \Delta t(\mathbf{f}_i(t^n) - K_i\mathbf{\Phi}_i^n) + M_i\mathbf{\Phi}_i^n$$

2. *Compute the Lagrange multiplier*: Solve the Schur complement system

$$\Delta t\Big(G_1 M_1^{-1} G_1^T + G_2 M_2^{-1} G_2^T\Big)\boldsymbol{\lambda} = G_1 M_1^{-1}\tilde{\mathbf{g}}_1^n - G_2 M_2^{-1}\tilde{\mathbf{g}}_2^n$$

   for $\boldsymbol{\lambda}$. Compute $\Delta t G_1^T\boldsymbol{\lambda}$ and $\Delta t G_2^T\boldsymbol{\lambda}$.

3. *Update the state variables*: For $i = 1, 2$ solve explicitly the equations

$$M_i\mathbf{\Phi}_i^{n+1} = \tilde{\mathbf{g}}_i^n + (-1)^i \Delta t G_i^T\boldsymbol{\lambda}$$

4. ROM Construction. We now extend the partitioned schemes described in Section 3 to the case where a reduced order model (ROM) on one of the subdomains is coupled with a traditional finite element scheme on the other subdomain. Since the main practical difference in our scenario between the IVR and IFR methods presented in Sections 3.1 and 3.2 is in the order of the time discretization and the Schur complement (elimination) steps we will explicitly present the ROM-FEM coupling only for the IVR case. In this section, we describe the construction of a reduced order model using Galerkin projection, a technique to deal with Dirichlet boundaries, and then the implementation of the partitioned scheme to include this ROM.

Without loss of generality, we choose to implement the reduced order model on Ω_1. First, a new, reduced order basis matrix $\tilde{U}$ is determined such that $\mathbf{\Phi}_1 = \tilde{U}\boldsymbol{\varphi}_R$, where $\boldsymbol{\varphi}_R$ is the vector of solution coefficients for the reduced order model. The matrix $\tilde{U}$ has dimensions $N_1 \times N_R$, with $N_R \ll N_1$.

To compute $\tilde{U}$, we obtain m snapshots in time of the original finite element solution on Ω_1, letting the boundary corresponding to the interface γ have Neumann instead of Dirichlet conditions. For problems with low advection, the full order model can be run on only Ω_1; however, it is our experience that for highly advective problems it is best to use a full order

model obtained on the entire domain Ω. This can be either a FEM solution across Ω or a coupled FEM-FEM solution obtained as described in Section 3. For the numerical results shown in Section 6, we use the FEM solution across Ω. Define $t_k = k(\Delta_s t)$, for $k = 1, \dots m$ and let $\Delta_s t$ be the size of the timestep chosen for obtaining the snapshots. Let X be the resulting $N_1 \times m$ snapshot matrix, with the kth column corresponding to the coefficient vector $\mathbf{\Phi}_1(t_k)$ of the full finite element solution at t_k.

To deal with the Dirichlet boundary condition, we consider removing the rows of the snapshot matrix X that correspond to Dirichlet nodes on Γ_1 to obtain the adjusted snapshot matrix X_0. This matrix contains solution coefficients corresponding to the free nodes, that is the interior and the Neumann nodes that correspond to the unknowns in our problem. Let $N_{1,0}$ denote the number of these free nodes. Then, we compute the singular value decomposition of the adjusted matrix, $X_0 = U_0 \Sigma_0 V_0^T$. The new basis matrix $\tilde{U}_0$ is obtained by retaining the columns of U_0 corresponding to the N_R largest singular values. Thus, the matrix $\tilde{U}_0$ is $N_{1,0} \times N_R$, instead of $N_1 \times N_R$. Note that the new basis functions, the columns of $\tilde{U}_0$, are globally supported rather than locally supported as is the case with traditional finite element basis functions. Most commonly, we choose a tolerance level δ so that we remove columns corresponding to singular values which are less than δ. Once we have the final reduced order solution vector $\boldsymbol{\varphi}_R$, we can recover the full finite element vector by computing $\mathbf{\Phi}_{1,0} = \tilde{U}_0 \boldsymbol{\varphi}_R$. However, the coefficients in $\mathbf{\Phi}_{1,0}$ correspond to the free nodes and so, this vector does not contain the information about the imposed Dirichlet boundary condition. We recover a full state coefficient vector $\mathbf{\Phi}_1$ by augmenting $\mathbf{\Phi}_{1,0}$ with the values at the Dirichlet nodes.

5. Extension of IVR and IFR to a ROM-FEM coupling. Since we now have the reduced basis matrix $\tilde{U}_0$, we can use $\tilde{U}_0^T$ to project the equations over Ω_1 into the smaller space. However, since $\tilde{U}_0$ only has the information from the non-Dirichlet nodes, the other matrices need to be adjusted correspondingly so that only the non-Dirichlet rows and columns are used. Let the matrices $\tilde{M}_1, \tilde{G}_1, \tilde{D}_1, \tilde{A}_1$ represent the FEM matrices with rows and columns corresponding to Dirichlet nodes removed. Substituting $\tilde{U}_0 \boldsymbol{\varphi}_R$ for $\mathbf{\Phi}_1$ into the constraint and the first equation in the system (2.7), multiplying the first equation by $\tilde{U}_0^T$, and expanding its right hand side yields the following ROM-FEM monolithic problem:

$$\begin{aligned} \tilde{U}_0^T \tilde{M}_1 (\tilde{U}_0 \dot{\boldsymbol{\varphi}}_R) + \tilde{U}_0^T \tilde{G}_1{}^T \boldsymbol{\lambda} &= \tilde{U}_0^T \tilde{\mathbf{f}}_1 - \tilde{U}_0^T (\tilde{D}_1 + \tilde{A}_1) \tilde{U}_0 \boldsymbol{\varphi}_R \\ M_2 \dot{\mathbf{\Phi}}_2 - G_2^T \boldsymbol{\lambda} &= \bar{\mathbf{f}}_2(\mathbf{\Phi}_2) \\ \tilde{G}_1 (\tilde{U}_0 \dot{\boldsymbol{\varphi}}_R) - G_2 \dot{\mathbf{\Phi}}_2 &= 0 \end{aligned} \tag{5.1}$$

Note that the first equation is now of size N_R. Let $y = (\boldsymbol{\varphi}_R, \mathbf{\Phi}_2)$ be the differential variable, and $z = \boldsymbol{\lambda}$ the algebraic variable. As in Section 3, the ROM-FEM monolithic system (5.1) is an index-1 DAE having the same canonical form as (3.2) but with:

$$f(t, y, z) = \begin{pmatrix} (\tilde{U}_0^T \tilde{M}_1 \tilde{U})^{-1} \Big(\tilde{U}_0^T \bar{\mathbf{f}}_1 (\tilde{U}_0 \boldsymbol{\varphi}_R) - \tilde{U}_0^T \tilde{G}_1{}^T \boldsymbol{\lambda} \Big) \\ M_2^{-1} \Big((\bar{\mathbf{f}}_2(\mathbf{\Phi}_2) + G_2^T \boldsymbol{\lambda} \Big) \end{pmatrix} \tag{5.2}$$

and

$$g(t, y, z) = \tilde{S} \boldsymbol{\lambda} - \tilde{G}_1 \tilde{U}_0 (\tilde{U}_0^T \tilde{M}_1 \tilde{U})^{-1} \tilde{U}_0^T \bar{\mathbf{f}}_1 (\tilde{U}_0 \boldsymbol{\varphi}_R) + G_2 M_2^{-1} \bar{\mathbf{f}}_2 (\mathbf{\Phi}_2) \tag{5.3}$$

where $\tilde{S} = \tilde{G}_1 \tilde{U}_0 (\tilde{U}_0^T \tilde{M}_1 \tilde{U}_0)^{-1} \tilde{U}_0^T \tilde{G}_1{}^T + G_2 M_2^{-1} G_2^T$ is the Schur complement of the upper 2×2 block of the ROM-FEM monolithic problem. At this juncture, we point out that we

may safely expect the matrix $\tilde{U}_0^T \tilde{M}_1 \tilde{U}_0$ to be invertible because M_1, and therefore $\tilde{M}_1$ are full rank. Multiplication by the orthogonal matrix $\tilde{U}_0$ preserves the rank of the matrix. Now, the system (5.1) can be equivalently written as:

$$\begin{aligned} \dot{y} &= f(t, y, z) \\ 0 &= g(t, y, z) \end{aligned} \tag{5.4}$$

Extension of the IVR scheme to the ROM-FEM system (5.1) requires the Jacobian $\partial_z g = \tilde{S}$ to be non-singular for all t. In the case of the FEM-FEM coupled system (3.1) conditions on the Lagrange multiplier space were given in [2] that correspond to properties of the matrices G_1, G_2, and ensure that the FEM-FEM Schur complement is symmetric and positive definite. In the case of the ROM-FEM coupled problem we have observed numerically that the corresponding Schur complement $\tilde{S}$ is nonsingular. A formal proof and a sufficient condition for $\tilde{S}$ to be symmetric and positive definite is in progress and will be reported in the journal version of this paper.

Extension of the IFR scheme to the ROM-FEM coupling proceeds along the same lines presented for the IVR extension. Specifically, in (3.8) we substitute $\tilde{U}_0 \boldsymbol{\varphi}_R^{n+1}$ and $\tilde{U}_0 \boldsymbol{\varphi}_R^n$ for $\boldsymbol{\Phi}_1^{n+1}$ and $\boldsymbol{\Phi}_1^n$, respectively and then multiply the first equation in this system by $\tilde{U}_0^T$. This will result in a ROM-FEM version of (3.8) in which the first equation has dimension N_R.

5.1. Algorithm. In this section, we present the extension of the IVR partitioned scheme to the ROM-FEM monolithic system (5.1). Extension of the IFR scheme is very similar and is omitted for the sake of brevity.

Note that once $\tilde{U}_0$ is computed, the reduced size matrices $\tilde{U}_0^T \tilde{M}_1 \tilde{U}_0$, $\tilde{U}_0^T \tilde{D}_1 \tilde{U}_0$, $\tilde{U}_0^T \tilde{A}_1 \tilde{U}_0$ can all be precomputed. For the explicit time-stepping methods, we set $D_t^n(\boldsymbol{\varphi}) = (\boldsymbol{\varphi}^{n+1} - \boldsymbol{\varphi}^n)/\Delta t$.

The IVR ROM-FEM partitioned algorithm has two phases: an offline phase to compute the reduced order model and an online phase where the ROM is used in the partitioned scheme to solve the coupled system. For example, in the context of a PDE-constrained optimization algorithm that requires multiple solutions of the coupled problem, the first phase would be conducted offline before the optimization loop, and then the second phase would run at each optimization iteration.

Computation of the reduced order model (Offline)

1. Solve an appropriate full order model, either FEM on Ω_1 with Neumann conditions on γ or the full FEM on Ω, to collect samples for the snapshot matrix X. Compute the SVD $X_0 = U_0 \Sigma_0 V_0^T$ of the modified snapshot matrix X_0 containing the non-Dirichlet rows of X.
2. Given a threshold $\delta > 0$ define the reduced basis matrix $\tilde{U}_0$ by discarding all columns in U_0 corresponding to singular values less than δ.

Solution of the coupled ROM-FEM system for $t \in [0, T_f]$. (Online)

1. Use $\boldsymbol{\varphi}_R^n$ to compute the vector
$$\tilde{\boldsymbol{f}}_1^n := \tilde{U}_0^T \overline{\mathbf{f}}_1(\tilde{U}_0 \boldsymbol{\varphi}_R^n) = \tilde{U}_0^T \tilde{\mathbf{f}}_1 - \tilde{U}_0^T (\tilde{D}_1 + \tilde{A}_1) \tilde{U}_0 \boldsymbol{\varphi}_R^n$$
2. Use $\boldsymbol{\Phi}_2^n$ to compute the vector $\tilde{\boldsymbol{f}}_2^n := \overline{\mathbf{f}}_2(\boldsymbol{\Phi}_2^n) = \mathbf{f}_2 - (D_2 + A_2)\boldsymbol{\Phi}_2^n$
3. Solve the Schur complement system
$$\begin{aligned} \big(\tilde{G}_1 \tilde{U}_0 (\tilde{U}_0^T \tilde{M}_1 \tilde{U}_0)^{-1} \tilde{U}_0^T \tilde{G}_1{}^T + G_2 M_2^{-1} G_2^T\big) \boldsymbol{\lambda}^n = \\ \tilde{G}_1 \tilde{U}_0 (\tilde{U}_0^T \tilde{M}_1 \tilde{U}_0)^{-1} \tilde{\boldsymbol{f}}_1^n - G_2 M_2^{-1} \tilde{\boldsymbol{f}}_2^n \end{aligned}$$
for $\boldsymbol{\lambda}^n$. Compute $\tilde{U}_0^T \tilde{G}_1{}^T \boldsymbol{\lambda}^n$ and $G_2^T \boldsymbol{\lambda}^n$.

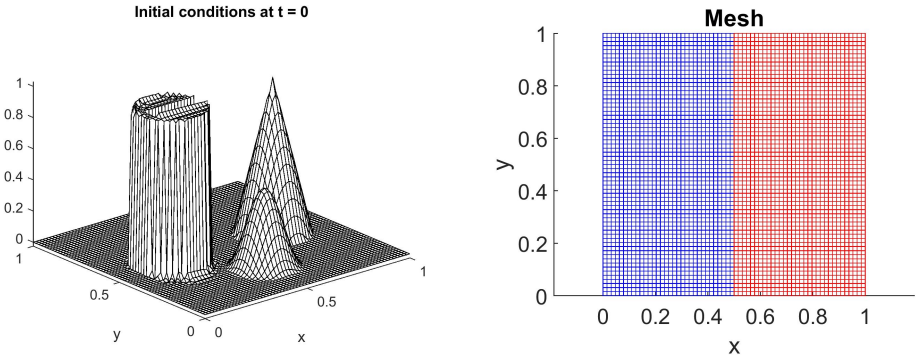
4. Solve the system $(\tilde{U}_0^T \tilde{M}_1 \tilde{U}_0) D_t^n(\boldsymbol{\varphi}_R) = \tilde{\boldsymbol{f}}_1^n - \tilde{U}_0^T \tilde{G}_1{}^T \boldsymbol{\lambda}^n$
5. Solve the system $M_2 D_t^n(\boldsymbol{\Phi}_2) = \tilde{\boldsymbol{f}}_2^n + G_2^T \boldsymbol{\lambda}^n$

6. Numerical examples. Let Ω be the unit square divided in half vertically by the line $x = 0.5$. Denote by Ω_1 the left side of the domain and Ω_2 the right side of the domain; see Figure 6.1(b). Let γ denote the interface ($x = 0.5$) between the two sides, and let $\Gamma_i = \partial\Omega_i \backslash \gamma$ for $i = 1, 2$. We take $\mathbf{n}_\gamma$ to be the unit normal on the interface pointing toward Ω_2.

In this section, we present select results for solving the model advection-diffusion interface problem (2.1) by coupling a ROM on Ω_1 with a traditional FEM on Ω_2. The coupled ROM-FEM problem is then solved by using the extension of the IFR partitioned scheme.

We impose Dirichlet boundaries conditions on the non-interface boundaries Γ_i, $i = 1, 2$. Results are shown for models with higher diffusion ($\epsilon_i = 0.01$) and lower diffusion ($\epsilon_i = 10^{-5}$) as well as pure advection ($\epsilon_i = 0$). RK4 is used as the explicit time discretization method, as it is a more accurate method than Forward Euler. The ROM is developed as described in Section 4.

Our experience suggests that for highly advective problems the best practice to obtain the m snapshots required to define the ROM on Ω_1 is to use a full order solution obtained on the entire domain Ω rather than on the target subdomain Ω_1 where we desire to construct the ROM. There are two options from this point. First, one can compute the SVD of the full snapshot matrix X and extract only the entries from U that correspond to the degrees of freedom from Ω_1. The second method is to first extract the rows from X that correspond to the desired degrees of freedom, and then compute the SVD of that reduced matrix. In practice, both methods tend to give similar results. We implement the second method in order to reduce the computational expense of solving the SVD.



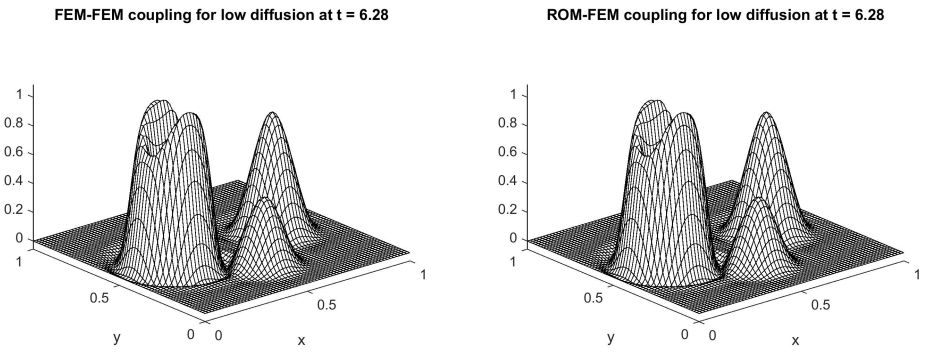
(a) Initial conditions (b) Uniform meshes in both domains

Fig. 6.1: Initial conditions and mesh

The initial conditions for our test problem, a combination of a cone, cylinder, and a smooth hump, are seen in Figure 6.1(a). We apply a rotating advection field $(0.5-y, x-0.5)$ and allow two full rotations. The results below are obtained using the IFR method described in Section 3.2, extended to the ROM-FEM coupling. In Ω, the spatial discretization is $1/64$ in both the x and y directions, yielding 4225 DOFs in Ω and 2145 DOFs in each of Ω_1, Ω_2. In the following results, we have retained a relatively large basis (roughly 150 modes) for the low diffusion and pure advection cases. Since our goal is to test the coupling method,

we do not focus here on the ROM development and basis size. However, trials have been run with a significantly smaller number of snapshots (m) as well as a smaller basis (down to $N_R = 22$), and the ROM and FEM solutions continue to show strong agreement on the interface.

First we examine the case with low diffusion, $\epsilon_i = 10^{-5}$ for $i = 1, 2$. For the FEM-FEM and ROM-FEM couplings, $\Delta t = 3.367 \times 10^{-3}$, but for the ROM snapshot construction, we set $\Delta_s t = \Delta t/3 = 1.123 \times 10^{-3}$. This creates a snapshot matrix X of size 4225×5597. Truncating all singular values less than $\delta = 10^{-10}$, the resulting basis is of size $N_R = 151$. Results for the FEM-FEM coupling in Figure 6.2(a) are compared with the ROM-FEM coupling in Figure 6.2(b)



(a) FEM-FEM coupling for $\epsilon_i = 10^{-5}$ (b) ROM-FEM coupling for $\epsilon_i = 10^{-5}$

Fig. 6.2: Comparison of FEM-FEM and ROM-FEM IFR couplings for model with low diffusion

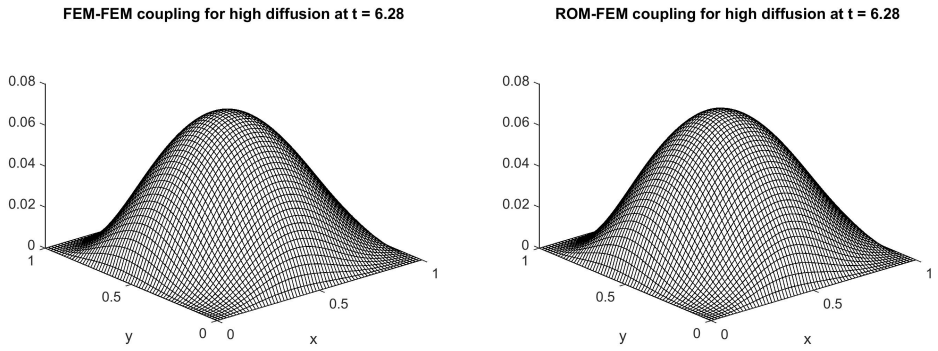
For high diffusion, $\epsilon_i = 0.01$ for $i = 1, 2$ the snapshot matrix X is of size 4225×6862 due to $\Delta t = 1.83 \times 10^{-3}$ and $\Delta_s t = \Delta t/2 = 9.16 \times 10^{-4}$. The resulting reduced basis size is $N_R = 59$. Results for the FEM-FEM coupling are seen in Figure 6.3(a) and the ROM-FEM coupling in Figure 6.3(b). If the tolerance for the singular values is increased from $\delta = 10^{-10}$ to $\delta = 10^{-4}$, the ROM basis size is now 32 and the solution looks the same as results with a basis of size 59.

We see in Figures 6.4(a) and 6.4(b) that the ROM solution in Ω_1 and the traditional FEM solution in Ω_2 align well on the interface for both low and high diffusion. Next, we examine the pure advection case ($\epsilon_i = 0$) for $i = 1, 2$. For the FEM-FEM and ROM-FEM couplings, $\Delta t = 3.367 \times 10^{-3}$, and we set $\Delta_s t = \Delta t/2 = 1.684 \times 10^{-3}$ for the ROM snapshot construction. This produces a snapshot matrix X with dimensions 4225×3732. With $\delta = 10^{-10}$, the size of the reduced basis is $N_R = 147$. Results in Figures 6.5(a) and 6.5(b) closely resemble those of the low diffusion case in Figures 6.2(a) and 6.2(b). For pure advection, we may compute errors using the FEM-FEM coupling solution as a reference. Let $\mathbf{\Phi}^h_{FF}(t) = \begin{bmatrix}\mathbf{\Phi}^h_{1,FF}(t) & \mathbf{\Phi}^h_{2,FF}(t)\end{bmatrix}^T$ denote the finite element solution over Ω of the coupled FEM-FEM model at time t. Likewise, let $\mathbf{\Phi}^h_{RF}(t) = \begin{bmatrix}\mathbf{\Phi}^h_{1,RF} & \mathbf{\Phi}^h_{2,RF}(t)\end{bmatrix}^T$ denote the finite element solution over Ω of the coupled ROM-FEM model at time t. In both, let the value

of the function on the interface come from the solution on Ω_1. The errors are as follows:

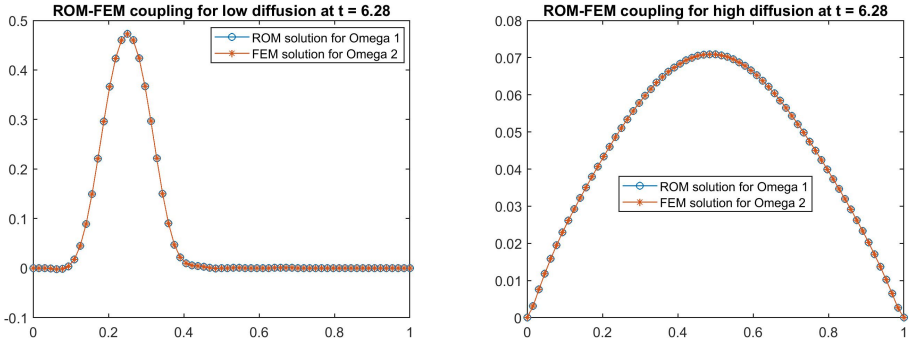
$$||\mathbf{\Phi}^h_{FF}(2\pi) - \mathbf{\Phi}^h_{RF}(2\pi)||_0 = 7.595 \times 10^{-5} \qquad ||\mathbf{\Phi}^h_{FF}(2\pi) - \mathbf{\Phi}^h_{RF}(2\pi)||_1 = 4.072 \times 10^{-3}$$
$$||\mathbf{\Phi}^h_{FF}(0) - \mathbf{\Phi}^h_{FF}(2\pi)||_0 = 7.4940 \times 10^{-2} \qquad ||\mathbf{\Phi}^h_{FF}(0) - \mathbf{\Phi}^h_{FF}(2\pi)||_1 = 8.28143$$
$$||\mathbf{\Phi}^h_{RF}(0) - \mathbf{\Phi}^h_{RF}(2\pi)||_0 = 7.4942 \times 10^{-2} \qquad ||\mathbf{\Phi}^h_{RF}(0) - \mathbf{\Phi}^h_{RF}(2\pi)||_1 = 8.28137$$

Lastly, we display results for low diffusion on a non-uniform mesh. For the fine mesh, we leave the spatial discretization at $1/64$ in the y direction and $1/32$ in the x direction in Ω_i. The coarse mesh is obtained by setting the spatial discretization to $1/32$ in the y direction and $1/16$ in the x direction in the other subdomain. In Figures 6.6(a) and 6.7(a), we implement the coarse mesh on Ω_1 where the ROM is constructed and the fine mesh on Ω_2. The degrees of freedom on the interface are always set as the degrees of freedom from Ω_2; thus in this example there are 65 degrees of freedom on the interface. In Figures 6.6(b) and 6.7(b), we set the fine mesh on Ω_1 and the coarse mesh on Ω_2. In this case, there are 33 degrees of freedom on the interface. For both of these non-uniform meshes, there is strong agreement between the ROM and FEM solutions on the interface, but the results are a bit more diffused than the results from implementing the fine mesh on both Ω_1 and Ω_2.



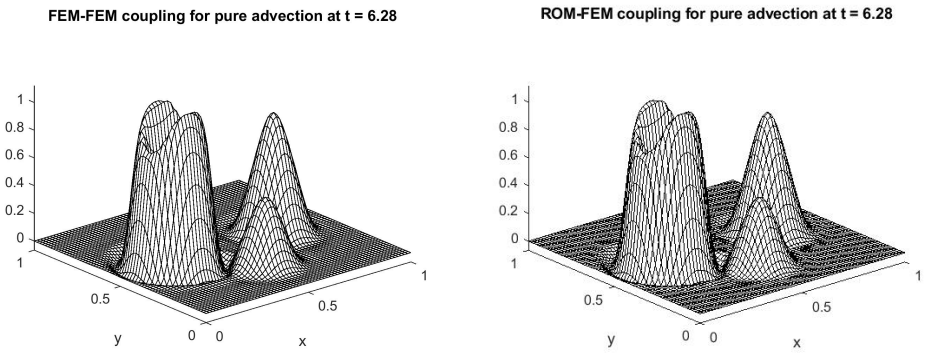
(a) FEM-FEM coupling for $\epsilon_i = 0.01$ (b) ROM-FEM coupling for $\epsilon_i = 0.01$

Fig. 6.3: Comparison of FEM-FEM and ROM-FEM IFR couplings for model with high diffusion



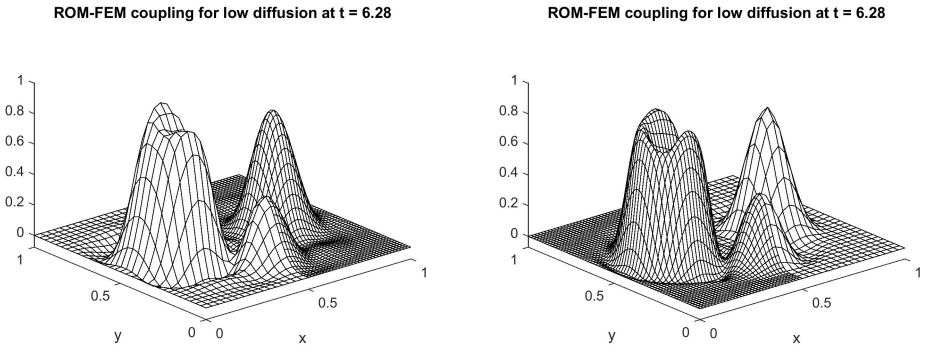
(a) ROM-FEM interface solution for $\epsilon_i = 10^{-5}$ (b) ROM-FEM interface solution for $\epsilon_i = 0.01$

Fig. 6.4: Interface solutions for low and high diffusion ROM-FEM IFR coupling



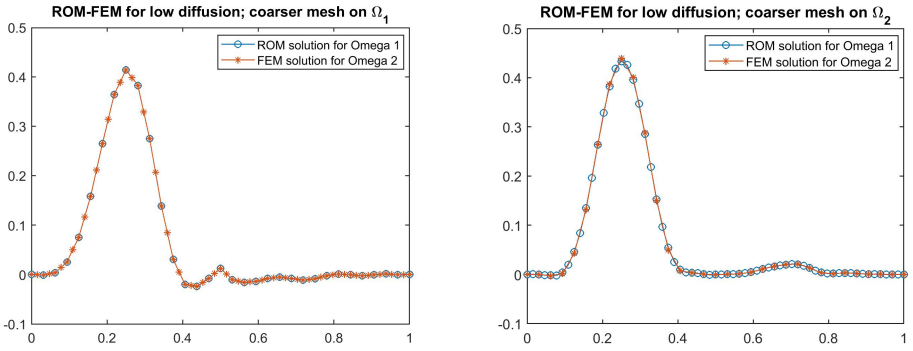
(a) Advective FEM-FEM coupling (b) Advective ROM-FEM coupling

Fig. 6.5: Comparison of FEM-FEM and ROM-FEM IFR couplings for the pure advection case



(a) Coarse mesh on Ω_1 for $\epsilon_i = 10^{-5}$ (b) Coarse mesh on Ω_2 for $\epsilon_i = 10^{-5}$

Fig. 6.6: Comparison of non-uniform mesh results for low diffusion



(a) Coarse mesh on Ω_1 for $\epsilon_i = 10^{-5}$

(b) Coarse mesh on Ω_2 for $\epsilon_i = 10^{-5}$

Fig. 6.7: Comparison of non-uniform mesh results on interface for low diffusion

7. Conclusions. We presented an explicit partitioned scheme for a transmission problem that extends the approaches developed in [2] and [3] to the case of coupling a reduced order model with a traditional finite element scheme. In particular, the scheme begins with a monolithic formulation of the transmission problem and then employs a Schur complement to solve for a Lagrange multiplier representing the interface flux as a Neumann boundary condition. We constructed a reduced order model from a full finite element solution and then presented an algorithm to couple this reduced model with a traditional finite element scheme. Our numerical results show that the ROM-FEM coupling produces solutions which strongly agree with those produced by a FEM-FEM coupling. Implementing the reduced order model reduces the time and computational cost of solving the coupled system. In principle, this coupling method should extend to other discretizations such as finite volume; this will be studied in future work. Additionally, extensions to nonlinear problems and predictive runs can be considered. In a follow up work, we plan to study the coupling of two reduced order models.

References.

[1] C. Hoang, Y. Choi, and K. Carlberg, *Domain-decomposition least-squares petrov-galerkin (dd-lspg) nonlinear model reduction*, Computer Methods in Applied Mechanics and Engineering, 384 (2021).

[2] K. Peterson, P. Bochev, and P. Kuberry, *Explicit synchronous partitioned algorithms for interface problems based on lagrange multipliers*, Computers & Mathematics with Applications, 78 (2019), pp. 459–482.

[3] K. C. Sockwell, K. Peterson, P. Kuberry, P. Bochev, and N. Trask, *Interface flux recovery coupling method for the ocean–atmosphere system*, Results in Applied Mathematics, 8 (2020), pp. 100–110.

A SURVEY OF THE KOOPMAN OPERATOR AND MOMENT PROPAGATION

THOMAS Z. DEAN* AND EDGAR GALVAN†

Abstract. This article surveys several techniques and results that make use of the application of the Koopman operator to the study of dynamical systems. Close attention is paid to the study of the spectrum of the Koopman operator and the propagation of moments for uncertainty quantification. The Koopman Operator theoretic approach described in this article is an alternative formalism for the study of nonlinear dynamical system that provides unique advantages, especially in data-driven analysis. The Koopman operator is a linear transformation on a space of observables of the nonlinear dynamical system that evolves the system with time. The state space is *lifted* to the space of (possibly infinite) observables via functions belonging to the Koopman invariant subspace of functions. The surveyed data-driven techniques have a background in the mathematical field of Operator Theory where spectral and Hilbert Space theory are applied to the Koopman operator. We include examples to show the advantage of working with several Koopman operator theoretic methods.

1. History and Development of the Koopman Operator. Dynamical systems, and in particular nonlinear dynamical systems, have been intensively studied in physics and math literature since the late 1800s. The classical geometric perspective, originating with Poincaré, has been a powerful tool in aiding our understanding of linear systems and the local behavior of nonlinear systems (through linear approximation in a neighborhood of a fixed point). Recently, this classical perspective is being supplemented by a *operator-theoretic* perspective. In a seminal paper written in 1931, B.O. Koopman [4] made a connection between the spectrum of the Koopman (composition) operator and various dynamical system properties and demonstrating that it is possible to represent a nonlinear dynamical system in terms of an infinite-dimensional composition operator acting on a Hilbert space of state space observables (also called measurements). In 1932, Koopman and von Neumann extended this theory to dynamical systems with a continuous spectrum. It is notable that Koopman's 1931 paper led to the proofs of the ergodic theorem by von Neumann and Birkhoff. [9]

Recently, the Koopman approach to the study of dynamical systems has led to data-driven techniques to understand the nonlinear dynamics of the system by recovering the spectrum of the Koopman operator [5], [13]. This approach *lifts* the original nonlinear dynamical system (state space), denoted by $\mathcal{M}$, to a new dynamical system, denoted by $\mathcal{F}$, for which the Koopman operator governs the evolution of *observables* as a linear transformation.

These observables are functions defined in the state space (a Hilbert space). The evolution of the observables provide an understanding of the evolution of the dynamical system. The Koopman view of dynamical systems is again gaining interest due to the seminal work by Rowley, Mezic et al. demonstrating Koopman operator theoretic data-driven methods to analyze nonlinear dynamical systems [11]. Several authors have since investigated this Koopman view of dynamical systems which has been at the forefront of many data-driven methods, such as dynamic mode decomposition (DMD) and extended mode decomposition (EDMD) [1],[13].

The methods of DMD and EDMD are implemented through a finite dimensional approximation of the Koopman operator to understand the behavior of dynamical systems, which we will discuss in detail in this survey. We will also address the Koopman expectation, which does not require the explicit construction of the Koopman operator nor a finite-dimensional approximation of the Koopman operator [2]. This method, along with the method discussed in [6], is used to calculate higher-order statistics (e.g., calculate cen-

*University of Wyoming, tdean3@uwyo.edu
†Sandia National Laboratories, egalvan@sandia.gov

tral moments). The advantage of the Koopman approach to nonlinear dynamical systems is that we are able to utilize the richness of the mathematical field of Operator Theory in a linear setting to recover the spectrum and moments of a given dynamical system.

The purpose of this survey is to provide an updated review of the current state of the Koopman operator as it pertains to the use of the Koopman operator to study spectral properties and moment propagation of dynamical systems. In this survey we will discuss the data-driven techniques of DMD, EDMD, the Christoffel function, and Generalized Laplace Analysis. These data-driven techniques allow the approximation and recovery of the spectrum of the Koopman operator, which govern the nonlinear dynamical system. Lastly, we present the Koopman expectation and Koopman moment propagation which allow for the calculation of higher order statistics of a given nonlinear dynamical system. The applications of the Koopman operator are vast and this survey only touches upon a handful of these. Throughout this survey, citations to relevant works will be provided for further reading when needed.

2. Overview of the Koopman Operator. Consider dynamical systems of the form

$$\dot{x}(t) = F(x) \tag{2.1}$$

where $x \in \mathbb{R}^n$ and $F : \mathcal{M} \to \mathcal{M}$ is the *flow map* defined on the state space $\mathcal{M}$. It is important to note that $\mathcal{M}$ is a measure space. For finite-dimensional systems, we take $M \subset \mathbb{R}^n$ and for infinite-dimensional systems we take M to be a subset of a Banach space. We can also include an input (forcing) function u such that our dynamical system may be of the form

$$\dot{x}(t) = F(x, u) \tag{2.2}$$

Dynamical systems of the form (2.1) and (2.2) are continuous dynamical systems. When implementing the data-driven methods in section 4, snapshots of the data are used to construct a discrete representation of the dynamical system. This is expressed as follows:

$$x_{n+1} = F(x_n)$$

That is, the flow map moves the state x_n one timestamp ahead to x_{n+1}. The Koopman operator U acting on observables behaves as

$$Uf_n = f_{n+1}$$

Throughout this article, we assume that we are in an invertible measure preserving setting. That is, M is endowed with a Borel sigma algebra Σ and a measure ν such that

$$\nu(F^{-1}(A)) = \nu(A), \quad \forall A \in \Sigma$$

The Koopman operator is a composition operator $U : \mathcal{H} \to \mathcal{H}$ where $\mathcal{H}$ is an arbitrary Hilbert space. We define U for all $h : \mathcal{M} \to \mathbb{C}$ and $h \in \mathcal{H}$ by

$$(Uh)(x) = [h \circ F](x) = h(F(x)) \tag{2.3}$$

where h is referred to as an *observable* function. Furthermore, note that the Koopman operator is a *composition* operator. The composition operator acts as a change of basis, which is precisely what we are doing by lifting the dynamical system from the state space to the observable space. If we choose $\mathcal{H} = L^2(\nu)$, the space of complex-valued functions that

are square integrable for two arbitrary functions f and g with respect to the given measure ν have the following standard inner product.

$$\langle f, g \rangle = \int_{\mathcal{M}} f \bar{g} d\nu \tag{2.4}$$

With the assumption that we are in a measure-preserving setting, U is a unitary operator. That is, $U^{-1} = U^*$. Thus, a spectral representation of U exists and we can gain insight into the spectrum of U which we will denote by $\sigma(U)$. That is,

$$U = \int_{\mathbb{T}} z dE(z) \tag{2.5}$$

where $\sigma(U) \subset \mathbb{T}$ since U is unitary and its spectrum lies on the unit circle. Here, E represents the projection-valued measure which defines a real-valued positive spectral measure μ_f on $\mathbb{T}$.

$$\mu_f(A) = \langle E(A)f, f \rangle \tag{2.6}$$

By working with the spectral measure, we can understand the decomposition of the spectrum. That is,

$$\sigma(U) = \sigma_{pp}(U) \cup \sigma_{ac}(U) \cup \sigma_{sc}(U)$$

where the subscripts pp, ac, and sc denote the pure point, absolutely continuous, and singular continuous spectrum respectively. More will be discussed on the recovery of the spectrum of U in section 3 of this survey.

The power of the Koopman operator when studying nonlinear dynamical systems is that the Koopman operator is a linear operator in the lifted (observable) space. The problem setting consists of a flow map F that evolves the states to a future state in a nonlinear manner. As with most nonlinear problems, our goal is to develop a way to manipulate the problem such that we can work in a linear setting and use the vast knowledge of solving linear problems. We define the observable space for which the linear Koopman operator behaves according to equation (2.3). To achieve this, we find the spectrum of the Koopman operator in order to represent the states as observables which the Koopman operator acts on. More precisely, we calculate the *Koopman tuple* which consists of the Koopman eigenvalues, eigenfunctions, and modes. The Koopman eigenfunctions and modes are the "bridges" between the nonlinear state-space and linear observable space that we aim to construct. This can be shown through a commutative diagram as in figure 1 in [13]. In the literature, the observable space is often referred to as the *lifted* space or the *measurement* space. The importance of DMD and EDMD is that they allow for an approximation of the Koopman tuples. These two techniques allow for a deeper analysis of nonlinear dynamical systems, which we will discuss in the proceeding sections.

2.1. Eigenfunctions of Dynamical Systems and the Koopman Operator. Let $\varphi(x)$ denote a Koopman eigenfunction with its corresponding eigenvalue λ. Note the following equation.

$$\varphi(x_{n+1}) = U\varphi(x_n) = \lambda(x_n) \tag{2.7}$$

If we work in a continuous time dynamical system, we have a Lie operator eigenfunction. We will not discuss this case in this survey since our focus is on discrete time systems.

For further reading of the Lie operator eigenfunction, [1] discusses this topic and provides additional resources for further reading.

An important property of Koopman eigenfunctions is that they can be used to generate more eigenfunctions. That is,

$$U\varphi_1(x)\varphi_2(x)) = \varphi(F(x))\varphi_2(F(x)) = \lambda_1\lambda_2\varphi_1(x)\varphi_2(x)$$

where $\lambda_1\lambda_2$ is an eigenvalue given by the eigenfunction $\varphi_1(x)\varphi_2(x)$. The literature on the Koopman operator, and especially on the algorithms such as DMD and EDMD, has an intense focus on recovering or approximating the associated eigenfunctions of the Koopman operator. This is because if observables can be constructed as linear combinations, then the observables have a linear evolution with respect to the Koopman operator. That is, to understand a nonlinear dynamical system we study observables g. If $g \in \text{span}\{\varphi_i(x)\}$ for $i = 1, ..., m$ for some m, then the evolution of g is linear and can be expressed as follows:

$$g(x) = \sum_m v_m\varphi_m, \quad Ug(x) = \sum_m v_m\lambda_m\varphi_m \tag{2.8}$$

where v_m is a coefficient matrix, which is referred to as a *Koopman mode*. Thus, the subspace spanned by the Koopman eigenfunctions is an invariant subspace with respect to the action of the Koopman operator U. As we can see, by recovering the so-called Koopman tuples we can recover the observables of the system which behave linearly with respect to the action of U. The problem of solving for Koopman modes is extensively studied and is mentioned in section 3 of this survey and for further reading we refer to [8].

Ideally, the Koopman eigenfunctions are solved for analytically, but in practice this is rarely the case except for the simplest systems. The motivation behind the calculation of eigenfunctions is to provide the necessary coordinates for which the dynamics of the system behave linearly. Within the past decade, algorithms have been developed for discrete-time systems to approximate eigenfunctions such that we can often approximate the observables that live in the invariant subspace and behave linearly. Several prominent algorithms that aim to approximate or recover the eigenfunctions are subsequently discussed in this survey.

3. A Data-Driven Application. Data analysis and data science have shown increased interest due to the abundance of data that is available in a wide variety of problems. With this abundance of data, as it pertains to dynamical systems, the natural questions arises of what do we do with this data and how do we use the data to gain a deeper understanding of the system? A more recent and common approach has been to construct dynamical systems from data and then analyze, predict, and control the system [12]. This is exactly where the Koopman operator, and more specifically the spectrum of the Koopman operator, has been shown to be of importance. In [1], Kutz et al. state the following five application of Koopman analysis when working with dynamical systems.

1. *Diagnostics.* Use $\sigma(U)$ to understand complex dynamical systems. That is, systems that are nonlinear and high-dimensional.
2. *Prediction.* Understand how the system evolves as time moves forward. The Koopman operator can make the prediction easier to solve for certain observables because of linearity.
3. *Estimation and control.* This is at the forefront of studying dynamical systems and the fact that the Koopman method can linearize the system makes estimation and control easier.
4. *Uncertainty quantification and management.* There are various methods where the Koopman operator is used to quantify uncertainty. These include, but are not

limited to, the Koopman expectation and the use of EDMD which will be discussed later.

5. *Understanding.* The Koopman operator provides a more intuitive framework to understand complex dynamical systems.

The connection between data-driven methods and Koopman have been especially powerful in the methods of DMD and EDMD. The Koopman approach of solving a nonlinear system is to redefine the state space in terms of observables. That is, where the state space is nonlinear we apply a function g such that we obtain a linear system.

DMD	Koopman
X and X' are the data matrices, which we take to be observables.	Let g be a function such that $g(X), g(X') \to Y, Y'$
$A = X'X^\dagger$ $\downarrow$	$A_Y = Y'Y^\dagger$ $\downarrow$ DMD
$x_{k+1} = Ax_k$	$g(x_{k+1}) \approx Ug(x_k)$ $U = A_Y$

Note that for the Koopman approach we need to be able to define the function (observable) g so that we can approximate the nonlinear dynamics of the system. The key fact to remember is that g is in the span of the eigenfunctions of U as is expressed in (2.8). One drawback of DMD is that we are limited in the observables we choose, which are the data matrices X and X'. For some cases, DMD proves to be an inaccurate data-driven method to apply to nonlinear systems. To mitigate this problem, we can make use of observables (a dictionary function) to apply to the data matrices to gain a more accurate approximation of the nonlinear system. This method is EDMD, which will be discussed in detail in this section.

It is important to note that both DMD and EDMD are methods of providing finite-dimensional approximation of the infinite-dimensional Koopman operator U. Because of this, DMD and EDMD only provide insight into the point spectrum while neglecting the continuous part of the spectrum of U. Subsection 3.4 provides an example detailing the DMD and EDMD method. The method discussed in subsection 3.4 provides an algorithm of recovering the full spectrum (i.e. point, absolutely continuous, and singular continuous) of the Koopman operator. The Lorenz system is an example where by neglecting the continuous spectrum we lose all understanding of the dynamical system. Examples will provided in this section to show the reader the power and effectiveness of applying data-driven methods to understand nonlinear dynamical systems.

3.1. Dynamic Mode Decomposition (DMD). The goal of DMD is to build a linear dynamical system $\dot{x} = Ax$ that best approximates a nonlinear dynamical system. To do so, the work involved is with constructing a matrix A that is a best fit approximation of the Koopman operator. The key fact here is that while the Koopman operator is infinite dimensional, A is a matrix for which we can find the corresponding eigenvalues and eigenfunctions. To make the DMD algorithm computationally efficient, we also discuss how to reduce the rank of A as it is common for A to be a large matrix which results in computing its eigenvalues and eigenfunctions to be inefficient.

Let $\dot{x} = f(x, u)$ be the nonlinear dynamical system of interest. Our goal is to approximate $f(x, u)$ with the *best* linear dynamical system where *best* is meant in a least squares sense. Let $x \in \mathbb{R}^n$ be a state and x_n be the n^{th} *snapshot* of the system. A snapshot is a measurement of the state of a system at a fixed time step that captures all the necessary physical and mathematical information at that instant in time. Our first goal is to find the

matrix A that advances our snapshots forward in time Δt as follows:

$$Ax_n = x_{n+1}$$

where $x_n := x(t_n)$. The data-driven aspect of DMD, as well as with all the algorithms mentioned in this section, comes from taking snapshots (i.e. measurements) to build a model. From these snapshots of data we can construct two high-dimensional $m \times n$ matrices X and X'.

$$X = \begin{bmatrix} x_1 & | & x_2 & | & \cdots & | & x_n \end{bmatrix}$$

as the state matrix that denotes contains each snapshot at time t_n for each of its columns. Furthermore, denote

$$X' = \begin{bmatrix} x_2 & | & x_3 & | & \cdots & | & x_{n+1} \end{bmatrix}$$

which represents the shifted matrix of X by one snapshot. From these two large data matrices, we can express their relation to each other as follows:

$$X' \approx AX \tag{3.1}$$

The goal of DMD is to find a best fit operator in a least squares sense. That is, to find such an A that

$$A = \text{argmin}_A \|X' - AX\| = X'X^\dagger$$

where $X^\dagger$ represents the pseudo-inverse (Moore-Penrose) of the data matrix X. To calculate $X^\dagger$, we implement the singular value decomposition (SVD) method as

$$X = U\Sigma V^* \implies X^\dagger = V\Sigma^{-1}U^*$$

While we want to use SVD to calculate $X^\dagger$, it is also important in creating a low-rank matrix A that still holds the information of the dynamics of the system. It is common for X and X' to be large data matrices such that $A = XX'$ will be too large a matrix to efficiently apply spectral analysis. This rank reduction is known as rank$-r$ truncation. To do this, we look at the singular values of Σ and project A onto the first r modes in U and approximate $X^\dagger$. Implementing this gives the following:

$$X \approx U_r\Sigma_r V_r^* \implies X^\dagger \approx V_r\Sigma_r^{-1}U_r^*$$

While A is of dimension $m \times m$, we can construct an approximation of A that is of dimension $r \times r$ with $r << m$. We can express this as follows:

$$\tilde{A} = U_r^* A U_r = U^* X' V \Sigma^{-1}$$

which is our reduced matrix of A of dimension $r \times r$. The purpose of constructing the matrix $\tilde{A}$ is to capture the nonlinear dynamics of the system and allow for the application of linear techniques to understand the corresponding eigenvalues and eigenfunctions of $\tilde{A}$. Let W be a matrix where the columns are eigenvectors and Λ a diagonal matrix with eigenvalues along its diagonal. Then,

$$\tilde{A}W = \Lambda W \tag{3.2}$$

Given the eigendecomposition of $\tilde{A}$, $x(t)$ can be expressed in a succinct manner such that we can understand the behavior of the states at an arbitrary time t. This is done by the DMD modes Φ [1],

$$\Phi = X'V\Sigma^{-1}W \tag{3.3}$$

Once the DMD modes and eigenvalues are computed, we can express the state as follows:

$$x_k = \Phi\Lambda^{k-1}b = \sum_{i=1}^{r} \varphi_i \lambda_i^{k-1} b_i \tag{3.4}$$

where $b = \Phi^{\dagger}x_1$ represents the mode amplitudes. The DMD method allows for the matrix approximation of the Koopman operator so that we can accurately approximate the states x_k of the nonlinear dynamical system.

3.2. Extended Dynamic Mode Decomposition (EDMD). The strength of the DMD algorithm is that we are able to build a regression model from the state X from one snapshot to the next. This is also a computationally efficient method which is why it is popular for data-driven applications. However, it can fail to be accurate for even moderately nonlinear systems. The EDMD algorithm has been developed to understand the eigenfunctions of nonlinear systems.

The EDMD method can be broken down into three steps.

1. Construct a data set of snapshot pairs.
2. Choose a dictionary of observables.
3. Use a least-squares method to approximate U by a finite-dimensional matrix.

As with DMD, snapshot matricies are used to construct the Koopman approximation. We begin with

$$X = \begin{bmatrix} x_1 & \Big| & x_2 & \Big| & \cdots & \Big| & x_n \end{bmatrix}$$

$$X' = \begin{bmatrix} x_2 & \Big| & x_3 & \Big| & \cdots & \Big| & x_{n+1} \end{bmatrix}$$

where $x_{n+1} = F(x_n)$, F is our flow map of the state space, and each $x_n, x_n' \in \mathcal{M}$.

The dictionary of observables (i.e. basis function) is given by the set $\{\varphi_1, ..., \varphi_N\}$ while the vector-valued *dictionary function*

$$\Psi(x) = [\varphi_1(x), ..., \varphi_K(x)]$$

is a mapping $\Psi : \mathcal{M} \to \mathbb{C}^{1\times K}$. The dictionary function is needed to extend the state X in order to contain the nonlinear measurements. Because of this, applications of EDMD typically involve higher dimensional data sets (observables) than DMD to approximate the Koopman operator. While EDMD can be used for more complex dynamical systems, the difficulty is in choosing the correct dictionary function, ones that span the Koopman invariant subspace. The most common dictionary functions to choose from are polynomials of degree $n \in \mathbb{N}$ and radial basis functions (RBF). Even with using a polynomial dictionary function, we need to choose a degree such that we obtain an accurate approximation while also not resulting in a computationally expensive process.

As with DMD, the goal is to construct a matrix $\tilde{U} \in \mathbb{R}^{K\times K}$, which is a finite dimensional approximation of the Koopman operator. Consider an arbitrary function f which can be expressed as a linear combination of the dictionary function.

$$f = \sum_{k=1}^{K} a_k \varphi_k(x) = \Psi(x)^T a$$

where $a \in \mathbb{R}^K$ is a weight. Recall that in an exact sense, the Koopman operator propagates an observable forward in time. That is, $Ug(x_n) = g(x_{n+1})$ where g is an arbitrary observable. However, since our dictionary of observables $\{\varphi_1, ..., \varphi_K\}$ are not, in general, an invariant subspace of U, we may not have this exact relation. Thus, the Koopman operator is related by the following equation:

$$Uf = (\Psi \circ F)a = \Psi(\tilde{U}) + r \tag{3.5}$$

where r is a residual term since $\tilde{U} \in \mathbb{R}^{K \times K}$ is a finite-dimensional approximation of U. The purpose of EDMD is to find $\tilde{U}$ and minimize the residual term r. That is, find $\tilde{U}$ such that

$$\tilde{U} = \text{argmin}_{\tilde{U}} \|G\tilde{U} - A\| \tag{3.6}$$

where

$$G = \frac{1}{N} \sum_{k=1}^{N} \Psi(x_k)\Psi(y_k)^T, \quad A = \frac{1}{N} \sum_{k=1}^{N} \Psi(x_k)\Psi(x_k)^T$$

This yields the analytical solution $\tilde{U} = G^{\dagger}A$, where $G^{\dagger}$ is the Moore-Penrose inverse of G. The finite-dimensional (matrix) approximation of $\tilde{U}$ minimizes r at each data snapshot pair. Since $\tilde{U}$ is a matrix, we are able to calculate the corresponding eigenfunctions with its respective eigenvalue. As we will see in the next section, the EDMD method is used to propagate the expected value of nonlinear dynamical systems. For further reading on the choice of dictionary functions, we refer the reader to the following paper [13].

It is important to note that both DMD and EDMD are trying to solve the same minimization problem. However, the major difference is that for EDMD we now have lifting functions from the state space to the observable space for a more accurate approximation of the Koopman operator. While EDMD may be more accurate method compared to DMD for nonlinear systems, there are a few drawbacks. The first one is choosing the dictionary basis that accurately approximates the Koopman operator while also being computationally tractable. Furthermore, there are cases when EDMD fails due to closure issues. That is, if the nonlinear system has multiple fixed points or attractors we can get inaccurate dynamics since linear models only contain one such fixed point or attractor. This effect happens because the eigenfunctions are inaccurately projected onto the finite-dimensional subspace. Perhaps, the most important fact to note is the possibility of over-fitting the data with EDMD. This happens when one does not use enough training data on the model and compare it to test data. By neglecting this, a seemingly accurate $\tilde{U}$ can fail to predict future states of the system given novel conditions. Now that we have covered two data-driven methods, DMD and EDMD, to recover the spectrum we will discuss how we can recover the full spectrum in the following subsection.

3.3. A Full Spectral Analysis of the Koopman Operator. The DMD and EDMD algorithms are effective methods of approximating the Koopman operator and corresponding eigenfunctions in a finite-dimensional space. While these two methods are proven to be effective, the continuous spectrum of the Koopman operator is not considered. Recall that

the Koopman operator U is an infinite-dimensional operator. Thus, we cannot assume that it only consists of a point spectrum (i.e. eigenvalues). However, with the DMD and EDMD method we develop finite-dimensional approximations of U which has a spectrum that consists of only eigenvalues. To entirely understand the nonlinear dynamics of the system we need an algorithm to recover the continuous and singular parts of the spectrum of U. For this purpose we turn our attention to the Christoffel-Darboux kernel (CD-kernel) and Christoffel function as discussed in [5]. We should note at the outset that if only the eigenvalues of a dynamical system are of interest, the DMD and EDMD methods will be more efficient than the CD-kernel approach without losing accuracy. The advantage of the CD-kernel method is that we recover the full spectrum as well as it being a mathematically interesting method as it pertains to the spectrum of infinite-dimensional unitary operators.

The CD-kernel method, requires two independent assumptions discussed in further detail in [5]

1. The measure ν is ergodic. That is, the data lie on a *single* trajectory such that $x_{i+1} = F(x_i)$.
2. The samples are independently drawn from the measure ν.

As in the DMD and EDMD algorithms, we begin with snapshots of the observable. However, the goal here is to estimate the first $N+1$ moments from the given data. First, we define the moments of U with respect to the spectral measure as defined in (2.6). The n^{th} moment m_n of the spectral measure μ_f with respect to the observable $f \in \mathcal{H}$ is defined as follows:

$$m_n := \int_{\mathbb{T}} z^n d\mu_f(z), \quad n \in \mathbb{Z} \tag{3.7}$$

Furthermore, m_n satisfies

$$m_n = \langle U^n f, f \rangle \tag{3.8}$$

The importance of (3.8) lies in its use to compute moments from data.

As is the common theme with all data-driven methods, we need to take snapshots of the data. For this method we take snapshots of the observables $f \in \mathcal{H}$ such that

$$y_i = f(x_i), \quad i = 1, ..., N$$

Note the following:

$$m_n = \langle U^n f, f \rangle = \int_{\mathcal{M}} (f \circ F^n)\overline{f} d\nu, \quad k = 0, ..., N \tag{3.9}$$

Recall equation (3.8).

$$m_n = \int_{\mathcal{M}} (f \circ F^n)\overline{f} d\nu = \lim_{N\to\infty} \frac{1}{N-n} \sum_{i=1}^{N-n} (f \circ F^n(x_i))\overline{f(x_i)} \tag{3.10}$$

Using the fact that $y_i = f(x_i)$, we arrive at the following approximation of m_n for large N.

$$m_n \approx \frac{1}{N-n} \sum_{i=1}^{N-n} y_{i+n}\overline{y}_i \tag{3.11}$$

This approximation of m_n can be computed directly from data which is more efficient than using a numerical integration technique to calculate (3.9). The purpose of approximating the moments from data is that we can now recover the spectral measure μ_f.

The overarching problem that we are trying to solve is the moment problem. That is, given the moments of an operator we want to recover the spectral measure associated with the given operator. For the purposes of [5], we have a truncated moment problem. We will show how to recover the point and continuous spectrum from the CD-kernel and Christoffel function.

Definition 1: For each $N \in \mathbb{N}$ and $z, s \in \mathbb{C}$, we define the Christoffel-Darboux kernel as follows:

$$K_N(z,s) = \sum_{i=0}^{N} \overline{\varphi}_i(z)\varphi_i(s)$$

where φ_i's are orthonormal polynomials associated to the spectral measure μ_f.

We can also define the CD-kernel as follows:

$$K_N(z,s) = \psi_N(z)^H M_N^{-1} \psi_N(s) \tag{3.12}$$

where $\psi_N(z) = [1, z, z^2, ..., z^N]^T$, $\psi_N(z)^H$ is the Hermitian transpose of $\psi_N(z)$, and

$$M_N = \int_{\mathbb{T}} \psi_N \psi_N^H = \begin{bmatrix} m_0 & \overline{m}_1 & \overline{m}_2 & \cdots & \cdots & \overline{m}_N \\ m_1 & m_0 & \overline{m}_1 & \ddots & & \overline{m}_{N-1} \\ m_2 & m_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & m_1 & m_0 & \overline{m}_1 \\ m_N & \cdots & \cdots & m_2 & m_1 & m_0 \end{bmatrix} \tag{3.13}$$

is the Hermitian Toeplitz moment matrix. The entries of M_N are populated by the estimated moments from (3.11). Note that the (3.12) depends on the invertibility of M_N. To guarantee this, we construct the CD kernel by the modified moment

$$\tilde{m}_n = \begin{cases} m_n + 1 & \text{if } n = 0 \\ m_n & \text{if } n > 0 \end{cases}$$

So, we replace m_n by $\tilde{m}_n$ such that we define

$$\tilde{K}_N(z,s) = \psi_N(s)\tilde{M}_N^{-1}\psi_N(z) \tag{3.14}$$

Now we define the Christoffel function which is used to recover both the point and continuous spectrum of the Koopman operator. This is done by approximating μ_f and choosing diagonal values $\tilde{K}_N(z,z)$ on the unit circle such that $z = e^{2\pi i\theta}$.

Definition 2: The Christoffel function, $\zeta_N(z_0, z_0)$, is defined as follows for all $z_0 \in \mathbb{C}$ and polynomials $p_N \in \text{span}\{1, z, ..., z^N\}$ of degree at most N:

$$\zeta_N(z_0,z_0) = \frac{1}{\tilde{K}_N(z_0,z_0)} = \min\left\{ \int_{\mathbb{C}} |p_N(z)|^2 d\tilde{m}(z) : p_N(z_0) = 1,\ \deg(p_N) \le N \right\}$$

This definition leads us to the following two theorems.

Theorem 1: Recovery of Point Spectrum

If μ_f is a positive measure on $\mathbb{T}$, then for all $\theta \in [0,1]$

$$\lim_{N\to\infty}\left[\zeta_N(e^{2\pi i\theta}, e^{2\pi i\theta}) - \frac{1}{N+1}\right] = \mu(\{e^{2\pi i\theta}\}) = \mu_{pp}(\{e^{2\pi i\theta}\}) \tag{3.15}$$

Theorem 2: Recovery of Absolutely Continuous Spectrum
If $\mu = \mu_{pp} + \mu_{ac} + \mu_{sc}$ with $d\mu_{ac} = \rho d\theta$ is a positive measure supported on $[0,1]$ and $(m_k)_{k=0}^N$ its Fourier coefficients, then for Lebesgue almost all $\theta \in [0,1]$.

$$\lim_{N\to\infty}\left[(N+1)\zeta_N(e^{2\pi i\theta}, e^{2\pi i\theta}) - 1\right] = \rho(\theta) \tag{3.16}$$

While theorem 1 provides a way of recovering the point spectrum, theorem 2 provides a method of recovering the density ρ. The Christoffel function is at the heart of both of these theorems and the CD-kernel is necessary in order to define the Christoffel function. The reason why this approach is considered a data-driven problem because the moment matrix consists of moments computed from snapshots of our data.

The main focus of this survey, as it pertains the CD-kernel and the Christoffel function, are the theorems of the recovery of point and absolutely continuous spectrum. We refer the reader to [5] for a detailed discussion. These theorems are equivalent to showing that a distribution function $\mu_N([0,t])$ converges pointwise to $\mu([0,t])$ at every point of continuity. The two methods of constructing such a distribution function involve Cesàro sums and Quadrature. We have introduced two theorems that allow for the recovery of the point and absolutely spectrum of U. However, there is still the singular continuous spectrum to consider if we want to fully recover the spectral measure μ_f. The singular continuous part of the spectrum is defined by $\Delta_N(t_k)$ in [5]. The last important result from Korda et al. is that they develop a method of approximating the Koopman operator from spectral projections P_N. The advantage of this approach is that we also retain information on the continuous spectrum while DMD and EDMD lose this information of the spectrum. The one drawback of the methods discussed in this subsection is that it is limited to data from a single trajectory.

3.4. Generalized Laplace Analysis (GLA). As has been extensively discussed in the data-driven techniques in this section so far, calculating the spectrum of the Koopman operator is one of the main goals in Koopman analysis to understand nonlinear dynamical systems. The DMD and EDMD methods provide approximations of the Koopman operator while the CD methods discussed in [5] provide an additional recovery of the continuous spectrum, albeit more computationally intensive. Another method to recover the continuous spectrum is the Generalized Laplace Analysis (GLA) method. In this subsection we provide a brief overview of the GLA method and refer the reader to [8] and [7] for more details into GLA.

The Koopman modes, $v(x)$ are projections of the vector-valued observable $g(x)$ $\varphi_i(x)$ of U at the corresponding eigenvalue λ_i. Recall, that $\varphi_i(x)$ behave linearly with respect to the invariant subspace. We will discuss the following theorem stated in [8]. We denote an arbitrary vector-valued observable $f(x,z)$, where $z \in A \subset \mathbb{R}^n$ and $z \in M$ where M is a manifold. In general, we take M to be a compact metric space, which is our state space.

Theorem 3: Generalized Laplace Analysis
Let $\lambda_0, ..., \lambda_K$ be simple eigenvalues of U such that $|\lambda_0| \geq |\lambda_1| \geq \cdots \geq |\lambda_K|$, and there are no other points $\lambda \in \sigma(U)$ with $|\lambda| \geq |\lambda_K|$. Let φ_k be the k^{th} eigenfunction of U associated

with λ_k, for $k \in \{0, ..., L\}$. Then, the Koopman mode associated with λ_k is obtained by computing

$$\begin{aligned} f_k &= \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \lambda_k^{-i} \left(f(F^i x, z) - \sum_{j=0}^{k-1} \lambda_j^i \varphi_j(x) v_j(z) \right) \\ &= \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \lambda_k^{-i} \left(f^i(z) - \sum_{j=0}^{k-1} \lambda_j^i f_j \right) \end{aligned}$$

where $f_k = \varphi_k(x) v_k(z)$, φ_k is an eigenfunction of U such that $|\varphi_k| = 1$, and v_k is the k^{th} Koopman mode.

As discussed in [11], the GLA approach again computes Koopman modes from snapshots. The Arnoldi algorithm in this data-driven method is used to provide approximations to the Koopman eigenvalues and corresponding modes.

4. Moment Propagation.

4.1. The Koopman Expectation. The Koopman expectation is proposed in [2], where the authors utilize the duality between the Perron-Frobenius and Koopman operators. It is important to recall that the Perron-Frobenius operator, P, acts measure spaces and U acts on observables in a function space. Let f be a pdf and g an observable. Then, we have the following relation between P and U,

$$\langle Pf, g \rangle = \langle f, Ug \rangle \tag{4.1}$$

This is known as the duality relation of P and U.

The key contribution of [2] is that they introduce the relationship of the inner product representation in terms of expectations. We follow their notation and define the following expectations.

$$\mathbb{E}[g(X)|X \sim Pf] = \mathbb{E}[Ug(X)|X \sim f]$$

where

$$\mathbb{E}[g(X)|X \sim Pf] = \int_\Omega [Pf(x)] g(x) dx = \langle Pf, g \rangle$$

and X is a random variable and $\mathbb{E}[\cdot]$ represents the expectation. As discussed earlier, it is easier to calculate the action of U on an observable rather than P on a measure. However, intuitively the latter is more appropriate for understanding the behavior of the pdf as the dynamical system progresses forward in time. Rather than of working with P where we need to find the pre-image of the flow map, we can utilize the duality as expressed in (4.1).

The power of the Koopman expectation is that it allows us to compute higher-order moments with relative ease. For example, if we want to compute the second central moment of a random variable X, then we calculate

$$m_n = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

where $\mathbb{E}[X]$ is the first raw moment. Note that in this method we do not have to explicitly define the Koopman operator nor construct a finite-dimensional approximation as before. While using the Koopman expectation to calculate higher-order central moments is not as direct as above, we can still simplify the process of calculating higher order central moments for a given dynamical system by careful selection of observables.

First, we define the expected value operator as follows:

$$\begin{aligned} m_n &= \mathbb{E}\big[(X - \mu(X))^n\big] \\ &= \mathbb{E}\left[\sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} \mu(X)^{n-k} X^k\right] \\ &= \sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} \mu(X)^{n-k} \mathbb{E}[X^k] \end{aligned}$$

where we utilize the linearity property of the expected value operator and the fact that the mean $\mu(X)$ is a constant. Thus, in order to calculate a higher order moment, we need to be able to calculate $\mu(X)$, which is the first raw moment, and $\mathbb{E}[X^k]$. To relate this to the Koopman expectation and the Koopman approach of dynamical systems with observables, we are interested in observables $g(X)$ rather than random variables X. As describe in [2], first define define a vector-valued observable

$$\tilde{g}(x) = [g(x), g(x)^2, ...g(x)^n]^T$$

and then we can express the Koopman expectation as follows:

$$\mathbb{E}\big[\tilde{g}(F(x,\alpha))|f\big] = \int_{\Omega} \tilde{g}(F(x,\alpha)) f(\alpha) d\alpha$$

where α is a parameter where we have uncertainty and Ω is the domain of integration for which the uncertainty is contained within.

As we can see, the calculation of higher-order moments with the Koopman expectation condenses to a problem of integration. When solving a data-driven problem, the difficulty lies in accurately approximating the integral such that we can accurately calculate the n^{th} central moment with respect to the given observable. In [2], an example is done to show how the Koopman expectation is used to calculate the expected value of a dynamical system along with higher-order moments. Note that for most applications, the first four moments are of concern since they contain statistical importance. i.e. the second central moment is the variance of the data. Furthermore, this example compares the Koopman expectation to the Monte-Carlo method and a significant speedup occurs when working with the Koopman expectation. The main difficulty with the Koopman expectation is the selection of observables when one wants to calculate higher-order central moments. However, the benefits of this method is that we never have to explicitly define the Koopman operator or construct a finite-dimensional approximation of the Koopman operator such as with the DMD and EDMD methods.

4.2. Koopman Matrix Moment Propagation. Another method to quantify uncertainty in a dynamical system using the Koopman operator is by the propagation of moments as discussed in [6]. The authors study a nonlinear dynamical system and develop the technique to propagate the moments by the approximate Koopman operator, which we will denote by $\tilde{U}$. From the propagation of moments, we are able to reproduce the resulting pdf

at a future time. In order to propagate the moments of the system, the EDMD method is used to arrive at an approximate Koopman matrix. We provide a brief summary and implications of [6] and refer the reader to this paper for further reading if needed.

In this method, we make use of EDMD to construct the Koopman matrix $\tilde{U}$ that will be used to propagate the moments m_t forward in time. That is,

$$\tilde{U}m_t = m_{t+1} \tag{4.2}$$

Note that $m_t \in \mathbb{R}^K$ and $\tilde{U} \in \mathbb{R}^{K\times K}$, where K represents the number of basis functions that compose of Ψ. E.g., if Ψ is a degree-two polynomial in a dynamical system with two variables then $K = 5$ if we ignore the polynomial equal to 1.

Define $g_t(x) = \Psi(x)^T w_t$ where $w_t \in \mathbb{R}^K$. g_t represents the projection of the current observable at time t onto the basis function $\Psi(x)^T$ with the weight vector w_t. From this, we define the moment at an arbitrary time t

$$m_t = \langle \varphi_k, g_t \rangle \tag{4.3}$$

Note that the inner product is an inner product in the Hilbert space which we assume is L^2 with respect to a measure. Thus, we have

$$m_t^k = \int_X f_0(x)\varphi_k(x)dx = \langle \varphi_k(x), f_0(x) \rangle, \quad \text{for } k = 1, ..., K$$

where $f_0(x)$ is a pdf and $\Psi(x) = [\varphi_1(x), ..., \varphi_K(x)]$. Furthermore, m_t^k denotes the moment at time t corresponding to the k^th component of the dictionary function $\Psi(x)$.

In equation (4.2) we see how to propagate the moment m_t to the moment in the next time stamp at time m_{t+1}. What we need to understand is how to calculate m_t. We can look at equation (4.3), but there is a more succinct way of expressing m_t that the authors of [6] mention. That is,

$$m_t = \Lambda w_t$$

where $m_t, w_t \in \mathbb{R}^K$ and $\Lambda = \Psi(x)\Psi(x)^T$. Now, we show the implementation of this method of propagating moments.

5. Numerical Examples.

5.1. Analytical Koopman Operator. The following example taken from [1] demonstrates a case where the Koopman operator can be determined analytically and in finite dimensional space. We refer the reader to the original paper for a deeper analysis on the spectrum of the finite dimensional approximation of the Koopman operator. The nonlinear dynamical system we will study is as follows:

$$\dot{x}_1 = \mu x_1$$

$$\dot{x}_2 = \lambda(x_2 - x_1^2)$$

where we choose $\mu = .05$ and $\lambda = -1$. Note that when $\lambda < \mu < 0$, the system shows a slow attracting manifold given by $x_2 = x_1^2$. For a further discussion on the slow attracting manifold, we refer the reader to [1]. This is the key observation to make in order to define new coordinates such that the dynamics can be expressed as a linear system. We choose the observables (x_1, x_2, x_1^2) and thus can redefine the dynamical system as a linear system:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

where

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix}$$

are the "new" variables that we use in order to represent the problem as a linear dynamical system that fully captures the nonlinear behavior of the original system in state space, see Figure 5.1

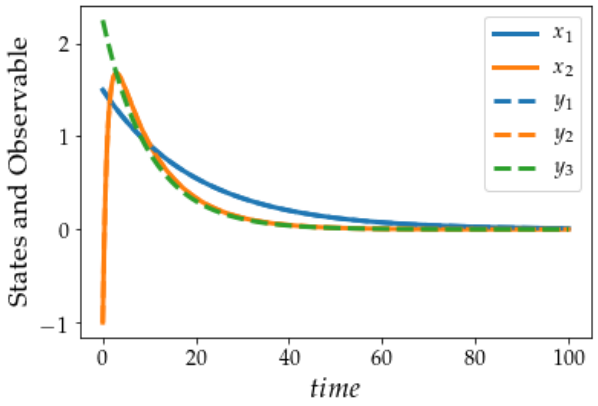


Fig. 5.1: Demonstration of an analytically determined Koopman operator for a simple nonlinear system

In general, the appropriate lifting function is considerably more difficult to discover. This analytical example is intended to demonstrate the intuition behind Koopman analysis.

5.2. DMD and EDMD. We can make use of the same example to demonstrate important performance differences between DMD and EDMD. A straightforward application of DMD to the previous example (recall that DMD does not make use of a lifting function) find a system that unsurprisingly does not capture well the nonlinearity in x_2, see Figure 5.2a. In contrast, the EDMD method is able to capture the nonlinearity using a second order polynomial dictionary, see Figure 5.2b.

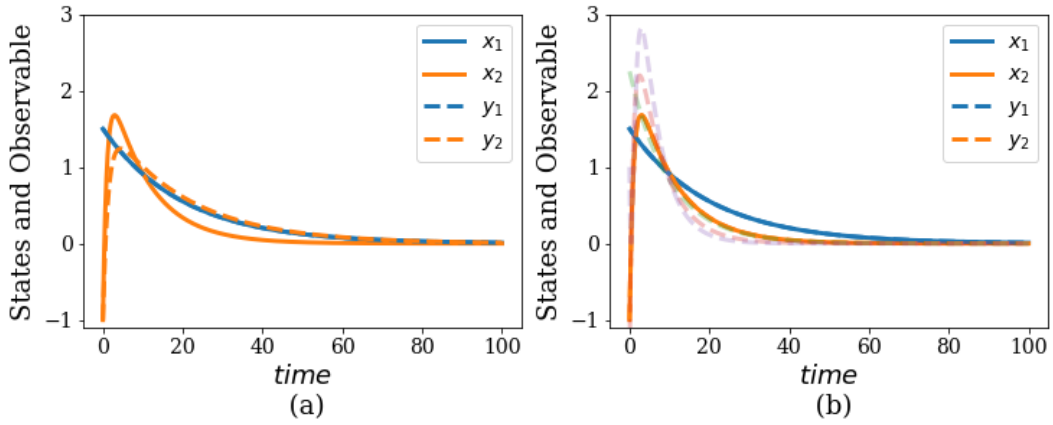


Fig. 5.2: Results from the application of (a) DMD and (b) EDMD with a second order polynomial to the example from the previous section. The higher order measurements are transparent to more easily see the states corresponding to x_1, and x_2

With the Koopman approximation found using EDMD, we can vary the initial conditions to graph the evolution of the variables x_1 and x_2. The three initial conditions (distinct from the training data, which is a single trajectory) that we chose are as follows:

$$\mathrm{IC}_1 = (1.5, -1), \quad \mathrm{IC}_2 = (1, -1), \quad \mathrm{IC}_3 = (2, -2)$$

The results are depicted in Figure 5.3. The solid lines in the figure above represent the solution of the nonlinear dynamical system using an ode solver. The dotted lines are the data points created from the EDMD algorithm to show us the future states of the system. As we can see, any error that exists is negligible and the EDMD algorithm provides an accurate approximation of the evolution of this simple nonlinear dynamical system for which we know we are using an appropriate dictionary.

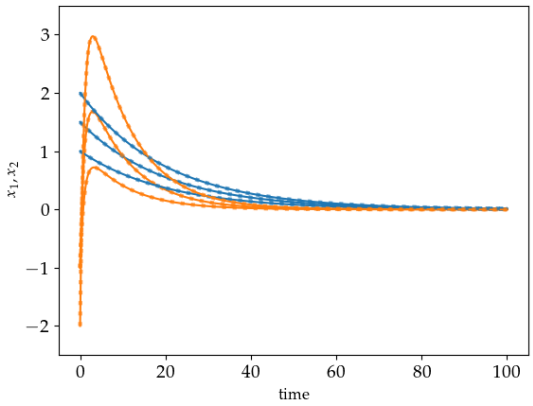


Fig. 5.3: Results from simulations at three initial conditions distinct from the training data. The figure shows the results in the original state space rather than the space of observables.

5.3. EDMD with Inputs. An extension to DMD to incorporate the effects of a control input is derived in [10]. Consider a discrete system such that

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$

for which we'll record snapshots as usual. We'll additionally record snapshots of the input function

$$\Upsilon = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_{m-1} \\ | & | & & | \end{bmatrix}$$

Such that

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X} + \mathbf{B}\boldsymbol{\Upsilon}$$

We can solve for A and B as

$$\mathbf{G} = [\mathbf{A}\ \mathbf{B}] = \mathbf{X}' \begin{bmatrix} \mathbf{X} \\ \Upsilon \end{bmatrix}^{\dagger}$$

or the projection

$$\mathbf{G} = \mathbf{X}'\tilde{\mathbf{V}}\Sigma^{-1}\tilde{\mathbf{U}}^*$$

And finally, the dynamic modes are

$$\Phi = \mathbf{X}'\tilde{\mathbf{V}}\tilde{\boldsymbol{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\hat{\mathbf{U}}\mathbf{w}$$

Such an extension can be applied readily to EDMD as well. To demonstrate, we consider another simple nonlinear system from "Model Order Reduction of Nonlinear Dynamical Systems" by Chenjie Gu [3] with a sinusoidal input.

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ x_1^2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u(t) \tag{5.1}$$

The results in Figure 5.4 demonstrate the performance of the EDMD method on this nonlinear system. The left most figure depicts the training data while the middle and right figures compare the predicted response to "test" inputs, namely a more complex sinusoidal input and a step input. Again, note that the EDMD results are depicted in the original state space rather than the space of observables.

5.4. Koopman Expectation Example. Here, we include an example of using the Koopman expectation to understand higher order moments. The example that we choose to show the propagation of moments is example 1 from [6]. We have a 2−D system governed by the following nonlinear system.

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{3}{2}x_1 - x_2 + \frac{x_2^3}{9} \end{aligned}$$

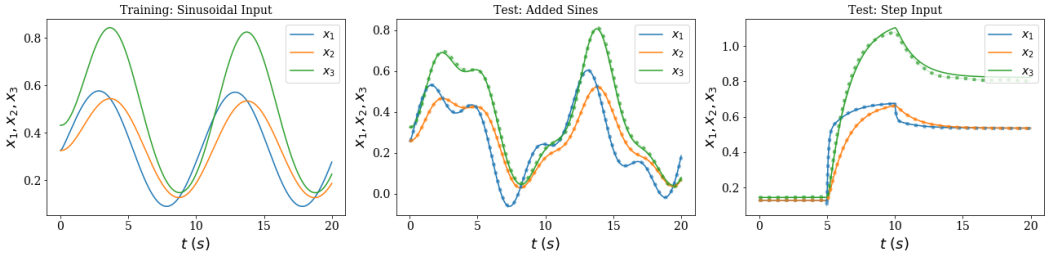
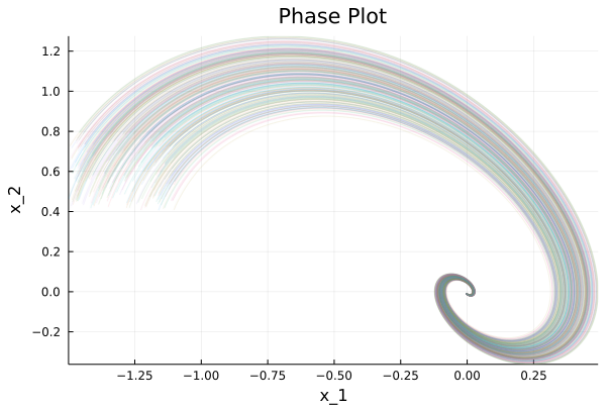
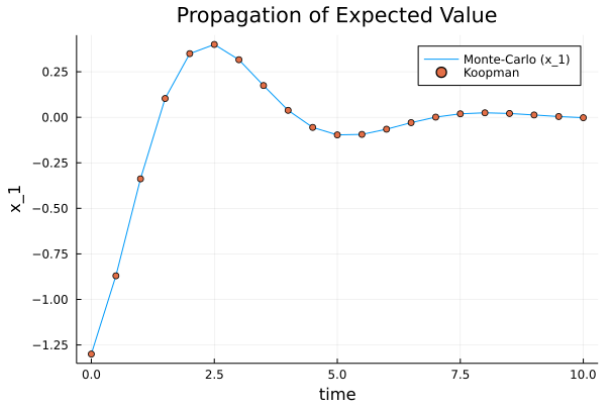


Fig. 5.4: The training data with a sinusoidal input is depicted in the left most figure. The middle and right figures compare the responses to varying inputs. The results of the EDMD simulation as dotted lines

We incorporate uncertainty in the system by allowing the initial conditions of the system to be a uniform random distribution in a region of interest. For our study we choose this region of interest to be the square $(-1.5, -1.1) \times (0.5, 0.8)$. The phase plot is shown below:



Using the Koopman expectation as described in [2], we can propagate higher-order moments and understand higher-order statistics at an arbitrary time in the future. We show the propagation of the expected value (first raw moment) in the following figure:

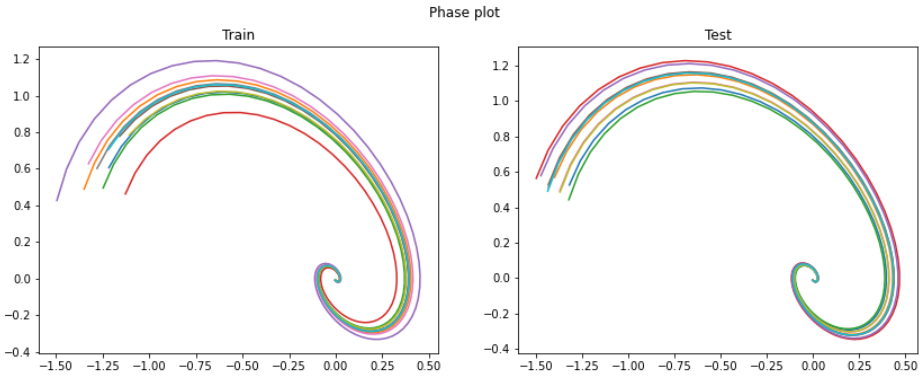


Here, we compare the Koopman method to the Monte-Carlo. We take snapshots every half second from zero to ten seconds. While we know the Monte-Carlo method works in propagating moments, it can suffer in the time it takes to run. While the dynamical system we consider in this example is small and straightforward, we can already see a dramatic speedup. It takes the Koopman expectation 1.57 seconds while it takes Monte Carlo with $100,000$ trajectories 2.55 seconds. When we change the snapshots to every 0.1 seconds, the speedup of the Koopman expectation is more apparent due to more data being captured. Here, Monte-Carlo takes 17.38 seconds while the Koopman expectation takes 2.69 seconds and is still an accurate approximation of the expected value. Another example, with more detail, is explained in [2] where they work with a more complicated dynamical system and show the accuracy of calculating higher-order moments compared with the Monte-Carlo method. With the Koopman expectation method shown, we will show another method of propagating moments which we refer to as the Koopman matrix moment propagation.

5.4.1. Koopman Matrix Moment Propagation Example. We will use the same example as shown for the Koopman expectation. This is to show that both methods are effective in propagating moments. Recall that we have a $2-$D system governed by the following nonlinear system.

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= -\frac{3}{2}x_1 - x_2 + \frac{x_2^3}{9}\end{aligned}$$

The region of interest where our uncertainty lives is the square $(-1.5,-1.1)\times(0.5,0.8)$. The phase plots of the training and test data is shown below. We can see that there is an equilibrium point at the origin.

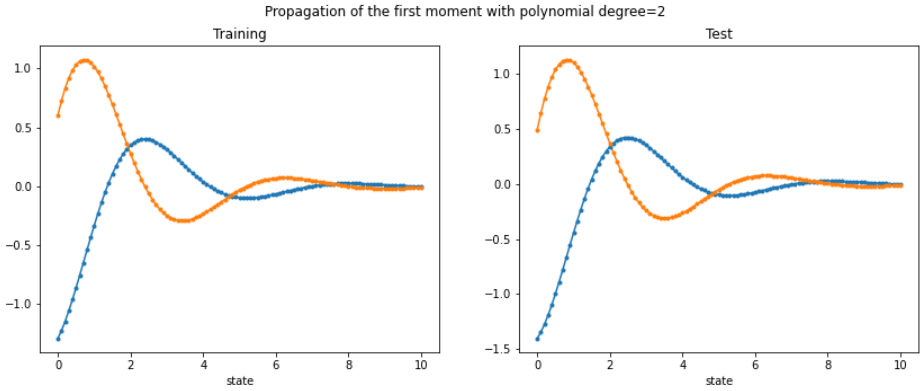


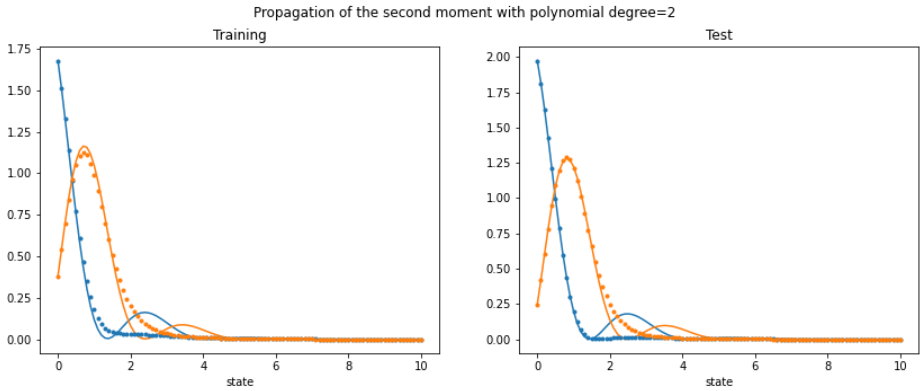
Following the method discussed in [6], we construct the Koopman matrix $\tilde{U}$, by the EDMD algorithm, that propagates the moments forward in time. i.e. $m_{t+1} = \tilde{U}m_t$. First, we choose the dictionary function to be polynomials of degree 2 with a scaling factor of 3.

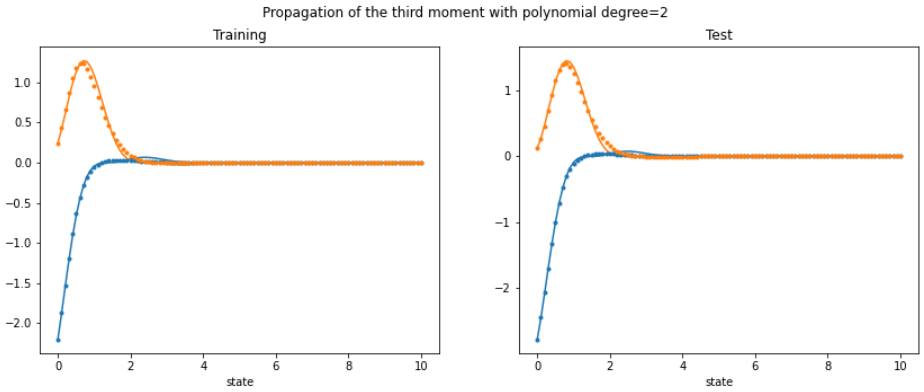
$$\Phi(x) = \left[\frac{x_1}{3}, \frac{x_2}{3}, \frac{x_1^2}{9}, \frac{x_1x_2}{9}, \frac{x_2^2}{9}\right]^T$$

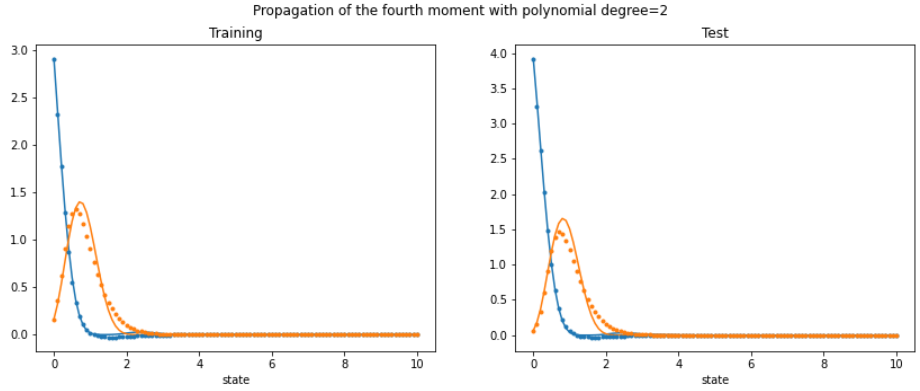
We choose 500 initial points in a uniform random distribution so that we can construct trajectories of the system for our training and test data. From here, we arrive at the

propagation of the first four moments, which provide statistical insight into the dynamical system.

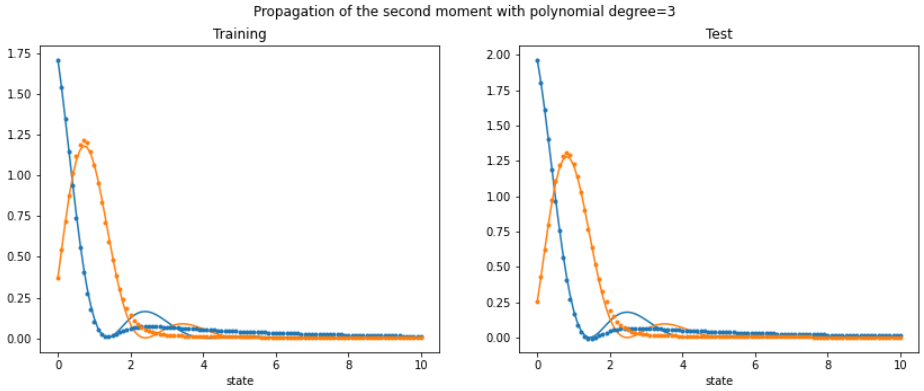








It can be seen that for the propagation of the first moment, the propagation of moments by the Koopman matrix is a near exact match to that of the Monte-Carlo method. The advantage of working with the Koopman matrix to propagate moments is that the computation involved is matrix multiplication, which is more efficient than the Monte-Carlo method. The reader may note that for the propagation of higher moments, specifically the second moment, the accuracy does not match as well. To solve this problem, we can use a higher order polynomial for our dictionary function. If we choose the degree to equal 3, then we get the following:



While there is still noticeable error, we have decreased the error compared to the previous dictionary function. In theory, we can increase the degree of the polynomial for the dictionary function to increase the accuracy. However, by doing this we increase the computation time because with an increase in the degree we have an increase in computations needed to construct the Koopman matrix $\tilde{U}$. For more complex dynamical systems, choosing polynomials for our dictionary function may not be accurate. A common dictionary function to choose when this is the case is the radial basis functions (RBFs). This is shown in example 3 in [6].

Here, it may be useful to directly compare the Koopman matrix propagation approach to the Koopman Expectation approach of the previous section. For this, we consider a phase plot of the first three moments in Figure 5.5. For this example, the Koopman expectation outperforms the Koopman Matrix propagation approach in terms of computational expense as well as accuracy.

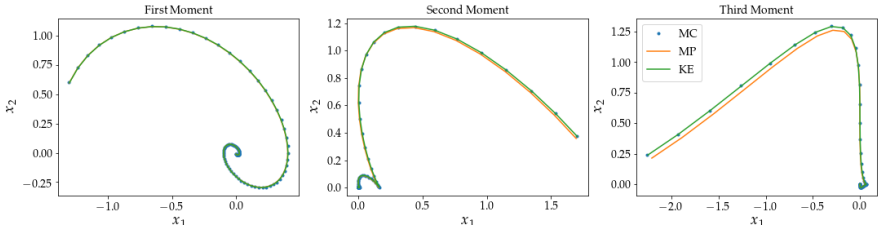


Fig. 5.5: Comparison of Koopman Matrix propagation (MP) approach and Koopman Expectation (KE) and MonteCarlo (MC). The figures show a phase plot of the first three moments of the system.

6. Conclusion. In this survey paper, the mathematical foundations of the Koopman operator are presented as it pertains to nonlinear dynamical systems. Alongside with the mathematical theory, numerical examples implementing the methods of DMD, EDMD, moment propagation and the Koopman Expectation are demonstrated and compared to a simple Monte Carlo method. To understand the power of these numerical methods, the mathematical tools involved are discussed. The methods of DMD and EDMD rely heavily on constructing an accurate finite dimensional approximation of the unbounded Koopman operator. This is done by creating a large data matrix from snapshots of the data and then implementing a least squares approach and the use of SVD. After obtaining such an approximation, the recovery of the eigenvalues, eigenfunction, and modes of the dynamical system is more accessible. The main difference between DMD and EDMD is that EDMD implements the use of dictionary (basis) functions to lift from the state space to the observable space. While EDMD is more accurate, it is not always the most desirable method since there are instances where it fails. Furthermore, it is often a difficult task to choose the correct dictionary functions. By approximating the Koopman operator as a finite dimensional matrix through the use of DMD or EDMD, one can understand and apply a proper analysis of a nonlinear dynamical system.

The numerical methods of moment propagation [6] and the Koopman Expectation [2] are useful tools in understanding how higher order statistics propagate through a dynamical system. The numerical method propagates the n^{th} moment, e.g., the first moment is expected value, the second moment is variance, the third moment is skewness, and the fourth moment is kurtosis, and so on. The moment propagation method utilizes EDMD to construct a matrix approximation of the Koopman operator along with constructing a vector that represents the n^{th} moment at a given time t. Then, basic matrix multiplication of the matrix and vector result in the n^{th} moment at the next time stamp. The Koopman Expectation satisfies a similar goal in that the authors develop a technique to calculate the n^{th} moment by the use of expectations. Following the representation of the expectation as a Lebesgue integral, we are able to calculate higher order moments. The difficulty in this approach is that we can encounter problems with accurately approximating the integral. Despite the fact that both the moment propagation and Koopman Expectation approaches are distinct methods, they both rely on the use of the Koopman operator. The moment propagation method utilizes an approximation of the Koopman operator while the Koopman Expectation uses a more theoretical use of the unbounded Koopman operator to set up the mathematical definition of a moment represented as a Lebesgue integral.

While we have discussed and provided examples of the numerical methods used to understand nonlinear dynamical systems, it can be useful to understand the Koopman operator at a more theoretical level. In [7], recovering the entirety of the spectrum of

the Koopman Operator is discussed in detail. In the numerical methods such as DMD, EDMD, and the moment propagation methods, we only recover the point spectrum (i.e. eigenvalues) of the finite dimensional approximation of the Koopman operator. Since the Koopman operator is unbounded, we have the spectral decomposition into the pure point, absolutely continuous and singular continuous spectrum. These parts of the spectrum also have interesting mathematical and physical implications. The numerical methods used to recover the full spectrum of the Koopman operator are the CD-kernel and the Christoffel function. To do so, one must construct a distribution related to the spectral measure. The methods of Cesàro sums and Quadrature are used to construct such a distribution function. In our survey paper, we do not go into detail on these methods and refer the reader to [7] for further reading.

Throughout this survey paper, the main focus has been on discussing the mathematical and numerical methods known to accurately and efficiently understand nonlinear dynamical systems with the use of the Koopman operator. It is important to note that we have only focused on a small subset of the applications of the Koopman operator. Thus, we refer the reader to the references provided for further applications and details that were omitted in this survey.

References.

[1] Steven L. Brunton, *Modern koopman theory for dynamical systems* (2021).
[2] Adam R. Gerlach, *The koopman expectation: An operator theoretic method for efficient analysis and optimization of uncertain hybrid dynamical systems* (2020), available at 2008.08737.
[3] Chenjie Gu, *Model order reduction of nonlinear dynamical systems*, University of California, Berkeley, 2011.
[4] B.O. Koopman, *Hamiltonian systems and transformation in hilbert space*, Proceedings of the National Academy of Sciences (1931).
[5] Milan Korda, *Data-driven spectral analysis of the koopman operator*, Vol. 48, 2020.
[6] Amarsagar Reddy Ramapuram Matavalam, *Data-driven approach for uncertainty propagation and reachability analysis in dynamical systems* (2020).
[7] Igor Mezić, *Spectrum of the koopman operator, spectral expansions in functional spaces and state-space geometry*, Journal of Nonlinear Science (2019), pp. 1–55.
[8] ———, *On numerical approximations of the koopman operator* (2020).
[9] Calvin C Moore, *Ergodic theorem, ergodic theory, and statistical mechanics*, Proceedings of the National Academy of Sciences **112** (2015), no. 7, 1907–1911.
[10] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz, *Dynamic mode decomposition with control*, SIAM Journal on Applied Dynamical Systems **15** (2016), no. 1, 142–161.
[11] Clarence W. Rowley, *Spectral analysis of nonlinear flows*, Journal of Fluid Mechanics (2009).
[12] Peter Schmid, *Dynamic mode decomposition of numerical and experimental data*, Journal of Fluid Mechanics (2010), pp. 5–28.
[13] Matthew O. Williams, *A data-driven approximation of the koopman operator: Extending dynamic mode decomposition*, Journal of Nonlinear Science (2014).

MAJORIZE-MINIMIZE ALGORITHMS FOR GENERALIZED CANONICAL POLYADIC TENSOR DECOMPOSITIONS

KYLE GILMAN* AND ERIC PHIPPS †

Abstract. In this work, we propose accelerated majorization-minimization (MM) algorithms for computing the generalized canonical polyadic (GCP) tensor decomposition from streaming multiway data. Our algorithms enjoy first-order computational complexity, rapidly converge to a minimizer of each subproblem in the GCP alternating schedule, and require no step-size hyperparameters to tune. We demonstrate our algorithms have superior performance compared to stochastic gradient descent with ADAM for a variety of nonnegative CP tensor factorization problems under non-Euclidean statistical losses. The success of these algorithms permits more efficient and rapid computation in large-scale tensor problems.

1. Introduction. Multiway data analysis has flourished in the era of modern data as data are increasingly collected over multiple modalities into multidimensional arrays called tensors, from functional MRI data [15] to plant-pollinator data [6]. This tensor data often exhibits low-dimensional latent structure describing the multilinear relationships between its modes, which can be leveraged for exploratory data analysis, missing data imputation, anomaly detection, data compression, and more. Hence, researchers have turned their focus to tensor decompositions and generalizations of principal component analysis (PCA) to higher-order data structures, the foremost methods being the Canonical Polyadic decomposition (CPD) and Tucker decomposition. Recently, the CPD was extended beyond least squares problems to fit models under losses of various statistical distributions called Generalized CP (GCP) or Non-Euclidean CP [10, 12, 17]. Since many real data are non-Gaussian, like sparse count data arising in computer network traffic monitoring [20], these tensor factorizations find great utility in practical modern applications.

Despite the great progress in GCP, these methods require significant tuning of problem-dependent hyperparameters like step sizes–nontrivial challenges especially in large data settings where hyperparameter tuning itself is costly–and the number of iterations and samplings of the tensor can be quite large in order to converge. In addition, tensor optimizers usually require stochastic methods in order to subsample the data and to avoid forming and storing the large matrices that arise in the original problem. Little work has gone towards leveraging the geometry of these problems and stochastic methods to more efficiently factor tensors under generalized losses.

Many GCP problem seek nonnegative factorizations to recover interpretable factors in practical applications, and the statistical losses are either convex or a difference of convex functions. We utilize these key properties to develop accelerated majorize-minimize algorithms for nonnegative generalized tensor factorization problems in both the batch and streaming settings. Our algorithms rapidly converge in fewer iterations than GCP with ADAM, and with similar first-order complexity per iteration. Importantly, our methods require no hyperparameters like step sizes, and our methods can be easily combined with stochastic sampling techniques for sparse tensor data. Extensive empirical results support that MM methods are more robust and efficient at recovering the tensor factors.

*University of Michigan, kgilman@umich.edu
†Sandia National Laboratories, etphipp@sandia.gov

1.1. Organization of this paper.

- Section 3 details related work in tensor decompositions for general losses and MM algorithms for nonnegative matrix and tensor factorization.
- Section 4 proposes our GCP tensor decomposition optimization algorithms using MM methods for statistically-motivated losses. We discuss the various root-solving strategies for obtaining the minimizers of the constructed surrogate functions.
- Section 5: Experimental results are given for synthetic data showing significant speedup of factor recovery in streaming GCP iterations compared to state-of-the-art ADAM optimizers.
- Section 6: We conclude the paper and outline next steps to finish the results as well as possible future directions.

2. Notation & Background. We shall denote all scalar quantities as s, vectors as $\boldsymbol{v}$, matrices as $\boldsymbol{A}$, and tensors as $\boldsymbol{\mathcal{X}}$. The (complex conjugate) transpose of a matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}'$. We denote the Frobenius norm of a matrix as $\|\boldsymbol{A}\|_F := \sqrt{\sum_{ij} |\boldsymbol{A}_{ij}|^2}$. The slice/hyperslice of $\boldsymbol{\mathcal{X}}$ at time t is denoted as $\boldsymbol{\mathcal{X}}_t := \boldsymbol{\mathcal{X}}[:, \ldots, :, t]$, where will assume in this paper that time is the last tensor dimension. The n-mode unfolding of a N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ into a $I_n \times \Pi_{k \neq n}^{d} I_k$ matrix is written as $\boldsymbol{X}^{(n)}$.

The Kronecker product is denoted as $\otimes$, the Khatri-Rao product as $\odot$, and $\circ$ as the outer product. The mode-n product of a tensor $\boldsymbol{\mathcal{X}}$ with matrix $\boldsymbol{A}$ is denoted as $(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A})^{(n)} = \boldsymbol{A}\boldsymbol{X}^{(n)}$. Refer to [11] for more on these products and their properties and identities.

We use the notation in [11] to express a N-way rank-r CP model as

$$\boldsymbol{\mathcal{M}} \in \mathbb{R}^{I_1 \times \cdots \times I_N} = [\![\boldsymbol{\lambda}, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!] = \sum_{j=1}^{r} \lambda_j \boldsymbol{a}_j^{(1)} \circ \cdots \circ \boldsymbol{a}_j^{(N)} \tag{2.1}$$

for weights $\boldsymbol{\lambda} \in \mathbb{R}^r$ and factors $\{\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times r}\}_{n=1}^{N}$ where $[\boldsymbol{a}_1^{(n)} \cdots \boldsymbol{a}_r^{(n)}]$ are the unit-norm column vectors of the n^{th} factor matrix $\boldsymbol{A}^{(n)}$. With unit-norm factors, $\boldsymbol{\lambda}$ contains the column norms, i.e. $\lambda_j = \|\boldsymbol{a}_j^{(1)}\|_2 \|\boldsymbol{a}_j^{(2)}\|_2 \cdots \|\boldsymbol{a}_j^{(N)}\|_2$; otherwise, we can also just express the CP factorization in terms of the (unnormalized) factors and omit $\boldsymbol{\lambda}$. For a 3-way CP tensor $\boldsymbol{\mathcal{X}}$ with factors $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, Fig. 2.1 illustrates the CP decomposition in the sum-of-outer-products form.

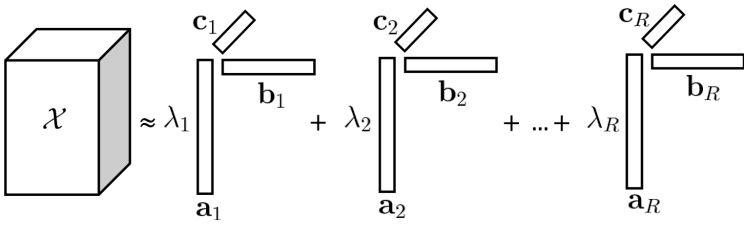


Fig. 2.1: CP Decomposition of $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.

3. Related Work. The work in [4] developed an MM algorithm for sparse count tensor data; in fact, their work is a special case of ours. After originally proposing GCP, Kolda & Hong [12] then developed stochastic gradient sampling techniques with ADAM for solving the GCP problem, including stratified sampling that significantly improves stochastic gradient methods on sparse tensor data. Their SGD method with ADAM draws data samples from the tensor $\boldsymbol{\mathcal{X}}$ either uniformly at random or by their stratified sampling technique and estimates the gradients $\boldsymbol{\mathcal{G}}_t^{(n)}$ in each mode n from current factor estimates $\boldsymbol{A}_t^{(1)}, \ldots, \boldsymbol{A}_t^{(N)}$. Then they jointly update the factors simultaneously at each iteration for each mode, i.e.

$$\boldsymbol{A}_{t+1}^{(n)} \leftarrow \boldsymbol{A}_t^{(n)} - \alpha \tilde{\boldsymbol{\mathcal{G}}}_t^{(n)} \quad n = 1, \ldots, d,$$

where $\tilde{\boldsymbol{\mathcal{G}}}_t^{(n)} = \boldsymbol{\mathcal{M}}_t^{(n)} / ((\sqrt{\boldsymbol{\mathcal{V}}_t^{(n)}} + \epsilon)$, $\boldsymbol{\mathcal{M}}_t^{(n)}$ and $\boldsymbol{\mathcal{V}}_t^{(n)}$ are the momentum terms in ADAM computed from the gradients $\boldsymbol{\mathcal{G}}_t^{(n)}$, $\alpha > 0$ is a step size, and arithmetic operations are elementwise.

Pu et al. [17] then proposed SmartCPD using stochastic mirror descent for nonnegative GCP with fiber sampling, which becomes an upper bounding MM surrogate given a certain choice of Bregman divergence and step size. However, the authors do not consider the inclusion of quadratic regularizers, which no longer make for closed-form solutions to the factor updates without becoming standard stochastic gradient descent. While they show fiber sampling performs well, in practice, sampling an entire tensor fiber may be cumbersome for large dense data, or in cases where the tensor entries are sparse or even missing, we wish to selectively sample only certain entries. In the work herein, we will propose tight surrogates for the streaming GCP objective that is augmented with regularization terms, and also discuss further exploiting the structure of sparse and dense tensors to more efficiently subsample the data.

The works in [7, 9, 16, 19] develop multiplicative update MM algorithms for nonnegative matrix factorization with α-,β-, generalized KL, IS, and other divergences, but they do not consider the Rayleigh, Bernoulli, or Negative Binomial losses, nor any losses with a quadratic regularization penalty like in this work, which makes solving for the roots of the resulting polynomials non-trivial. Zhao & Tan [21] unifed the large variety of MM surrogates for nonnegative matrix factorization to classes of functions represented by h-divergences, including the addition of ℓ_1 and ℓ_2 regularization terms. While their surrogates are flexible to many different types of losses, they are not necessarily tight upper-bounding surrogates. The work in [5] discusses multiplicative updates for difference of convex losses in generalized tensor factorizations, but the methods therein are heuristics which are not guaranteed to converge and do not necessarily form upper-bounding surrogates.

4. Methods.

4.1. Objectives.

GCP. For N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}_+^{I_1 \times \cdots \times I_N}$, we seek the rank-$r$ nonnegative GCP factorization for factors $\{\boldsymbol{A}^{(n)}\}_{n=1}^N \in \mathbb{R}_+^{I_n \times r}$ under loss $\ell(\cdot) : \mathbb{R}_+ \mapsto \mathbb{R}$:

$$L(\boldsymbol{\mathcal{X}}, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}) = \sum_{i=1}^{D} \ell(x_i, m_i) + \frac{\lambda}{2} \sum_{n=1}^{N} \|\boldsymbol{A}^{(n)}\|_F^2, \quad \lambda > 0. \tag{4.1}$$

Here, x_i is the i^{th} linear index of the tensor data, and m_i is the i^{th} index of the GCP model $\boldsymbol{\mathcal{M}} := [\![\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!]$. The loss $\ell(\cdot)$ usually represents the negative log-likelihood of the data parameterized by the CP factors in $\boldsymbol{\mathcal{M}}$. The Tikhonov regularizers with regularization parameter λ encourage low-rank factors and prevent the entries of factors from becoming too large.

Streaming GCP. Computing the factorization over the batch tensor may be memory-prohibitive or the data may be streaming in where we wish to update the model with every new (hyper)slice. Given tensor slice $\boldsymbol{\mathcal{X}}_t \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, our goal is to compute a GCP factorization for *the entire tensor* in a streaming way over T slices, i.e., we wish to factorize $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times T}$, but without revisiting or storing previous data. Each time a new data slice/hyperslice $\boldsymbol{\mathcal{X}}_t$ arrives, we optimize the following problem for the non-temporal factors $\{\boldsymbol{A}^{(n)}\}_{n=1}^N$ and the temporal weights $\boldsymbol{s}_t \in \mathbb{R}^r$, which are the rows of the temporal factor matrix $\boldsymbol{S} \in \mathbb{R}^{T \times r}$ that is estimated sequentially in time:

$$\begin{aligned} L(\boldsymbol{\mathcal{X}}_t, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}, \boldsymbol{s}_t) = \sum_{i=1}^{D} \{\ell(x_{it}, m_{it}) + \sum_{h \in \mathcal{H}_t} w_h \|\bar{m}_{ih} - m_{ih}\|_2^2\} \\ + \frac{\lambda}{2} \sum_{n=1}^{N} \|\boldsymbol{A}^{(n)}\|_F^2 + \frac{\lambda}{2} \|\boldsymbol{s}_t\|_2^2, \end{aligned} \tag{4.2}$$

where x_{it} is the i^{th} entry of the data slice/hyperslice $\boldsymbol{\mathcal{X}}_t$, $\bar{m}_{ih}$ is the i^{th} entry of the h^{th} historical model $\bar{\boldsymbol{\mathcal{M}}}_h = [\![\bar{\boldsymbol{A}}^{(1)}, \ldots, \bar{\boldsymbol{A}}^{(N)}, \boldsymbol{s}_h]\!]$ with previous estimates $\bar{\boldsymbol{A}}^{(1)}, \ldots, \bar{\boldsymbol{A}}^{(N)}$, and m_{ih} is the i^{th} entry of the h^{th} model $\boldsymbol{\mathcal{M}}_h = [\![\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}, \boldsymbol{s}_h]\!]$. The historical regularizer sums a weighted combination of least squares losses with weights $w_h \geq 0$ to penalize the model differences across time; this encourages fitting CP factors over the global tensor rather than the most recent data like in subspace tracking algorithms and acts as a "forgetting factor" to downweight older data. A memory bank of previously estimated GCP models $\bar{\boldsymbol{\mathcal{M}}}_h$ is stored in the set $\mathcal{H}_t$, where the set membership is updated in some way to include the most recently observed data and omit older data. The works in [1, 18] proposed a similar objective function for fitting CP models with the Gaussian log-likelihood–that is, a least squares objective. In that setting, the joint least squares problem permits efficient ADMM algorithms with recursive updates of the temporal memory. The same does not apply here, however, because of the mismatch between the loss $\ell(\cdot)$ and the least squares regularizer.

4.2. Problem for $\boldsymbol{A}^{(n)}$. The GCP problem is nonconvex between the factors due to their coupling, and algorithms can generally only be guaranteed to converge to a local minimizer. Consequently, tensor factorization optimizers typically resort to block coordinate descent or minimization in each factor $\boldsymbol{A}^{(n)}$. The general optimization problem for $\boldsymbol{A}^{(n)}$, or equivalently $\boldsymbol{a} := \text{vec}(\boldsymbol{A}^{(n)'}) \in \mathbb{R}^{I_n r}$, with all other factors and $\boldsymbol{s}_t$ held fixed can be recast as

$$\min_{\boldsymbol{a} \in \mathbb{R}_+^{I_n r}} \sum_{i=1}^{D} \{\ell(x_i, [\boldsymbol{\Phi}_t \boldsymbol{a}]_i)\} + r_{\mathcal{H}}(\boldsymbol{a}) + \frac{\lambda}{2} \|\boldsymbol{a}\|_2^2, \tag{4.3}$$

where $\boldsymbol{\Phi}_t = \boldsymbol{I}_{I_n} \otimes (\boldsymbol{Z}^{(n)} \odot \boldsymbol{s}_t)$. Here $\boldsymbol{Z}^{(n)} = \boldsymbol{A}^{(1)} \odot \cdots \odot \boldsymbol{A}^{(n-1)} \odot \boldsymbol{A}^{(n+1)} \odot \cdots \odot \boldsymbol{A}^{(N)}$ represents the Khatri-Rao product matrix of the other factors held fixed, and

$$r_{\mathcal{H}}(\boldsymbol{a}) := \sum_{h \in \mathcal{H}_t} \frac{w_h}{2} \sum_{i=1}^{D} (\bar{m}_{ih} - [\boldsymbol{\Phi}_h \boldsymbol{a}]_i)^2 \tag{4.4}$$

is the historical regularizer re-expressed in terms of $\boldsymbol{a}$. As [17] noted, all of general loss functions of interest in [10] (except the Poisson and Gaussian losses, which are convex) are difference of convex (DC) functions comprising of a function $\breve{\ell}(x, m)$ convex in m plus a concave function $\hat{\ell}(x, m)$. Furthermore, the most practically used losses employ link functions constraining the variables to be nonnegative, since nonnegativity typically recovers more interpretable CP factors. In other words, the loss in (4.3) is most commonly expressed as $\ell_i(\boldsymbol{a}) := \breve{\ell}(x_{it}, [\boldsymbol{\Phi}\boldsymbol{a}]_i) + \hat{\ell}(x_{it}, [\boldsymbol{\Phi}\boldsymbol{a}]_i)$ for $\boldsymbol{\Phi} \geq 0, \lambda > 0$. Note that the loss is not separable in each coordinate of $\boldsymbol{a}$, i.e., a_j for $j = 1, \ldots, R$, so (4.3) cannot be solved for a_j in closed-form. In this section, we develop additively separable majorizers of $\ell(\cdot)$, $g(\boldsymbol{a}; \boldsymbol{a}^{(k)}) : \mathbb{R}_+^R \mapsto \mathbb{R}$ for DC losses that are minimized at each iteration. Given current iterate $\boldsymbol{a}_k$, the majorizers are functions such that

$$\sum_{i=1}^{D} \ell_i(\boldsymbol{a}) \leq g(\boldsymbol{a}; \boldsymbol{a}^{(k)}) = \sum_{j=1}^{R} g_j(a_j; a_j^{(k)}), \quad \forall a_j \geq 0 \tag{4.5}$$

$$\sum_{i=1}^{D} \ell_i(\boldsymbol{a}^{(k)}) = g(\boldsymbol{a}^{(k)}; \boldsymbol{a}^{(k)}). \tag{4.6}$$

DC majorizers using Jensen's inequality and first order Taylor expansions are well-known in nonnegative matrix factorization literature for constructing majorizing surrogates of non-Euclidean functions [7, 9, 16, 19]. The SmartCPD algorithm of [17] also leverages this structure to derive MM/stochastic mirror descent updates for nonnegative GCP factorization, but their methods do not have closed-form updates for problems like streaming GCP that have historical and Tikhonov regularization terms. An important difference of this work will be the inclusion of these regularization terms which makes solving for the minimizers of the surrogates non-trivial.

We derive our majorizers in a similar fashion by upper-bounding ℓ with a DC surrogate that first linearizes $\hat{\ell}$ and then majorizes $\breve{\ell}$ using convexity and Jensen's Inequality [8]. For the Poisson loss where there is no concave term in the negative log-likelihood, this technique specializes to a regularized version of the CP-APR algorithm [4]. We can also use Jensen's inequality and convexity to majorize $r_{\mathcal{H}}$, yielding surrogate functions

$$g_j^{r_{\mathcal{H}}}(a_j; a_j^{(k)}) = \sum_j \sum_{h \in \mathcal{H}_t} \frac{w_h}{2} \sum_{i=1}^{D} \left(\frac{\phi_{ijh} a_j^{(k)}}{[\boldsymbol{\Phi}_h \boldsymbol{a}^{(k)}]_i} \right) \left(\bar{m}_{ih} - \frac{[\boldsymbol{\Phi}_h \boldsymbol{a}^{(k)}]_i}{a_j^{(k)}} a_j \right)^2, \tag{4.7}$$

where ϕ_{ij} denotes the $(i,j)^{th}$ entry of the matrix $\boldsymbol{\Phi}$. Combining all of the above surrogates, we obtain the composite upper-bounding surrogate

$$g_j(a_j; a_j^{(k)}) := g_j^{\mathrm{DC}}(a_j; a_j^{(k)}) + g_j^{r_{\mathcal{H}}}(a_j; a_j^{(k)}), \tag{4.8}$$

$$\begin{aligned} g_j^{\mathrm{DC}}(a_j; a_j^{(k)}) := \sum_{i=1}^{D} \{ & \alpha_{ij} \breve{\ell}_i \left(x_{it}, \frac{a_j}{a_j^{(k)}} [\boldsymbol{\Phi}\boldsymbol{a}^{(k)}]_i \right) \\ & + [\nabla \hat{\ell}_i(\boldsymbol{x}_t, \boldsymbol{\Phi}\boldsymbol{a}^{(k)})]_j (a_j - a_j^{(k)}) + \hat{\ell}_i(\boldsymbol{x}_t, \boldsymbol{\Phi}\boldsymbol{a}^{(k)}) \} + \frac{\lambda}{2} a_j^2 \end{aligned} \tag{4.9}$$

for nonnegative scalars $\alpha_{ij} = \frac{\phi_{ij} a_j^{(k)}}{[\mathbf{\Phi} \boldsymbol{a}^{(k)}]_i} \geq 0$ that sum to one by construction, i.e. $\sum_{j=1}^{R} \alpha_{ij} = 1$.

At each iteration k, we minimize the surrogate functions for each a_j as $\min_{a_j \geq 0} g_j(a_j, a_j^{(k)})$, which after setting the gradient equal to zero, requires solving the following equations for each a_j for $j = 1, \ldots, R$:

$$\sum_{i=1}^{D} \phi_{ijt} \nabla \breve{\ell}_i \left(x_{it}, \frac{[\mathbf{\Phi}_t \boldsymbol{a}^{(k)}]_i}{a_j^{(k)}} a_j \right) + \sum_{i=1}^{D} \phi'_{ijt} \nabla \hat{\ell}_i(x_{it}, [\mathbf{\Phi}_t \boldsymbol{a}^{(k)}]_i)) - \xi_j + \mu_j a_j = 0, \tag{4.10}$$

where $\nabla \breve{\ell}$ and $\nabla \hat{\ell}$ denote the elementwise gradients of $\breve{\ell}$ and $\hat{\ell}$, and

$$\xi_j := \sum_{h \in \mathcal{H}_t} w_h \sum_{i=1}^{D} \phi_{ijh} \bar{m}_{ih} \tag{4.11}$$

$$\mu_j := \left(\lambda + \sum_{h \in \mathcal{H}_t} w_h \sum_{i=1}^{D} \frac{\phi_{ijh} [\mathbf{\Phi}_h \boldsymbol{a}^{(k)}]_i}{a_j^{(k)}} \right). \tag{4.12}$$

Importantly, we note the $\mathbf{\Phi} \boldsymbol{a}^{(k)}$ terms are implemented and computed as $\texttt{vec}(\boldsymbol{Z}^{(n)} \boldsymbol{A}^{(k,n)'})$ since all $\mathbf{\Phi}$ are block-diagonal. Continuing below, we derive the factor updates in the individual cases for various statistical losses.

Rayleigh. For Rayleigh distributed data $x > 0$, $\ell(x, m) = \frac{\pi}{4}(\frac{x}{m})^2 + 2\log(m)$, $\nabla \breve{\ell}(x, m) = -\frac{\pi}{2} \frac{x^2}{m^3}$, and $\nabla \hat{\ell}(x, m) = \frac{2}{m}$. Solving (4.10) yields the following polynomial in the variable $z_j = 1/a_j$:

$$\left(-\frac{\pi}{2} {a_j^{(k)}}^3 \sum_{i=1}^{D} \frac{\phi_{ij} x_i^2}{[\mathbf{\Phi} \boldsymbol{a}^{(k)}]_i^3} \right) z_j^4 + \left(\sum_{i=1}^{D} \frac{2\phi_{ij}}{[\mathbf{\Phi} \boldsymbol{a}^{(k)}]_i} - \xi_j \right) z_j + \mu_j = 0. \tag{4.13}$$

The above polynomial is a twice-depressed quartic polynomial $z_j^4 + q_j z_j + r_j = 0$ since its cubic and square terms are zero. Analyzing the signs of its coefficients reveals there is at most one real positive root by Descartes' rule of signs, which can be analytically computed easily using Ferrari's method.

Gamma. Gamma-distributed data $x > 0$ has loss $\ell(x, m) = x/m + \log(m)$. (4.10) is then

$$\left(-{a_j^{(k)}}^2 \sum_{i=1}^{D} \frac{\phi_{ij} x_i}{[\mathbf{\Phi} \boldsymbol{a}^{(k)}]_i^2} \right) z_j^3 + \left(\sum_{i=1}^{D} \frac{\phi_{ij}}{[\mathbf{\Phi} \boldsymbol{a}^{(k)}]_i} - \xi \right) z_j + \mu = 0, \tag{4.14}$$

which is a depressed cubic with at most one real positive root that can be analytically found with Cardano's formula.

Bernoulli. For Bernoulli distributed data where $x \in \{0, 1\}$, $\ell(x, m) = \log(m + 1) - x \log(m)$, $\nabla \breve{\ell}(x, m) = -x/m$, and $\nabla \hat{\ell}(x, m) = 1/(m + 1)$. (4.10) becomes

$$\left(-a_j^{(k)} \sum_{i=1}^{D} \frac{\phi_{ij} x_i}{[\mathbf{\Phi} \boldsymbol{a}^{(k)}]_i} \right) z_j^2 + \left(\sum_{i=1}^{D} \frac{\phi_{ij}}{[\mathbf{\Phi} \boldsymbol{a}^{(k)} + 1]_i} - \xi \right) z_j + \mu = 0, \tag{4.15}$$

whose real positive root is given by the quadratic formula.

Poisson. Similar to binary data, the gradient equations of the majorizers for count data $x \in \mathbb{N}$ under a Poisson distribution are also described by quadratics, where $\ell(x, m) = m - x\log(m)$, $\nabla\check{\ell}(x, m) = -x/m$, and $\nabla\hat{\ell}(x, m) = 1$.

$$\left(-a_j^{(k)} \sum_{i=1}^{D} \frac{\phi_{ij} x_i}{[\boldsymbol{\Phi}\boldsymbol{a}^{(k)}]_i}\right) z_j^2 + \left(\sum_{i=1}^{D} \phi_{ij} - \xi\right) z_j + \mu = 0. \tag{4.16}$$

β-divergence ($\beta = 1/2$). Because of the quadratic regularizer, solving for the roots of (4.10) becomes nontrivial. Here, we focus on the cases when β is a rational number to derive polynomials with integer powers. For $\beta \in (0, 1)$, (4.10) takes the form

$$\left(-{a_j^{(k)}}^{2-\beta} \sum_{i=1}^{D} \frac{\phi_{ij} x_i}{[\boldsymbol{\Phi}\boldsymbol{a}^{(k)}]_i^{2-\beta}}\right) a_j^{\beta-2} + \left(\sum_{i=1}^{D} \phi_{ij} [\boldsymbol{\Phi}\boldsymbol{a}^{(k)}]_i^{\beta-1} - \xi\right) + \mu a_j = 0. \tag{4.17}$$

For $\beta = 1/2$, letting $z_j = a_j^{-1/2}$ yields depressed quintics with coefficients $a \in \mathbb{R}_-$ and $b, \mu \in \mathbb{R}_+$ corresponding to the terms above:

$$a z_j^5 + b z_j^2 + \mu = 0. \tag{4.18}$$

Polynomials of degree five and higher do not have analytic roots, but again by knowledge of Descartes' rule of signs, at most one real positive root exists. We use Julia's `Roots.jl` package and the `find_zero` function with the bisection method to compute the roots. The bisection method converges linearly, and the number of iterations can be further reduced by restricting the algorithm's initial search interval since we know the real positive root lies in $[0, 1 + \max\{-b/a, -\mu/a\}]$.

The key features of the majorizers obtained by our polynomial solvers include the regularization terms without upper-merging lower polynomial powers to become quadratic like the work in [21]. Hence, our majorizers for fully-sampled data are tighter upper bounds, as evidenced by plots of the loss and various surrogates in Fig. 4.1 for a one-dimensional scalar factor problem. We also remark how the quadratic surrogate, equivalently gradient descent, not only makes smaller progress with each iterate, but is highly sensitive to the choice of step size that dictates the curvature of the upper bounding quadratic. Each of our surrogates has only one unique minimizer, which in many cases can be found in closed-form arithmetic. We summarize the losses, their associated minimizing surrogate polynomials, and their solvers in table 4.1.

4.3. Optimization for $\boldsymbol{s}_t$. We note here that the optimization for $\boldsymbol{s}_t$ is the same as that for $\boldsymbol{A}^{(n)}$, where $\boldsymbol{s}_t$ itself can be treated as a $1 \times r$ factor for the time mode, and $w_h = 0$ for all h.

4.4. Stochastic Augmented Non-Euclidean Surrogates for Tensors Algorithm (SANESTA). The original objective function involves a very large sum of D terms, i.e. $\ell(\boldsymbol{x}, \boldsymbol{m}) = \sum_{i=1}^{D} \ell_i(\boldsymbol{x}_i, \boldsymbol{m}_i)$ and furthermore

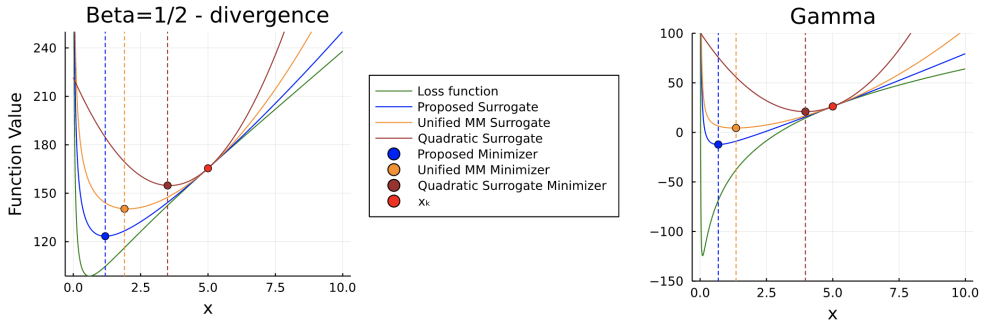
$$f(\boldsymbol{m}) = \mathbb{E}_{\boldsymbol{x}}[\ell(\boldsymbol{x}, \boldsymbol{m})], \tag{4.19}$$

where if the data samples are drawn i.i.d., each $\ell(\boldsymbol{x}_i, \boldsymbol{m})$ is an unbiased estimate of the objective function. The majorizers (in one block of variables, with the others held fixed) described thus far also involve this very large sum over D gradients of the loss in (4.10) that become the primary computational bottlenecks:

$$g(\boldsymbol{A}^{(n)}; \boldsymbol{A}_k^{(n)}) := \sum_{i=1}^{D} g_i(\boldsymbol{A}^{(n)}; \boldsymbol{A}_k^{(n)}). \tag{4.20}$$

Loss	Polynomial	Solver
$\beta = 1/2$-divergence	Thrice-depressed quintic	Bisection method
Rayleigh	Twice-depressed quartic	Ferrari's method*
Gamma	Depressed cubic	Cardano's formula*
Poisson	Quadratic	Quadratic formula*
Bernoulli	Quadratic	Quadratic formula*
Neg. Binomial	Quadratic	Quadratic formula*

Table 4.1: Statistical losses, the polynomial gradient equations to minimize their surrogates, and the solvers to obtain the minimizers. * = closed-form solution.



(a) $\beta = 1/2$-divergence (b) Gamma loss

Fig. 4.1: Upper bounding majorizers for a one-dimensional problem. The x-axis represents the scalar variable, and x_k represents the current iterate value. Unified MM uses the update from [21]. The quadratic surrogate (i.e. gradient descent) is illustrated for a step size of 0.1.

This large sum requires forming and storing the entire Khatri-Rao product matrix of factors $\boldsymbol{Z}^{(n)}$, which itself is unfeasible for large tensor data. The works in [13] and [14] showed that such MM algorithms defined over a large sum of surrogates can be solved stochastically or incrementally, respectively. However, the work in [14] still requires storing a memory of the previous surrogates, which for our problem is not feasible. Instead, we perform stochastic updates like in [13] by drawing a random sample $i \in D$ to form surrogate $g_k(\boldsymbol{A}; \boldsymbol{A}_k, [i])$ (where $g(\boldsymbol{A}; \boldsymbol{A}_k, \Omega) := \sum_{i \in \Omega} g_i(\boldsymbol{A}; \boldsymbol{A}_k)$ denotes forming surrogate $g(\boldsymbol{A}; \boldsymbol{A}_k)$ from the samples in set Ω) and then only storing and recursively updating one extra surrogate in memory:

$$\boldsymbol{A}^{(n)}_{k+1} = \min_{\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times r}} \bar{g}_k(\boldsymbol{A}^{(n)}), \quad \bar{g}_k = (1-\theta_k)\bar{g}_{k-1} + \theta_k g_k(\boldsymbol{A}^{(n)}; \boldsymbol{A}^{(n)}_k, i), \quad k \geq 1. \tag{4.21}$$

We use weights $\theta_k = 1/\sqrt{k}$ which satisfy the conditions for $(\theta_k)_{k \geq 1}$ in [13]. The MM updates for (4.21) are very similar to stochastic gradient averaging, where the polynomial coefficients in (4.10) become weighted gradient averages.

Rather than single sample updates, it is preferable to take small mini-batches of samples to expedite convergence. Because our MM algorithms are first-order gradient methods, we can form surrogates from unbiased stochastic gradients like Kolda & Hong [12]. Below we describe two sampling strategies for incorporating unbiased gradients from mini-batches into the stochastic MM approach.

Uniform sampling is the simplest method to implement and works well with densely distributed data, like Rayleigh or Gamma, where the s number of sampled gradients are upscaled by D/s. For samples Ω_s drawn uniformly at random, the surrogates are updated as

$$\bar{g}_k = (1-\theta_k)\bar{g}_{k-1} + \frac{D}{s}\theta_k g_k(\boldsymbol{A}^{(n)}; \boldsymbol{A}_k^{(n)}, \Omega_s), \quad k \geq 1. \tag{4.22}$$

In the case of sparse data, the authors in [12] showed uniform sampling performs poorly since it often misses the nonzero values, which are more important to the model and its gradients. In that work, the authors proposed stratify sampling the gradients based on the sets of nonzero and zero data indices, denoted Ω_{nnz} and Ω_{z} respectively (where $|\Omega_{\text{nnz}}| + |\Omega_{\text{z}}| = D$). Here, the gradients corresponding to the p sampled nonzero indices out of $\eta := |\Omega_{\text{nnz}}|$ total nonzeros are uniformly chosen at random from Ω_{nnz} and upscaled by η/p, and the q gradients of the zeros are uniformly chosen from Ω_{z} and upscaled by $|\Omega_{\text{z}}|/q = (D-\eta)/q$. We note that choosing the zeros is done like described in [12] where we overdraw a random sample of all tensor entries and then check and omit the samples if they are nonzero. Adapting to the MM framework, the surrogates are updated as

$$\bar{g}_k = (1-\theta_k)\bar{g}_{k-1} + \frac{\eta}{p}\theta_k g_k(\boldsymbol{A}^{(n)}; \boldsymbol{A}_k^{(n)}, \Omega_{\text{nnz}}) + \frac{D-\eta}{q}\theta_k g_k(\boldsymbol{A}^{(n)}; \boldsymbol{A}_k^{(n)}, \Omega_{\text{z}}), \quad k \geq 1. \tag{4.23}$$

Forming the subsampled $\boldsymbol{Z}^{(n)}$ can then done efficiently as described in [3] for computing the subsampled-Khatri-Rao product matrix from the chosen Cartesian indices. We emphasize special care must be taken in sampling, however, since the MM updates are multiplicative. If a gradient with respect to a_j is zero, then the minimizing root to the polynomial is also zero, and a_j will be set as such. Since the MM updates are multiplicative, setting any a_j to zero will trap the algorithm at this stationary point. To avoid these scenarios, this simply requires just sampling at least once on every row of the mode-n matricized tensor $\boldsymbol{X}^{(n)}$ to guarantee a gradient with respect to each element of $\boldsymbol{A}^{(n)}$. Another important implementation detail is that we prohibit any entry from being set exactly to numerical zero; instead, we limit any entry to a minimum of 10^{-6}; [4] gives more details on "scooching" the entries away from numerical zero in the CP-APR algorithm.

4.4.1. Block-coordinate descent. Here, we put together all the pieces to express the surrogates for each factor in an iteration of MM optimization. In the block-coordinate descent framework of tensor optimization, our heuristic will be to update the surrogates in an alternating way such that the most current surrogate is formed from the updated factors of the previous iterations, the historical weights, the previous temporal weights, and the current estimate of $\boldsymbol{s}_t$. In the case of uniform sampling with samples Ω, now denote the surrogate of the n^{th} mode at MM iteration k as

$$g_k^{(n)} = \frac{D}{|\Omega|} g\left(\boldsymbol{A}^{(n)}; \{\boldsymbol{A}_k^{(j)}\}_{j=1}^N, \boldsymbol{s}_{t,k}, \{\bar{\boldsymbol{A}}^{(j)}\}_{j=1}^N, \{w_h, \boldsymbol{s}_h\}_{h\in\mathcal{H}_t}, \Omega\right) \tag{4.24}$$

where $\boldsymbol{A}_k^{(n)}$ is the current iterate of the factor being estimated to form the majorizer, $\boldsymbol{A}_k^{(j)}$ for $j \neq n$ and $\boldsymbol{s}_{t,k}$ are the current iterates for the factors and current temporal weights, respectively, forming the Khatri-Rao product matrix, $\{\bar{\boldsymbol{A}}^{(j)}\}_{j=1}^N$ are the previous factor estimates from the $t-1^{th}$ streaming iteration, and $\{w_h, \boldsymbol{s}_h\}_{h\in\mathcal{H}_t}$ are the historical terms. Then form the averaged surrogate $\bar{g}_k^{(n)} = (1-\theta_k)\bar{g}_{k-1}^{(n)} + \theta_k g_k^{(n)}$ and compute

$$\boldsymbol{A}_k^{(n)} \leftarrow \underset{\boldsymbol{A}^{(n)}\in\mathbb{R}_+^{I_n\times r}}{\operatorname{argmin}} \bar{g}_k^{(n)}(\boldsymbol{A}^{(n)}).$$

The next mode for $n+1$ in the block coordinate descent scheduling will then use the updated $\boldsymbol{A}_k^{(n)}$.

The surrogates for stratified-sampled data, with nonzero samples Ω^{nnz} and zero samples Ω^{z}, are similar:

$$\begin{aligned} g_k^{(n)} = &\frac{\eta}{p} g\left(\boldsymbol{A}^{(n)}; \{\boldsymbol{A}_k^{(j)}\}_{j=1}^N, \{\bar{\boldsymbol{A}}^{(j)}\}_{j=1}^N, \{w_h, \boldsymbol{s}_h\}_{h\in\mathcal{H}_t}, \Omega^{\text{nnz}}\right) \\ &+ \frac{D-\eta}{q} g\left(\boldsymbol{A}^{(n)}; \{\boldsymbol{A}_k^{(j)}\}_{j=1}^N, \{\bar{\boldsymbol{A}}^{(j)}\}_{j=1}^N, \{w_h, \boldsymbol{s}_h\}_{h\in\mathcal{H}_t}, \Omega^{\text{z}}\right). \end{aligned} \tag{4.25}$$

We detail SANESTA in algorithm 1.

4.5. Complexity analysis. SANESTA is a first-order optimization method with computational complexity on par with gradient methods. For a CP rank-r tensor slice of sizes $I_1 \times \cdots \times I_N$, computing the first polynomial coefficient for each mode costs $\mathcal{O}(\text{nnz}(\boldsymbol{\mathcal{X}}_t) I_n r)$, since the multiplication with the data only depends on the number of nonzero entries, so this computation can be carried out efficiently with sparse arrays. Computing the second coefficient incurs $\mathcal{O}(|\Omega_t| I_n r)$ flops, where $|\Omega_t|$ is the number of samples drawn. Solving for the root of each polynomial costs K flops, where K is either the number of bisection iterations for β-divergences or otherwise a small constant for algebraically computing the root. In total, the computational cost of one SANESTA iteration is $\sum_{n=1}^N \mathcal{O}((|\mathcal{H}_t|+1)(\text{nnz}(\boldsymbol{\mathcal{X}}_t)+|\Omega_t|+K) I_n r)$, where $|\mathcal{H}_t| = 0$ when solving for $\boldsymbol{s}_t$.

Algorithm 1 Stochastic Augmented Non-Euclidean Surrogates for Tensors Algorithm (SANESTA)

Require: Previous factor estimates $\{\bar{\boldsymbol{A}}^{(n)}\}_{n=1}^N$, weights and previous temporal weights $w_h, \boldsymbol{s}_h, \quad \forall h \in \mathcal{H}$, number of samples S, maximum number of iterations K, tolerance $\epsilon > 0$, $\lambda > 0$.
Require: $\boldsymbol{\mathcal{X}}_t \in \mathbb{R}_+^{I_1 \times \ldots \times I_N}$ (data).
Initialize $\boldsymbol{A}_0^{(n)} \in \mathbb{R}_+^{I_n \times r}$ for $n = 1, \ldots, N$ and $\boldsymbol{s}_t \in \mathbb{R}_+^r$. D $= \prod_{n=1}^N I_n$
Initialize $g_0^{(n)} = 0, \quad \forall n = 1, \ldots, N \bigcup t$.
while iterations $k < K$ **do**
  $\theta_k = 1/\sqrt{k}$;
  for $n = 1, \ldots, N \bigcup t$ **do**
    if Uniform Sampling **then**
      Draw S number of samples in Ω uniformly at random;
      Form $g_k^{(n)}$ in (4.24);
    if Stratified Sampling **then**
      Draw S number of samples in Ω^{nnz} and Ω^{z} uniformly at random;
      Form $g_k^{(n)}$ in (4.25);
    $\bar{g}_k^{(n)} = (1-\theta_k)\bar{g}_{k-1}^{(n)} + \theta_k g_k^{(n)}$;
    if $n \neq t$ **then**
      $\boldsymbol{A}_k^{(n)} \leftarrow \text{argmin}_{\boldsymbol{A}^{(n)} \in \mathbb{R}_+^{I_n \times r}} \bar{g}_k^{(n)}(\boldsymbol{A}^{(n)})$;
    else
      $\boldsymbol{s}_{t,k} \leftarrow \text{argmin}_{\boldsymbol{s}_t \in \mathbb{R}_+^r} \bar{g}_k^{(n)}(\boldsymbol{s}_t)$;
  $k \leftarrow k+1$
return $\{\boldsymbol{A}^{(n)}\}_{n=1}^N, \boldsymbol{s}_t$

5. Experiments.

5.1. Setup. We synthetically generate tensor data observed under various probability distributions to empirically test our algorithm SANESTA against stochastic gradient descent with ADAM (SGD-ADAM) [12]. Each experiment generates non-temporal factors $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$ and temporal factor $\boldsymbol{S} \in \mathbb{R}_+^{T \times r}$. For densely-distributed tensor data, like Rayleigh, Gamma, and $\beta = 1/2$, we generate sinusoidal factors whose columns are cosine waves with different frequencies and a bias of 1.1. Using Julia indexing notation, the entries of factor $\boldsymbol{A}^{(n)}$ are

$$\boldsymbol{A}^{(n)}[t, c] = 1.1 + \cos\left(\frac{2\pi c(t-1)}{D}\right), \quad t = 1, \ldots, I_n, \quad c = 1, \ldots, r, \quad D = \Pi_{j=1}^{N} I_j.$$

We generate $\boldsymbol{S}$ in the same manner. From the synthetic factors, we form the full batch CP model tensor $\boldsymbol{\mathcal{M}} \in \mathbb{R}_+^{I_1 \times \cdots \times I_N \times T}$ and draw samples under the chosen distribution to obtain observations $\boldsymbol{\mathcal{X}}$. For the Gamma and β distributions, the data are scaled to be unnormalized, since Julia's `Distributions.jl` package generates data from the normalized distribution.

For sparse Bernoulli and Poisson distributed tensor data, we follow the procedure in [4] and in the TensorToolbox [2] for generating "spikey" factors with a small number of large factor entries and the rest being a small positive constant. We form $\boldsymbol{\mathcal{M}}$ from the factors and either draw samples from a Poisson distribution, or for binary data, we convert the odds to probabilities $\boldsymbol{\mathcal{M}}/(1 + \boldsymbol{\mathcal{M}})$ and draw binary samples as $\boldsymbol{\mathcal{X}}_{i_1,\ldots,i_N,t} = \mathbb{I}\{p < \boldsymbol{\mathcal{M}}_{i_1,\ldots,i_N,t}\}$ for some random number $p \in [0, 1]$ where $\mathbb{I}\{\cdot\} \in \{0, 1\}$ is the indicator function.

Our synthetic experiments examine the recovery of non-temporal factors $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$ and temporal weights $\boldsymbol{s}_t$ for one iteration t of streaming GCP given $\boldsymbol{\mathcal{X}}_t$. For $\bar{\boldsymbol{\mathcal{M}}}_h$, we simulate previous estimates of $\bar{\boldsymbol{A}}^{(j)}$ as noisy-perturbations of the planted model factors $\boldsymbol{A}^{(j)}$ and $\boldsymbol{s}_h$ as the true rows of $\boldsymbol{S}$ for all $h \in \mathcal{H}$, where we set $\mathcal{H}$ to be the most recent time indices.

We generate a 25 3-way tensor slices of sizes $30 \times 30 \times 30$ from a CP rank-3 planted model and estimate the factorization for the last slice under Rayleigh and $\beta = 1/2$ distributions. We set $w_h = \omega * (0.9)^(25 - h)$ for $h = 1, \ldots, 24$ and choose a small ω to recover factors that interpolate between the previous factors and the factors that maximize the log-likelihood for just the most current slice.

In a manner similar to how we estimate the gradients, we also estimate the log-likelihood value at iteration k, $\mathcal{L}_k$, from uniform or stratified samples as described in [Sec 5.1, [12]]. We typically sample 10% of the entries to obtain a more reliable estimate that can differentiate between the performances of the two algorithms. From the log-likelihood value estimate, we normalize by the log-likelihood of the planted model, i.e. we compute $\mathcal{L}_k - \mathcal{L}^*$. In addition, we track the CPD factor score $\epsilon_t \in [0, 1]$ given in algorithm 2. The CPD factor score takes into consideration the scaling and permutation ambiguities in the CP factorization to compute how aligned the estimated factors are to the planted model. The factor score returns 1 as the best score and 0 as the poorest.

We implemented both SGD-ADAM [12] and SANESTA in Julia and ran our experiments on a MacBook Pro with a 2.6 GHz 6-Core Intel Core i7 processor. For each experiment, we initialize both algorithms from the same set of nonnegative randomly-generated factors, with 20 different initializations.

5.2. Results.

Dense data. We compare SANESTA and ADAM on performing one streaming iteration to compute the t^{th} factorization for dense Rayleigh and $\beta = 1/2$-distributed data. We set $\omega = 0.01, \mu = 0.01$ and $\omega = 0.001, \mu = 0.1$, respectively. For ADAM, we found the learning

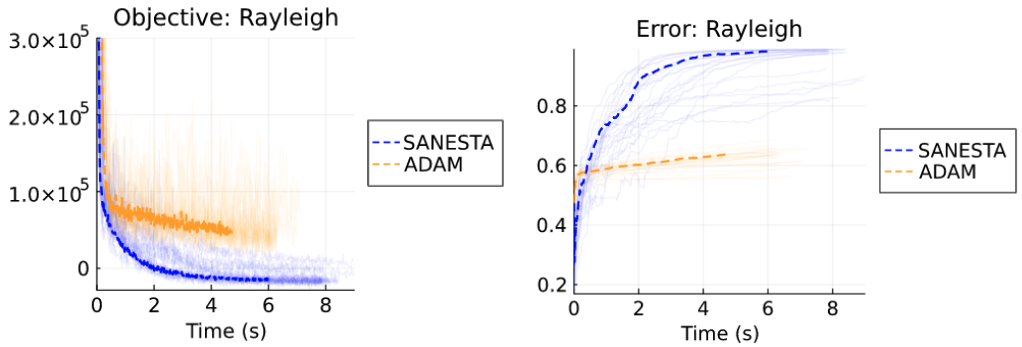
Algorithm 2 CPD factor score

Require: Estimated factors $\{\boldsymbol{F}^{(n)}\}_{n=1}^{N}$, true factors $\{\boldsymbol{A}^{(n)}\}_{n=1}^{N}$ where $\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times r}$

1: $\texttt{scores} = \mathbf{1}^r$
2: **for** $n = 1, \ldots, N$ **do**
3: **for** $i = 1, \ldots, r$ **do** ▷ Normalize factors to have unit-norm columns.
4: $\boldsymbol{F}^{(n)}[:, i] \leftarrow \boldsymbol{F}^{(n)}[:, i] / \|\boldsymbol{F}^{(n)}[:, i]\|_2^2$
5: $\boldsymbol{A}^{(n)}[:, i] \leftarrow \boldsymbol{A}^{(n)}[:, i] / \|\boldsymbol{A}^{(n)}[:, i]\|_2^2$
6: $\texttt{scores} = \texttt{scores} * \max\left\{\left|\boldsymbol{F}^{(n)'}\boldsymbol{A}^{(n)}\right|, \text{dims} = 1\right\}$ ▷ Compute inner products of the factors.
7: **return** $\epsilon_t = \frac{1}{r}\sum_{i=1}^{r} \texttt{scores}[i]$

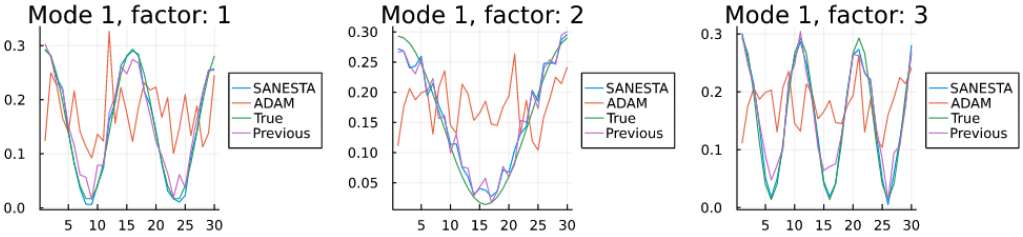
rates to be 0.1 and 0.01 for the two problems respectively to work best. Both algorithms sample 0.1% of the samples uniformly at random, estimate the log-likelihood from 1% of the samples, and execute for 400 iterations. We plot each of the individual trials' statistics by wall-clock time, as well as their medians in Fig. 5.1. The estimated log-likelihoods in Fig. 5.1(a) show SANESTA rapidly converges whereas ADAM slowly makes progress with large variance in its iterates. Fig. 5.1(b) reveals SANESTA's iterates converge to estimates far closer to the planted model in factor score, and Fig. 5.1(c) confirms this by visually plotting the recovered factors for one the trials at the termination of optimization. These plots are read as "Mode (n), factor: (j)" for referring to the j^{th} column of the factor $\boldsymbol{A}^{(n)}$. The plotted factors' interpolation between the planted model factors and the previous factors demonstrate the streaming algorithm's ability to modify upon the estimate from the previous factors. Likewise, we observe similar results for $\beta = 1/2$-distributed data in Fig. 5.2.

Sparse binary data. We generate $30 \times 35 \times 40$-size binary data from the "spikey" factor planted model with 7 spike components in each factor, generating data that is just over 12% sparse. For $\omega = 0.0001, \mu = 0.1$, both algorithms optimize the streaming objective for the Bernoulli log-odds likelihood and sample 420 nonzeros and zeros (equivalently 2% of the total entries) for the gradients and 4200 nonzeros and zeros (equivalently 20% of the total entries) for the log-likelihood estimate. We set ADAM's learning rate to 0.01. Fig. 5.3 displays our results, again demonstrating the superior optimization performance of SANESTA over ADAM for the same sampling schedule with similar computation costs for the gradients. Fig. 5.3(b)(c) show SANESTA's estimates lie closer to both the previous and planted model factors.



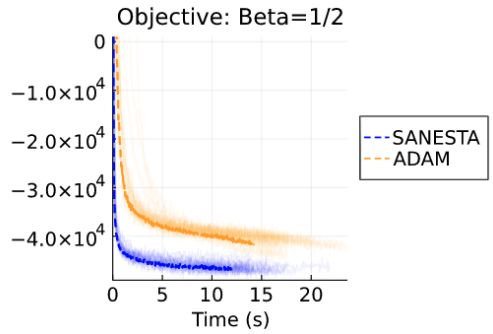
(a) Estimated log-likelihood

(b) Factor scores.

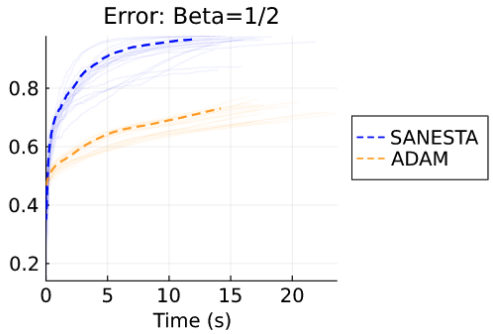


(c) Factor scores.

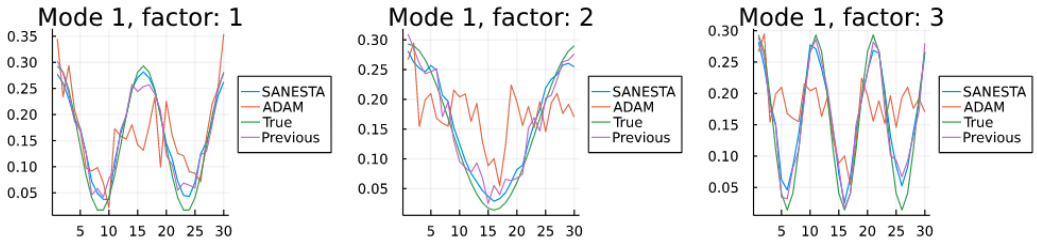
Fig. 5.1: Rayleigh-distributed $30 \times 30 \times 30$ tensor slices.



(a) Estimated log-likelihood

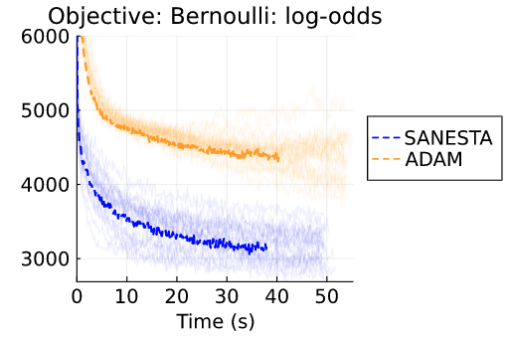


(b) Factor scores.

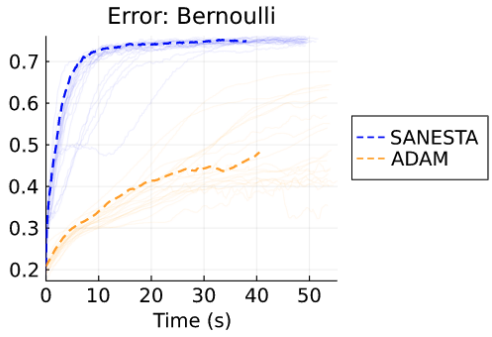


(c) Factor scores.

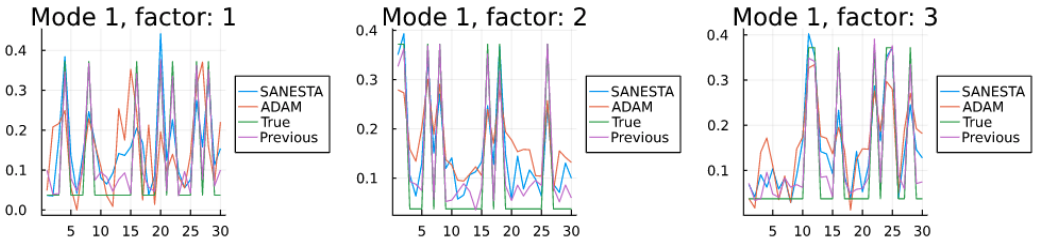
Fig. 5.2: $\beta = 1/2$-distributed $30 \times 30 \times 30$ tensor slices.



(a) Estimated log-likelihood



(b) Factor scores.



(c) Factor scores.

Fig. 5.3: Bernoulli-distributed $30 \times 35 \times 40$ tensor slices.

6. Conclusions. In this paper, we presented an algorithm, SANESTA, that minimizes tightly majorizing surrogate functions of the streaming GCP problem, requires no tunable optimization hyperparameters, can easily be adapted with stochastic sampling for large-scale data, and achieve convergence in far fewer iterations with similar computational complexity as ADAM.

Next, we anticipate testing both SANESTA and ADAM in a fully-streaming problem to compare the CP factor estimates across time iterations. Each streaming iteration itself takes considerable time to optimize, even with our algorithm that improves upon ADAM, so studying how inexactly solving the problem for each iteration is of interest. Other future work includes further understanding the interplay of the least squares historical regularizers with the log-likelihood of the statistical loss, and how the choice of weights w_h affect convergence and quality of factor estimates. The effects of initialization also play an important role in the optimization. In a practical application, one would likely initialize with the previous factors, but how the optimum is characterized when the previous factors are far away from the factors generating the slice at time t is not well-understood yet.

References.

[1]
[2] B. Bader, T. Kolda, et al., *Tensor Toolbox for MATLAB, version 3.2.1*, 2021.
[3] C. Battaglino, G. Ballard, and T. G. Kolda, *A practical randomized CP tensor decomposition*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 876–901.
[4] E. C. Chi and T. G. Kolda, *On tensors, sparsity, and nonnegative factorizations*, SIAM Journal on Matrix Analysis and Applications, 33 (2012), pp. 1272–1299.
[5] A. Cichocki, R. Zdunek, A. H. Phan, and S.-I. Amari, *Nonnegative Matrix and Tensor Factorizations*, John Wiley & Sons, Ltd, 2009, pp. i–xxi.
[6] S. E., J. Jones, R. Hutchinson, and V. Pfeiffer, *Plant pollinator data at HJ Andrews Experimental Forest, 2011 to 2018 ver 6. Environmental Data Initiative.*
[7] C. Févotte and J. Idier, *Algorithms for nonnegative matrix factorization with the β-divergence*, Neural computation, 23 (2011), pp. 2421–2456.
[8] N. Gillis, *Chapter 4: Majorization and Minorization*, pp. 93–121.
[9] ———, *Chapter 8: Iterative algorithms for NMF*, pp. 261–305.
[10] D. Hong, T. G. Kolda, and J. A. Duersch, *Generalized canonical polyadic tensor decomposition*, SIAM Review, 62 (2020), pp. 133–163.
[11] T. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[12] T. Kolda and D. Hong, *Stochastic gradients for large-scale tensor decomposition*, SIAM J. Math. Data Sci., 2 (2020), pp. 1066–1095.
[13] J. Mairal, *Stochastic majorization-minimization algorithms for large-scale optimization*, arXiv preprint arXiv:1306.4650, (2013).
[14] J. Mairal, *Incremental majorization-minimization optimization with application to large-scale machine learning*, SIAM J. Optim., 25 (2015), pp. 829–855.
[15] M. Mardani, G. Mateos, and G. Giannakis, *Subspace learning and imputation for streaming big data matrices and tensors*, Signal Processing, IEEE Transactions on, 63 (2014).
[16] M. Nakano, H. Kameoka, J. Le Roux, Y. Kitano, N. Ono, and S. Sagayama, *Convergence-guaranteed multiplicative algorithms for nonnegative matrix factorization with β-divergence*, in 2010 IEEE International Workshop on Machine Learning for Signal Processing, 2010, pp. 283–288.
[17] W. Pu, S. Ibrahim, X. Fu, and M. Hong, *Stochastic mirror descent for low-rank tensor decomposition under non-Euclidean losses*, ArXiv, abs/2104.14562 (2021).
[18] S. Smith, K. Huang, N. D. Sidiropoulos, and G. Karypis, *Streaming Tensor Factorization for Infinite Data Sources*, pp. 81–89.
[19] Z. Yang and E. Oja, *Unified development of multiplicative algorithms for linear and quadratic nonnegative matrix factorization*, IEEE Transactions on Neural Networks, 22 (2011), pp. 1878–1891.
[20] C. Ye and G. Mateos, *Online tensor decomposition and imputation for count data*, in 2019 IEEE Data Science Workshop (DSW), 2019, pp. 17–21.
[21] R. Zhao and V. Y. F. Tan, *A unified convergence analysis of the multiplicative update algorithm for nonnegative matrix factorization*, in 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 2562–2566.

STUDY OF THE RECOVERY DISCONTINUOUS GALERKIN METHOD AND ITS APPLICATION TO EMPIRE

MEGAN F. MCCRACKEN* AND SEAN MILLER†

Abstract. The recovery discontinuous Galerkin method recovers a smooth local solution that is weakly equal to the discrete discontinuous solution of the diffusive flux terms. This method can be applied to both structured and unstructured grids and is used to resolve the smooth solution between the union of two cells. This project focuses on the implementation of a recovery algorithm for the diffusive fluxes seen in the Navier-Stokes equation set, as well as the application of RDG in EMPIRE the relevant viscous terms. Tests on the achieved accuracy and stability are conducted to see improvements made over previous calculation methods.

1. Introduction. The recovery method for computing diffusive fluxes was first described in van Leer [5] in 2007. It uses a method in which the fluxes between cells are approximated by a smooth, locally recovered solution based on the weak equality of the discontinuous Galerkin method (DGM) [1] [3] [4] [8]. The recovery method is used to overcome previous deficiencies in DGM by taking the conventional advection-diffusion scheme and making it into a 2k+1 order of accuracy in an advection dominated problem and a 2k, where k is polynomial order, regime in a diffusion dominant problem [7]. Using the recovery algorithm for systems such as the compressible Navier-Stokes equations allows for higher order accuracy to be achieved while retaining the smooth components through discontinuities [6].

The recovery-based discontinuous Galerkin (RDG) method [2] has been proven to be a stable scheme for smooth elements within the DGM. This method uses a projection method to generate an interpolation function representing the gradient of a degree of freedom on the mesh [6].

One of the main drivers for exploring the recovery-based discontinuous Galerkin method is to address diffusive terms that occur in parabolic equations, which a standard DG does not address well. Using an interface-centered recovery scheme will allow the underlying DG algorithm to better handle the diffusive terms better than previous methods. Some of the benefits of this scheme include higher-order accuracy, weaker time-stepping restrictions given the methods need to only have information from the adjacent cells and relative ease of implementation into existing code. In order to test the ability of the recovery method as applied to the Navier-Stokes equations a Blasius solution will be explored.

To describe the recovery method used it is necessary to define the weak equality for the recovery algorithm we say that two functions f and g are weakly equal across some interval I if :

$$\int_I (f-g)\Psi_k dx = 0, \quad \forall k = 1, \ldots, N \tag{1.1}$$

where Ψ_k are test functions spanning a polynomial space P. The weak equality can be written as $f \doteq g$.

To handle a 1-D diffusion equation for the DG approach we define the test function Ψ_k for:

$$\int_I \psi_k \frac{\partial u_i^k}{\partial t} dx = -\int_{\partial I} \psi_k (G(\widetilde{u})D) dx + \int_I \nabla\psi_k (G(u)D) dx + \int_I \psi_k S dx \tag{1.2}$$

*Megan McCracken, Department of Aerospace and Ocean Engineering, meganfm@vt.edu
†Sandia National Laboratories, seamill@sandia.gov

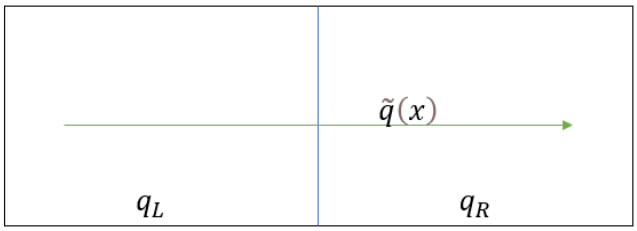


Fig. 1.1: The figure above represents the recovery interface where q_L is the left interface, q_R is the right interface, and $\tilde{q}$ is the recovered solution across the interface (represented by the blue line).

where u are the conserved variables to be passed, G is the general function form being solved for (e.i. the advection-diffusion, or Navier-Stokes), ψ_k is the test function of order k, D is the diffusion coefficient, and S is a given source term [2]. Below is a more in depth look at the application of the recovery algorithm in the context of a 1-D example with polynomial order of 1.

1.1. Recovery Algorithm. In this section the implementation of the recovery algorithm for a 1-D representation will be introduced. In order to apply the 1-D recovery solution it is necessary to define a representation of the solution in terms of q_L, or the left state at the cell interface, represented by K_L, and q_R, the right state of the interface, represented by K_R. A representation of these states can be seen in Fig. 1.1. The recovered state $\widetilde{q}$ is a continuous function satisfying the recovery definition in equation 1.1, using as the recovered state (1.3) and (1.4).

$$\widetilde{q}(x) \doteq q_L(x), x \in K_L \tag{1.3}$$
$$\widetilde{q}(x) \doteq q_R(x), x \in K_R \tag{1.4}$$

For the 1D case at polynomial order 1, the orthonormal Legendre basis functions are defined as:

$$\psi_1(\eta) = \frac{1}{\sqrt{2}} \tag{1.5}$$
$$\psi_2(\eta) = \frac{\sqrt{3}\eta}{\sqrt{2}} \tag{1.6}$$

Using this set of basis functions the recovery must be shifted to both the left and the right of the cell in order to recover the polynomial across the cell-interface. The basis set must first be shifted to the left at $K_L = [-2, 0]$, and to the right at $K_R = [0, 2]$. The basis set now takes on a form:

$$\psi_{L1}(\eta) = \psi_{R1}(\eta) = \frac{1}{\sqrt{2}} \tag{1.7}$$

$$\psi_{L2}(\eta) = \frac{\sqrt{3}(\eta+1)}{\sqrt{2}} \tag{1.8}$$

$$\psi_{R2}(\eta) = \frac{\sqrt{3}(\eta-1)}{\sqrt{2}} \tag{1.9}$$

Given the shifted polynomial basis set the q_L and q_R functions can now be defined as:

$$q_L(\eta) = \frac{1}{\sqrt{2}} q_{L1} + \frac{\sqrt{3}(\eta+1)}{\sqrt{2}} q_{L2}, \quad \eta \in [-2, 0] \tag{1.10}$$

$$q_R(\eta) = \frac{1}{\sqrt{2}} q_{R1} + \frac{\sqrt{3}(\eta-1)}{\sqrt{2}} q_{R2}, \quad \eta \in [0, 2] \tag{1.11}$$

Next, the solution state is projected onto onto a monomial basis set centered on the interface between cells. At p=1 there are 2 degrees of freedom in each of the cells making the the recovered polynomial 4th order:

$$\widetilde{q(\eta)} = \widetilde{q_1} + \widetilde{q_2}\eta + \widetilde{q_3}\eta^2 + \widetilde{q_4}\eta^3 \tag{1.12}$$

Giving a system of four equations with four unknowns:

$$\int_{-2}^{0} \left(\widetilde{q}(\eta) - q_L(\eta)\right) \psi_{L1}(\eta) d\eta = 0, \tag{1.13}$$

$$\int_{-2}^{0} \left(\widetilde{q}(\eta) - q_L(\eta)\right) \psi_{L2}(\eta) d\eta = 0, \tag{1.14}$$

$$\int_{0}^{2} \left(\widetilde{q}(\eta) - q_R(\eta)\right) \psi_{R1}(\eta) d\eta = 0, \tag{1.15}$$

$$\int_{0}^{2} \left(\widetilde{q}(\eta) - q_R(\eta)\right) \psi_{R2}(\eta) d\eta = 0. \tag{1.16}$$

We can invert this projection to define the solution to the monomial expansion coefficients in terms of the modal coefficients in the left and right cells:

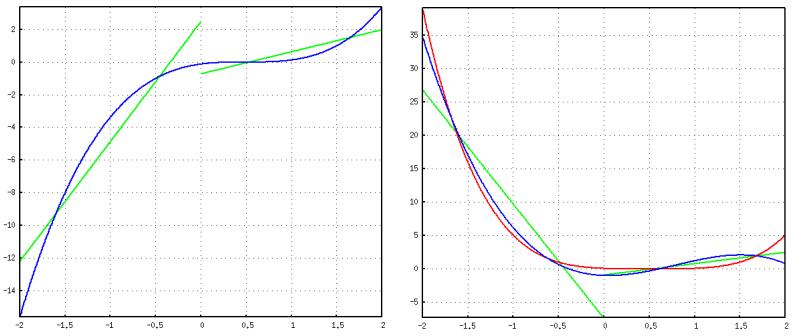
$$\widetilde{q_1} = \frac{\sqrt{2}\left(-2\sqrt{3}q_{R2} + 2\sqrt{3}q_{L2} + 3q_{R1} + 3q_{L1}\right)}{12}, \tag{1.17}$$

$$\widetilde{q_2} = -\frac{\sqrt{2}\left(5\sqrt{3}q_{R2} + 5\sqrt{3}q_{L2} - 9q_{R1} + 9q_{L1}\right)}{16}, \tag{1.18}$$

$$\widetilde{q_3} = -\frac{\sqrt{3}\left(q_{L2} - q_{R2}\right)}{\sqrt{2}^5}, \tag{1.19}$$

$$\widetilde{q_4} = \frac{\sqrt{2}\left(5\sqrt{3}q_{R2} + 5\sqrt{3}q_{L2} - 5q_{R1} + 5q_{L1}\right)}{32}. \tag{1.20}$$

This recovery form is referred to as a 2-cell recovery and it is used to recover the solution on an interface between two cells. In order to extend this to a two-dimensional system the recovery cells increase from 2 to 6, in which the solution is recovered not only across the x-direction but also the y-direction. A further explanation of this method is found in section 1.2. A representation of the recovery algorithm in 1D can be seen in Fig. 1.2. Here the original polynomial is represented in red, the projection in green, and the recovered polynomial in blue. In Fig. 1.2(a) the red line in fully recovered and thus is hidden beneath the blue.



(a) P = 2 (b) P = 3

Fig. 1.2: 2-cell recovery. Red line is the original polynomial, green are the projections of the polynomial to the P=1 representation in each cell, and blue is the recovered 4th order polynomial spanning the left and right cells.

1.2. Multi-dimensional Recovery. For multi-dimensional recovery there are a few methods that can be explored in order to achieve a full recovery in two or three dimensions[1].The first way is building a recovery solution traditionally by forming a multidimensional polynomial, as well as a systems of equations analogous to 1.13-1.16. For $p = 1$ in 2D such a polynomial would look like equation 1.21 below[2]

$$\widetilde{q}(\eta,\xi) = \widetilde{q}_1 + \widetilde{q}_2\eta + \widetilde{q}_4\eta^2 + \widetilde{q}_4\eta^3 + \widetilde{q}_5\xi + \widetilde{q}_6\eta\xi + \widetilde{q}_7\eta^2\xi + \widetilde{q}_8\eta^3\xi. \tag{1.21}$$

Using the representation above an equation system such as 1.22-1.23 can be established.

$$\int_{-2}^{0}\int_{-1}^{1} \left(\widetilde{q}(x,y) - q_L(\eta)\right)\psi_{L,k}(x,y)dydx = 0, \tag{1.22}$$

$$\int_{0}^{2}\int_{-1}^{1} \left(\widetilde{q}(x,y) - q_R(\eta)\right)\psi_{R,k}(x,y)dydx = 0. \tag{1.23}$$

This method can require a large amount of computing power, since it creates 8 equations for each face, therefore the RDG allows for an alternate solution to multi-dimensionality. In the second method the recovery can be accomplished in 1D and the spatial variations on the other dimensions can be constructed separately since they are in an orthogonal

[1] Here it is meant that a recovered polynomial is across one face but using data for multiple dimensions.
[2] η and ξ are in logical space, which was chosen instead of writing η_x or η_y

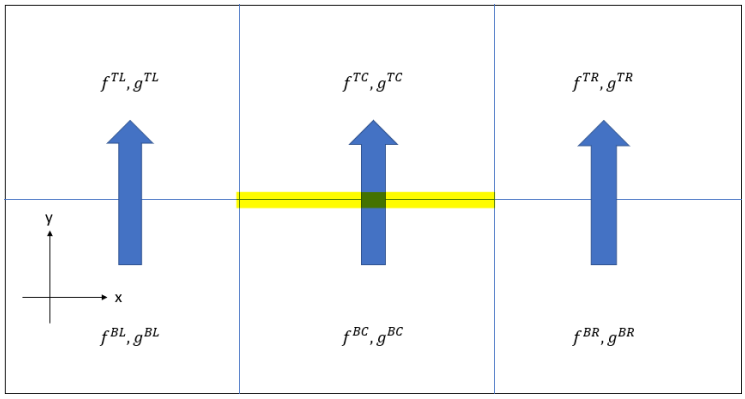


Fig. 1.3: In the figure above $f^{TL,TC,TR}$ and $g^{TL,TC,TR}$ represent the Top left, center and right expansion coefficient f and g, and $f^{BL,BC,BR}$, $g^{BL,BC,BR}$ are the Bottom Left, Center, and Right representations respectively. The blue arrows represent a 2-cell recovery in the y-direction. The highlighted line in the center is the section where we evaluate the recovered solution.

dimension. This is niot restricted to Cartesian meshes, however for simplicity the algorithm is represented as such in this research. This representation can be better visualized in figure 1.3, where a multidimensional recovery can be seen as a system of 9, 2-cell recoveries, one recovery at each edge. As shown previously, the recovered polynomial is given in terms of the expansion coefficients of the DG representation

$$\widetilde{q}_1 = \frac{\sqrt{2}\left(-2\sqrt{3}q_{R2}^{1D} + 2\sqrt{3}q_{L2}^{1D} + 3q_{R1}^{1D} + 3q_{L1}^{1D}\right)}{12}, \tag{1.24}$$

$$\widetilde{q}_2 = -\frac{\sqrt{2}\left(5\sqrt{3}q_{R2}^{1D} + 5\sqrt{3}q_{L2}^{1D} - 9q_{R1}^{1D} + 9q_{L1}^{1D}\right)}{16}, \tag{1.25}$$

$$\widetilde{q}_3 = -\frac{\sqrt{3}\left(q_{L2}^{1D} - q_{R2}^{1D}\right)}{\sqrt{2}^5}, \tag{1.26}$$

$$\widetilde{q}_4 = \frac{\sqrt{2}\left(5\sqrt{3}q_{R2}^{1D} + 5\sqrt{3}q_{L2}^{1D} - 5q_{R1}^{1D} + 5q_{L1}^{1D}\right)}{32}. \tag{1.27}$$

where $q_{R,L:1,2}^{1D}$ represent the monomial coefficients to the right or the left respectively in a single dimension(1D).
The 2D representation can then be taken as, $\sum_{i=1,\ldots,4} q_i^{2D}\psi_i^{2D}(x,y)$, and projected onto 1D basis function[3],

$$q_k^{1D}(y) = \int_{-1}^{1}\left(\sum_{i=1,\ldots,4} q_i^{2D}\psi_i^{2D}(x,y)\right)\psi_k^{1D}dx. \tag{1.28}$$

[3]Here the mass matrix is an identity

Here, the 1D expansion coefficient is still a function of y, therefore for $p = 1$, the following basis coefficients can be represented as:

$$q_1^{1D}(y) = \frac{\sqrt{3}q_3^{2D}y + q_1^{2D}}{\sqrt{2}}, \tag{1.29}$$

$$q_2^{1D}(y) = \frac{\sqrt{3}q_4^{2D}y + q_2^{2D}}{\sqrt{2}}. \tag{1.30}$$

Substituting these expansion coefficients into the 1D recovered polynomial then produces the same result as directly solving Eq. 1.22. Working through multiple 1D recoveries as opposed to solving for one large matrix allows for more manageable equation sets. In higher dimensions the recovery solution is still represented by a large matrix, but each recovery is more simply represented and calculated. for an unstructured grid the recovery algorithm requires solving for a local matrix system as well, and is not discussed in this research.

2. Results. Below are the initial tests run on the python script for the test recovery algorithm. Test cases are run in 1D but will be extended to 2D with the addition of the recovery algorithm for the diffusive terms, as well as the 2D Navier-Stokes equation set. Initial testing shown in Fig. 2.1 shows a test of a smooth advection wave, represented below in equations 2.1 and 2.2.

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} \doteq 0 \tag{2.1}$$

$$\int \frac{\partial f}{\partial t}\psi_t + \underbrace{\int \frac{\partial f}{\partial x}\psi_t}_{[f\psi_t] - \int f\frac{\partial \psi_t}{\partial x}} = 0 \tag{2.2}$$

Fig. 2.1 shows the advection wave run over 100 iterations on the top and a single iteration on the bottom, comparing the initial wave to recovery flux algorithm, central flux algorithm and the upwind flux algorithm to show a comparison between the most common methods of handling diffusive terms.

A second one-dimensional test run on the heat equation represented here in equation 2.3. The test simulated the heat flow as distributed across a uniform metal plate, where the boundary conditions were taken as a zero gradient. In Fig. 2.2 we see the convergence results with the calculated analytical solution for this test and that of the recovered solution. Here, we see good agreement between the first and second order polynomials for this test case. The heat equation is defined as:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{2.3}$$

where $\kappa = \frac{K_0}{c\rho}$ represents thermal diffusivity, t is time, and u are the conserved quantities.

2.1. Blasius solution testing. The recovery algorithm is tested in 2D by solving the Navier-Stokes equations for the semi-analytic Blasius solution.In order to test the recovery algorithm in two dimensions a simple Blasius solution selected. The flow over a plate problem for aerodynamics was selected since it is one of the most simplistic cases available. The goal of the flow over a flat plate solution is to analyze the basis solution for flow over a plate as the pressure gradient and suction velocities vary. In order to do this a viscosity was calculated using Sutherland's Law and ideal gas constants were assumed. For other

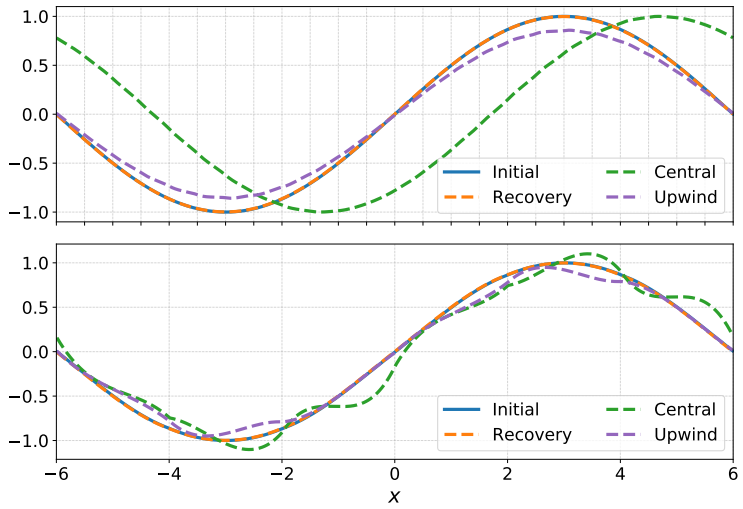


Fig. 2.1: Comparison of classical methods for DG ($p = 1$) 1D advection. Plotted are results after 100 periods (top) and 1 period (bottom) for different resolutions.

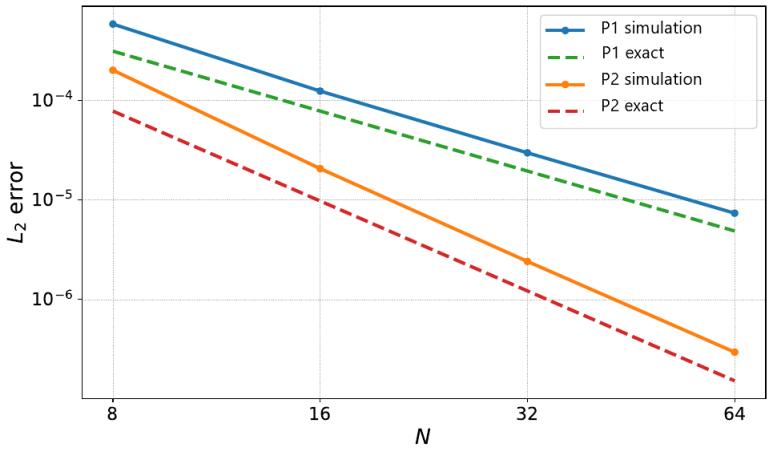


Fig. 2.2: Convergence study of the heat equation in 1D for P1 and P2 as compared to the exact.

constants the thermal conductivity of air, $c_p = 0.3$, as well as the specific heat of air, $k = 0.02$, were assumed. A non-dimensional form of the 1D Navier-Stokes equations are represented below. These were extended to the 2D form and are the main equation set used in the set up of the Blasius solutions.

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} = 0 \tag{2.4}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial (\rho u^2 + p)}{\partial x} - \frac{1}{Re}\frac{\partial}{\partial x}(\frac{4}{3}\frac{\tilde{\mu}}{\mu_{ref}}\frac{\partial u}{\partial x}) = 0 \tag{2.5}$$

$$\frac{\partial E}{\partial t} + \frac{\partial (u(E+p))}{\partial x} - \frac{1}{Re}\frac{\partial}{\partial x}(\frac{4}{3}u\frac{\tilde{\mu}}{\mu_{Ref}}\frac{\partial u}{\partial x}) - \frac{1}{ReEcPr}\frac{\partial}{\partial x}(\frac{\tilde{k}}{k_{Ref}}\frac{\partial T}{\partial x}) = 0 \tag{2.6}$$

Using these equations the free-stream flow constants are defined as $u_e = 10$ m/s, Re $=$ 63750.0, Pr $= 0.003$ and $\delta = 0.00396$ (m). Where u_e is the free stream velocity, Re is the Reynolds's number, Pr is the Prandtl number, and δ is the boundary layer height with a plate length, L, of 1. In order to initially test the Blasius solution for laminar flow a semi-infinite flat plate boundary layer must first be established, this is done using the Blasius equation below:

$$u = Ug(\eta) \tag{2.7}$$

where η is defined as

$$\eta = \frac{y}{\delta(x)} \tag{2.8}$$

with,

$$\delta(x) = (\frac{\nu x}{U})^{(1/2)}, \tag{2.9}$$

Establishing the basis structure of the Blasius solution used to test the recovery algorithm, a simplified case for the Navier-Stokes equations were chosen, where a stream function, defined here as $\Psi(x, y)$ with velocities u, and v.

$$u = \frac{\partial \Psi}{\partial y}, v = -\frac{\partial \Psi}{\partial x} \tag{2.10}$$

Other Laminar flow parameters needed to establish the boundary layer expressions are defined below:

$$\frac{u}{U} = f'(\eta) \tag{2.11}$$

$$v = \sqrt{\frac{\nu u_e(x)}{(2-\beta)x}}[(1-\beta)\cdot \eta f' - f(\eta)] \tag{2.12}$$

$$\delta = 0.994 \cdot f'(\eta) \cdot x\sqrt{\frac{2}{Re_x}} \tag{2.13}$$

$$\delta^* = x\sqrt{\frac{2-\beta}{Re_x}} \lim_{\eta\to\inf} \eta - f(\eta) \tag{2.14}$$

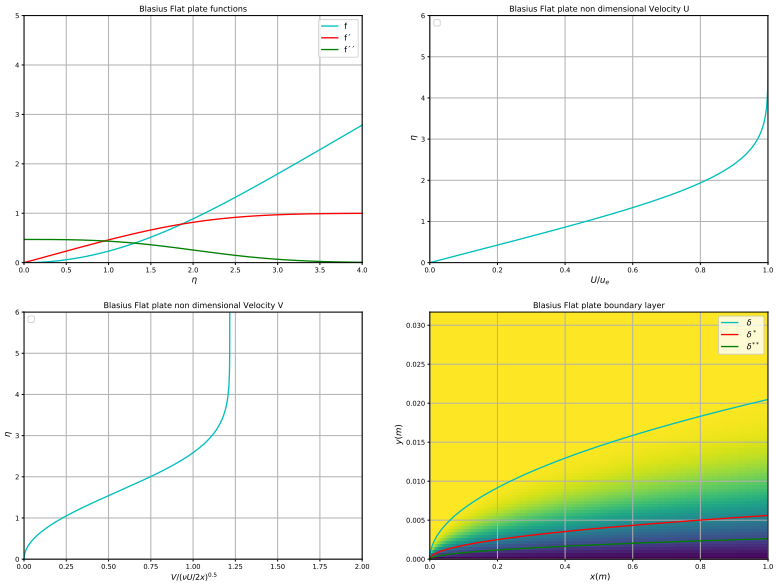
$$\delta^{**} = x\sqrt{\frac{2-\beta}{Re_x}} \frac{f"(0,\beta) - \beta \lim_{\eta\to\inf} \eta - f(\eta)}{1+\beta} \tag{2.15}$$

$$\tau_p = \rho \cdot u_e^2(x) \cdot f"(0,\beta)\sqrt{\frac{m+1}{2\cdot Re_x}} \tag{2.16}$$

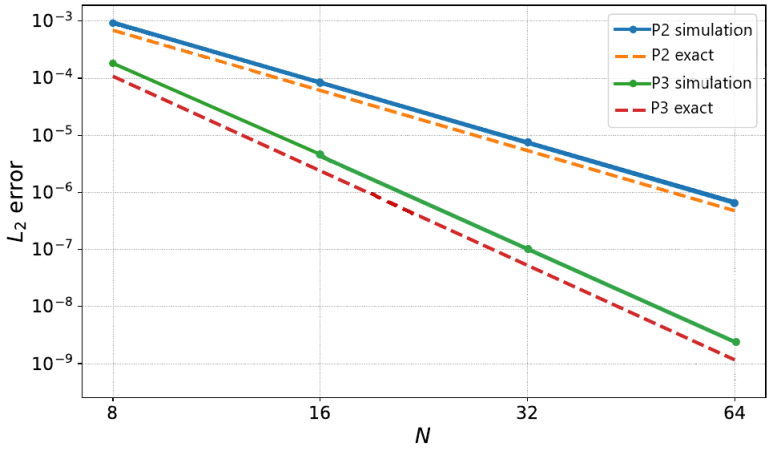
$$c_f = \frac{\tau_p}{\frac{1}{2}\cdot \rho \cdot u_e^2(x)} \tag{2.17}$$

$$Nu = \frac{q_p \cdot x}{k\cdot(T_p - T_{\inf})} = -\sqrt{\frac{Re_x}{2-\beta}}(\frac{d\Theta}{d\eta}_{\eta=0}) \tag{2.18}$$

Where eq. 2.5 is the Horizontal velocity, 2.6 is the vertical velocity, 2.7 is the boundary layer thickness, 2.8 is the boundary layer displacement thickness, 2.9 is the boundary layer momentum thickness, 2.10 is the shear stress at the wall, 2.11 is the skin friction coefficient, and 2.12 is the Nusselt number. The flow over a plate problem, also known as a Blasius problem, has no closed form analytical solution for calculating the boundary layer. Instead it requires solving for a set of nonlinear differential equations that are then used to interpret the final result. The above expressions were used to calculate the final analytical result of the Blasius solution being tested. Fig. 2.3 shows the analytical and convergence results from the Blasius solution testing. The convergence results seen in Fig. 2.3b show the L_2 error norm results between the analytical and the recovery simulation results for the Blasius solution. These results show that the recovery and the analytical forms have good agreement.



(a) Analytical solution for Blasius equations



(b) Convergence study for Blasius and recovery formula

Fig. 2.3: (a) Analytical solutions to the Blasius solutions. (b) Convergence of the recovery solution as compared to the Blasius solutions. Here simulation is the order achieved by the simulation and exact is the convergence order.

3. Conclusions. The main focus of this project was to establish the recovery-based discontinuous Galerkin algorithm as a viable option in calculating the diffusive terms in a discontinuous system. Tests were run in one and two dimensions on a structured mesh framework using python scripting to establish the ability of RDG to handle an advection wave, the heat equation, and Blasius solution test cases. Given the complexity of EMPIRE some changes in the structure of the prototypes are necessary. The tests run in this paper used a structured mesh, while EMPIRE is an unstructured mesh code with only face connectivity level support, as opposed to nodal support required by the structured mesh version of RDG. These are all factors that need to be taken into account when developing the recovery method in EMPIRE. This research shows that the recovery solution allows for diffusive fluxes to be captured accurately in a DG algorithm, and that extending this method to EMPIRE would be beneficial to the code by establishing a relatively simple method to handle the diffusive flux terms in a discontinuous system. Future work will include the establishments of the recovery algorithm in EMPIRE with both the surface and volume integrals to be recovered for the viscous terms.

References.

[1] F. L. A. Ferrero and G. Puppo, *A robust and adaptive recovery-based discontinuous galerkin method for the numerical solution of convection–diffusion equations*, 2014.
[2] M. L. Bram van Leer and M. van Raalte, *A discontinuous galerkin method for diffusion based on recovery*, (2007).
[3] K. Ho and M. Lo, *A space-time discontinuous galerkin method for navier-stokes with recovery*, 2011.
[4] E. Johnsen, A. G. Nair, and S. Varadan, *Recovery discontinuous galerkin method for compressible turbulence*, 2013.
[5] B. V. Leer, M. Lo, and M. V. Raalte, *A discontinuous galerkin method for diffusion based on recovery*, 2007.
[6] E. J. Loc Khieu and B. van Leer, *The recovery-based discontinuous galerkin method for the navier-stokes equations*, (2014).
[7] L. H. K. Philip E. Johnson and E. Johnson, *Analysis of recovery-assisted discontinuous galerkin methods for the compressible navier-stokes equations*, Journal of Computational Physics, (2020).
[8] S.Nomura and B. van Leer, *Discontinuous galerkin for diffusion*, AIAA Paper, (2011).

DATA-DRIVEN MODEL REDUCTION FOR PHYSICS-CONSTRAINED OPTIMIZATION

SHANE A. MCQUARRIE[§], JOSEPH HART[¶], BART VAN BLOEMEN WAANDERS[‖], AND KAREN WILLCOX[**]

Abstract. Harnessing the full power of physics-based models for decision making is challenging but critical in many engineering analyses. Accordingly, surrogate models with reduced computational complexity are necessary to combat the prohibitive costs of high-fidelity simulations. Operator Inference has recently emerged as a data-driven paradigm of constructing physics-based reduced-order models non-intrusively. In this work, we use Operator Inference as a cost-effective surrogate for dynamical constraints in a control setting. We improve the effectiveness of the reduced-order model by directly augmenting the underlying low-dimensional representation with known information from the objective function. The framework is demonstrated on two control problems constrained by convection-diffusion dynamics.

1. Introduction. Many critical problems in computational science and engineering applications may be solved as an optimization problem constrained by a physics-based model. In large-scale applications, governing physics modeled by partial differential equations or systems of ordinary differential equations present a computational challenge because solving such equations at high fidelity is often prohibitively expensive. Further, efficient optimization algorithms require access to derivatives of the constraint to determine search directions in high-dimensional spaces [1, 5, 6, 9, 13]. Such calculations demand access to the underlying physics models, which is not always feasible for complex application codes.

Model reduction seeks to alleviate the computational burden of solving large dynamical systems by identifying latent low-dimensional structure in the system of interest, then exploiting that structure to construct a representative dynamical system in the identified low-dimensional space. The new system, called a reduced-order model (ROM), is less computationally expensive to solve than the original problem due to the dimensionality reduction. Our objective in this article is to use Operator Inference, a data-driven model reduction technique introduced in [15], to generate computationally efficient surrogates for the dynamical constraints of an optimization problem. We focus on a particular structure in the constraint equations, but the framework is highly generalizable and applies to a wide class of physics-constrained optimization problems.

Classical model reduction methods are *intrusive*, meaning they make a direct reduction of high-fidelity numerical solvers [2]. In contrast, Operator Inference is a *non-intrusive* (data-driven) model reduction strategy: the form or structure of the reduced model is dictated by the known governing dynamics, but the actual operators defining the ROM are learned through a residual-based regression on a set of observed states [9, 15]. This capability is essential for reducing systems for which we have a computationally expensive numerical solver for the original dynamics but no internal access to the solver code. To extend the Operator Inference framework to the control setting, we also consider the hybrid setting in which some (but not all) elements of the high-dimensional dynamics are known; in particular, we explicitly construct control terms through intrusive projection while learning reduced state operators through Operator Inference. Additional problem-specific information can be injected into the ROM by augmenting the intrinsic low-dimensional representation with

[§]Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, shanemcq@utexas.edu

[¶]Sandia National Laboratories, joshart@sandia.gov

[‖]Sandia National Laboratories, bartv@sandia.gov

[**]Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, kwillcox@oden.utexas.edu

desirable states. To derive a ROM that is tailored to a particular optimization problem, we ensure that target states defined by the objective function can be well represented by the reduced state. Thus, our contribution leverages Operator Inference in a goal-oriented fashion through physics-constrained optimization.

The remainder of the article is organized as follows. Section 2 outlines the traditional projection-based approach to model reduction and the Operator Inference framework. In Section 3, we accelerate constrained optimization problems by substituting the physics constraints with an appropriate ROM learned through Operator Inference. The methodology is demonstrated in Section 4 with two prototypical convection-diffusion control problems. Concluding remarks and an outline for future work are given in Section 5.

2. Projection-based Model Reduction. Our first objective is to construct a physics-based, computationally efficient surrogate model for a complex physical process. Such a surrogate will serve as a cost-effective constraint in an optimization problem of interest. Consider an ordinary differential equation (ODE)—obtained by spatially discretizing a partial differential equation (PDE)—that is linear in both the state and control,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}(t) = \mathbf{A}\mathbf{u}(t) + \mathbf{B}\mathbf{z}(t), \qquad \mathbf{u}(0) = \mathbf{u}_0, \qquad t \in [t_0, t_f], \tag{2.1}$$

where $\mathbf{u}(t) \in \mathbb{R}^m$ is the dynamic state with initial condition $\mathbf{u}_0 \in \mathbb{R}^m$, $\mathbf{z}(t) \in \mathbb{R}^n$ is a control vector, and $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B} \in \mathbb{R}^{m\times n}$ are the finite-dimensional operators that define the dynamics of the system. Our approach is general, and more complicated ODEs can be considered.

A reduced-order model of (2.1) is an ODE of reduced dimension which approximates its dynamics. The goal is to construct a ROM that achieves a computational speedup but with minimal accuracy loss in the state over the range of admissible controls. To do so, we require samples of the state solution space, which we will make precise shortly. We assume that a) the ODE (2.1) has a unique solution for all admissible controls $\mathbf{z}(t) \in \mathbb{R}^n$, and b) we have a numerical solver for (2.1) that we can use to generate solutions, albeit at high computational cost.

2.1. Intrusive Galerkin Projection. Projection-based model reduction methods approximate the state variable $\mathbf{u}(t)$ as a linear combination of $r \ll m$ vectors [2]. To that end, let $\mathbf{V} = \begin{bmatrix}\mathbf{v}_1 & \cdots & \mathbf{v}_r\end{bmatrix} \in \mathbb{R}^{m\times r}$ be a matrix with orthonormal columns and $\widehat{\mathbf{u}}(t) = \begin{bmatrix}\hat{u}_1(t) & \cdots & \hat{u}_r(t)\end{bmatrix}^\top \in \mathbb{R}^r$ be a time-dependent vector such that the state can be approximately represented as

$$\mathbf{u}(t) \approx \mathbf{V}\widehat{\mathbf{u}}(t) = \sum_{i=1}^{r} \hat{u}_i(t)\mathbf{v}_i. \tag{2.2}$$

We call $\mathbf{v}_i$ the basis vectors and $\mathbf{V}$ the basis matrix. Inserting this approximation into (2.1), multiplying both sides by $\mathbf{V}^\top$, and using the orthogonality of $\mathbf{V}$ yields the low-dimensional system

$$\frac{\mathrm{d}}{\mathrm{d}t}\widehat{\mathbf{u}}(t) = \widehat{\mathbf{A}}\widehat{\mathbf{u}}(t) + \widehat{\mathbf{B}}\mathbf{z}(t), \qquad \widehat{\mathbf{u}}(0) = \mathbf{V}^\top\mathbf{u}_0, \qquad t \in [t_0, t_f], \tag{2.3}$$

where $\widehat{\mathbf{A}} = \mathbf{V}^\top\mathbf{A}\mathbf{V} \in \mathbb{R}^{r\times r}$ and $\widehat{\mathbf{B}} = \mathbf{V}^\top\mathbf{B} \in \mathbb{R}^{r\times n}$. The ODE (2.3) is the (well-known) projection-based Galerkin ROM for (2.1) and is intrusive because computing $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{B}}$ requires explicit access to $\mathbf{A}$ and $\mathbf{B}$.

The basis matrix $\mathbf{V}$ can be computed in a variety of ways [2]; in this work, we use proper orthogonal decomposition (POD) [4, 10, 17], which takes the dominant left singular

vectors of an observed state space as the orthogonal basis. That is, given a set of state vectors $\{\mathbf{u}_j\}_{j=1}^k \subset \mathbb{R}^m$, we compute the singular value decomposition (SVD) of the states, $\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Psi}^\top = \left[\begin{array}{cccc} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k \end{array}\right] \in \mathbb{R}^{m\times k}$, and set $\mathbf{V}$ to the first r columns of $\boldsymbol{\Phi}$. The integer r is typically determined by the decay of the singular values $(\sigma_1, \ldots, \sigma_k) = \operatorname{diag}(\boldsymbol{\Sigma})$, for example by choosing the minimal number of singular values such that

$$\sum_{i=r+1}^{k} \sigma_i^2 \Bigg/ \sum_{j=1}^{k} \sigma_j^2 < \varepsilon \tag{2.4}$$

for some $\varepsilon > 0$. The left-hand side of (2.4) is interpreted as the amount of residual energy in the system that is neglected by truncating to r basis vectors.

2.2. Non-intrusive Operator Inference. As an alternative to intrusive projection, Operator Inference [15] seeks to learn a ROM of the form (2.3) by inferring $\widehat{\mathbf{A}}$ and/or $\widehat{\mathbf{B}}$ from the same data that is required to compute the POD basis. This kind of data-driven approach is needed when the high-dimensional operators $\mathbf{A}$ and/or $\mathbf{B}$ are unknown or otherwise unavailable for computation. See [16, 19] for examples of Operator Inference applied to more general (nonlinear) systems.

Let $\{(\mathbf{z}_j, \mathbf{u}_j, \dot{\mathbf{u}}_j)\}_{j=1}^k \subset \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$ be a set of k control-state-derivative triples satisfying $\dot{\mathbf{u}}_j = \mathbf{A}\mathbf{u}_j + \mathbf{B}\mathbf{z}_j$, which we obtain by solving the high-dimensional system (2.1). Since the $\dot{\mathbf{u}}_j \in \mathbb{R}^m$ represent the time derivative of the states, they may be estimated by finite differences if they are not provided by the numerical solver. The states $\{\mathbf{u}_j\}_{j=1}^k \subset \mathbb{R}^m$ are used to compute the POD basis $\mathbf{V} \in \mathbb{R}^{m\times r}$, then the states and the corresponding time derivatives are projected to the r-dimensional subspace defined by the basis as

$$\widehat{\mathbf{u}}_j = \mathbf{V}^\top \mathbf{u}_j, \qquad\qquad \dot{\widehat{\mathbf{u}}}_j = \mathbf{V}^\top \dot{\mathbf{u}}_j.$$

The projected data and the controls drive the learning of the ROM operators.

Consider first the task of learning both $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{B}}$, which is the standard problem in the Operator Inference literature. The learning problem is posed as the following minimal-residual regression:

$$\min_{\widehat{\mathbf{A}},\widehat{\mathbf{B}}} \sum_{j=1}^{k} \left\| \widehat{\mathbf{A}}\widehat{\mathbf{u}}_j + \widehat{\mathbf{B}}\mathbf{z}_j - \dot{\widehat{\mathbf{u}}}_j \right\|_2^2. \tag{2.5}$$

This is an ordinary least-squares problem with standard form

$$\min_{\widehat{\mathbf{A}},\widehat{\mathbf{B}}} \left\| \widetilde{\mathbf{D}} \left[\begin{array}{cc} \widehat{\mathbf{A}} & \widehat{\mathbf{B}} \end{array}\right]^\top - \widetilde{\mathbf{R}}^\top \right\|_F^2,$$

where $\|\cdot\|_F$ is the Frobenius matrix norm and

$$\widetilde{\mathbf{D}} = \left[\begin{array}{cccc} \widehat{\mathbf{u}}_1 & \widehat{\mathbf{u}}_2 & \cdots & \widehat{\mathbf{u}}_k \\ \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_k \end{array}\right]^\top \in \mathbb{R}^{k\times(r+m)},$$
$$\widetilde{\mathbf{R}} = \left[\begin{array}{cccc} \dot{\widehat{\mathbf{u}}}_1 & \dot{\widehat{\mathbf{u}}}_2 & \cdots & \dot{\widehat{\mathbf{u}}}_k \end{array}\right] \in \mathbb{R}^{r\times k}.$$

As long as $\widetilde{\mathbf{D}}$ has full column rank, the solution can be written explicitly as

$$\left[\begin{array}{cc} \widehat{\mathbf{A}} & \widehat{\mathbf{B}} \end{array}\right]^\top = \left(\widetilde{\mathbf{D}}^\top\widetilde{\mathbf{D}}\right)^{-1}\widetilde{\mathbf{D}}^\top\widetilde{\mathbf{R}}^\top.$$

Tikhonov regularization may be introduced to avoid overfitting in (2.5); we refer to [14,15,19] for details.

In optimization settings, the control $\mathbf{B}$ is often known a priori. To infer $\widehat{\mathbf{A}}$ while making use of a known $\mathbf{B}$, we set $\widehat{\mathbf{B}} = \mathbf{V}^\top \mathbf{B} \in \mathbb{R}^{r\times n}$ as in the intrusive case and consider the regression problem

$$\min_{\widehat{\mathbf{A}}} \sum_{j=1}^{k} \left\| \widehat{\mathbf{A}}\widehat{\mathbf{u}}_j + \widehat{\mathbf{B}}\mathbf{z}_j - \dot{\widehat{\mathbf{u}}}_j \right\|_2^2. \tag{2.6}$$

Note that (2.6) is identical to (2.5), except that $\widehat{\mathbf{A}} \in \mathbb{R}^{r\times r}$ is the only unknown. The standard form of this problem is

$$\min_{\widehat{\mathbf{A}}} \left\| \mathbf{D}\widehat{\mathbf{A}}^\top - \mathbf{R}^\top \right\|_F^2,$$

where now

$$\mathbf{D} = \begin{bmatrix} \widehat{\mathbf{u}}_1 & \widehat{\mathbf{u}}_2 & \cdots & \widehat{\mathbf{u}}_k \end{bmatrix}^\top \in \mathbb{R}^{k\times r},$$
$$\mathbf{R} = \begin{bmatrix} \dot{\widehat{\mathbf{u}}}_1 - \widehat{\mathbf{B}}\mathbf{z}_1 & \dot{\widehat{\mathbf{u}}}_2 - \widehat{\mathbf{B}}\mathbf{z}_2 & \cdots & \dot{\widehat{\mathbf{u}}}_k - \widehat{\mathbf{B}}\mathbf{z}_k \end{bmatrix} \in \mathbb{R}^{r\times k}.$$

Similar to the previous case, the solution is explicitly given by

$$\widehat{\mathbf{A}}^\top = \left(\mathbf{D}^\top\mathbf{D}\right)^{-1}\mathbf{D}^\top\mathbf{R}^\top,$$

provided $\mathbf{D}$ has full column rank. Incorporating the known $\mathbf{B}$ in this fashion makes the size of the regression problem independent of the control dimension n, which is important for problems with high dimensional controls (as frequently occurs when controls are spatially distributed in PDEs).

3. Optimization with Reduced-order Constraints. Reduced-order models yield the greatest benefit in situations where the high-cost model of interest must be evaluated many times with various initial conditions or control schemes. Modern constrained optimization routines require multiple forward and adjoint solves of the constraint equations to make progress [1,5,6,9,13]; if this is computationally expensive, it is highly advantageous to replace the constraint with an efficient ROM. This approach is not new (see, e.g., [3,7,8,20]), but our use of Operator Inference to construct a ROM of the constraint in an optimization formulation is novel. In this section, we consider the constrained optimization problem

$$\texttt{FullOpt} \quad \begin{cases} \min\limits_{\mathbf{u}\in\mathbb{R}^m, \mathbf{z}\in\mathbb{R}^n} J(\mathbf{u},\mathbf{z}) & \text{(3.1a)} \\ \text{s.t.} \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}(t) = \mathbf{A}\mathbf{u}(t) + \mathbf{B}\mathbf{z}(t), & \text{(3.1b)} \end{cases}$$

where J is a scalar-valued function of the dynamic state and control. For demonstration purposes we write the constraint the same form as (2.1), but we emphasize again that the framework can be generalized to more complicated ODE structures [14,16,19].

To overcome computational limitations associated with the high-dimensional constraint equations, we replace (3.1b) with a ROM of the form (2.3). This yields the new problem

$$\texttt{RomOpt} \quad \begin{cases} \min\limits_{\widehat{\mathbf{u}}\in\mathbb{R}^r, \mathbf{z}\in\mathbb{R}^n} J(\mathbf{V}\widehat{\mathbf{u}},\mathbf{z}) & \text{(3.2a)} \\ \text{s.t.} \quad \frac{\mathrm{d}}{\mathrm{d}t}\widehat{\mathbf{u}}(t) = \widehat{\mathbf{A}}\widehat{\mathbf{u}}(t) + \widehat{\mathbf{B}}\mathbf{z}(t), & \text{(3.2b)} \end{cases}$$

which we solve to approximate the solution of the original problem `FullOpt`. This substitution introduces model form error from the ROM that then propagates through the optimization process; part of our ongoing work is to explore how to mitigate this error through the use of post-optimality sensitivity analysis [11].

If the objective function J has an explicit representation, we can incorporate known information from J into the basis $\mathbf{V}$ a priori so that the resulting Operator Inference problem is cognizant of the optimization goal. There are several ways to manage this; we elect to use the follow strategy and leave alternatives to future work. Suppose J measures the similarity of the state to a target profile,

$$J(\mathbf{u},\mathbf{z}) = \int_{t_0}^{t_f} \frac{1}{2}\left\|\mathbf{g}(t)-\mathbf{u}(t)\right\|^2 + \frac{\beta}{2}\left\|\mathbf{B}\mathbf{z}(t)\right\|^2 \, dt, \tag{3.3}$$

where $\mathbf{g} : \mathbb{R} \to \mathbb{R}^m$ is the target function, $\beta > 0$ is a regularization constant, and the norm $\|\cdot\|$ encodes a quadrature rule on the spatial mesh. The cost reduction of the ROM (3.2b) comes from expressing the state as the linear combination of relatively few elements in (2.2). For `RomOpt` to be an accurate approximation of `FullOpt` with J as in (3.3), it is critical that the reduced state be able to represent the target function $\mathbf{g}(t)$. In other words, it is desirable that $\mathbf{g}(t) \in \text{range}(\mathbf{V})$. Since $\mathbf{g}(t)$ is known, we can augment the basis matrix $\mathbf{V}$ obtained through POD by adding orthonomal components of $\mathbf{g}(t)$ to the column space. We first compute the principle components of the target profile by computing the SVD $\mathbf{\Phi}_g\mathbf{\Sigma}_g\mathbf{\Psi}_g^\top = \begin{bmatrix} \mathbf{g}(t_0) & \mathbf{g}(t_1) & \cdots & \mathbf{g}(t_f) \end{bmatrix}$, then appending the first few columns of $\mathbf{\Phi}_g$ to $\mathbf{V}$ in orthonormal fashion using the Gram-Schmidt procedure. An appropriate number of columns can be selected, for example, as in the strategy of (2.4). The resulting basis is then used to construct the ROM (3.2b) via Operator Inference as described in Section 2.2.

4. Numerical Examples. We now demonstrate the methodology on two source control problems constrained by convection-diffusion physics. Though relatively simple, these problems are ideal prototypes for more complicated applications.

4.1. Convection-Diffusion in One Dimension. We consider the problem `FullOpt` with objective function (3.3), target profile $g(x,t) = t\cos(2\pi x)$, and constrained by the following PDE in one spatial dimension:

$$\frac{\partial}{\partial t}u(x,t) = \kappa\frac{\partial^2}{\partial x^2}u(x,t) - \alpha\frac{\partial}{\partial x}u(x,t) + \sum_{j=1}^{n} z_j(t)b_j(x), \qquad x\in[0,1],\ t\in(0,1],$$
$$u(x,0) = \frac{\partial}{\partial x}u(x,t)\big|_{x=0} = \frac{\partial}{\partial x}u(x,t)\big|_{x=1} = 0,$$

where $\kappa > 0$ and $\alpha \in \mathbb{R}$ are diffusion and convection constants, respectively. The functions $b_j(x) = \exp\left(-10(x-x_j)^2\right)$ are Gaussian source terms with associated control coefficients $z_j(t)$, where $0 = x_1 < x_2 < \cdots < x_{n-1} < x_n = 1$ is an equidistant grid of the spatial domain. Discretizing the state with m points leads to a high-fidelity model of the form (2.1), with $\mathbf{A} \in \mathbb{R}^{m\times m}$ representing convection and diffusion, $\mathbf{z}(t) = \begin{bmatrix} z_1(t) & \cdots & z_n(t) \end{bmatrix}^\top \in \mathbb{R}^n$, and $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^{m\times n}$ where $\mathbf{b}_j \in \mathbb{R}^m$ is the spatial discretization of $b_j(x)$.

For our numerical experiments, we discretize the spatial domain with finite differences on $m = 80$ points, use $n = 30$ Gaussian sources, and set $\kappa = \alpha = 1$. The high-fidelity model (3.1b) is integrated once over $k = 100$ time steps with the implicit Euler method to generate training data from which a POD basis with $r = 4$ modes is computed (r chosen by setting $\varepsilon = 10^{-12}$ in (2.4)). We explicitly construct and project $\mathbf{B}$ but learn a reduced operator corresponding to $\mathbf{A}$ with Operator Inference. To study the effect of the basis augmentation

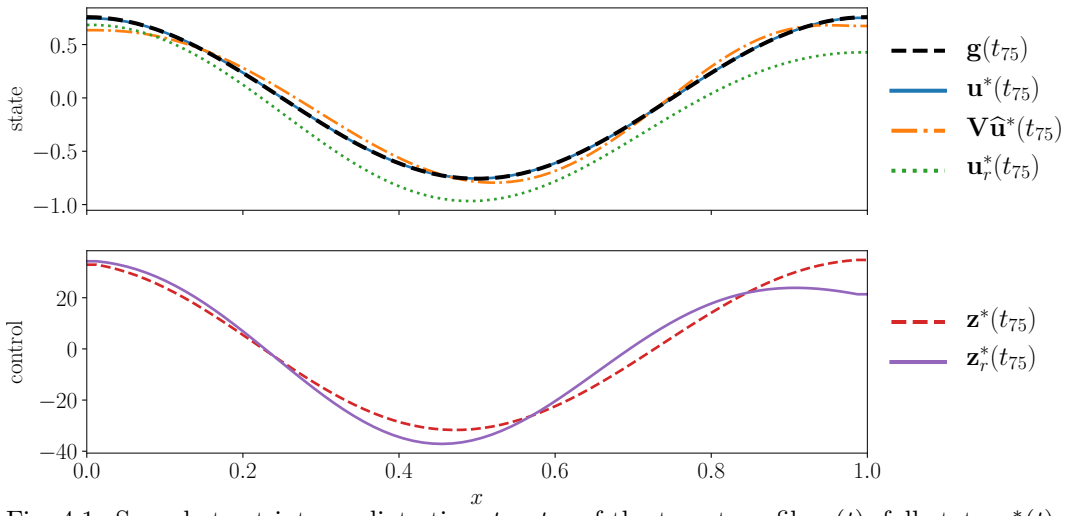


Fig. 4.1: Snapshots at intermediate time $t = t_{75}$ of the target profile $\mathbf{g}(t)$, full state $\mathbf{u}^*(t)$, reduced state $\mathbf{V}\widehat{\mathbf{u}}^*(t)$, and reconstructed full state $\mathbf{u}_r^*(t)$ (top), along with the full and reduced control solutions $\mathbf{z}^*(t)$ and $\mathbf{z}_r^*(t)$ (bottom), using the standard POD basis with $r = 4$ modes.

strategy, we construct the ROM using 1) the usual POD basis computed from state data alone, and 2) the POD basis augmented with the dominant mode of the discretized target profile. In each case, we first solve the high-fidelity problem `FullOpt` via a trust region conjugate gradient solver, obtaining the (optimal) control profile $\mathbf{z}^*(t)$ and the corresponding state $\mathbf{u}^*(t)$. This requires $\mathcal{O}(100)$ solves of the m-dimensional ODE (3.1b). We similarly solve the problem `RomOpt`, which has reduced-order constraints, yielding the control profile $\mathbf{z}_r^*(t)$ and the state $\mathbf{V}\widehat{\mathbf{u}}^*(t)$. This only requires a single solve of the m-dimensional ODE system (3.1b) (to generate training data) and $\mathcal{O}(100)$ solves of the r-dimensional ROM (3.2b), resulting in a significant computational speedup. Finally, we also evaluate the high-fidelity constraint (3.1b) with the control $\mathbf{z}_r^*(t)$ to produce the state $\mathbf{u}_r^*(t)$. Figures 4.1–4.2 compare the various states and controls in the case where the basis has not been augmented with the target profile; Figures 4.2–4.3 show results for the case when the basis has been augmented with the target.

Figure 4.1 plots a representative snapshot in time of the target profile $\mathbf{g}(t)$, the high-fidelity state solution $\mathbf{u}^*(t)$, the ROM state solution $\mathbf{V}\widehat{\mathbf{u}}^*(t)$, and the state $\mathbf{u}_r^*(t)$ corresponding to the high-fidelity constraint with the ROM-driven control profile. The high-fidelity state is indistinguishable from the target profile, but there are visible errors in the ROM state. The control solutions $\mathbf{z}^*(t)$ of `FullOpt` and $\mathbf{z}_r^*(t)$ of `RomOpt` differ throughout the time domain by about 20%. Figure 4.2 plots the absolute state error in time, which show that the ROM solution is an order of magnitude less accurate than the high-fidelity solution. Augmenting the POD basis with the target profile has a significant impact on the performance of `RomOpt`: Figures 4.2 and 4.3 show that the ROM state solution now matches the target almost exactly as well as the high-fidelity model, and the control solutions $\mathbf{z}^*(t)$ and $\mathbf{z}_r^*(t)$ now differ by only about 10% throughout the time domain. However, using `RomOpt` to determine the control solution and plugging that solution back into the high-fidelity model (3.1b) (labeled $\mathbf{u}_r^*(t)$) increases the state-target error, as shown in Figure 4.2. This is not unexpected and is indicative of the propagation of error introduced by the dimension

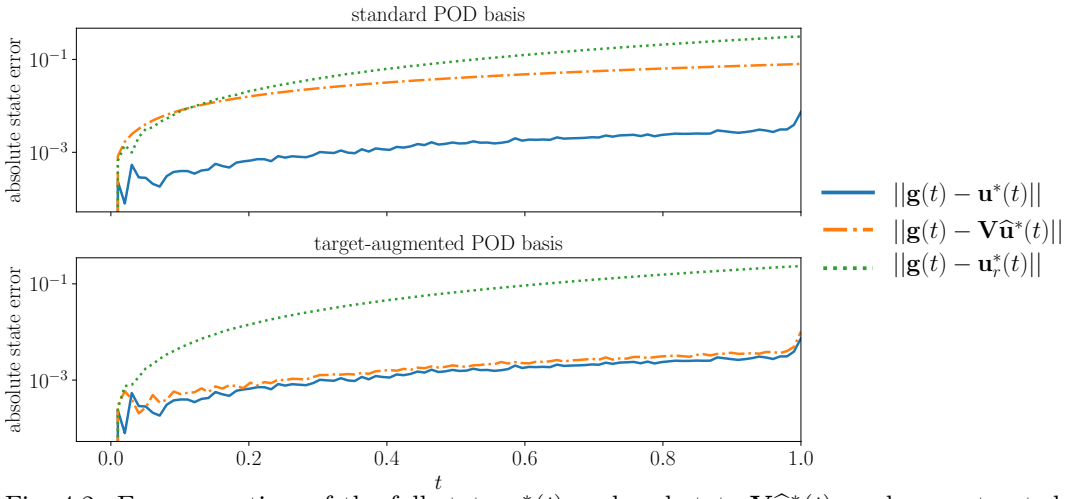


Fig. 4.2: Error over time of the full state $\mathbf{u}^*(t)$, reduced state $\mathbf{V}\widehat{\mathbf{u}}^*(t)$, and reconstructed full state $\mathbf{u}_r^*(t)$ compared to the target $\mathbf{g}(t)$, using the standard POD basis with $r = 4$ modes (top) and the POD basis augmented with an additional mode representing the target (bottom).

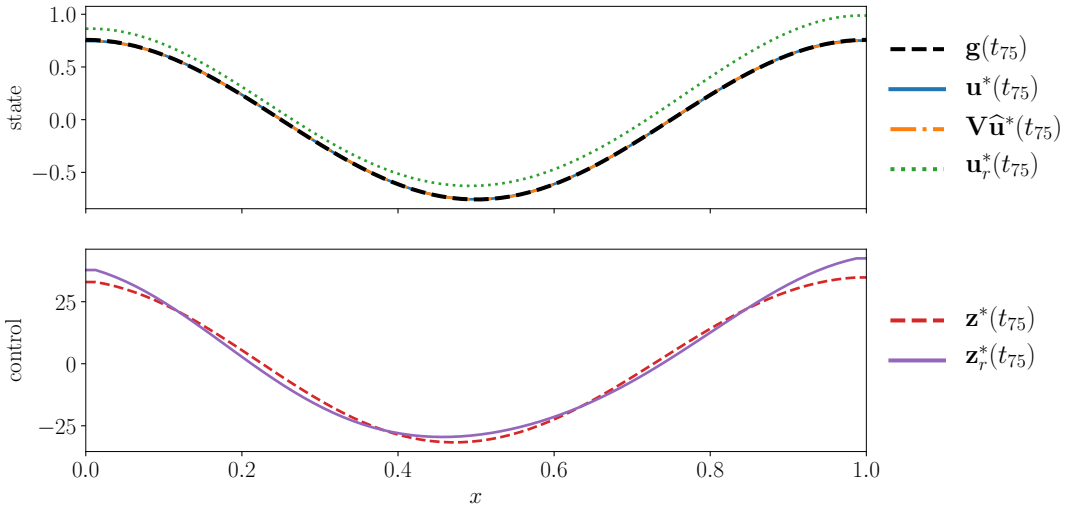


Fig. 4.3: Snapshots at intermediate time $t = t_{75}$ of the target profile $\mathbf{g}(t)$, full state $\mathbf{u}^*(t)$, reduced state $\mathbf{V}\widehat{\mathbf{u}}^*(t)$, and reconstructed state $\mathbf{u}_r^*(t)$ (top), along with full and reduced control solutions $\mathbf{z}^*(t)$ and $\mathbf{z}_r^*(t)$ (bottom), using a POD basis augmented by the target profile.

reduction. Nevertheless, the real-time performance benefit and accuracy of `RomOpt` are significant given that the ROM is constructed with relatively limited training data.

4.2. Convection-Diffusion in Two Dimensions. We next examine a problem with a more complex computational infrastructure, with an eye toward tackling applications in future work that require scalable solvers and sophisticated software. We again consider the

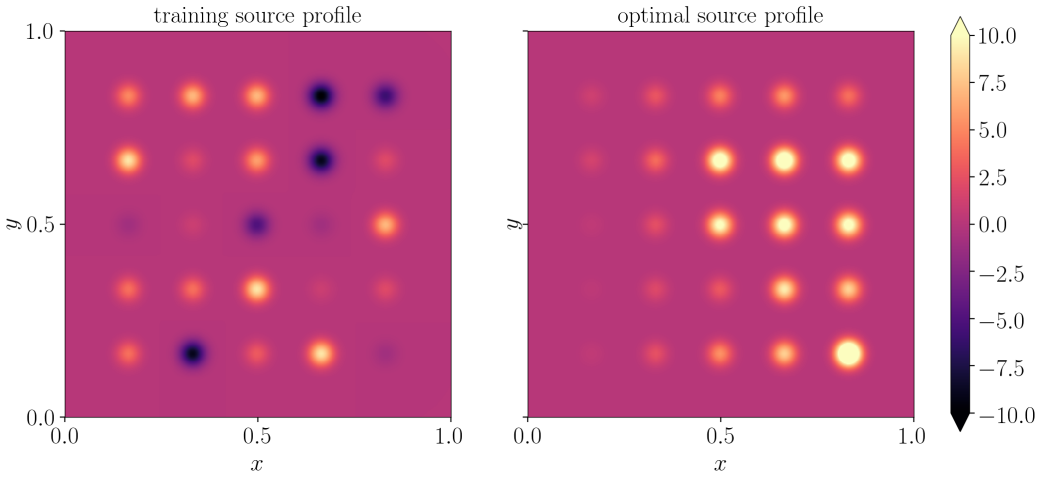


Fig. 4.4: Source profile with the control used to generate training data for Operator Inference (left) and with the optimal control scheme determined by solving the optimization problem with ROM constraints in the two-dimensional convection-diffusion system.

problem `FullOpt`, but with the objective function

$$J(\mathbf{u}, \mathbf{z}) = \frac{1}{2} \left\| \mathbf{u}(t_f) - \mathbf{g} \right\|^2 + \frac{\beta}{2} \left\| \mathbf{B}\mathbf{z} \right\|^2, \tag{4.1}$$

where $\mathbf{g} \in \mathbb{R}^m$ is a discretization of the target function $g(x, y) = t_f x$ and $\beta > 0$ is a regularization constant as before. The control vector $\mathbf{z} \in \mathbb{R}^n$ is independent of time, and the state $\mathbf{u}(t) \in \mathbb{R}^m$ is a spatial discretization of $u(x, y, t)$ satisfying the PDE

$$\begin{aligned} \frac{\partial u}{\partial t} &= \kappa \Delta u - \mathbf{v} \cdot \nabla u + \sum_{j=1}^{n} z_j b_j(x, y), && x \in \Omega,\ t \in (t_0, t_f], \\ \boldsymbol{\eta} \cdot \nabla u &= 0, && x \in \partial\Omega,\ t \in (t_0, t_f), \\ u(x, 0) &= 0, && x \in \Omega, \end{aligned}$$

where $\Omega = (0, 1) \times (0, 1)$ is the 2D spatial domain with boundary $\partial\Omega$ and outward-pointing normal $\boldsymbol{\eta} \in \mathbb{R}^2$, $\kappa > 0$ is a diffusion constant, and $\mathbf{v} \in \mathbb{R}^2$ specifies constant x- and y-velocities. The source functions $b_j(x, y) = \exp\left(-\frac{1}{1000}((x - x_j)^2 + (y - y_j)^2)\right)$ are isotropic Gaussians centered at equidistantly spaced interior points (see Figure 4.4). Similar to the previous example, spatially discretizing the state yields the system (2.1) where $\mathbf{A} \in \mathbb{R}^{m \times m}$ encodes the convection and diffusion operators, $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$ concatenates the spatially discretized source functions $b_j(x, y)$, and $\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}^\top \in \mathbb{R}^n$ is the control vector.

We discretize the governing equations with 40×40 finite elements and $n = 25$ sources and set $\kappa = 0.1$ and $\mathbf{v} = \begin{bmatrix} 0.8 & -0.2 \end{bmatrix}^\top$. The PDE constraint with the control shown in Figure 4.4 is solved in a parallelized `C++` framework, and the resulting states are used to compute a POD basis of $r = 14$ modes. We then augment the basis with the target profile and learn a ROM with Operator Inference for use in the reduced optimization problem, `RomOpt`. In our current computational framework, the control matrix $\mathbf{B}$ is easy to construct explicitly, but the diffusion and convection operators are not as immediately accessible from

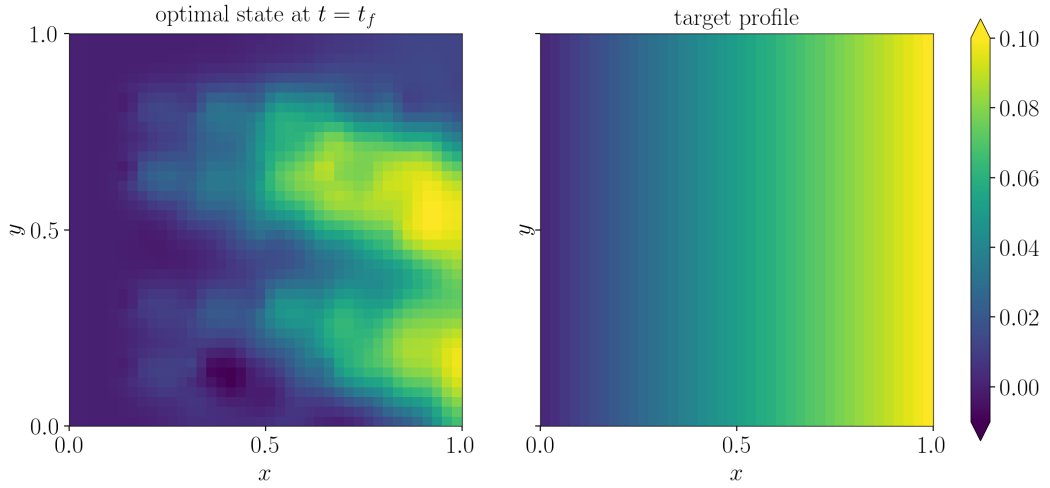


Fig. 4.5: Optimal state profile at the final time (left) and the linear target profile (right) for the two-dimensional convection-diffusion optimization problem with ROM constraints.

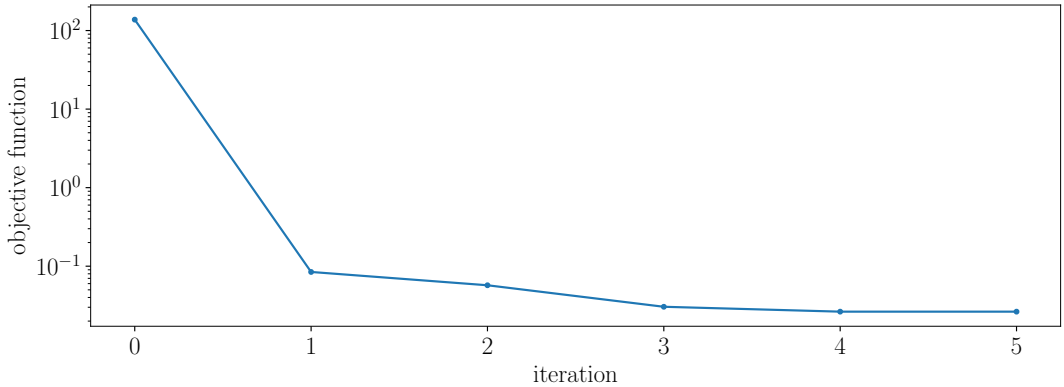


Fig. 4.6: Value of the objective function (4.1) at each iteration of the optimization routine.

the implementation. In more complex problems with black-box solvers for the constraint equations, the requisite matrices may not be accessible at all. This makes Operator Inference advantageous in the workflow of the problem, since we are not required to assemble $\mathbf{A}$ explicitly.

Figure 4.4 shows the control solution to `RomOpt` with objective (4.1), and Figure 4.5 compares the associated end state $\mathbf{u}(t_f)$ to the target profile $\mathbf{g}$. The control solution is reasonable but clearly suboptimal. Figure 4.6 shows that the objective function (4.1) does decrease by several orders of magnitude as the optimization routine progresses, but Figure 4.5 makes it clear that the quality of the ROM-constrained end state varies throughout the domain because not all features of the governing dynamics are represented in the limited training set. Despite the limitations, these preliminary results are encouraging considering the ROM is derived from a single high-fidelity solve. In addition, the use of time-dependent controllers will increase controllability and combined with a more heterogeneous target, the results will reveal more insight.

5. Conclusions and Future Work. Control problems with dynamical constraints are ubiquitous in computational science and engineering, but such problems are often computationally expensive due to the high dimensionality of the constraint equations. In this work we leveraged Operator Inference, a model reduction framework for generating reduced-order models from known physics and observed data, not from intrusive modifications of existing codes. Specifically, we specialized Operator Inference to the physics-constrained optimization setting by 1) incorporating known control data into the inference problem and 2) expanding the underlying basis to be able to represent well the target profile(s) defined by the minimization objective.

We presented two prototypical convection-diffusion examples that utilize Operator Inference ROMs as surrogates for the optimization constraints. In the first example, we improved the real-time performance of the optimization routine without sacrificing accuracy loss in the evolution of the state, although some error remains in the ROM-driven control solution and the corresponding solution to the high-fidelity model. We connected all of the mechanical components of the methodology in a more demanding computational framework for the second example and also achieved a real-time performance gain in that setting. The reduced-order model solutions in this problem have room for improvement due to the limited expressiveness of the time-independent controls and the small size of the training set used to drive the reduced model learning.

Given the successes and current limitations of our ROM-driven optimization paradigm, the stage is set for the next important step toward solving mission critical problems at scale: using post-optimality analyses to diagnose and correct the model form error introduced by the ROM and intelligently guide the collection of further training data [11]. We plan to leverage hyper-differential sensitivity analysis (HDSA) [12, 18] for these purposes, which will aid in reducing the error in the ROM-constrained optimization while preserving the computational efficiency of only requiring a small number of solves with the high-fidelity model.

Acknowledgments. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND2021-XXXX. This work was supported by the US Department of Energy, Office of Advanced Scientific Computing Research, Field Work Proposal 20-023231.

References.

[1] H. Antil, D. P. Kouri, M. Lacasse, and D. Ridzal, eds., *Frontiers in PDE-Constrained Optimization*, Springer, 2018.

[2] P. Benner, S. Gugercin, and K. Willcox, *A survey of projection-based model reduction methods for parametric dynamical systems*, SIAM Review, 57 (2015), pp. 483–531.

[3] P. Benner, E. Sachs, and S. Volkwein, *Model order reduction for PDE constrained optimization*, in Trends in PDE Constrained Optimization, G. Leugering, P. Benner, S. Engell, A. Griewank, H. Harbrecht, M. Hinze, R. Raanacher, and S. Ulbrich, eds., Cham, 2014, Springer International Publishing, pp. 303–326.

[4] G. Berkooz, P. Holmes, and J. Lumley, *The proper orthogonal decomposition in the analysis of turbulent flows*, Annual Review of Fluid Mechanics, 25 (1993), pp. 539–575.

[5] L. T. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, eds., *Real-Time PDE-Constrained Optimization*, vol. 3, SIAM Computational Science and Engineering, 2007.

[6] G. Biros and O. Ghattas, *Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization. Parts I-II*, SIAM Journal of Scientific Computing, 27 (2005), pp. 687– 738.

[7] T. Bui-Thanh, K. Willcox, O. Ghattas, and B. van Bloemen Waanders, *Goal-oriented, model-constrained optimization for reduction of large-scale systems*, Journal of Computational Physics, 224 (2007), pp. 880–896.
[8] M. Fahl and E. W. Sachs, *Reduced order modelling aproaches to PDE-constrained optimization based on proper orthogonal decomposition*, in Large-Scale PDE-Constrained Optimization, L. T. Biegler, M. Heinkenschloss, O. Ghattas, and B. van Bloemen Waanders, eds., Berlin, Heidelberg, 2003, Springer Berlin Heidelberg, pp. 268–280.
[9] O. Ghattas and K. Willcox, *Learning physics-based models from data: perspectives from inverse problems and model reduction*, Acta Numerica, 30 (2021), pp. 445–554.
[10] W. R. Graham, J. Peraire, and K. Y. Tang, *Optimal control of vortex shedding using low-order models. Part I—Open-loop model development*, International Journal for Numerical Methods in Engineering, 44 (1999), pp. 945–972.
[11] J. Hart and B. van Bloemen Waanders, *Hyper-differential sensitivity analysis with respect to model form error*, (2021). In preparation.
[12] J. Hart, B. van Bloemen Waanders, and R. Herzog, *Hyperdifferential sensitivity analysis of uncertain parameters in PDE-constrained optimization*, International Journal for Uncertainty Quantification, 10 (2020), pp. 225–248.
[13] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*, Springer, 2009.
[14] S. A. McQuarrie, C. Huang, and K. E. Willcox, *Data-driven reduced-order models via regularised operator inference for a single-injector combustion process*, Journal of the Royal Society of New Zealand, 51 (2021), pp. 194–211.
[15] B. Peherstorfer and K. Willcox, *Data-driven operator inference for nonintrusive projection-based model reduction*, Computer Methods in Applied Mechanics and Engineering, 306 (2016), pp. 196–215.
[16] E. Qian, B. Kramer, B. Peherstorfer, and K. Willcox, *Lift & Learn: Physics-informed machine learning for large-scale nonlinear dynamical systems.*, Physica D: Nonlinear Phenomena, 406 (2020), p. 132401.
[17] L. Sirovich, *Turbulence and the dynamics of coherent structures. I. Coherent structures*, Quarterly of Applied Mathematics, 45 (1987), pp. 561–571.
[18] I. Sunseri, J. Hart, B. van Bloemen Waanders, and A. Alexanderian, *Hyper-differential sensitiviy analysis for inverse problems constrained by partial differential equations*, Inverse Problems, 36 (2020), p. 125001.
[19] R. Swischuk, B. Kramer, C. Huang, and K. Willcox, *Learning physics-based reduced-order models for a single-injector combustion process*, AIAA Journal, 58 (2020), pp. 2658–2672.
[20] M. J. Zahr and C. Farhat, *Progressive construction of a parametric reduced-order model for PDE-constrained optimization*, International Journal for Numerical Methods in Engineering, 102 (2015), pp. 1111–1135.

COMPARISON OF TEMPERED AND TRUNCATED FRACTIONAL MODELS

HAYLEY A. OLSON*, MARTA D'ELIA†, MIKIL FOSS‡, MAMIKON GULIAN§, AND PETRONELA RADU¶

Abstract. Tempered fractional operators are able to model effects that classical partial differential equations cannot capture, such as the super- and sub-diffusive effects that are present in hydrology and geophysics models. However, tempered fractional operators are computationally intensive due to the infinite range of interaction for the integral operator. We analyze a truncated variation of the fractional operators, which are less computationally intensive, in an effort to use them in place of the more complex tempered variation. In particular, we train parameters of the truncated operator using neural networks in order to optimize the difference of the actions of the two operators.

1. Introduction. Fractional models are nonlocal models that can be utilized in place of standard partial differential equations (PDEs) when trying to model anomalous effects that standard PDEs fail to describe. For example, fractional models have found applications in subsurface diffusion and transport, turbulence, and machine-learning algorithms. Here, we are considering the tempered variation of the fractional Laplacian introduced in [5] which has applications in; e.g. hydrology and geophysics [1, 6].

These nonlocal operators, such as the tempered fractional Laplacian, are integral operators that act on a so-called "interaction horizon" that determine the radius of interaction between points in the domain. The tempered fractional Laplacian has an infinite interaction horizon. This is useful for modelling long range forces and it reduces the regularity requirements on the solution. However, the infinite interaction horizon causes the operator to be computationally complex. This work explores whether a similar operator can be generated that mimics the tempered fractional Laplacian while being less expensive computationally.

In particular, we investigate a truncated version of the fractional Laplacian, restricting the interaction horizon to a ball of radius $0 < \delta < \infty$. This truncation allows for a reduction in the computational cost of the numerical evaluation of the operator. The authors have studied these two operators previously in [3] and determined that the nonlocal fractional energy norms of the tempered fractional Laplacian and truncated fractional Laplacian, in its simplest form, are equivalent. In this work, we introduce a modified form of the truncated fractional Laplacian and we parametrize it with the goal of identifying the parameters that minimize the difference of the actions of the operators. Similar parameter identification problems for fractional models can be found in e.g. [2, 8]. In this work the parameters are trained using deep neural networks (DNNs). Machine learning has been used to tackle many aspects of nonlocal models, we mention [4] as an example of the use of DNNs for the approximation of the solution of a nonlocal equation.

This report is organized as follows. Section 2 includes relevant definitions and previous results that will be referenced throughout. The formulation of the problem as a loss function to optimize and characterization of the learned parameters is defined in Section 3. Section 4 has information about the discretization of the problem for the numerical analysis. Computational results of learning the parameters can be found in Section 5 with conclusions in Section 6.

*University of Nebraska-Lincoln, hayley.olson@huskers.unl.edu
†Sandia National Laboratories, mdelia@sandia.gov
‡University of Nebraska-Lincoln, mikil.foss@unl.edu
§Sandia National Laboratories, mgulian@sandia.gov
¶University of Nebraska-Lincoln, pradu@unl.edu

2. Notation and Previous Work. Let $\Omega \in \mathbb{R}^n$ be an open bounded domain. Define the corresponding *interaction domain* as

$$\begin{aligned}\Omega_I &= \{\mathbf{y} \in \mathbb{R}^n \backslash \Omega \text{ such that } \mathbf{x} \text{ interacts with } \mathbf{y} \text{ for some } \mathbf{x} \in \Omega\} \\ &= \{\mathbf{y} \in \mathbb{R}^n \backslash \Omega : |\mathbf{x}-\mathbf{y}| \leq \delta \text{ for some } \mathbf{x} \in \Omega\},\end{aligned}$$

where $\delta > 0$ is the so-called interaction radius or horizon. For the tempered fractional operator, the operator has an infinite radius of interaction. Hence, $\delta = \infty$ and thus $\Omega_I = \mathbb{R}^n \setminus \Omega$.

We will be considering two fractional operators of the general form

$$\mathcal{L}u(\mathbf{x}) = \int_{\Omega \cup \Omega_I} \frac{u(\mathbf{y}) - u(\mathbf{x})}{|\mathbf{x}-\mathbf{y}|^{n+2s}} \gamma(\mathbf{x},\mathbf{y}) d\mathbf{y},$$

where n is the dimension and $0 < s < 1$ is the fractional order. In particular, we will consider the tempered fractional Laplacian, introduced in [5], which has the kernel

$$\gamma_{\text{tem}}(\mathbf{x},\mathbf{y};\lambda) = e^{-\lambda|\mathbf{x}-\mathbf{y}|} \tag{2.1}$$

alongside a truncated version of the fractional Laplacian which, in the simplest form, has the kernel

$$\gamma_{\text{tr}}(\mathbf{x},\mathbf{y};\sigma,\delta) = \sigma \mathcal{X}(|\mathbf{x}-\mathbf{y}| < \delta), \tag{2.2}$$

where σ and δ are positive real numbers that represent a scaling constant and the horizon of truncation, respectively. The function $\mathcal{X}(\cdot)$ represents the indicator function; thus, the support of the truncated kernel is limited to a Euclidean ball of radius δ centered at $\mathbf{x}$.

With the purpose of improving the descriptive power of the truncated operator, we propose a modified version of γ_{tr} (that we denote by the same symbol) where both the scaling parameter and the horizon depend on the space variable. Keeping in mind that we will use optimization algorithms to learn these parameters, we substitute the indicator function with a smooth approximation, so as to facilitate the use of available machine-learning tools (this is discussed in more detail in Section 4). Thus, we define the new truncated kernel as follows:

$$\gamma_{\text{tr}}(\mathbf{x},\mathbf{y};\sigma,\delta) = \sigma(\mathbf{x})\eta(|\mathbf{x}-\mathbf{y}|,\delta(\mathbf{x})), \tag{2.3}$$

where σ and δ are now space-dependent parameters. Here, for the sigmoid function $S(x) = \frac{1}{1+e^{-x}}$ and a fixed sharpness parameter $\alpha > 0$, we define η as

$$\eta(|\mathbf{x}-\mathbf{y}|,\delta(\mathbf{x})) = 1 - S(\alpha(|\mathbf{x}-\mathbf{y}| - \delta(\mathbf{x}))). \tag{2.4}$$

The fractional Laplacians with the tempered and truncated kernels will be referred to as $\mathcal{L}_{\text{tem}}$ and $\mathcal{L}_{\text{tr}}$, respectively.

Note that the use of the sigmoid function, supported in $\mathbb{R}$, might seem in contrast with the goal of restricting the domain of integration in the definition of the integral operator. However, such function decays to very small values which are negligible at the numerical level, i.e. values that are below machine precision. Thus, when evaluating the sigmoid-truncated operator, one is allowed to restrict the integration domain to the set corresponding to function values above machine precision.

The tempered fractional Laplacian and the truncated fractional Laplacian in the simple form of equation (2.2) have been compared previously in [3], where their energy norms are

shown to be equivalent. We summarize that result below. For a kernel γ, the *nonlocal fractional energy norm* is defined as

$$E(u;\mu) = \iint_{(\Omega\cup\Omega_I)^2} \frac{(u(\mathbf{x})-u(\mathbf{y}))^2}{|\mathbf{x}-\mathbf{y}|^{n+2s}}\gamma(\mathbf{x},\mathbf{y},\mu)d\mathbf{y}d\mathbf{x} \tag{2.5}$$

where μ is a parameter that determines the kernel. Then for γ_{tem} and γ_{tr} as defined in (2.1) and (2.2), we have the following result [3].

THEOREM 2.1. *There exist positive constants* $\underline{A}$ *and* $\overline{A}$ *such that, given* $\lambda > 0$,

$$\underline{A}E_{\text{tr}}(u;\delta) \le E_{\text{tem}}(u;\lambda) \le \overline{A}E_{\text{tr}}(u;\delta), \quad \forall\, u \in H^s_\Omega(\mathbb{R}^n),\ \delta < \infty. \tag{2.6}$$

This result, although valid only for the symmetric kernel in (2.2), provides the foundation for our proposed kernel representation and sets the groundwork for its mathematical analysis, which is the subject of our future work.

3. Formulation of the learning problem as the minimization of a loss function. We describe our procedure in a one-dimensional setting and we refer to the representation in equation (2.3). The goal is to learn functions $\sigma(x)$ and $\delta(x)$ such that they minimize the difference of the actions of the tempered and truncated fractional Laplace operators. We consider the operators acting on a set of training functions $\{u_i\}_{i=1}^N$ and minimize the following loss function, which is a percent error of the L^2 norm of the difference of the operators,

$$\text{Loss}(\delta,\sigma) = \frac{1}{N}\sum_{u_i} \frac{\|\mathcal{L}_{\text{tem}}u_i(x) - \mathcal{L}_{\text{tr}}u_i(x)\|_{L^2(\Omega)}}{\|\mathcal{L}_{\text{tem}}u_i(x)\|_{L^2(\Omega)}} \quad \text{for } x \in [-A, A]. \tag{3.1}$$

The training functions are constructed as linear combinations of basis functions, chosen with increasing levels of complexity and different regularity properties. In a one-dimensional setting, they are defined as follows. Let $\{x_i\}_{i=1}^N$ be a uniform partition of $[-a, a]$ and Δx the distance between the points. Note that $a \neq A$; indeed, in our numerical tests, $A < a$. The following functions are utilized, for some covering or scale parameter c:

1. Linear hat functions ($C^0(\mathbb{R})$):

$$u_i(x) = \begin{cases} \frac{1}{c}(x - x_i) + 1, & x_i - c < x \le x_i \\ \frac{-1}{c}(x - x_i) + 1, & x_i < x < x_i + c \\ 0, & \text{else.} \end{cases} \tag{3.2}$$

2. Exponential bump functions ($C^\infty(\mathbb{R})$):

$$u_i(x) = \begin{cases} \exp\left(\frac{c^2}{(c)^2-(x-x_i)^2}\right), & x_i - c < x < x_i + c \\ 0, & \text{else.} \end{cases} \tag{3.3}$$

3.1. Characterization of σ and δ. The parameters σ and δ are learned as the outputs of a deep neural networks (DNNs), i.e. $\delta(x) = DNN_1(x)$ and $\sigma(x) = DNN_2(x)$. The networks have the following structure. In addition, in some tests the parameter δ is trained as a constant.

The DNN for $\sigma(x)$ is a fully connected DNN with m nodes in each layer, referred to as the width of the network, and n-many hidden layers, referred to the depth of the network. All hidden layers have the same activation function, which is specified for each result below.

Finally, the output layer is associated with a linear activation function (i.e., an affine map, which is common practice when using NNs for regression purposes). The DNN for $\delta(x)$ has an almost identical structure. However, the output layer is an affine map, composed with a custom ReLU activation to force the DNN to have a positive lower bound. This is to reflect the fact that $\delta(x)$ cannot take negative values. In place of a standard ReLU function that maps to $f(x) = \max\{0, x\}$, the custom ReLU function maps to $f(x) = \max\{\epsilon, x\}$ Here, $\epsilon = 0.05$ throughout.

4. Discretization of Operators, and Optimization. Several methods of discretization were employed in order to approximate the actions of the operators and minimize their difference. Here, we outline some of the main discretizations and approximations employed in our numerical tests. In all of the following experiments, we fix the fractional order $s = 0.25$.

For the computation of both the tempered and truncated fractional operators, the integrals are approximated as follows. Let $I = [-b, b]$ and $\{y_k\}_{k=1}^{S}$ a uniform partition of I such that $y_k \neq 0$ for any k and let Δy be the distance between the points. Then for a point x in the domain, we approximate the value of the operator acting on a function u at a the point x using a Riemann sum over the interval $[x - b, x + b]$. That is,

$$\mathcal{L}_\ell u(x) \approx \mathcal{A}_\ell u(x) := \sum_{k=1}^{S} \frac{u(x + y_k) - u(x)}{|y_k|^{n+2s}} \gamma_\ell(x, x + y_k) \Delta y,$$

where ℓ can refer to either the tempered or truncated operator. In the following results, $I = [-10, 10]$ and $S = 1000$, and thus $\Delta y = 0.02$.

Likewise, the L^2 norm of the operators is approximated using the ℓ^2 norm on a set of discrete points in the domain $\Omega = [-A, A]$. Specifically, let $\{x_j\}_{j=1}^{R}$ be a uniform partition of Ω.

Additionally, index the training functions as $\{u_i\}_{i=1}^{N}$. Let

$$f_{\gamma_\ell}(u_i(x_j))(x_j + y_k) = \frac{(u(x_j + y_k) - u(x_j))}{|y_k|} \gamma_\ell(x_j, x_j + y_k), \tag{4.1}$$

Then approximate the loss function (3.1) as

$$\text{Loss} \approx \frac{1}{N} \sum_{u_i} \left[\frac{\sum_{x_j} |\mathcal{A}_{\text{tem}} u_i(x_j) - \mathcal{A}_{\text{tr}} u_i(x_j)|^2}{\sum_{x_j} |\mathcal{A}_{\text{tem}} u_i(x_j)|^2} \right]^{1/2} \tag{4.2}$$

We recall that we are using the smoothed version of the truncated kernel (2.2), as discussed in Section 2. The reason of this approximation resides in the fact that δ cannot be a boolean variable during training, which is how a standard piecewise indicator function is defined. The indicator is then approximated using the sigmoid function as in equation (2.4), with $\alpha = 10$.

5. Computational Results. We report the results of the regression algorithm. In all our tests, the hidden layers have a hyperbolic tangent (tanh) activation function and we use the Adam optimizer.

For the first test, we train $\delta(x)$ and $\sigma(x)$ as NNs with 8 nodes in each layer and 2 hidden layers. The training and testing functions both utilize the exponential bump basis function as defined in (3.3). There are 32 training and 32 testing functions formed as linear combinations of 10 exponential bump basis functions spread evenly across the interval $[-8, 8]$.

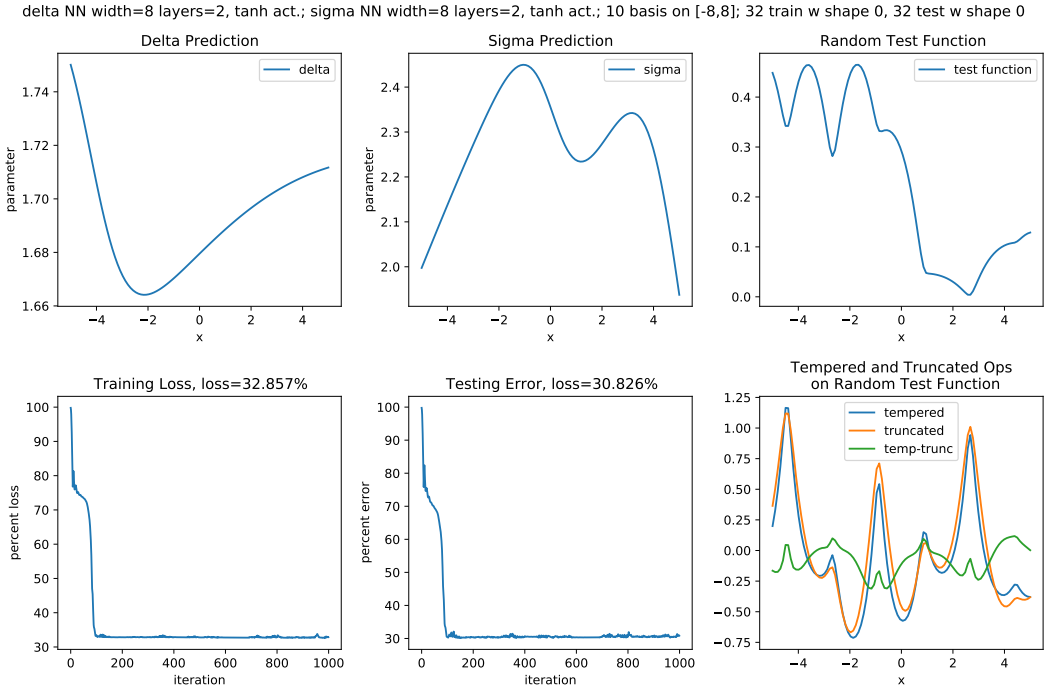


Fig. 5.1: Result of training and testing NNs with width 8 and depth 2. Both training and testing functions are generate using bump basis functions.

Results are reported in Figure 5.1. The upper left and upper center plots contain the DNN prediction for the $\delta(x)$ and $\sigma(x)$ parameters, respectively, on the last iteration of the optimization. The lower left plot displays the training loss, which is the percent error defined in (4.2) for the set of training functions. The lower center plot displays the testing loss, which is computed in the same manner as the training loss but on a set of testing functions. The testing functions are distinct from the training functions (which are used to learn the parameters), but formed using the same basis functions as the training functions. This graph illustrates how well the learned parameters can generalize to functions outside the training set. The upper right plot is a random test function from the set of testing functions. In the lower right plot we report the values of the tempered and truncated operators acting on the random test function from the upper right plot, as well as the difference of the actions of the operators. The final value of the training loss indicates a percent error of 30%. While this result is not entirely satisfactory, we believe that by using a more stable optimization algorithm, such as L-BFGS, after training with Adam, lower loss values could be achieved. The value of the testing loss is of the same order of the training loss; this indicates that the learnt operator is as accurate when evaluated on (new) functions belonging to the same family, i.e. the set of bump functions.

We performed the same test with varying choices of depth and width of the neural networks used to train δ and σ. Result of these simulations are reported in Table 5.1. We note that the training and testing loss are not sensitive to the sizes of the NNs.

Due to the near-constant behavior of the $\delta(x)$ when trained as DNNs, we perform tests where this parameter is a positive real number, and learn this values as well as the neural

Width × Depth	**Training Loss**	**Test Error**
8×2	32.857 %	30.826 %
16×4	30.790 %	28.952 %
32×8	30.031 %	30.511 %
128×16	29.589 %	29.581 %

Table 5.1: Training and testing loss for different neural network sizes.

network for $\sigma(x)$ using the same algorithm as described above. An example of these results is reported in Figure 5.2. Here, the plots represent the same quantities as in Figure 5.1.

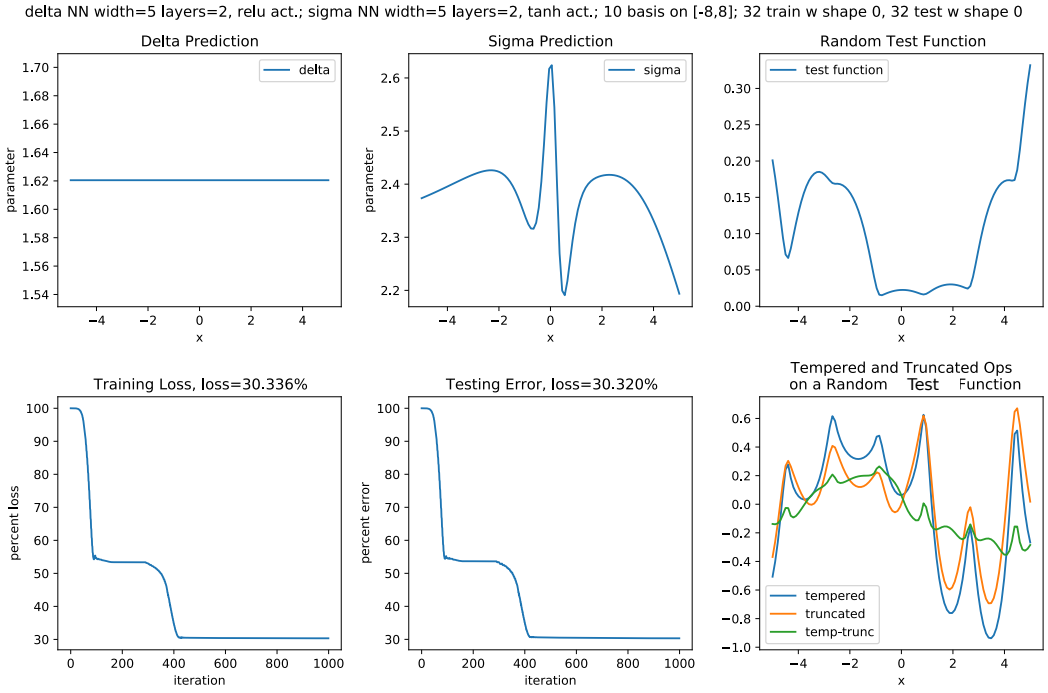


Fig. 5.2: Results of training with the parameter δ trained as a constant value, and σ trained as neural network.

With the purpose of testing the generalization properties of our learning procedure, we also perform cross-validation tests, i.e. we train the parameters using a set of basis functions and test the learnt operator on a set of functions generated with a different basis. In this test we kept all the parameters the same as in the first test with the exception of the shape of the basis functions used for the testing functions, which are generated by using the linear hat functions defined in (3.2). The corresponding results, for a new run of the algorithm, that results in slightly different $\delta(x)$ and $\sigma(x)$ than in Figure 5.1, are reported in Figure 5.3. We observe a testing percent error of 40%; the higher value is justified by the fact that the test functions are substantially different from the training functions, as they are generated using different basis functions.

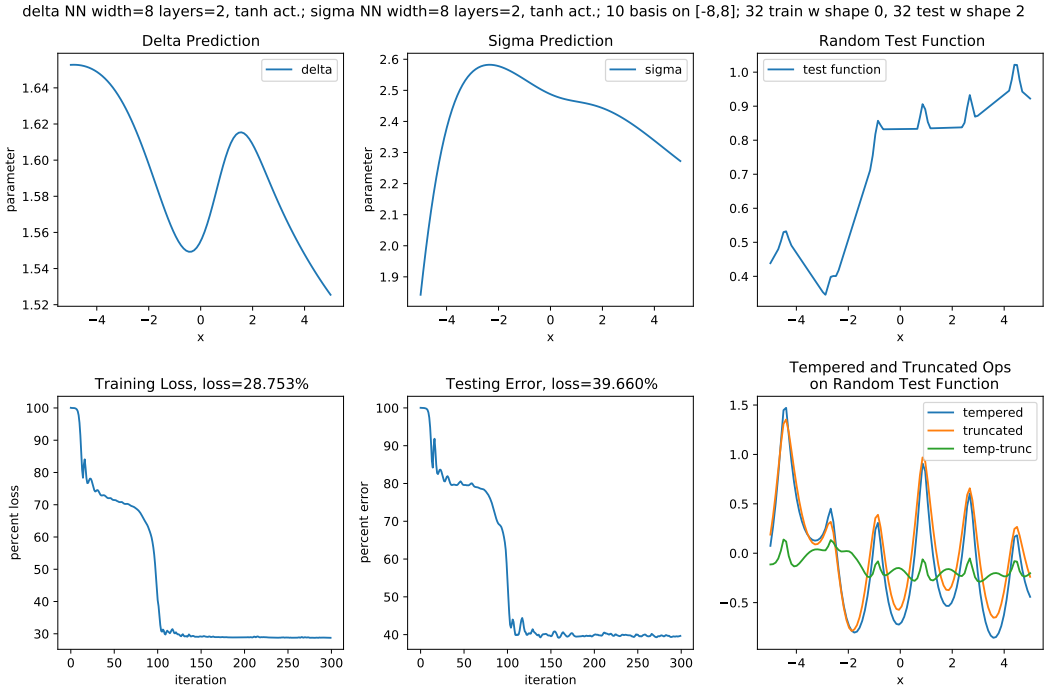


Fig. 5.3: Result of training and cross-validation tests for NNs with width 8 and depth 2. The training functions are generated using bump basis functions whereas the testing functions are generated using hat functions.

6. Conclusions. In this work we explored the approximation of the tempered fractional Laplacian by means of a truncated fractional Laplacian operator with the purpose of improving the efficiency of numerical computations. We propose a modified truncated fractional Laplacian operator parametrized by a scaling parameter and a horizon and we learn, via deep learning, the optimal values of these parameters such that the action of the truncated operator is as close as possible to the one of the tempered fractional Laplacian. Our results indicate that, for the studied basis set and training/test functions, the percent error between the truncated and the tempered operator is of the order of 30%, and that this is quite independent of the neural network architecture, or even whether δ is a constant value. The same accuracy is achieved when testing the optimal operator on new functions belonging to the same family of functions used for training. However, when tested on functions of a different family, the percent error increases to 40%.

While a deeper architecture does not improve the accuracy of the learnt operator, we believe that more sophisticated optimization algorithms may yield accuracy improvements. Thus, part of our planned follow-up work includes the use of improved optimization algorithms such as L-BFGS after training with the Adam optimizer to improve final test error. If this succeeds in improving training and test error, we also plan to run additional cross-validation tests to study how the qualitative differences between the training and test sets effects generalization. Furthermore, we plan to utilize a symmetric form of the truncated operator by symmetrizing the truncated kernel as done in, e.g., [7], to reflect the symmetric nature of the tempered fractional kernel. Furthermore, we plan to study the differences in

solutions to exterior value problems using the tempered Laplacian and the learned truncated operator, which may be significantly different that the observed test errors due to regularity.

7. Acknowledgments. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs) project. M.D. and M.G. are supported by Sandia National Laboratories (SNL), SNL is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration contract number DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References.

[1] D. A. Benson, S. W. Wheatcraft, and M. M. Meerschaert, *Application of a fractional advection-dispersion equation*, Water Resources Research, 36 (2000), pp. 1403–1412.

[2] O. Burkovska, C. Glusa, and M. D'Elia, *An optimization-based approach to parameter learning for fractional type nonlocal models*, Computers & Mathematics with Applications, (2021).

[3] H. Olson, M. D'Elia, and M. Gulian, *The tempered fractional Laplacian as a special case of the nonlocal Laplace operator*, Computer Science Research Institute Summer Proceedings 2020. Technical Report SAND2020-12580R, (2020), pp. 111–126.

[4] G. Pang, M. D'Elia, M. Parks, and G. E. Karniadakis, *nPINNs: nonlocal Physics-Informed Neural Networks for a parametrized nonlocal universal Laplacian operator. Algorithms and Applications*, Journal of Computational Physics, 422 (2020), p. 109760.

[5] F. Sabzikar, M. M. Meerschaert, and J. Chen, *Tempered fractional calculus*, J. Comput. Phys., 293 (2015), pp. 14–28.

[6] R. Schumer, D. A. Benson, M. M. Meerschaert, and B. Baeumer, *Multiscaling fractional advection-dispersion equations and their solutions*, Water Resources Research, 39 (2003), pp. 1022–1032.

[7] Y. Tao, X. Tian, and Q. Du, *Nonlocal models with heterogeneous localization and their application to seamless local-nonlocal coupling*, Multiscale Modeling & Simulation, 17 (2019), pp. 1052–1075.

[8] H. You, Y. Yu, N. Trask, M. Gulian, and M. D'Elia, *Data-driven learning of robust nonlocal physics from high-fidelity synthetic data*, Computer Methods in Applied Mechanics and Engineering, (2021).

A REDUCED-SPACE FORMULATION FOR NONLINEAR PROGRAMMING WITH INDEX-1 DIFFERENTIAL ALGEBRAIC EQUATION SYSTEMS

ROBERT B. PARKER*, BETHANY L. NICHOLSON†, JOHN D. SIIROLA‡, CARL D. LAIRD§, AND LORENZ T. BIEGLER¶

Abstract. A reduced space formulation for optimization of index-1 differential algebraic equation systems (DAEs) is described and implemented in the PyNumero extension to the Pyomo algebraic modeling environment. The formulation defines implicit functions from algebraic equations and uses them to remove algebraic variables and equations from the optimization problem. The formulation is used to solve dynamic optimization problems with the Ipopt algorithm, which uses first and second derivatives calculated by the implicit function theorem. In challenging case studies involving the simulation and optimization of a chemical looping combustion reactor, the reduced space formulation is more robust, solving 92 out of 116 problem instances in the first case and 52 out of 72 instances in the second case. In these cases the full space simultaneous formulation can solve 40 and 25 problem instances. The results indicate the potential of this reduced space formulation to be more reliable than the full space formulation for challenging nonlinear DAE optimization problems.

1. Introduction. Differential algebraic equation systems (DAEs) are an expressive class of equations for chemical and process systems engineers. They are capable of representing processes that evolve over continuous time and/or space domains and involve nonlinear physical and chemical phenomena. Complicated interactions among differential states are often described by large numbers of highly nonlinear algebraic equations that represent the thermodynamics, chemical reactions, and transport phenomena of the system. Due to their number and complexity, the ability to simulate or optimize a large-scale DAE relies heavily on the ability to solve these algebraic equations.

Optimization of DAE systems is typically done by solving nonlinear programming (NLP) problems, as solvers for these problems are capable of handling large scale systems. Two common approaches for these optimization problems are sequential approaches [8] and simultaneous approaches [6]. In the former, the optimization algorithm is only aware of control inputs and computes the objective function by simulating a DAE system. Gradient information is returned by the DAE simulator and used to update the input variables. In addition to using first-order gradient information, Vassiliadis et al. [14] calculate second derivatives with respect to control inputs, and a sequential optimization approach with an exact Hessian is implemented by Balsa-canto et al [1]. Hybrid approaches have been proposed, including multiple shooting [2], which partitions the continuous domain and simulates multiple DAEs sequentially, and the quasi-sequential approach of Hong et al. [10], which eliminates all equality constraints from the optimization algorithm but retains inequality constraints that may involve state variables.

We propose a hybrid approach for DAEs that are index-1 in which only the algebraic variables and equations are removed from the nonlinear optimization problem using implicit functions. Our approach is simultaneous in the sense that the continuous domain is discretized and that all discretization points are considered simultaneously by the optimizer, but has the characteristic of sequential approaches that square problems are solved by an embedded solver within each iteration of the optimization solve. This is a reduced-space

*Carnegie Mellon University, rparker1@andrew.cmu.edu
†Sandia National Laboratories, blnicho@sandia.gov
‡Sandia National Laboratories, jdsiiro@sandia.gov
§Carnegie Mellon University, claird@andrew.cmu.edu
¶Carnegie Mellon University, biegler@cmu.edu

formulation as only the differential, input, and derivative variables are seen by the optimization algorithm. The implicit function subproblems admit exact first and second derivatives in terms of these variables and are thus compatible with second-order optimization algorithms. Our approach is conceptually similar to that of Bongartz and Mitsos [3], where external functions are used to solve global flowsheet optimization problems in a reduced space. Rather than implementing a fully implicit function, however, they use a sequential modular function evaluation for which they compute convex relaxations. We also differentiate our approach from reduced space optimization algorithms, for instance of Cervantes et al. [5], by noting that our optimization algorithm is not aware of any particular partition of the variables to which it has access. That is, it calculates its search direction in the space of variables that it is aware of rather than in a further reduced space.

2. Background and Implementation. A DAE has the form given by Equation (2.1), where (2.1a) are the differential equations, (2.1b) are the algebraic equations, (2.1c) are the discretization equations, and (2.1d) are the initial conditions. The equations are in terms of differential variables x, algebraic variables y, input variables u, and derivatives $\dot{x}$ with respect to a continuous domain. We consider the DAE after discretization of this domain. With the exception of (2.1d), these variables and equations are repeated at every point along the discretized domain. Functions f and g are assumed to be twice continuously differentiable in all arguments.

$$\dot{x} = f(x, y, u) \tag{2.1a}$$
$$0 = g(x, y, u) \tag{2.1b}$$
$$0 = d(x, \dot{x}) \tag{2.1c}$$
$$x(0) = x_0 \tag{2.1d}$$

We say that a DAE is index-1 if the Jacobian of algebraic equations with respect to algebraic variables, $\nabla_y g$, is nonsingular for all values of differential, algebraic, and input variables. In this case, by the implicit function theorem, there exists a function g_y which maps the vector (x, u) to y such that (2.1b) are satisfied.

A simultaneous nonlinear programming formulation for the optimization of a DAE model has the form given by Equation (2.2). We refer to this as the full space formulation.

$$\begin{aligned} \min_{(\dot{x},x,y,u)} \quad & \varphi(x, u) \\ \text{s.t.} \quad & \dot{x} - f(x, y, u) = 0 \\ & g(x, y, u) = 0 \\ & d(x, \dot{x}) = 0 \\ & x(0) = x_0 \\ & x \geq 0 \end{aligned} \tag{2.2}$$

A dynamic optimization problem of this form can be constructed by an algebraic modeling language if the constraints and objective are explicit functions of the variables. The optimization problem may then be solved with a generic nonlinear programming solver.

Our reduced space formulation takes advantage of the index-1 property to eliminate algebraic variables and equations from the NLP. The formulation is given by Equation (2.3).

$$\begin{aligned} \min_{(\dot{x},x,u)} \quad & \varphi(x,u) \\ \text{s.t.} \quad & \dot{x} - f(x, g_y(x,u), u) = 0 \\ & d(x,\dot{x}) = 0 \\ & x(0) = x_0 \\ & x \geq 0 \end{aligned} \tag{2.3}$$

An optimization problem of this form cannot be constructed directly in an algebraic modeling environment because g_y is an implicit function. This formulation can still be solved by a nonlinear programming algorithm, however, if g_y may be evaluated and admits sufficient derivative information. We implement an interface that evaluates an implicit function and computes first and second derivatives via the implicit function theorem. Our implementation is described in Section 2.1. These values and derivatives can plug into the PyNumero CyIpopt interface, allowing us to solve NLPs in the form of Equation (2.3) using the Ipopt algorithm [15].

2.1. Implicit Function Implementation. Our implementation makes use of the PyNumero extension to the Pyomo algebraic modeling environment [4]. The implicit function implementation allows users to specify Pyomo constraints and variables that will be solved to evaluate the implicit function. The variables and constraints that the user supplies are shown in Equation (2.4), where Equation (2.4a) are the residual equations, which are exposed to the NLP, and Equation (2.4b) are the external equations, which will be removed as an implicit function. Variables x are inputs into the system, and variables y are external – they are solved for by the implicit function and eliminated from the NLP.

$$f(x,y) = 0 \tag{2.4a}$$
$$g(x,y) = 0 \tag{2.4b}$$

Assuming that the Jacobian of external equations with respect to external variables, $\nabla_y g$, is nonsingular, Equation (2.4b) can be used to solve for y as a function of x. In our implementation, this square system solve is performed with Ipopt. The residual equations can then be written as shown in Equation (2.5).

$$f(x, y(x)) = \bar{f}(x) = 0 \tag{2.5}$$

To include the constraints of Equation (2.5) in an NLP, we need to calculate first and second derivatives of $\bar{f}$. We do this using the implicit function theorem and the chain rule, applied to Equations (2.4). The derivatives we calculate are given by Equation (2.6).

$$\begin{aligned} \nabla_x y =& -\nabla_y g^{-1} \nabla_x g \\ \nabla_x \bar{f} =& \nabla_x f - \nabla_y f \nabla_x y \\ \nabla^2_{xx} y =& -\nabla_y g^{-1} \otimes_{2,3} \left(\nabla^2_{xx} g + (\nabla^2_{xy} g \otimes_1 \nabla_x y + \nabla_x y^T \otimes_1 \nabla^2_{yx} g) \right. \\ & \left. + \nabla_x y^T \otimes_1 \nabla^2_{yy} g \otimes_1 \nabla_x y \right) \\ \nabla^2_{xx} \bar{f} =& \nabla^2_{xx} f + (\nabla^2_{xy} f \otimes_1 \nabla_x y + \nabla_x y^T \otimes_1 \nabla^2_{yx} f) \\ & + \nabla_x y^T \otimes_1 \nabla^2_{yy} f \otimes_1 \nabla_x y + \nabla_y f \otimes_{2,3} \nabla^2_{xx} y \end{aligned} \tag{2.6}$$

In Equation (2.6), the "$\otimes$" product refers to the product between a third-order tensor and a matrix. The result is a third-order tensor. In particular, $\otimes_1$ produces a tensor obtained by a matrix-matrix product along each coordinate of the first rank of the tensor, while $\otimes_{2,3}$ produces a tensor obtained by a matrix-vector product along all coordinates of the second and third ranks of the tensor. Equation (2.7) illustrates these products with index and slice notation, where A and B are third-order tensors, C is a matrix, and "$\cdot$" is the standard inner product between vectors.

$$\begin{aligned} A = B \otimes_1 C &\Rightarrow A_{i,j,k} = B_{i,j,:} \cdot C_{:,k} \\ A = C \otimes_1 B &\Rightarrow A_{i,j,k} = C_{k,:} \cdot B_{i,:,j} \\ A = C \otimes_{2,3} B &\Rightarrow A_{i,j,k} = C_{i,:} \cdot B_{:,j,k} \end{aligned} \tag{2.7}$$

We note that for second-order optimization algorithms, only the Hessian of the Lagrangian, rather than the Hessian of each constraint, is required. In our implementation, we compute the Hessian of the Lagrangian by first computing that of each constraint, although it may be possible to more efficiently compute the Hessian of the Lagrangian directly. In particular, it may be possible to extend the adjoint approach described by Heinkenschloss [9] to the case where implicit functions are embedded in vector-valued constraint functions rather than a scalar-valued objective function. We defer this potential improvement to future work.

All Jacobian and Hessian matrices are calculated by PyNumero's interface to the Ampl Solver Library (ASL) [7], and tensor products are performed with NumPy. The Jacobian and Hessian matrices of $\bar{f}$ are sent to CyIpopt via the PyNumero interface, allowing us to solve NLPs in the form of Equation (2.3) with the Ipopt algorithm.

2.2. Application to Index-1 DAEs. With an implicit function interface for systems in the form of Equation (2.4), the application to DAE systems in the form of Equation (2.1) is straightforward. Residual constraints (2.4a) are differential equations (2.1a), external constraints are algebraic equations (2.1b), external variables are algebraic variables y, and inputs into the external constraints are differential variables x and DAE inputs u. A separate implicit function is used to describe the differential and algebraic equations at each point in the discretized domain.

2.3. Sparsity of Derivative Matrices. Because derivative matrices and tensors $\nabla_x y$, $\nabla_x \bar{g}$, $\nabla^2_{xx} y$, and $\nabla^2_{xx} \bar{f}$ are the result of products with the matrix inverse $\nabla_y g^{-1}$, they are in general dense. Because the Ipopt algorithm requires knowledge of all possible nonzeros of the Jacobian and Hessian matrices, we provide a nonzero for every coordinate of these matrices, regardless of whether they are zero for a particular function evaluation. This is an inefficiency in our implementation as there may be several cases in which it is possible to determine a sparse set of possible nonzeros for the Jacobian and Hessian matrices of our implicit function constraints. For instance, if $\nabla_y g$ is block diagonal, its inverse has bounded fill-in and the derivatives of y and $\bar{f}$ may be sparse. In addition, Hessian matrices may be low rank, as is the case if a set of constraints is mostly linear. In this case Hessian tensors $\nabla^2_{xx} y$ and $\nabla^2_{xx} \bar{f}$ may be sparse even if $\nabla_y g^{-1}$ is dense. We defer the handling of these cases to future work. If sparsity of Jacobian and Hessian matrices cannot be sufficiently preserved, matrix-free optimization solvers that solve the KKT system using iterative methods will be considered. We have not used them so far as factorization of the KKT system does not appear to be a bottleneck of our current implementation.

Note that the density of the Jacobian and Hessian of implicit function constraints does not imply that the entire problem is dense. In our application, a separate implicit function is

used for each point in the discretized domain, which limits potentially superfluous nonzeros to blocks of variables and constraints at a given point in this domain.

3. Case Studies. To demonstrate the viability of our reduced-space index-1 DAE formulation, we apply it to chemical engineering optimization and simulation problems via the Ipopt algorithm. This section presents these results, which show that the reduced space formulation is more time-consuming than the full space formulation, but that there are many problem instances for which only the reduced space formulation converges.

3.1. Optimal Control of a Distillation Column. Optimal control of a binary distillation column is used as an example of the dynamic optimization capabilities of Pyomo.DAE [11]. In this section we solve a simple instance of this problem with full and reduced space formulations to demonstrate some general characteristics of our approach. The code for a full-space implementation of the problem we solve can be accessed from the `examples/dae/` directory of the Pyomo repository, `https://github.com/pyomo/pyomo`.

Table 3.1: Characteristics of the optimization problem with full and reduced space formulations

Formulation	Variables	Constraints	Iterations	Solve time	Jacobian nonzeros
Full	5,119	5,068	14	0.49 s	19,554
Reduced	3,282	3,232	16	242 s	108,832

Table 3.1 shows the number of variables, constraints, and Jacobian nonzeros for the full and reduced space formulations, as well as the solve time and number of iterations required to reach the solution. We also note that full and reduced space formulations lead to the same solution. Even though the reduced space formulation does not consider algebraic variables during the optimization, the algebraic variables and constraints are solved internally at every iteration, so their values may still be compared with those obtained by the full space formulation. The average relative error between variable values in the solutions reached by the two formulations is 2.8×10^{-6}.

Despite reaching the same solution, the optimization problems solved in the two formulations are very different. Compared with the full space, the reduced space formulation has fewer constraints and variables, but a much larger number of nonzeros in the constraint Jacobian, and a solve time that is larger by a factor of approximately 500. Despite having fewer constraints and variables, the reduced space solve converges in approximately the same number of NLP iterations as the full space solve. A breakdown of computational expenses for full and reduced space solves of this problem are shown in Table 3.2. This breakdown demonstrates that our implementation is bottlenecked by Hessian computation and would benefit from an approach that avoids computations with tensors.

Table 3.2: Breakdown of solve times of the distillation example for full and reduced space formulations

Formulation	I/O	Ipopt	Interface	Evaluation	Jacobian	Hessian	Other
Full	62 %	28 %	5 %	–	–	–	5 %
Reduced	12 %	1 %	–	20 %	1 %	56 %	10 %

This case study indicates that although it can solve a dynamic optimization problem to the same solution as a full space formulation, the reduced space formulation pays a high price for repeating implicit function evaluations each iteration. The following two case studies indicate the potential benefit of the reduced space approach by demonstrating improved robustness of the NLP over a range of parameters.

3.2. Chemical Looping Combustion Simulation. A chemical looping combustion (CLC) reactor is a gas-solid hydrocarbon reactor in which fuel and air reactions occur in separate chambers, with a metal-oxide oxygen carrier looping between the two. Here we present results for the simulation of a reduction reactor involving methane and iron oxide operating at steady state in a counter-current moving bed configuration. The reduction reaction is shown in Equation (3.1).

$$CH_4 + 12Fe_2O_3 \rightarrow 2H_2O + CO_2 + 8Fe_3O_4 \tag{3.1}$$

The model equations form a DAE in which the continuous domain is distance along the length of the reactor. Differential variables are material and enthalpy flow rates and pressure. Algebraic equations and variables describe thermodynamics, hydrodynamics, transfer correlations, and the reaction rate. In the current simulation case study, the model has no inputs; inlet conditions are fixed. Further details are given by [13], and the model may be accessed via the IDAES repository at `https://github.com/idaes/idaes-pse`.

To compare robustness and solve times of the full and reduced space formulations, we perform a parameter sweep, varying the number of finite elements (NFE) in the spatial discretization and the gas phase inlet temperature. We attempt to solve the simulation problem in the full and reduced space formulations for every combination of four NFEs and 29 temperatures, all with identical initialization routines and scaling factors. Figure 3.1 shows the solve times for each successful simulation. A failed simulation may be due to timeout, iteration limit, function evaluation error, or converging to an infeasible point. Solve times in these cases are omitted from Figure 3.1.

The data indicate that while the reduced space formulation takes much more time for the solver to converge, it can also solve significantly more instances of the simulation problem than the full space formulation. Of the 116 problem instances, the reduced space formulation can solve 92, while the full space formulation can solve 40. However, when it succeeds, the full space formulation solves in less than one second, while the reduced space formulation takes 160 seconds in the most time-consuming instance.

3.3. Chemical Looping Combustion Optimization. We now present results for optimization of operating conditions of the chemical looping combustion reactor. The model is the same as in Section 3.2, but now inlet conditions are degrees of freedom and outlet conditions participate in an objective function penalizing their deviation from target conditions. The target outlet conditions are taken from the moving bed process described in Table 5 of [12].

Because inlet gas temperature is no longer a fixed parameter, we now vary the gas temperature used for model initialization, as well as the number of finite elements, in a parameter sweep. Solve times are plotted for instances that converge in Figure 3.2.

The data again indicate that although the reduced space formulation takes much longer to converge than the full space formulation, there are many instances for which only the reduced space formulation converges. Out of 72 problem instances for CLC optimization, the reduced space formulation can solve 52, and the full space formulation can solve 25. While the reduced space formulation is slower than the full space formulation by approximately a

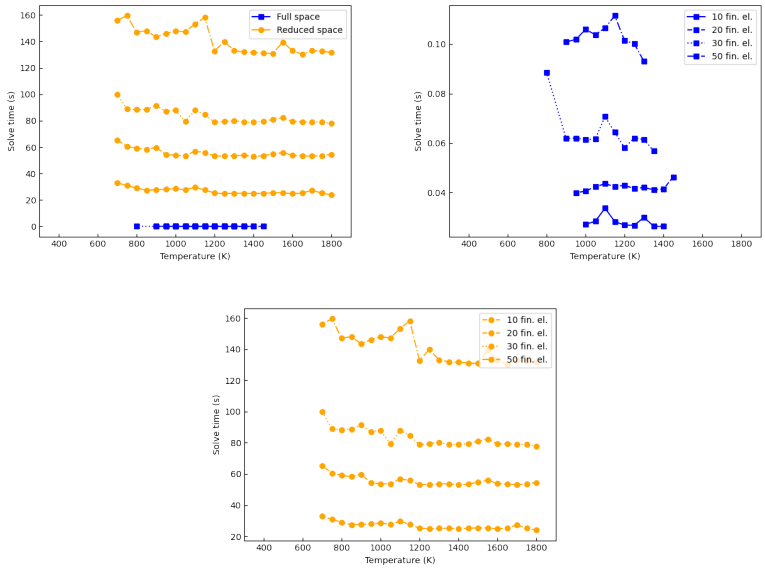


Fig. 3.1: Solve times for sample temperatures and numbers of finite elements in the full and reduced space models. Data points are omitted when the solve does not converge.

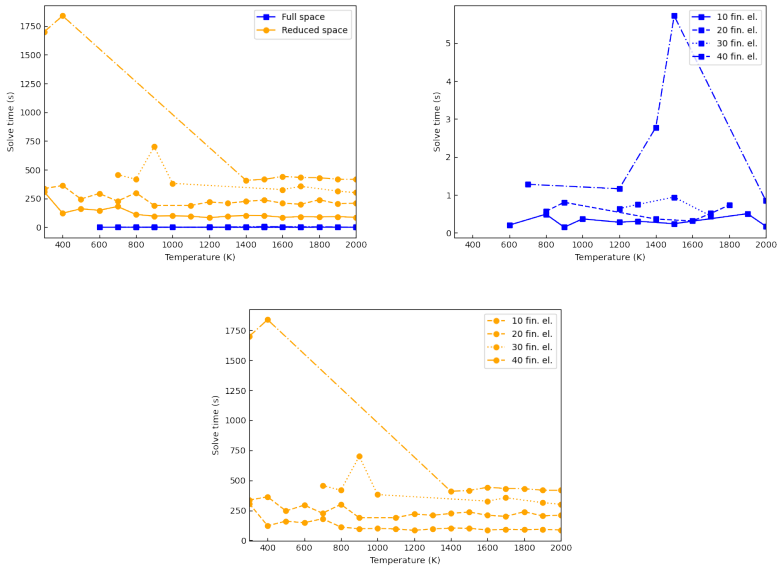


Fig. 3.2: Solve times for sample temperatures and numbers of finite elements in the full and reduced space models. Data points are omitted when the solve does not converge.

factor of 100, we note that there is much that can be done to speed up a solve in the reduced space. The implicit function and derivative evaluations for each point in the discretized domain can be performed in parallel, implicit function computations can be performed entirely in compiled code by operating on a PyNumero `PyomoNLP` object rather than Pyomo variables and constraints, and potential sparsity in the derivative matrices can be exploited. We defer these improvements to future work.

4. Conclusions. We have implemented a novel formulation for the simulation and optimization of index-1 DAE systems via nonlinear programming. The code for our implementation may be accessed in the `pyomo/contrib/pynumero/interfaces` directory of the Pyomo repository, `https://github.com/pyomo/pyomo`, as of the 6.1 release of Pyomo. The application of our formulation to the simulation and optimization of a chemical looping combustion reactor illustrates that it has the potential to be much more robust than a full-space formulation, although when it converges, the full space formulation is faster by a factor on the order of 100. We hypothesize that the improved robustness of the reduced space formulation is because the optimization algorithm cannot fail due to an inability to converge the algebraic equations. The embedded implicit function evaluations, however, can fail, but they are much smaller problems than the full or reduced space optimization problems and are guaranteed to have an isolated solution so we consider this less likely.

There are many extensions to the current work that can be explored. The computational performance can be sped up by exploiting potential sparsity in implicit function evaluation, evaluating different implicit functions in parallel, and processing floating point values exclusively in compiled code. Other extensions include removing only a subset of the algebraic equations from the optimization problem and applying implicit functions with exact first and second derivatives to other types of decomposable optimization problems. A comparison with a sequential formulation with exact first and second derivatives, performed in our current PyNumero/Ipopt computational framework as well as with matrix-free sequential methods, will also be valuable.

5. Acknowledgment. This work was supported through the Simulation-Based Engineering, Crosscutting Research Program within the U.S. Department of Energy's Office of Fossil Energy. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References.

[1] E. Balsa-Canto, J. R. Banga, A. A. Alonso, and V. S. Vassiliadis, *Dynamic optimization of chemical and biochemical processes using restricted second-order information*, Computers & Chemical Engineering, 25 (2001), pp. 539–546.

[2] H. Bock and K. Plitt, *A multiple shooting algorithm for direct solution of optimal control problems*, IFAC Proceedings Volumes, 17 (1984), pp. 1603–1608. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

[3] D. Bongartz and A. Mitsos, *Deterministic global optimization of process flowsheets in a reduced space using mccormick relaxations*, Journal of Global Optimization, 69 (2017), pp. 761–796.
[4] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, vol. 67, Springer Science & Business Media, third ed., 2021.
[5] A. M. Cervantes, A. Wächter, R. H. Tütüncü, and L. T. Biegler, *A reduced space interior point strategy for optimization of differential algebraic systems*, Computers & Chemical Engineering, 24 (2000), pp. 39–51.
[6] J. E. Cuthrell and L. T. Biegler, *On the optimization of differential-algebraic process systems*, AIChE Journal, 33 (1987), pp. 1257–1270.
[7] D. M. Gay, *Hooking your solver to ampl*, tech. rep., Bell Laboratories, 1997.
[8] C. Goh and K. Teo, *Control parametrization: A unified approach to optimal control problems with general constraints*, Automatica, 24 (1988), pp. 3–18.
[9] M. Heinkenschloss, *Numerical solution of implicitly constrained optimization problems*, tech. rep., Rice University, 2008.
[10] W. Hong, S. Wang, P. Li, G. Wozny, and L. T. Biegler, *A quasi-sequential approach to large-scale dynamic optimization problems*, AIChE Journal, 52 (2006), pp. 255–268.
[11] B. Nicholson, J. D. Siirola, J.-P. Watson, V. M. Zavala, and L. T. Biegler, *`pyomo.dae`: A modeling and automatic discretization framework for optimization with differential and algebraic equations*, Mathematical Programming Computation, 10 (2018), pp. 187–223.
[12] C. O. Okoli, A. Ostace, S. Nadgouda, A. Lee, A. Tong, A. P. Burgard, D. Bhattacharyya, and D. C. Miller, *A framework for the optimization of chemical looping combustion processes*, Powder Technology, (2020), pp. 149–162.
[13] A. Ostace, A. Lee, C. O. Okoli, A. P. Burgard, D. C. Miller, and D. Bhattacharyya, *Mathematical modeling of a moving-bed reactor for chemical looping combustion of methane*, in Proc. 13th International Symposium on Process Systesm Engineering (PSE 2018), Computer-Aided Chemical Engineering, San Diego, United States, 2018, Elsevier, pp. 325–330.
[14] V. S. Vassiliadis, E. B. Canto, and J. R. Banga, *Second-order sensitivities of general dynamic systems with application to optimal control problems*, Chemical Engineering Science, 54 (1999), pp. 3851–3860.
[15] A. Wächter and L. T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Math. Program., (2006), pp. 25–27.

MULTIFIDELITY DATA FUSION IN CONVOLUTIONAL ENCODER/DECODER ASSEMBLY NETWORKS FOR COMPUTATIONAL FLUID DYNAMICS APPLICATIONS

LAUREN M. PARTIN*, GIANLUCA GERACI†, AHMAD A. RUSHDI‡, MICHAEL S. ELDRED§, AND DANIELE E. SCHIAVAZZI¶

Abstract. We investigate the predictive performance and associated uncertainty of convolutional neural networks for dense regression problems, for the case where training datasets are available from multiple information sources. Specifically, we analyze convolutional neural networks assembled from encoders, decoders and skip connections. These networks benefit from a significant reduction in the number of trainable parameters with respect to an equivalent fully connected network and are versatile with respect to the dimensionality of the inputs and outputs. For example, encoder-decoder, decoder-encoder or decoder-encoder-decoder architectures are well suited to learn mappings between inputs and outputs of any dimensionality, and here we adapt their use to multifidelity modeling. We demonstrate the accuracy produced by such multifidelity convolutional architectures on the approximation of one-dimensional functions and the solution of a partial differential equation in a two-dimensional domain. In these examples, the networks are trained on a few high-fidelity and many low-fidelity examples and we investigate the performance for cases where the low- and high-fidelity representations are either implicitly or explicitly linked. Finally, we quantify the predictive uncertainty using a dropblock regularizer both during network training and evaluation, and investigate the factors responsible for the amount of variability generated in the network output.

1. Introduction. Computational simulation has revolutionized engineering design and constitutes an indispensable tool to advance our understanding of complex engineering and scientific problems. Simulating complex phenomena, for example characterized by multiple interacting physics, may however require a substantial computational effort such that broad usage of these models is hindered by the available computational resources, ultimately reducing their impact in answering the science questions of interest. However, it is often the case that there exists a trade-off between accurate but expensive high-fidelity simulations and lower-fidelity simulations that provide approximations at lower cost, and through the effective combination of these model fidelities, one can often optimize efficiency while retaining accuracy.

This research focuses on generating multifidelity surrogate models designed to combine information from a few high-fidelity model solutions with a large number of low-fidelity predictors of varying accuracy. More specifically, we focus on data-driven multifidelity surrogates in the machine learning context. Existing approaches within this context include the approach in [4], which considers a *student* network and a *teacher* Gaussian process associated with datasets of variable annotation quality. The teacher generates soft labels from weak labels using the representation learned by the student, and modulates the student learning process from the estimated predictive uncertainty. This technique is compared with two other approaches for transfer learning in [3], showing superior accuracy as a surrogate for physical systems, analyzed with models of two different fidelities. Another approach combines a sequence of three networks which are designed to learn a low-fidelity mapping, the correlation between a low- and a high-fidelity model, and a network designed to minimize the residual of an underlying PDE [16]. A Bayesian neural network pre-trained with variational inference and fine-tuned using Hamiltonian Monte Carlo is also discussed in [15]. Another approach combines convolutional and fully connected neural networks [22] to learn the

*University of Notre Dame, lhensley@nd.edu
†Sandia National Laboratories, ggeraci@sandia.gov
‡Sandia National Laboratories, arushdi@sandia.gov
§Sandia National Laboratories, mseldre@sandia.gov
¶University of Notre Dame, dschiavazzi@nd.edu

discrepancy between increasingly fine discretizations, projected on a common mesh.

Our approach is inspired by recent successes in image classification and segmentation shown by Deep Convolutional encoder-decoder Networks (DCNN) (see, e.g., [17]). In addition, the effectiveness of combining data from multiple fidelities has been mainly demonstrated for dense networks or separately for an ensemble of hybrid convolutional and dense networks [22]. No approach has however investigated encoding-decoding deep convolutional neural networks where the model fidelities are learned together using an all-at-once training approach. We introduce convolutions as an essential tool to reduce the number of weights with respect to fully connected networks when the input, the output or both are high-dimensional.

The context for our multifidelity predictions involves uncertainty, and hence Uncertainty Quantification (UQ) considerations, at two levels. First, there is *predictive uncertainty*, which provides an indication of the quality of the prediction by analyzing the variability in trained network outcomes. This case can be referred to as "UQ for ML" in the sense it analyzes the uncertainty in predictions that are inherent when using a machine-learned surrogate model, potentially targeting a trustworthy formulation with quantified and controlled surrogate modeling errors. Second, there is analyzing the effect that uncertainty in the parameter inputs has on the response outputs when using the network to perform the forward mapping. This case can be referred to as "ML for UQ" in the sense that it represents the traditional forward uncertainty quantification workflow for computing statistics on quantities of interest (QoI) using an ML surrogate. In this paper, we focus on the former as a precursor to the latter. That is, endowing our multifidelity network predictions with predictive uncertainty estimates will facilitate future work in effective surrogate model management for forward UQ analyses.

For quantifying predictive uncertainty, a large number of approaches has been proposed in the literature [1, 8, 11]. Among these, dropout layers [21] offer a simple and computationally appealing solution to drop network nodes with some probability, creating variance estimates complementing point estimate predictions, with well understood theoretical properties [2, 5]. However, their performance has been mainly assessed on neural networks with dense layers. In this study, we use dropblocks [6], i.e., adaptations of dropout layers showing improved performance on convolutional architectures.

This paper is organized as follows: §2 describes the problem domains of interest including one-dimensional functions and high-dimensional problems in Computational Fluid Dynamics (CFD), §3 describes the network architectures to be explored. In §4 and §5, we describe the estimation of predictive uncertainty and the use of implicit versus explicit fidelity connections, respectively. In §6, we focus on two test cases characterized by a small and a large number of inputs and outputs. Finally, we close with concluding remarks and plans for future work in §7.

2. Problem description. In this study, we focus on building a surrogate model for *dense* regression for both low- and high-dimensional inputs/outputs, to examine how low-fidelity information can be leveraged to accelerate the training and, consequently, to improve predictions on a high-fidelity model for which limited information is available.

2.1. One-dimensional multifidelity function approximation. We start with low-dimensional regression, analyzing multiple examples proposed in [16]. These problems consist of two linearly and/or nonlinearly correlated low- (LF) and high-fidelity (HF) functions, where fewer training examples are available for the HF model than for the LF model. In this work, we specifically focus on linearly correlated function pairs. The first pair we analyze is given by two linearly correlated continuous functions, defined in Equation (2.1) and Equation (2.2), with $A = 0.5$, $B = 10$, $C = -5$, with 11 and 4 samples, respectively, as

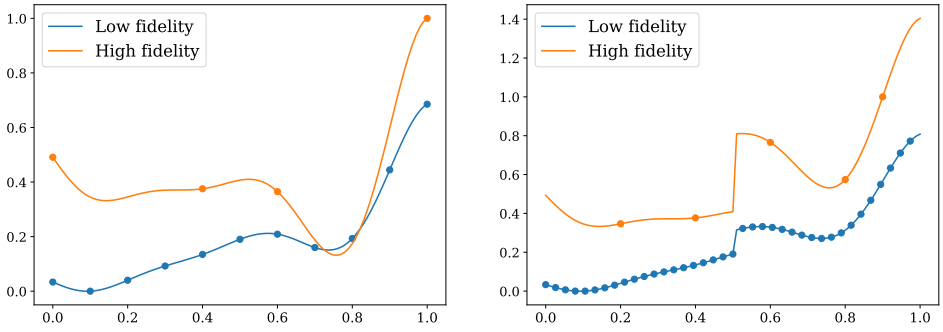
shown in Figure 2.1(a)

$$y_L(x) = A(6x-2)^2 \sin(12x-4) + B(x-0.5) + C \tag{2.1}$$
$$y_H(x) = (6x-2)^2 \sin(12x-4). \tag{2.2}$$

The second set of functions we consider involves a linear correlation between two discontinuous functions given by

$$y_L(x) = \begin{cases} 0.5(6x-2)^2 \sin(12x-4) + 10(x-0.5) - 5 := l(x) & 0 \le x \le 0.5 \\ 3 + l(x) & 0.5 < x \le 1 \end{cases} \tag{2.3}$$
$$y_H(x) = \begin{cases} 2y_L(x) - 20x + 20 := h(x) & 0 \le x \le 0.5 \\ 4 + h(x) & 0.5 < x \le 1 \end{cases} \tag{2.4}$$

with 38 and 5 training samples, respectively, as shown in Figure 2.1(b).



(a) (b)

Fig. 2.1: Low and high-fidelity functions, and available training samples, from (a) continuous pair given by Equations (2.1), (2.2) and (b) discontinuous pair given by Equations (2.3), (2.4), where function values are rescaled such that $y_H(x) \in [0,1]$ and $y_L(x) \in [0,1]$ for x in the training set.

2.2. High-dimensional dense regression for CFD. For the high-dimensional dense regression case, we focus on an application in computational fluid dynamics, where, at any given time, we measure a noisy realization of a fluid domain indicator function (e.g., noisy pixel intensities from magnetic resonance imaging) and a noisy velocity field on such domain, and we would like to estimate the fluid pressure distribution over the domain. The governing equations are the incompressible Navier Stokes (NS) equations, and the pressure can be computed (up to a constant, see, e.g., [20]) from their reformulation into a Poisson pressure equation. This approach, however, may require the solution of a PDE on a typically large computational grid and training a neural network to perform such task has the potential of being a much faster and computationally inexpensive alternative. Note how the present approach differs from previously proposed physics-informed neural networks (PINN [18]) as our network is designed to learn a relation between concentration plus velocity and pressure, rather than pressure as a function of space and time.

Consider a three-dimensional domain $\Omega_f \subset \mathbb{R}^3$ occupied by fluid and characterized by a sufficiently smooth boundary $\partial\Omega_f$. If no body force is assumed to be acting on the fluid, the NS equations can be written as

$$\begin{cases} \nabla p = -\rho \left(\dfrac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) + \mu \, \Delta \mathbf{u} := \mathbf{f} \\ \boldsymbol{\nabla} \cdot \mathbf{u} = 0, \end{cases} \quad \text{in } \Omega_f, \tag{2.5}$$

where ρ, μ and $\mathbf{u}$ are the fluid density, viscosity and velocity, respectively. The Poisson equation for the pressure p can be obtained by taking the divergence of both sides of the momentum equations in (2.5) (see, e.g, [20])

$$\Delta p = \boldsymbol{\nabla} \cdot \mathbf{f}, \tag{2.6}$$

which is equivalent to (2.5) provided the velocity field $\mathbf{u}$ is solenoidal, and only Neumann boundary conditions are applied equal to the flux of $\mathbf{f}$ on $\partial\Omega_f$. We also consider a fluid region Ω_f immersed in a structured grid Ω with $m_x \cdot m_y \cdot m_z = m$ cells, and identified through a *concentration* binary map $c_b : \mathbb{R}^3 \to \{0, 1\}$, where a value of 1 is associated with fluid. In practice, noise makes this function non-binary, with *measured* concentration expressed as $c : \mathbb{R}^3 \to \mathbb{R}$. The problem we would like to solve relates to the development of a neural network surrogate that can be used to quickly evaluate the map

$$f(\mathbf{c}, \mathbf{u}) = \mathbf{p}, \text{ where } f : \mathbb{R}^m \times \mathbb{R}^{m \times 3} \to \mathbb{R}^m, \tag{2.7}$$

which, given the concentrations and velocity distributions over Ω, returns the spatial pressure distribution on Ω_f. Our main goal in this work is to investigate the possibility to train the network with a multifidelity approach based on pressures available on images of increasingly coarser resolutions. The multifidelity training is preformed by combining a small number of HF examples, resulting from a Poisson pressure equation finite element solver, with a larger number of solutions from the same solver, but evaluated on coarser meshes.

3. Network architecture and training data. Our proposed surrogate architecture is a network assembled from convolutional encoders and decoders, for both low- and high-dimensional dense regression. A convolutional encoder [14] is composed of alternating layers of convolutions and pooling (i.e., downsampling), which generates a compressed feature representation. A convolutional decoder, on the other hand, is composed of alternating layers of convolutions and upsampling. For dense regression the encoder and decoder are symmetric, so that the input and output dimensionality of the network is the same. Thus, the network layout depends on both the input and output dimensionality. A detailed description of the two architectures and hyperparameters is offered next.

3.1. DropBlock regularization. As explained in §1, dropout regularization drops network nodes at random, simulating an ensemble of architectures without the extra computational burden of testing them individually. Therefore, it is widely employed to avoid overfitting during training. In practice, a dropout layer is used in hybrid convolutional/dense networks only after the final fully connected hidden layer. Since the network architecture used here does not include a fully connected layer, we utilize DropBlock [6], an alternative to dropout layers that is better suited to convolutional layers and has shown success regularizing convolutional networks for higher accuracy. Dropblock layers are designed to drop a continuous group of pixels (by leveraging the spatial coherence between adjacent pixels). As these layers are still parameterized in terms of dropout probability, the relation between

Block size	Drop prob.	Feature size	γ	Percent dropped
3	0.2	3	0.200	0.451
3	0.9	3	0.900	0.997
3	0.2	4	0.133	0.318
3	0.9	4	0.600	0.912
3	0.2	8	0.088	0.227
3	0.9	8	0.400	0.747
3	0.2	16	0.076	0.197
3	0.9	16	0.342	0.680
5	0.2	8	0.080	0.294
5	0.9	8	0.360	0.840
5	0.2	16	0.053	0.208
5	0.9	16	0.240	0.678
7	0.2	8	0.114	0.481
7	0.9	8	0.514	0.974
7	0.2	16	0.046	0.227
7	0.9	16	0.206	0.713

Table 3.1: Dropout probability vs. the average percentage of features dropped. The percent dropped is calculated across 1000 dropblock realizations and averaged. This is calculated for a one-dimensional layer, where each channel has a length of 8.

such hyperparameter and the actual percentage of features being dropped should be first clarified. A Bernoulli mask is generated in [6], using a probability γ expressed as

$$\gamma = \begin{cases} \dfrac{(1-p)F}{b(F-b+1)}, & \text{for 1D} \\ \\ \dfrac{(1-p)F^2}{b^2(F-b+1)^2}, & \text{for 2D,} \end{cases} \tag{3.1}$$

where p is the *keep* probability (one minus the drop probability), F represents the feature size and b is the block size. The drop probability $1-p$ may be an inaccurate representation of the percentage of elements dropped. As an example, for a drop probability equal to 1, γ may be less than 1 whenever $F \neq b$, meaning that, on average, not all features will be dropped despite the deceptively high drop probability. As seen in Table 3.1, even when γ is equal to the dropout probability, the percent dropped may not be equal to the dropout probability, since the Bernoulli mask generated with γ is expanded by the block size.

3.2. Decoder-encoder architecture for low-dimensional regression. For low-dimensional dense regression, we employ a decoder followed by an encoder, connected by skip connections and having a scalar input x_i and a scalar output $f(x_i)$.

We selected a convolutional network with a number of kernels per convolutional layer in the decoder equal to 16, 16, 8, 8 (4 layers), in the encoder equal to 8, 8, 16, 16, 8 (5 layers), and in the nonlinear correlation equal to 8, 1 (2 layers). We added dropblock regularization with a drop probability of 0.2 and a block size of 3, where the drop probability is increased by equal increments for 5000 steps, until it reaches 0.2, or, in other words, through a linear scheduler. One step is equivalent to processing a pre-defined number (batch) of the

training data. A linear scheduler for the drop probability is recommended in [6] to achieve more accurate predictions. Each convolution layer is followed by a *tanh* activation, with the exception of the final layer where we use a linear activation. The convolutions are performed with a kernel size of 1, 1, 2, 2 in the decoder, 2, 2, 1, 1 in the encoder, 2, 1 in the nonlinear correlation, and 1 in the linear correlation. A stride of 1 and padding are used to keep inputs and outputs of the same size. The kernel size is chosen to induce a convolutional behavior, as opposed to equivalency to a fully connected network. Training is performed using the Adam optimizer [12] utilizing a step learning rate scheduler with decay 0.9, where the step size and initial learning rate is determined by the dataset. In order to achieve robust results for the given datasets, the input x must be fed to downstream parts of the network and L_2 regularization penalty carefully selected. Therefore, L_2 regularization penalty is different depending on the dataset. A batch size of 1 is used due to the limited number of training examples. In addition, the functions are rescaled to the range $[0, 1]$ based on the maximum and minimum value in the training set, which allows for a consistent use of the same optimizer across functions. The network layout is illustrated in Figure 3.1.

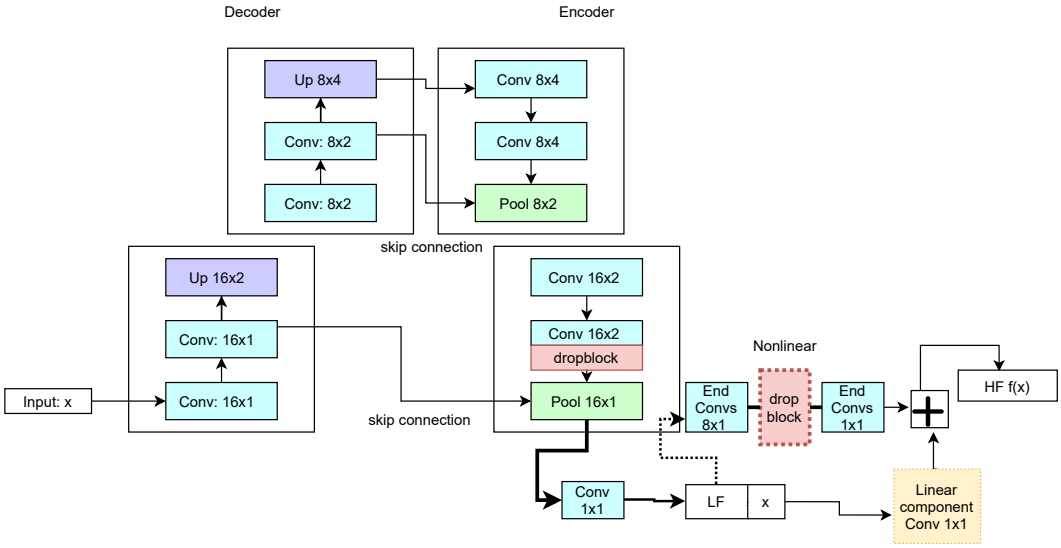


Fig. 3.1: Multi-fidelity decoder-encoder convolutional network architecture for low-dimensional regression with optional explicit feedback. The HF output is a linear combination of the nonlinear and linear portion of the network, $HF = \gamma Linear(LF, x) + (1 - \gamma)Nonlinear(LF, x)$, where γ is a parameter learned by the network.

3.3. Encoder-decoder architecture for high-dimensional regression. A widely adopted encoder-decoder (ED) architecture is the U-Net [19], a convolutional neural network characterized by an encoder which produces a compressed, *i.e.* low dimension, feature representation, followed by a decoder, which outputs a representation with the same dimension as the original input image. The U-Net architecture and its variants have shown great performance in terms of accuracy and training speed for segmentation tasks even under limited training data [10]. We have extended the U-Net architecture to predict the relative pressure on a two-dimensional structured grid where a velocity field and a fluid region indicator function are defined on each pixel.

More specifically, we consider both input and output images with 64×64 pixels. The

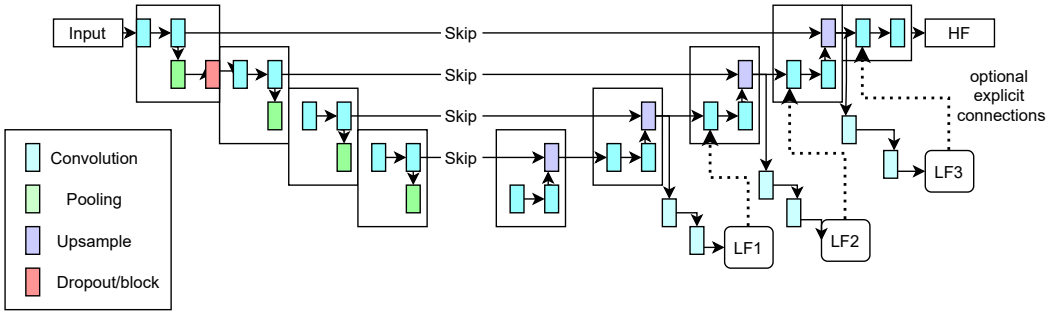


Fig. 3.2: Multi-fidelity decoder-encoder convolutional network architecture for high-dimensional dense regression. For explicit LF-HF feedback, the connections drawn with dotted lines are included, while they are omitted when the LF-HF feedback is implicit. Outputs $LF1$, $LF2$, $LF3$, HF are ordered in terms of resolution from coarsest to finest.

number of kernels per convolution layer in the encoder are 16, 16, 32, 32, 64, 64, 128, 128 (8 layers), respectively, while those in the decoder are instead 64, 64, 32, 32, 16, 16, 32, 32, 16, 1 (10 layers). We use a drop probability of 0.25 and a block size of 3 for the dropblock with a linear scheduler, where the drop probability starts at 0 and is increased by equal increments for 300 steps until it reaches 0.25. Each convolution layer is followed by a batch normalization layer and $ReLU$ activation, which becomes a simple identity after the final convolution layer. Additionally, each convolution is performed with a kernel of size 3, padding 1 (i.e. same size output) and stride 1 and the output is downsampled by max pooling. Training is performed using the Adam optimizer with learning rate 0.01 and no learning rate scheduler, using a batch size of 16.

3.4. Training datasets and multifidelity loss. For low-dimensional dense regression we adopt the same training data as detailed in [16], for the two datasets described in Equations (2.1), (2.2) and (2.3), (2.4). For the high-dimensional case, we have tested our approach using a two-dimensional slice from a Poiseuille flow with parabolic velocity profile, where the fluid is confined within a cylindrical domain Ω_f. We note here that the solution is symmetric with respect to the cylindrical axis and therefore a two-dimensional spatial representation, for a generic plane including the axis, is sufficient to fully describe the flow. The Poiseuille dataset is generated by randomizing the maximum velocity and cylinder radius parameters, as shown in Figure 3.3. Examples for the training, validation and testing datasets are randomly generated using 60/20/20 split ratios, resulting in 116, 49, 35 HF images, respectively. The two high-fidelity only training datasets contain subsets of the 116 HF training images: 32 and 116 high-fidelity data, respectively. For the multifidelity training dataset, we consider 32 high-fidelity and 116 low-fidelity representations, each of which contains noisy, subsampled representations of dimensions 32×32, 16×16, and 8×8 corresponding to a HF sample from the full 116 HF training set, for a total of 380 images.

After training, every dataset's accuracy is evaluated on the same validation and test sets as described previously (49, 35 HF images). Finally, the integral of the Mean Square Error (MSE), assembled from the contribution of all four fidelities with equal weight of 1/4, is used for the loss function when training. The integral here is obtained by multiplying each pixel's contribution to the MSE by the size of associated pixel. We also tested using a weighted average for the loss, where the high-fidelity samples are weighted more. This showed some

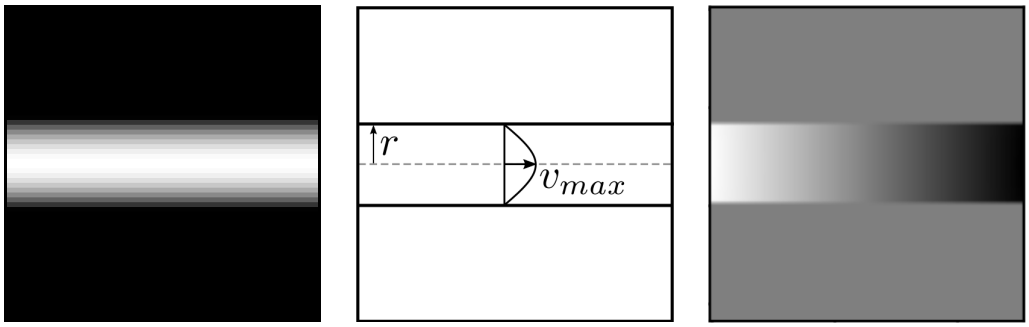


Fig. 3.3: Poiseuille flow test case. Velocity profile (a). Test case parameterization in terms of the fluid region radius and maximum velocity (b). Pressure result (c).

improvements in the final high-fidelity accuracy, but the results were not consistently better than the equally weighted case across different weight initializations. Final accuracy results are reported for predictions on the test set using the model with the lowest validation loss during training.

4. Network prediction estimation. If used during the evaluation of an optimally trained network, dropblocks can inject stochasticity in each network evaluation, and, combined with Monte Carlo sampling, provide a tool to quantify the uncertainty in the network outputs (so-called MC dropout, see [5]). As noted in §1, it is important to emphasize that this notion of uncertainty reflects the variability introduced in the network by hyperparameters (in this case changes in the network architecture induced by randomly dropping neurons) and not the impact of any uncertainty in the network inputs. For this work, we focus on determining the variability in the network predictions as a result of changes in the convolutional architecture induced by dropblock layers. We do not consider variability due to the network weights, as done, for instance, in Bayesian neural networks [13, 15].

5. Multifidelity data fusion. We use two different architectures for multifidelity information fusion; an encoder-decoder architecture for high-dimensional regression and a decoder-encoder architecture for low-dimensional regression. In the high-dimensional case, the network produces a LF prediction of increasing resolution at each stage of the decoder. A term for each LF predictor is then added to the loss function, so these LF representations are accurately learned. In Figure 3.2, the models are ordered as $LF1$, $LF2$, $LF3$, HF, i.e., from the coarsest to the finest resolutions. The low-dimensional regression network in Figure 3.1 only predicts a single low-fidelity estimator LF.

For each of these two networks, we also considered both an implicit and an explicit coupling between low- and high-fidelity representations. In the first case, the LF to HF information feedback in the network is only *implicit*, meaning these low-fidelity predictors are not propagated downstream (i.e., towards the network output, see Figure 3.2, omitting the propagation of the LF representations along the dotted arrows). However, forcing the upstream stages to learn accurate coarse pressure representations clearly affects the accuracy of the high-fidelity prediction. In the second case, this feedback mechanism is instead *explicit*, meaning that the LF predictions are propagated through the following stages of the decoder (see Figure 3.2, including the propagation of the LF representations along the dotted arrows). Similarly, the network selected for low-dimensional regression is shown in

Figure 3.1, where implicit and explicit feedback is again obtained by omitting or including the information transfer through the dotted arrow.

When the HF and LF truth belong to the same space and are correlated, an explicit connection helps in capturing the relationship between the LF and HF, such as in the two low-dimensional regression problems discussed in Section 3.2. However, in the high-dimensional regression problem examined in this work, it's unclear that an explicit connection would be beneficial, since the LFs and HF live on different spaces. In such a case, an implicit connection appears to force the network between two successive LFs to learn their discrepancy, ultimately leading to improved HF predictions.

6. Results.

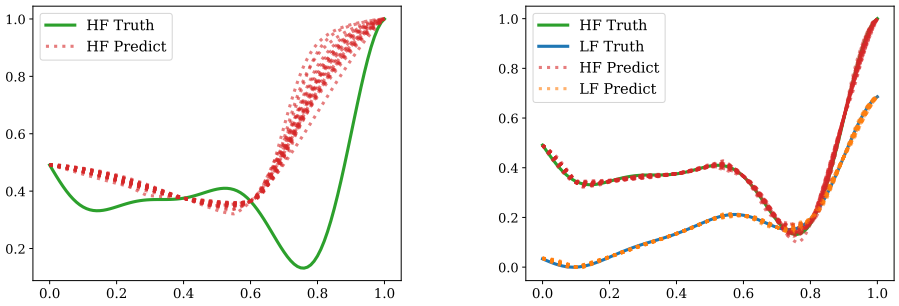
6.1. Low-dimensional dense regression. The network is first trained on the LF data only, since the LF contains more training points than the HF, to ensure there was no detriment to using a convolutional network as opposed to a fully-connected network. After obtaining accurate results on the LF, we focused on the multifidelity predictions. The explicit network outperformed the implicit network, which seems reasonable given the significant correlation between the LF and HF models from the reference publication.

The multifidelity network is able to correctly leverage the LF data to influence the HF prediction to more closely resemble the true HF function, as shown in Figure 6.1(b), compared to the predictions from the network trained with HF data only in Figure 6.1(a). Similarly, in Figure 6.2, the multifidelity network is able to capture the discontinuity by extracting this information from the LF data, since this feature could not be learned from the limited HF data. In Figures 6.1(a) and 6.2(a), the model weights are initialized with different realizations of the same weight initialization scheme; these figures show the behavior is consistent across realizations in that the network is unable to capture the unknown information when trained only with HF data. The weights and biases were initialized through $U(-s, s)$, where $s = nk_0k_1$, n is the number of input channels, and k is the kernel shape, but the behavior was consistent across other weight initialization schemes, such as Xavier [7].

Including x as an additional input downstream of the LF predictor was a necessary adjustment needed to separate the LF into a linear and a non linear contribution, facilitating their combination into an optimal HF predictor; in this regard, note that in (2.4), $y_H(x) = cy_L(x) + F_l(x)$, where c is a constant and $F_l(x) = -20x + 20$ is linear in x.

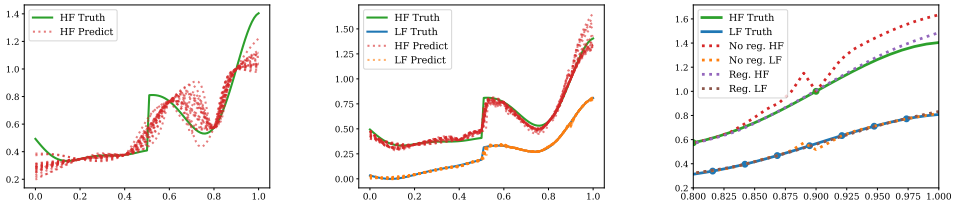
A significant difference arose in comparing the two problem sets for low-dimensional regression; the locations of the LF data chosen for (2.1) was a subset of the locations chosen for the HF data (2.2), whereas in (2.3)-(2.4), the locations of the LF data were distinct from the locations of the HF data. For different LF and HF x values, spikes were observed in the multifidelity network predictions of 6.2(c). The LF prediction overcompensated by creating spikes at the locations of the HF data, to improve the HF training loss, without altering the training loss at the LF points. Therefore, the LF portion of the network required more regularization to prevent this behavior. This does not happen instead when using the same x values for the LF and HF, since a spike reducing the HF loss would necessarily increase the LF loss. Although [15] reported robust results, its network required the regularization penalty to be tuned and the size of the network carefully chosen to siphon the HF prediction into the true HF-LF correlation; since no validation set was included, this would require some degree of manual tuning, which might not be possible in a realistic application, since HF data may not be readily available. Therefore, our network's sensitivity to the regularization penalty for the example in Figure 2.1(b) does not appear to be a limitation of this specific architecture. Lastly, results in this section utilize all-at-once training, by which the LF and HF sections of the network are both trained for every epoch. Simulating the separation of these two networks by freezing the weights (i.e. train on LF data first, freeze the weights,

and train the downstream weights only based on the HF data) produced reasonable visual results, but performed worse than the all-at-once training as determined by the mean error and variance of the error; these results are not reported due to space limitations.



(a) (b)

Fig. 6.1: Predictions from (a) high-fidelity only network, (b) multifidelity network, for different random initializations. True function values are from Equations (2.1), (2.2).



(a) (b) (c)

Fig. 6.2: Predictions from (a) network trained with only high-fidelity data (b) multifidelity network trained with low- and high-fidelity data, for different random initializations. (c) A single prediction both with or without regularization for a limited range of the prediction. True function values are from Equations (2.3), (2.4).

6.2. High-dimensional dense regression. We trained the multifidelity networks with implicit and explicit feedback using 32 high-fidelity and 116 low-fidelity pressure results. We compared their results with a network trained from 116 and 32 high-fidelity examples only. Results are shown in Table 6.1 where we report both the final validation accuracy as R^2

$$R^2 = 1 - RMSE = 1 - \frac{N-1}{N} \cdot \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{6.1}$$

where $RMSE$ stands for the relative mean squared error, y_i is the true pixel value, $\hat{y}_i$ is the predicted pixel value, $\bar{y}$ is the mean of the true pixel value across all pixels and all validation

samples, and N is the total number of pixels in the validation dataset (i.e., the number of pixels in a single image times the number of available validation samples). We also report a normalized accuracy with respect to the cost of generating the training data set, quantified as the total number of pixels in the data. Thus, 116 high-fidelity images with resolution 64×64 have a cost of 475,136 pixels, while 32 high-fidelity images have a cost of 131,072 pixels (a cost ratio of 0.276 to the 116 high-fidelity case). The multifidelity dataset with 32 images at high-fidelity, and 116 images at each of the three low fidelities would result in a cost of 286,976 pixels (a cost ratio of 0.604 to the 116 high-fidelity case).

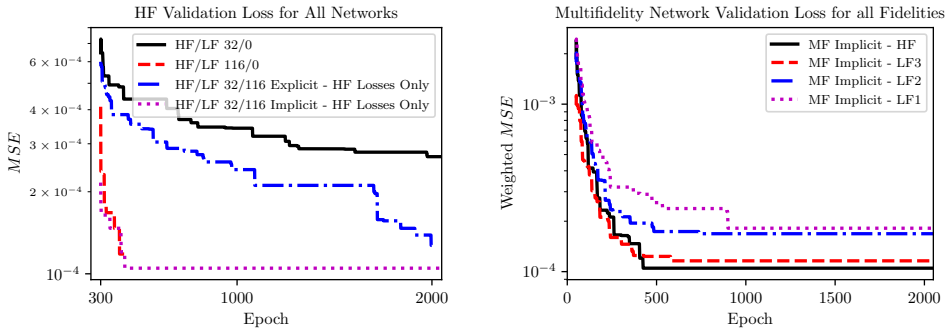
The multifidelity network with explicit feedback significantly improves the accuracy with respect to training with 32 high-fidelity examples. Its normalized accuracy is also superior to that produced by a network trained on 116 high-fidelity examples. High-fidelity and multifidelity validation loss profiles are shown in Figure 6.3. Figure 6.3(a) compares the high-fidelity contribution to the validation loss for networks trained from collections of examples containing a single or multiple fidelities. It emphasizes the acceleration in convergence produced by the multifidelity networks. Figure 6.3(b), on the other hand, demonstrates the ability of the network to learn representations at multiple fidelities (resolutions) during training. When calculating the loss, a weighted integral of mean squared error is used so that the low-fidelity error is compared on the same scale as the high-fidelity. Finally, Figure 6.4 shows the difference in the predicted pressure profiles for a network trained from 32 high-fidelity realizations and a multifidelity network with 32 high-fidelity and 116 low-fidelity examples.

Skip Conn.	**Network Type**	**HF/LF**	R^2	**Normalized** R^2
Concat	MF, explicit feedback	32/116	0.924898	3.223e-06
Add	MF, explicit feedback	32/116	0.930846	3.244e-06
Concat	MF, implicit feedback	32/116	0.917597	3.197e-06
Add	MF, implicit feedback	32/116	0.940792	3.278e-06
Concat	High-fidelity only	32/0	0.909241	6.937e-06
Add	High-fidelity only	32/0	0.874646	6.673e-06
Concat	High-fidelity only	116/0	0.94422	1.987e-06
Add	High-fidelity only	116/0	0.935619	1.969e-06

Table 6.1: Comparison of high-fidelity and multifidelity network performance on Poiseuille test case. Accuracy is computed for the test set. The cost for multifidelity training is 286976 pixels (32 high fidelity images, 116 samples of each of the 3 coarse low fidelity sets). The cost for the 116 high-fidelity samples is 116×64×64=475136. The normalized accuracy is R^2/C where C is the cost. The terms *Concat* and *Add* refer to how the information from a skip connection is assembled into the decoder.

6.3. Prediction estimate accuracy. Uncertainty in the network predictions is quantified, in this study, using MC dropblock. For every input at testing, the network is fed the same input for $N_{UQ} = 50$ times, producing an ensemble of 50 predictions induced by randomness in the dropblock layers.

The 5%-95% confidence interval computed by MC dropblock for the function in Equation (2.2) is amplified around $x = 0.2, 0.8$ where we lack HF samples. In addition, the estimated confidence interval in Figure 6.5(a) does not include the true underlying HF function. To better understand the reason why this happens, we investigated the factors that



(a) (b)

Fig. 6.3: Validation loss profiles for Poiseuille test case (a). The plot compares the losses resulting from HF only and multifidelity training, using the best model for each category, i.e. with the highest test acccuracy. The profiles for the weighted mean squared losses integrated over the fluid domain are shown in (b) for the best performing multifidelity approach. Only the decreasing losses are shown.

are mostly responsible for the amount of uncertainty generated by MC dropblock. Use of *tanh* activation functions produce relatively narrow uncertainty intervals both within and beyond the training locations, as shown in Figure 6.5(b). By replacing the activations with *ReLU*, the intervals widen, particularly away from the training examples, as shown in Figure 6.6(b). This behavior for the dropblock observed in this study is consistent with the observations in [5] for dropout layers in fully connected architectures, where they observed bounded and unbounded intervals with *tanh* and *ReLU* activation functions, respectively. In general, the *ReLU* activation functions produce wider uncertainty intervals, which are more likely to capture the true HF model.

The uncertainty estimates for the high-dimensional case are plotted in Figure 6.8 for the centerline of the fluid region. The mean prediction for the multifidelity network is generally closer to the truth, and its uncertainty estimate captures more of the truth than the high-fidelity only network. In Figures 6.8(c) and 6.8(f) specifically, we can see that the uncertainty increases further from the center as the prediction becomes less accurate. As shown in Figure 6.7, the variability of the predictions increases for the HF network as compared to the multifidelity network. The multifidelity network shows more consistently accurate predictions across the entire image instead of a localized region. The outliers near the fluid boundary appear typical of convolutional neural networks which often report lower accuracy near the boundary [9]; the original U-net architecture overcomes this through reflective padding the input layer and not padding any subsequent layers [19] (whereas we zero pad each convolutional layer), although other approaches exist to overcome this limitation for convolutional networks (e.g. [9]).

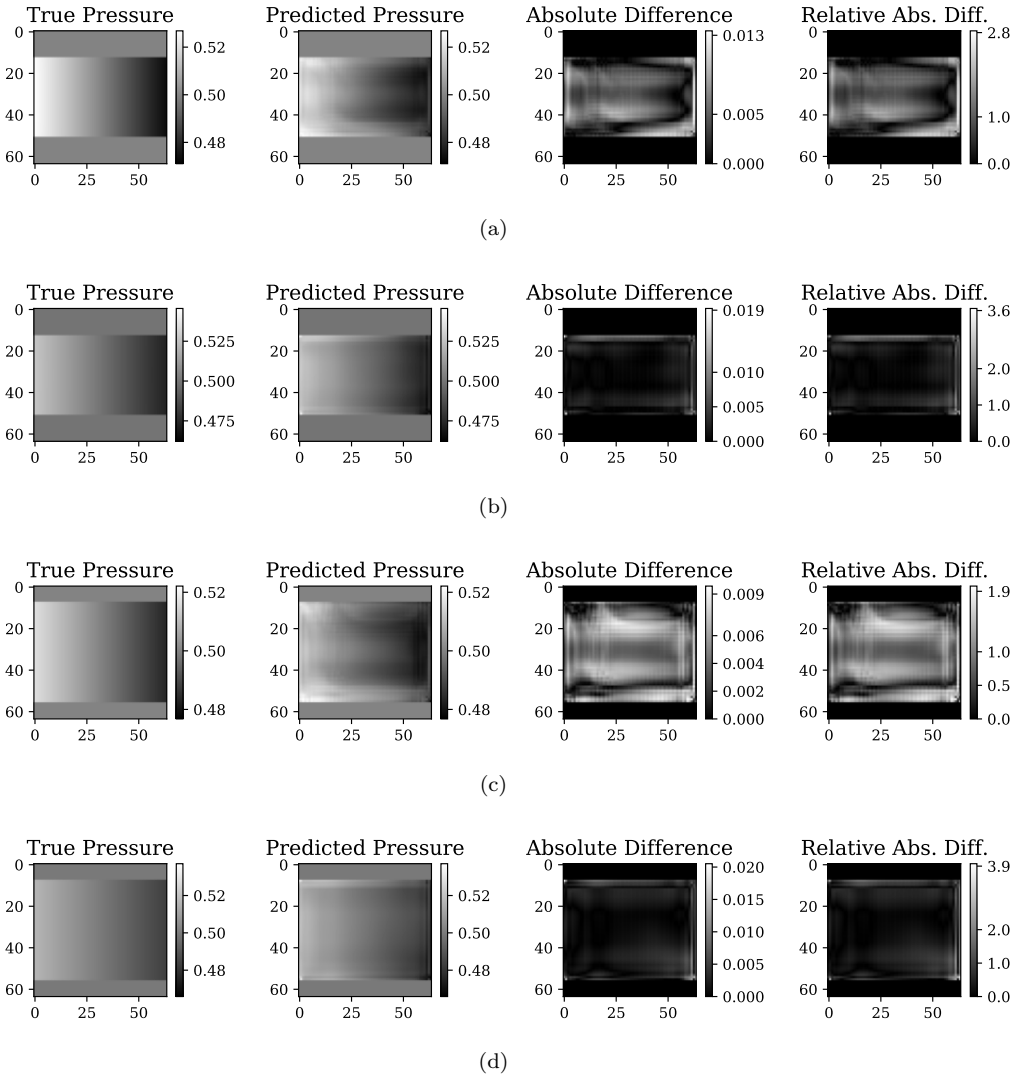
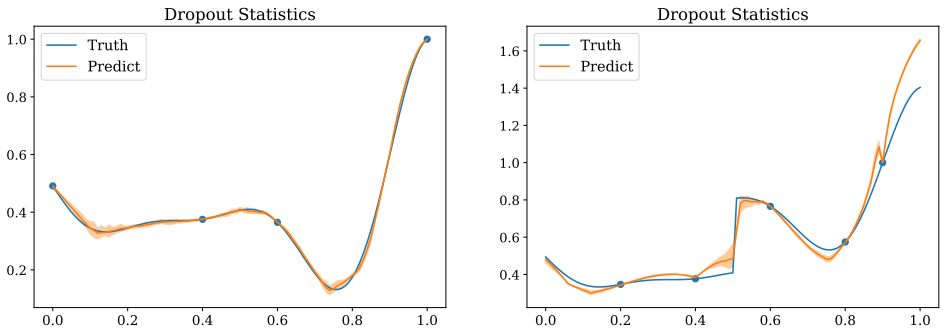
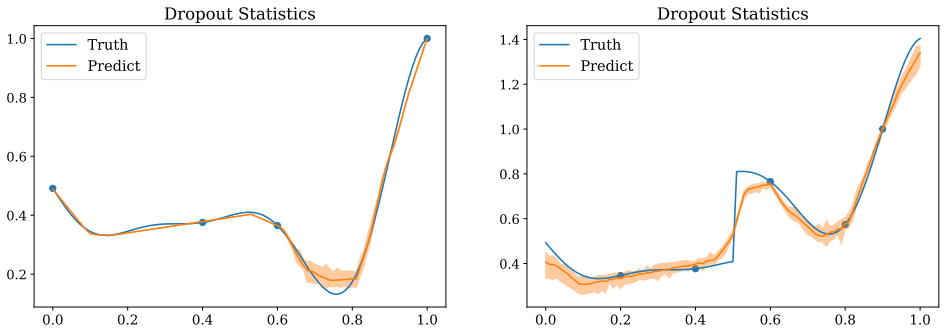


Fig. 6.4: Predictions from training with 32 HF pressure result examples on two Poiseuille flow configurations (a,c), respectively, from the test set. Predictions from implicit multifidelity network architecture trained with 32 high-fidelity and 116 low-fidelity pressure result examples (b,d).



(a) (b)

Fig. 6.5: Mean prediction and estimated 5%-95% confidence interval from multiple dropblock realizations in the prediction of (a) Equation (2.2) and (b) Equation (2.4), from a network with *tanh* activation functions.



(a) (b)

Fig. 6.6: Mean prediction and uncertainty 5%-95% confidence interval estimates from an ensemble of 50 dropblock realizations. A network with *ReLU* activations after each convolution layer except the last is used to predict (a) Equation (2.2) and (b) Equation (2.4).

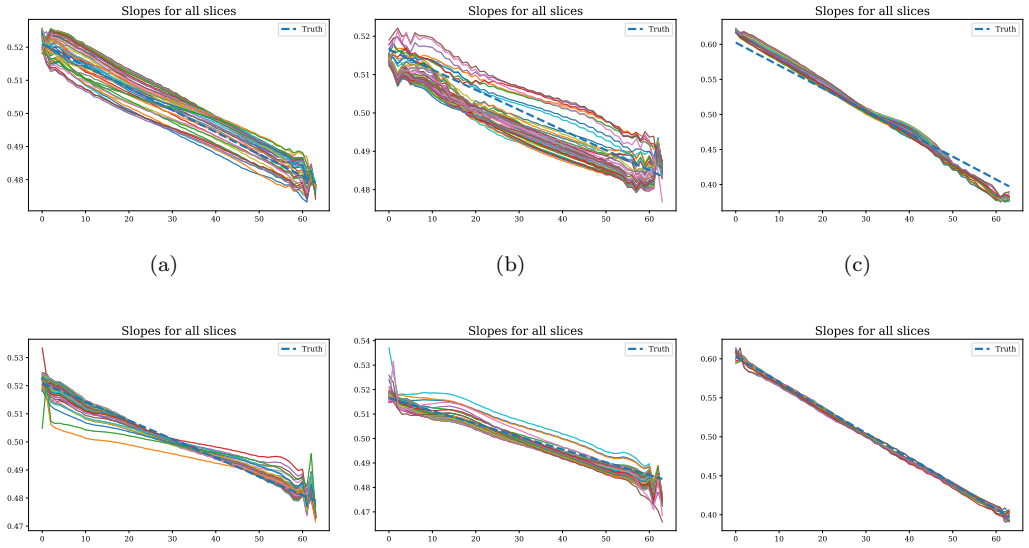


Fig. 6.7: Poiseuille predictions of test data along each slice inside the fluid region of the image. For a sample cylinder with a larger radius, there exists more slices. (a)-(c) are HF 32/0 network with highest test accuracy. (d)-(f) are multifidelity network with highest test accuracy.

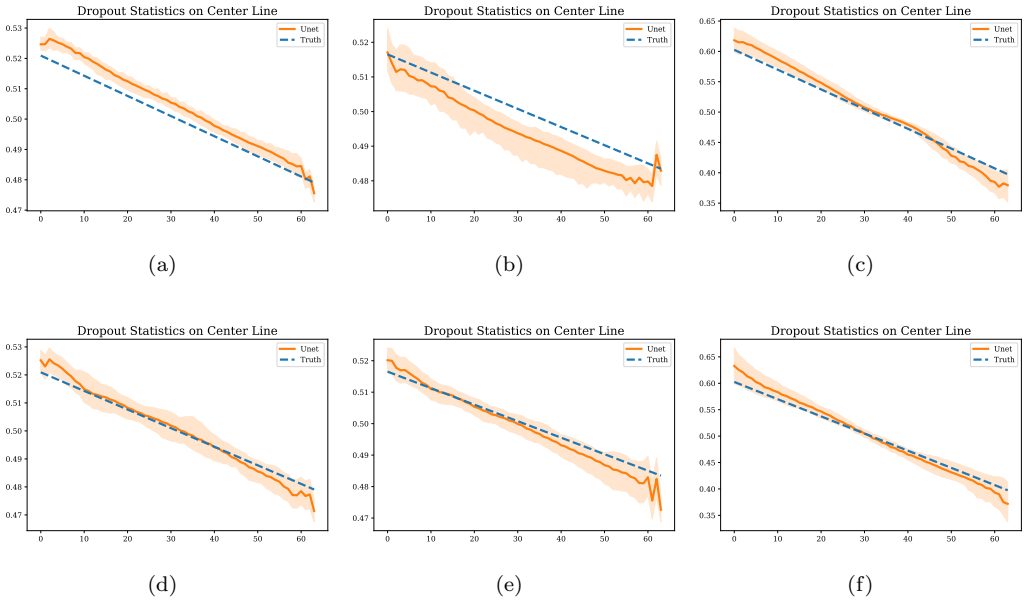


Fig. 6.8: Mean prediction and uncertainty 5%-95% confidence interval estimates from an ensemble of 50 dropblock realizations for Poiseuille predictions of test data along the centerline (center of the cylinder). (a)-(c) are HF 32/0 network with highest test accuracy. (d)-(f) are multifidelity network with highest test accuracy.

7. Conclusions and future work. Preliminary results indicate the possibility of improving the accuracy in dense regression using multifidelity networks with explicit or implicit feedback, by augmenting the training dataset through a large collection of inexpensive low-fidelity examples. In this work, we focus on convolutional neural networks, since they require a significantly smaller number of parameters with respect to fully connected layers having the same number of neurons. This property is crucial for predicting results from high-fidelity physics-based solvers having either high-dimensional inputs, high-dimensional outputs or both, where the number of parameters resulting from fully connected networks would simply be too large. Although these convolutional networks offer significant computational savings for high-dimensional inputs/outputs, we show that their performance is comparable to fully-connected multifidelity networks even for low-dimensional problems.

Additionally, an architecture resulting from an assembly of encoders and decoders has the flexibility to accommodate training data from a number of sources, including arbitrary LF predictors, ground-truth HF model results, LF data for coarse discretizations and general response surface surrogates. As an example, we combine in Section 6.2 a HF ground truth with its coarsened representations.

Our approach is designed to quantify variability in the prediction of the network. This is achieved through a dropblock regularizer that requires an end-to-end training task with multifidelity data. Given the ability of the proposed network to generate multiple predictions every time it is evaluated, we will investigate multifidelity variants of MC dropouts. Finally, this work focuses on networks where the inputs and outputs have the same dimensionality and will be extended in future work to more general cases.

8. Acknowledgements. The authors would like to thank Ishani Aniruddha Karmarkar for her assistance with testing fully connected multi-fidelity network implementations proposed in the literature, and would also like to acknowledge the three anonymous reviewers whose careful review and insightful comments greatly contributed to improve the quality of this article.

References.

[1] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. Acharya, V. Makarenkov, and S. Nahavandi, *A review of uncertainty quantification in deep learning: Techniques, applications and challenges*, arXiv preprint arXiv:2011.06225, (2020).

[2] P. Baldi and P. Sadowski, *Understanding dropout*, Advances in neural information processing systems, 26 (2013), pp. 2814–2822.

[3] S. De, J. Britton, M. Reynolds, R. Skinner, K. Jansen, and A. Doostan, *On transfer learning of neural networks using bi-fidelity data for uncertainty propagation*, International Journal for Uncertainty Quantification, 10 (2020).

[4] M. Dehghani, A. Mehrjou, S. Gouws, J. Kamps, and B. Schölkopf, *Fidelity-weighted learning*, arXiv preprint arXiv:1711.02799, (2017).

[5] Y. Gal and Z. Ghahramani, *Dropout as a bayesian approximation: Representing model uncertainty in deep learning*, in international conference on machine learning, PMLR, 2016, pp. 1050–1059.

[6] G. Ghiasi, T.-Y. Lin, and Q. Le, *Dropblock: A regularization method for convolutional networks*, arXiv preprint arXiv:1810.12890, (2018).

[7] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Y. W. Teh and M. Titterington, eds., vol. 9 of Proceedings of Machine Learning Research, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010, PMLR, pp. 249–256.

[8] E. Hüllermeier and W. Waegeman, *Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods*, Machine Learning, 110 (2021), pp. 457–506.

[9] C. Innamorati, T. Ritschel, T. Weyrich, and N. J. Mitra, *Learning on the edge: Explicit boundary handling in cnns*, CoRR, abs/1805.03106 (2018).

[10] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein, *nnu-net: a self-configuring method for deep learning-based biomedical image segmentation*, Nature methods, 18 (2021), pp. 203–211.

[11] A. Kendall and Y. Gal, *What uncertainties do we need in bayesian deep learning for computer vision?*, arXiv preprint arXiv:1703.04977, (2017).
[12] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017.
[13] H. Langseth and L. Portinale, *Bayesian networks in reliability*, Reliability Engineering & System Safety, 92 (2007), pp. 92–108.
[14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.
[15] X. Meng, H. Babaee, and G. Karniadakis, *Multi-fidelity bayesian neural networks: Algorithms and applications*, Journal of Computational Physics, 438 (2021), p. 110361.
[16] X. Meng and G. Karniadakis, *A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems*, Journal of Computational Physics, 401 (2020), p. 109020.
[17] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, *Image segmentation using deep learning: A survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2021).
[18] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics, 378 (2019), pp. 686–707.
[19] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in International Conference on Medical image computing and computer-assisted intervention, Springer, 2015, pp. 234–241.
[20] D. Schiavazzi, A. Nemes, S. Schmitter, and F. Coletti, *The effect of velocity filtering in pressure estimation*, Experiments in Fluids, 58 (2017), p. 50.
[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The journal of machine learning research, 15 (2014), pp. 1929–1958.
[22] Y. van Halder, B. Sanderse, and B. Koren, *Multi-level neural networks for pdes with uncertain parameters*, arXiv preprint arXiv:2004.13128, (2020).

ALGEBRAIC MULTIGRID FOR HIGHER-ORDER DISCRETIZATIONS OF THE STOKES EQUATIONS

ALEXEY VORONIN*, RAYMOND S. TUMINARO†, LUKE N. OLSON‡, AND SCOTT MACLACHLAN §

Abstract. This article focuses on developing monolithic algebraic multigrid (AMG) preconditioners for mixed finite-element discretizations of coupled partial differential equations (PDEs). In particular, we aim to construct robust AMG preconditioners for the Taylor-Hood, $\mathbb{Q}_2/\mathbb{Q}_1$ and $\mathbb{P}_2/\mathbb{P}_1$, discretizations of the Stokes equations. The key idea is to not directly apply AMG to the higher-order systems. Instead, AMG is applied to a stable low-order system based on either the $\mathbb{Q}_1$iso$\mathbb{Q}_2/\mathbb{Q}_1$ or the $\mathbb{P}_1$iso$\mathbb{P}_2/\mathbb{P}_1$ discretizations. The resulting AMG method is then used to precondition the high-order system. We explore a number of AMG coarsening parameters, identifying suitable choices that can be used to precondition finite-element problems on structured and unstructured grids.

1. Introduction. This study focuses on developing efficient monolithic algebraic multigrid (AMG) methods for the solution of mixed-finite element (MFE) discretizations of PDE systems. A common example of such systems is the Stokes equations, used to simulate incompressible viscous flow. Their discretization results in highly indefinite and block-structured saddle-point systems, which makes it challenging to construct an effective multigrid solver [6].

The most successful monolithic multigrid approaches for saddle-point type systems are of geometric multigrid (GMG) type [2,8,10], where the multigrid hierarchy is composed of a sequence of coarser discretizations on nested meshes, connected by canonical interpolation operators for each field. In the case of GMG, the ability to construct a grid-hierarchy is limited by the complexity of the domain with increasing difficulty from highly structured to unstructured meshes.

While scalar AMG solvers can easily address many matrix systems associated with unstructured meshes, the adaptation of AMG to saddle-point systems is not well understood. One difficulty is that AMG does not perform well on stiffness matrices constructed with higher-order bases, since they deviate from the M-matrices for which AMG was designed. A second difficulty is that the independent coarsening of each type of variable may lead to coarse discretization operators that violate an inf-sup (or LBB) stability condition [4]. Some AMG saddle-point approaches are described in [12, 18, 21, 22]. Previously, we proposed a preconditioner for a $\mathbb{Q}_2/\mathbb{Q}_1$ discretization of the Stokes equations based on applying GMG to a $\mathbb{Q}_1$iso$\mathbb{Q}_2/\mathbb{Q}_1$ problem, demonstrating that the solver convergence rate does not deteriorate as the mesh is refined [20]. The main goal of the current manuscript is to show that similar mesh-independent convergence rates can be achieved with AMG instead of GMG.

Two challenges must be addressed to develop a suitable AMG solver in our context. The first challenge centers on the AMG coarsening algorithm. While AMG convergence rates can be sensitive to coarsening heuristics even on scalar problems, we have observed that this sensitivity is much more pronounced in the context of mixed-finite element discretizations of Stokes equations. In particular, we observe (in unpublished experiments) that convergence rates may suffer significantly when the velocity unknowns are coarsened more rapidly than the pressure unknowns, as happens with some choices of coarsening algorithms. The second challenge centers on the choice of relaxation damping parameters. Commonly, local Fourier analysis has been used to guide the choice of these relaxation damping parameters within

*University of Illinois Urbana-Champaign, voronin2@illinois.edu
†Sandia National Laboratories, rstumin@sandia.gov
‡University of Illinois Urbana-Champaign, lukeo@illinois.edu
§Memorial University of Newfoundland, smaclachlan@mun.ca

GMG solvers. Unfortunately, it is not clear how to adapt Fourier analysis ideas to the types of highly irregular coarse meshes created within the AMG hierarchy. Thus, some alternative strategy is needed to determine the relaxation damping parameters.

The remainder of the report is organized as follows. section 2 summarizes the considered PDEs and discretizations, while section 3 describes the low-order preconditioning idea along with some of the AMG details necessary to address the low-order Stokes discretization. section 4.1 provides numerical results demonstrating optimal SA-AMG parameters for 2D Poisson problems on structured and unstructured meshes. section 4.2 provides results for the monolithic SA-AMG solvers applied to the Stokes problem. section 5 presents the conclusions and future steps.

2. Problem formulation.

2.1. Discretization of the Stokes equations. The two-dimensional Stokes equations are given by

$$-\nabla^2 \boldsymbol{u} + \nabla p = \boldsymbol{f} \tag{2.1a}$$

$$-\nabla \cdot \boldsymbol{u} = 0, \tag{2.1b}$$

where $\boldsymbol{u}$ is a vector-valued function representing the velocity of the fluid, p is a scalar pressure function, and $\boldsymbol{f}$ is a vector forcing term. For simplicity, we assume homogeneous Dirichlet conditions on $\boldsymbol{u}$ over the boundary of a domain Ω.

We consider a mesh over Ω and two finite-dimensional spaces of the form $\boldsymbol{\mathcal{X}}^h \subset \boldsymbol{H}_0^1(\Omega)$ and $\mathcal{M}^h \subset L_2(\Omega)$, where $\boldsymbol{\mathcal{X}}^h$ satisfies the appropriate homogeneous Dirichlet boundary conditions. The resulting discrete weak formulation of (2.1) corresponds to finding $\boldsymbol{u} \in \boldsymbol{\mathcal{X}}^h$ and $p \in \mathcal{M}^h$ such that

$$\int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - \int_\Omega p \nabla \cdot \boldsymbol{v} = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \tag{2.2a}$$

$$-\int_\Omega q \nabla \cdot \boldsymbol{u} \qquad\qquad = 0, \tag{2.2b}$$

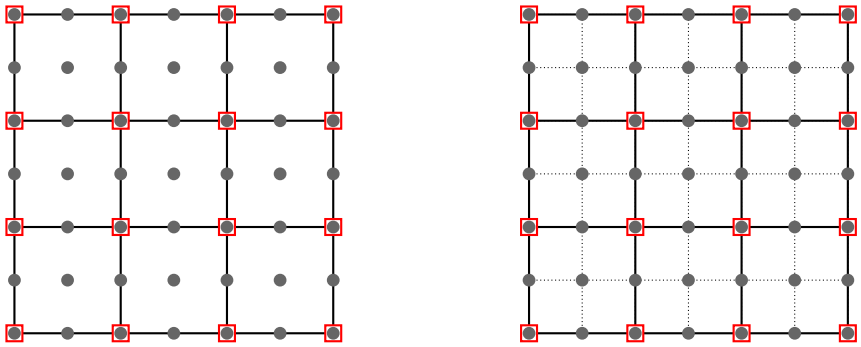
for all $q \in \mathcal{M}^h$ and $\boldsymbol{v} \in \boldsymbol{\mathcal{X}}^h$.

In this report, we focus on four types of stable mixed finite-element discretizations for $\boldsymbol{\mathcal{X}}^h$ and $\mathcal{M}^h$. The first discretization is the $\mathbb{Q}_2/\mathbb{Q}_1$ discretization (also known as the *Taylor-Hood* discretization), which uses a biquadratic representation for the velocity components and a bilinear representation for the pressure on quadrilateral meshes. The second discretization, $\mathbb{Q}_1\text{iso}\mathbb{Q}_2/\mathbb{Q}_1$, replaces the $\mathbb{Q}_2$ space for velocities with a linear $\mathbb{Q}_1$ approximation on a once-refined mesh. Figure 2.1 depicts the spatial location associated with different unknown types on a sample mesh for the $\mathbb{Q}_2/\mathbb{Q}_1$ and $\mathbb{Q}_1\text{iso}\mathbb{Q}_2/\mathbb{Q}_1$ discretizations. In addition to quadrilateral elements ($\mathbb{Q}$), we study AMG convergence for the analogous triangular element ($\mathbb{P}$) discretizations: $\mathbb{P}_2/\mathbb{P}_1$ and $\mathbb{P}_1\text{iso}\mathbb{P}_2/\mathbb{P}_1$.

All four of the above discretizations satisfy the *inf-sup* (or LBB) stability conditions, thereby yielding a stable discretization of the Stokes problem [6, 7]. The following saddle point matrix system emerges from (2.2)

$$K \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ 0 \end{bmatrix} = b, \tag{2.3}$$

where matrix A corresponds to the discrete vector-Laplacian and B represents the negative of the discrete divergence operator. Here, we overload the notation and use $\boldsymbol{u}$ and p to denote the discrete velocities and pressure for the remainder of the report.



(a) $\mathbb{Q}_2/\mathbb{Q}_1$ (b) $\mathbb{Q}_1\text{iso}\mathbb{Q}_2/\mathbb{Q}_1$

Fig. 2.1: Meshes and degrees of freedom for the $\mathbb{Q}_2/\mathbb{Q}_1$ and $\mathbb{Q}_1\text{iso}\mathbb{Q}_2/\mathbb{Q}_1$ discretizations. Dark circle [●] corresponds to the velocity locations (two components of velocity per marker), and red squares □ correspond to the pressure locations.

3. Low-Order Preconditioner. Lower-order systems have been used to precondition higher-order operators in many different contexts based on the defect-correction method [5, 11, 14, 17]. Generally, the idea centers on developing an auxiliary low-order operator that is used for preconditioning purposes. Specifically, a preconditioning strategy such as GMG or AMG is applied to the auxiliary low-order operator instead of being applied to the original high-order system. A preconditioner for the high-order system can be developed by first applying an inexpensive relaxation procedure directly to the high-order system. A residual is then computed and transferred to the basis associated with the low-order operator. A correction for the high-order system can be generated by applying a GMG or AMG V-cycle to the low-order system and, finally, transferring the resulting correction back to the high-order basis. In this way, one avoids applying GMG or AMG directly to the high-order system.

There are several reasons why it might be undesirable to apply a similar preconditioning strategy directly to the high-order system. In our context, we find that AMG performs poorly when applied directly to the high-order system. This is partially related to the fact that AMG methods tend to aggressively coarsen high-order systems (due to the denser sparsity pattern associated with high-order discretizations). In the case of the Stokes operator, this can result in a much more aggressive coarsening of the velocity variables than the pressure variables, which has a tendency to produce coarse discretizations with poor stability properties.

3.1. AMG Grids and Interpolation. In this report, we focus on *Smoothed Aggregation* (SA-)AMG. Applied to scalar linear systems, AMG setup first determines the coarse-grid and then defines an appropriate interpolation operator, P. For a 2D system of PDEs like Stokes, we construct a total of three scalar SA-AMG hierarchies: one for each component of velocity, and one for the pressure field. The individual interpolation operators, P_{v_x} and P_{v_y} for velocity and P_p for pressure, are then combined in a block-diagonal matrix that has the form

$$P = \begin{bmatrix} P_{v_x} & & \\ & P_{v_y} & \\ & & P_p \end{bmatrix} \tag{3.1}$$

on each level of the hierarchy. The block-diagonal structure of the interpolation operator preserves the saddle-point structure of the coarse-grid operator, $K_c = P^T K P$.

In scalar SA-AMG, each coarse-grid DoF is defined as a collection of fine-level DoFs (also known as an aggregate), where the aggregation of the fine-level DoFs is based on the undirected adjacency graph of the matrix. The aggregate information is then used to define an interpolation operator that accurately interpolates error that is not effectively reduced by relaxation. The interpolation is then used to compute the coarse-grid via the Galerkin product. The SA-AMG setup phase is outlined in algorithm 1.

Algorithm 1 SA based AMG Setup

Input: A_1, $n \times n$ fine-level matrix
C_1, $n \times c$ vectors representing smooth error components on fine grid
Output: $A_1,\ldots,A_{l_{max}}$, Grid hierarchy
$P_1,\ldots,P_{l_{max}-1}$, Interpolation Operators

for $l = 1, ..., l_{max} - 1$ **do**
  $S \leftarrow \text{strength}(A_k)$ // Compute strength-of-connection
  $Agg \leftarrow \text{aggregate}(S)$ // Aggregate nodes in the strength graph
  $T_k, C_{k+1} \leftarrow \text{tentative}(C_k, Agg)$ // Construct tentative interpolation operator
  $P_k \leftarrow \text{smooth\_interp}(A_k, T_k)$ // Improve interpolation operator
  $A_{k+1} = P_k^T A_k P_k$ // Compute coarse-grid operator

While each step of the algorithm plays an important role in the construction of an efficient AMG solver, in this report we will focus on the strength of connection and tentative interpolation smoothing steps.

The strength-of-connection (SoC) step is used to determine sparsity patterns for the grid-transfer operators. The standard SA algorithm uses a symmetric SoC, which is based directly on the matrix stencil. The symmetric SoC measure says that DoFs i and j are strongly connected if

$$|A_{ij}| \geq \theta \sqrt{|A_{ii} A_{jj}|}, \tag{3.2}$$

where θ is the threshold value and A_{ij} are the entries of the matrix being coarsened. For the rest of the report, we take $\theta = 0$, identifying any nonzero connection as a strong connection. However, for more complex problems and discretizations, the matrix coefficients alone do not contain sufficient information to correctly identify the directionality of algebraically smooth errors, which is a crucial step for a robust AMG solver. In those situations, we turn to the *evolution* SoC measure, which makes point-wise SoC choices based on the local algebraically smooth error and on the local behavior of tentative interpolation [15].

The second SA-AMG component examined in this work is the choice of tentative interpolation smoothing routine, which helps to determine coefficients for the final grid-transfer operators. For generally well-formulated systems, such as low-order discretizations of the Poisson equation, a Jacobi smoothing operation (with a damping parameter of 4/3) tends to perform well at improving the grid transfer [3]. Thus, we use this to improve the P_{v_x} and P_{v_y} grid transfers where the diagonal blocks of the discrete vector Laplacian, A, from (2.3) are used in the Jacobi operation. However, no such matrix is available for the pressure field interpolation operator as the diagonal block associated with pressure in (2.3) is identically zero.

Instead, we define $A_p = BB^T$, which leads to a pressure Poisson-like operator, albeit with a much wider stencil than desired. We also assume that a user supplies a pressure mass

matrix, M_p. To avoid coarsening difficulties associated with A_p's wide stencil, we use M_p for the AMG coarsen/aggregation algorithm, as it has a more amenable sparsity pattern. In addition to using Jacobi operator for propagator smoothing, we explore the energy-minimizing framework (EMIN-AMG). EMIN-AMG avoids some difficulties associated with A_p's wider stencil and has been shown to be effective within AMG for Stokes problems in [18].

3.2. Relaxation. Classical multigrid relaxation methods such as Jacobi and Gauss-Seidel are not well-defined for saddle-point systems due to the large zero block and the indefinite system. As a result, we turn to additive Vanka [8, 9], a coupled box-relaxation scheme.

Let n_p be the number of pressure DoFs. Algebraic Vanka relaxation partitions the $n \times n$ system matrix, K, into n_p overlapping saddle-point problems. Each saddle-point problem, or patch, consists of all of the velocity degrees of freedom with non-zero coefficients in row i of the divergence matrix, B, along with the pressure degree-of-freedom corresponding to this row. The adjacent velocity rows and columns are then extracted from the discrete vector Laplacian matrix, A, to form the local patch matrix.

For each patch, indexed by pressure DoF i, we form a (binary) restriction operator, V_i, which selects those entries in a global vector that appear on patch i. The system matrix is then projected onto the patch DoFs by a triple matrix product, $V_i K V_i^T$. The matrix representation of single iteration of additive Vanka relaxation is then given by

$$M_V^{-1} = \omega \sum_{i=1}^{n_p} V_i^T W_i \left(V_i K V_i^T \right)^{-1} V_i,$$

where ω is the global damping parameter and W_i is a diagonal weighting matrix defined so that each diagonal entry is equal to the reciprocal of the number of patches that contain the associated degree of freedom. In this report, we perform a parameter scan on ω, identifying the optimal ω value based on the iteration count of FGMRES preconditioned with the 2-level AMG method. Alternatively, one can avoid setting a relaxation parameter (i.e., take $\omega = 1$) by embedding the Vanka scheme within an outer Chebyshev polynomial or a Krylov method to accelerate the relaxation scheme without a damping parameter [1].

4. Numerical Results. In this section, we consider the numerical solution of 2D Poisson and Stokes problems. For convenience, the higher-order discretization matrices, A_0 and K_0, are assembled using Firedrake [13, 19]. In case of the Poisson problem, the lower-order discretization matrix, A_1, is assembled by forming a finer-mesh $\mathbb{Q}_2$ discretization, then applying coarsening-in-order to the DoFs. For Stokes problem, a similar approach is followed. The $\mathbb{Q}_1\text{iso}\mathbb{Q}_2/\mathbb{Q}_1$ discretization matrices are assembled by forming a finer-mesh $\mathbb{Q}_2/\mathbb{Q}_1$ discretization, and then applying coarsening-in-order to the velocity DoFs and coarsening-in-space to the pressure DoFs. The low-order AMG preconditioner is implemented using the PyAMG library [16].

All the convergence results are for FGMRES preconditioned with a single V(2,2)-cycle of AMG, where (2, 2) are the number of pre- and post- relaxation sweeps. All the convergence plots report the absolute Euclidean norm of the residual based on the A_0 or K_0 operators. The solver convergence tolerance is selected to be reducing this norm below 10^{-12} or 100 iterations, whichever is reached first.

In order to demonstrate the effect of element/mesh choice on the convergence, we consider structural quadrilateral and unstructured triangular grids, displayed in Fig. 4.1 on the left and right, respectively. We omit results for the structured triangular meshes because they give similar convergence patterns as the unstructured triangular meshes.

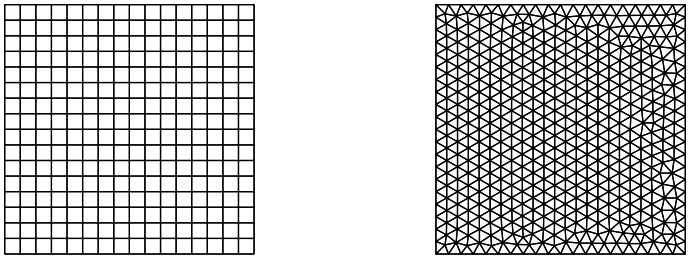


Fig. 4.1: Structured Quadrilateral Mesh (left) and Unstructured Triangular Mesh (right).

4.1. 2D Poisson. We start by verifying that SA-AMG low-order preconditioners can achieve grid-spacing, h-, independent convergence for the 2D Poisson problems. A 2D Poisson problem is equivalent to solving

$$A_x u = f_x, \tag{4.1}$$

where A_x is a single component of the vector-Laplacian operator A in (2.3) and u and f_x are corresponding scalar vectors. We assume Dirichlet boundary conditions along the boundaries of a square mesh.

The SA-AMG hierarchy is set up using symmetric SoC, standard aggregation, and Jacobi tentative interpolation smoothing with damping parameter 4/3. For pre- and post-relaxation, we use 2 sweeps of weighted Jacobi with optimal damping parameters, ω, determined by performing line-search for two-grid AMG hierarchies of various sizes. The correction on the coarsest grid is computed using a direct solver. The left plot in Fig. 4.2 shows that FGMRES preconditioned with AMG converges in an h-independent fashion for structured quadrilateral meshes. However, when we solve the same problem on an unstructured triangular mesh, as seen in the right plot of Fig. 4.2, the convergence is only h-independent until an absolute residual of 10^{-5}, after which the convergence rate drops off in proportion to the depth of AMG hierarchy. Similar results were observed for structured triangular meshes (not depicted). This suggests that the coarse-grid correction is not effectively reducing some of the smooth modes.

We examine the AMG aggregation patterns for the two types of problems to try to explain the convergence difference in Fig. 4.2. Fig. 4.4 shows that for structured quadrilateral meshes we get "nice" rectangular aggregates. Meanwhile in unstructured case, shown in Fig. 4.5(a), the aggregates become significantly more irregular, containing many aggregates with "offshoot" DoFs.

To improve the aggregation, we turn to the evolution strength measure, while keeping the rest of the AMG parameters the same. The aggregation plot in Fig. 4.5(b) demonstrates that this modification results in more cohesive aggregates which, in turn, helps recover the h-independent convergence as seen in the right plot of Fig. 4.3. The convergence for the structured quadrilateral meshes is unaffected, as shown in the left plot of Fig. 4.3. The effect of evolution strength measure on aggregation is further recorded in table 4.1. Here we measure the coarsening rate (CR) between two consecutive levels of the AMG hierarchy. The CR is computed as the number of fine-level DoFs divided by the number of coarse-level DoFs. For the 2D Poisson problem, SA-AMG with standard aggregation should achieve a coarsening rate of about 9. That is clearly the case for structured quadrilateral meshes. For unstructured triangular meshes, the symmetric strength of connection leads to more

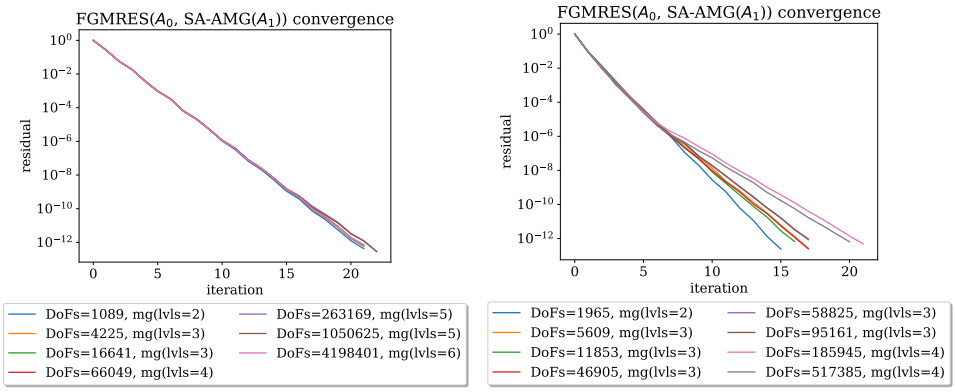


Fig. 4.2: Convergence of FGMRES preconditioned with AMG, which uses symmetric SoC, on structured quadrilateral (left) and unstructured triangular (right) meshes. The weighted-Jacobi relaxation damping parameters are 4/3 and 1.6, respectively.

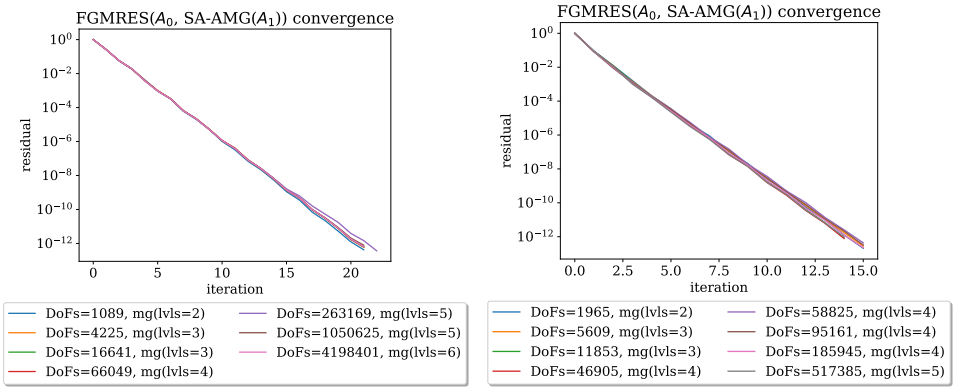


Fig. 4.3: Convergence of FGMRES preconditioned with AMG, which uses evolution SoC, on structured quadrilateral (left) and unstructured triangular (right) meshes. The weighted-Jacobi relaxation damping parameters are 4/3 and 1.6, respectively.

Table 4.1: Poisson Problem: AMG coarsening rates for each coarse-grid.

		Structured Quadrilateral		Unstructured Triangular	
	SoC	Symmetric	Evolution	Symmetric	Evolution
lvl	1	9.00	9.00	7.96	7.97
	2	8.97	8.95	15.9	9.04
	3	9.00	9.07	21.2	9.30
	4	8.80	8.67		9.30
	5	8.86	7.62		

aggressive coarsening than desired. The evolution strength measure helps rectify this by more accurately identifying the direction of smooth error on the coarser grids.

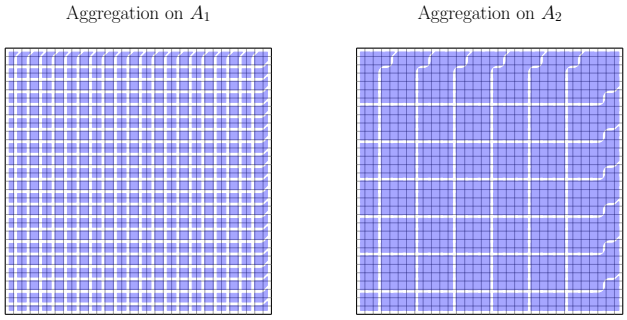


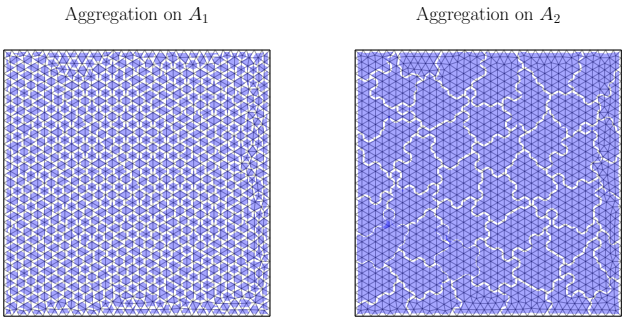
Fig. 4.4: Aggregates for the 3-level AMG hierarchy based on the 4225 DoF structured quadrilateral Poisson problem.

4.2. 2D Stokes. We next examine how the approach described in section 3 works on structured and unstructured Stokes problems on a uniform mesh. The boundary conditions are of lid-driven cavity problem with symmetric parabolic speed profile for the velocity of the lid. The velocity field interpolation operators are constructed using symmetric SoC, standard aggregation, and Jacobi tentative interpolation smoothing. The pressure interpolation is constructed using symmetric SoC and standard aggregation on a pressure mass-matrix, followed by EMIN-AMG on $A_p = BB^T$ to determine interpolation weights. For relaxation, we use two pre- and post- Vanka sweeps with optimal damping parameters, ω, determined by performing line-search for two-grid AMG hierarchies of various sizes. To address the singularity of this system the correction on the coarsest grid is computed using a pseudoinverse. The convergence plots for the solvers using these sets of parameters are depicted in Fig. 4.6.

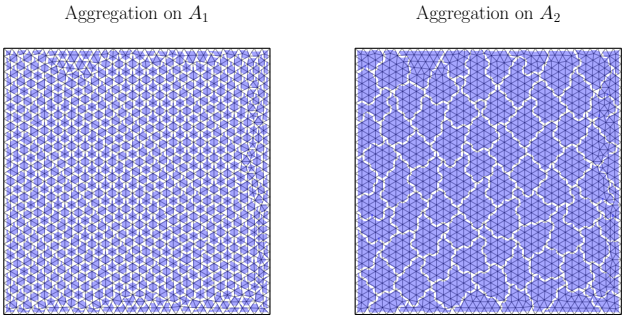
For the structured domains, we obtain close to h-independent convergence, as seen in the left plot of Fig. 4.6. In case of the unstructured triangular grids, the right plot of the Fig. 4.6, the convergence degrades with the depth of AMG hierarchy in a similar fashion to the Poisson problem in section 4.1. The convergence for the unstructured problems can once again be improved using the evolution strength measure. Performing a parameter search on SoC, we identified that the best convergence is achieved when the evolution SoC is applied to the pressure field and the velocity fields still use the symmetric SoC. The convergence plots in Fig. 4.7 suggest that the choice of tentative interpolation smoothing plays a significant role.

Our original intuition was that preserving the 4-to-1 ratio of DoFs between the individual velocity components and the pressure (matching the ratio prescribed by $\mathbb{Q}_2/\mathbb{Q}_1$ and $\mathbb{P}_2/\mathbb{P}_1$ discretizations) would be necessary for obtaining robust monolithic AMG solvers for the Stokes problem. Maintaining that ratio without additional code modifications proved to be difficult. The number of DoFs on each level depends on the SoC choices as well as the tentative interpolation smoothing choices. This is supported by the data in table 4.2, which shows the DoF ratio (DR), defined as the mean number of DoFs for each component of the velocity divided by the number of pressure DoFs on that level.

table 4.2 also demonstrates that the 4:1 ratios on all the levels of AMG hierarchy do not necessarily result in optimal convergence. The AMG hierarchy with the (Symmetric,



(a) Unstructured Triangular Mesh, Symmetric SoC



(b) Unstructured Triangular Mesh, Evolution SoC

Fig. 4.5: Effect of SoC choice on the shape of the aggregates. Both symmetric and evolution SoC plots are based on 3-level AMG hierarchy for a 5609 DoF unstructured Poisson problem.

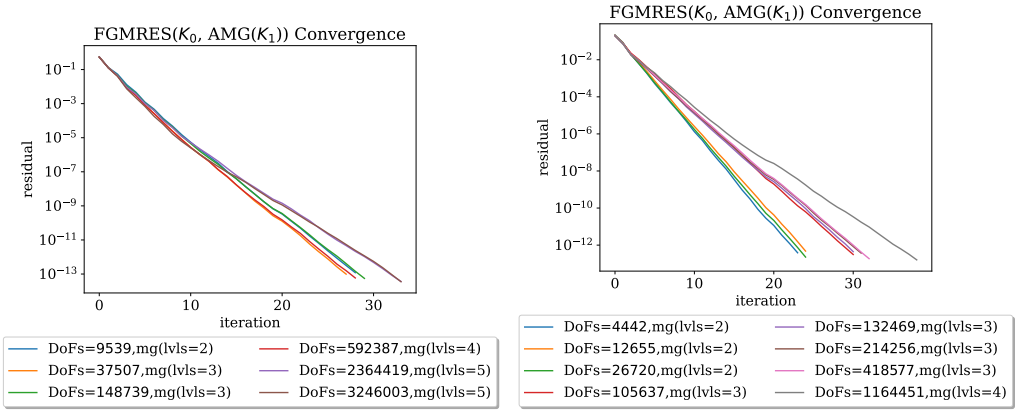


Fig. 4.6: Structured Quadrilateral Mesh (left) and Unstructured Triangular Mesh (right). The Vanka relaxation parameters, ω, are 0.54 and 0.44, respectively.

Symmetric) SoC pair matches the desired coarsening rates the closest, while still resulting in hierarchy-depth dependent convergence.

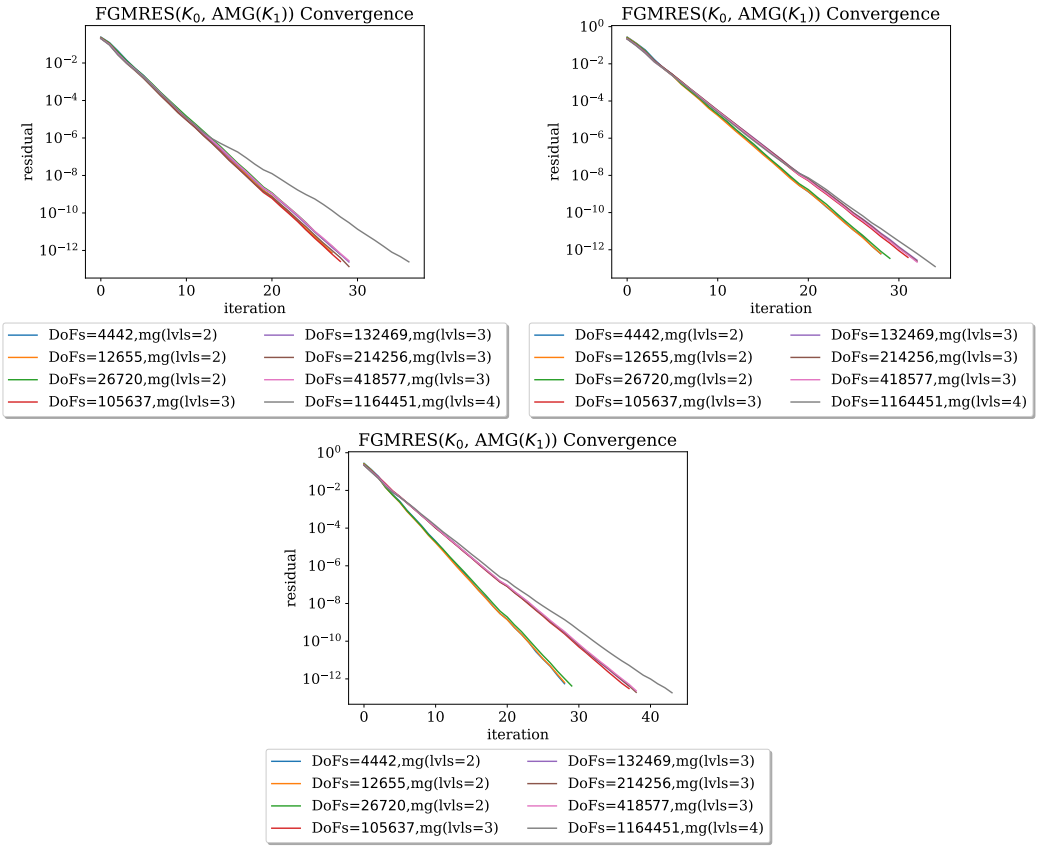


Fig. 4.7: Convergence of FGMRES preconditioned with AMG. All the plots above utilize symmetric strength of connection for the velocity field and evolution strength for the pressure field. The choices for tentative interpolation operator smoothing parameters are listed in the following order (velocity, pressure) for each plot: top left (Jacobi, Jacobi), top right (Energy, Jacobi), and bottom (Energy, Energy). The Vanka relaxation parameters, ω, are 0.39, 0.29, and 0.29, respectively.

Table 4.2: The DR ratios for the AMG hierarchies based on the largest problem size (DoFs=1164451). The SoC and T(entative interpolation) smoothing parameters are listed in the (velocity fields, pressure field) order and are abbreviated using the first 2-3 letters of the parameter's name: Jac(obi), En(ergy), Sym(mmetric), Ev(olution).

SoC	(Sym, Sym)	(Sym, Ev)		
T Smooth.	(Jac, En)	(Jac, Jac)	(En, Jac)	(En, En)
DR(lvl=1)	4.0	4.0	4.0	4.0
DR(lvl=2)	3.9	7.1	7.1	7.1
DR(lvl=3)	4.1	4.3	4.3	7.5
DR(lvl=4)	3.8	2.0	2.0	4.4

5. Conclusions and Future Work. In this report, we demonstrate that the low-order, $\mathbb{Q}_1\text{iso}\mathbb{Q}_2/\mathbb{Q}_1$and $\mathbb{P}_1\text{iso}\mathbb{P}_2/\mathbb{P}_1$, finite-element discretizations can be used to construct effective AMG preconditioners for the higher-order, $\mathbb{Q}_2/\mathbb{Q}_1$ and $\mathbb{P}_2/\mathbb{P}_1$, finite-element discretizations of the Stokes equations. To achieve this, we first explore the convergence of AMG preconditioned with FGMRES for the Poisson problem on structured and unstructured grids. We find that the convergence of the AMG solvers for the unstructured grids degrades with each additional level of the AMG hierarchy. Switching from symmetric to evolution strength of connection measures helps to recover h-independent convergence. In the case of the Stokes problem, we observe a similar AMG-depth-dependent convergence drop-off. While switching to the evolution strength measure on the pressure field helps to improve the convergence, we still observe some drop-off for larger problems.

As the next step, we plan on wrapping the relaxation methods within an outer Chebyshev polynomial, to eliminate the possibility of convergence drop-off due to level-dependent relaxation parameters. In addition, we have been exploring AMG coarsening techniques that enforce the 4:1 velocity-to-pressure DoF ratio.

References.

[1] J. H. Adler, T. R. Benson, E. C. Cyr, P. E. Farrell, S. P. MacLachlan, and R. S. Tuminaro, *Monolithic multigrid methods for magnetohydrodynamics*, SIAM Journal on Scientific Computing, (2021), pp. S70–S91.

[2] J. H. Adler, T. R. Benson, and S. P. MacLachlan, *Preconditioning a mass-conserving discontinuous Galerkin discretization of the Stokes equations*, Numerical Linear Algebra with Applications, 24 (2017), p. e2047.

[3] M. Brezina, *Robust iterative methods on unstructured meshes*, University of Colorado at Denver, 1997.

[4] F. Brezzi and M. Fortin, *Mixed and hybrid finite element methods*, Springer-Verlag, New York, 1991.

[5] M. O. Deville, P. F. Fischer, E. Mund, et al., *High-order methods for incompressible fluid flow*, vol. 9, Cambridge University Press, 2002.

[6] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*, Oxford University Press, USA, 2014.

[7] A. Ern and J.-L. Guermond, *Theory and Practice of Finite Elements*, vol. 159 of Applied Mathematical Sciences, Springer-Verlag New York, 2004.

[8] P. E. Farrell, Y. He, and S. MacLachlan, *A local Fourier analysis of additive Vanka relaxation for the Stokes equations*, Numer. Linear Alg. Appl., 28 (2021). https://doi.org/10.1002/nla.2306.

[9] P. E. Farrell, M. G. Knepley, L. Mitchell, and F. Wechsung, *PCPATCH: software for the topological construction of multigrid relaxation methods*, ACM Trans. Math. Softw., 47 (2021). https://doi.org/10.1145/3445791.

[10] B. Gmeiner, M. Huber, L. John, U. Rüde, and B. Wohlmuth, *A quantitative performance study for Stokes solvers at the extreme scale*, J. Comput. Sci., 17 (2016), pp. 509–521.

[11] J. Heys, T. Manteuffel, S. F. McCormick, and L. Olson, *Algebraic multigrid for higher-order finite elements*, Journal of Computational Physics, 204 (2005), pp. 520–532.

[12] A. Janka, *Smoothed aggregation multigrid for a Stokes problem*, Comput. Vis. Sci., 11 (2008), pp. 169–180.

[13] R. C. Kirby and L. Mitchell, *Solver composition across the PDE/linear algebra barrier*, SIAM Journal on Scientific Computing, 40 (2018), pp. C76–C98.

[14] L. Olson, *Algebraic multigrid preconditioning of high-order spectral elements for elliptic problems on a simplicial mesh*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2189–2209.

[15] L. N. Olson, J. Schroder, and R. S. Tuminaro, *A new perspective on strength measures in algebraic multigrid*, Numerical Linear Algebra with Applications, 17 (2010), pp. 713–733.

[16] L. N. Olson and J. B. Schroder, *PyAMG: Algebraic multigrid solvers in Python v4.0*, 2018. Release 4.0.

[17] S. A. Orszag, *Spectral methods for problems in complex geometrics*, in Numerical Methods for Partial Differential Equations, Elsevier, 1979, pp. 273–305.

[18] A. Prokopenko and R. S. Tuminaro, *An algebraic multigrid method for Q_2-Q_1 mixed discretizations of the Navier–Stokes equations*, Numerical Linear Algebra with Applications, 24 (2017), p. e2109.

[19] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, *Firedrake: Automating the finite element method by composing abstractions*, ACM Trans. Math. Softw., 43 (2016), pp. 24:1–24:27.

[20] A. Voronin, Y. He, S. MacLachlan, L. N. Olson, and R. Tuminaro, *Low-order preconditioning of the Stokes equations*, arXiv preprint arXiv:2103.11967, (2021).

[21] M. Wabro, *Coupled algebraic multigrid methods for the Oseen problem*, Comput. Vis. Sci., 7 (2004), pp. 141–151.
[22] ———, *AMGe—coarsening strategies and application to the Oseen equations*, SIAM J. Sci. Comput., 27 (2006), pp. 2077–2097.

II. Software & High Performance Computing

Articles in this section discuss the implementation of high performance computing (HPC) and productivity software. In many cases, performance improvements and portability are demonstrated for many-core architectures, such as conventional multicore CPUs, the Intel Many Integrated Core coprocessor (MIC), and graphical processing units (GPU).

1. *Fox, Modine,* and *Rajamanickam* employ **Atom-Decomposed Neural Modeling** techniques to reduce the training burden for molecular dynamics electronic density of state prediction modeling, improving by order of magnitude when compared to other machine learning density of state models.
2. *Gilbert, Madduri, Boman,* and *Rajamanickam* investigate **Fine-Grained Parallel Refinement Algorithms** that are applicable to GPUs. These algorithms are benchmarked on an NVIDIA GPU and an AMD processor.
3. *Kruse, Marts,* and *Dosanjh* explore application design for exascale computing by measuring the overhead of **MiniMod** and additionally consider three communication granularities.
4. *Li* and *Kolla* improve the computation of **Higher Order Joint Moment Tensors** by using Khatri-Rao products. They demonstrate a significant speed up when comparing the approach against state-of-the-art computation.
5. *Logan, Lofstead, Levy, Widener, Sun,* and *Kougkas* investigate the potential of **Persistent Memory (PMEM)** devices as storage through the use of the portable I/O library **pMEMCPY**. They demonstrate faster performance in a comparison to other parallel I/O libraries.
6. *Low* and *Wilson* showcase the geo-spatial data visualization capabilities of **Dash**, a Python package providing the ability to create interactive web apps.
7. *Luca* and *Wang* develop a simple fully connected neural network model to learn **Grid Cell Encodings**, providing a baseline for comparisons against other network architectures in future work.
8. *McCrary, Devine,* and *Younge* interface Chapel, a language for **Productive Parallel Computing using Global Address Spaces**, with the Grafiki and Trilinos libraries. Two approaches are considered and scalability and performance metrics are given for each.
9. *Pereyra* and *Wood* test different classes of machine learning models for use within production level **LAMMPS-MD** simulations. Benchmark speed and stability tests are performed and methods are provided for diagnosing reliable machine learning models for use within the application space.
10. *Woods* and *Curry* execute performance testing of the **Advanced Tri-Lab Software Environment** under various testing conditions.

J.D. Smith
E. Galvan

November 1, 2021

ACCELERATING ELECTRONIC STRUCTURE CALCULATION WITH ATOM-DECOMPOSED NEURAL MODELING

JAMES FOX *, N. A. MODINE †, AND SIVASANKARAN RAJAMANICKAM ‡

Abstract. Advances in deep learning has opened new and exciting opportunities for modeling fundamental properties of materials at much lower cost than is possible with existing quantum mechanical tools. One such property is the electronic density of states (DOS), a key component of electronic structure calculations in molecular dynamics simulation. Existing ML approaches for accurately predicting the DOS through LDOS supervision involves millions of samples for resolving the DOS of a single configuration of atoms, which poses formidable computational demands. This work presents a novel atom-centered decomposition of DOS for supervision, which reduces the number of samples for training and evaluation by orders of magnitude compared to LDOS supervision. Combined with a new model for learning atomic environment descriptions end-to-end, our approach allows resolving downstream quantities such as band energy of melting point aluminum at a fraction of the cost of LDOS, with matching or greater accuracy.

1. Introduction. The ability to perform accurate materials modeling across different length and time scales holds promise in advancing key directions of material science research. Example applications include the discovery of new desirable materials, or their behavior under extreme conditions. Molecular dynamics (MD) simulations and their efficient implementation provide a principled framework towards this goal. However, the primary challenge is being able to faithfully extend information from quantum mechanical calculation at atomic scales to simulations operating at larger system and time scales.

Kohn-Sham density functional theory (DFT) has been the quantum mechanical method of choice for calculations fundamental to driving simulations at the atomic scale. Important outputs from DFT include the energy and forces of a system as a function of the atomic positions, which enable moving forward the dynamics of the simulation in time according to physical principles. Despite the widespread use of DFT, its effectiveness is limited to systems on the scale of hundreds of atoms, as its computational cost scales as the cube of the system size and becomes prohibitively expensive for larger systems. The fundamental bottleneck of DFT calculations is the Kohn-Sham differential equations [5], which has inspired recent efforts to use ML to approximate its solutions [5–7]. However, achieving this goal requires accurately resolving the electronic structure of the system. One of the key quantities characterizing the electronic structure is the electronic density of states (DOS), which describes the energy distribution of electrons of an atomic system.

Recent methods have had success using the local density of states (LDOS) as the supervised target [5, 7], from which the DOS can be computed inexpensively. While accurate, the LDOS is computationally expensive as it is defined over a 3D grid containing tens of thousand of points per atom, requiring that many predictions to resolve the electronic structure properties of a single system configuration. The size of the grid also needs to scale up with the size of the system in order to maintain accuracy, presenting a formidable scalability challenge. We show that this is an unnecessary price to pay in calculating DOS through machine learning by proposing a new approach for atom-level supervision, atom-centered density of states, *ADOS*, that reduces the total work for prediction by orders of magnitude by comparison.

Additionally, existing ML approaches for resolving DOS have so far relied on handcrafted descriptors to extract features (*fingerprints*) from local atomic environments, as the input to their ML model. While much progress has been made in the development of

*Georgia Institute of Technology, jfox43@gatech.edu
†Sandia National Laboratories, namodin@sandia.gov
‡Sandia National Laboratories, srajama@sandia.gov

fingerprinting techniques, they share in common the constraint of being limited to fitting to fixed basis functions. This work proposes to instead use trainable neural descriptors for fingerprinting, specifically focusing on the Concentric Spherical GNN (CSGNN) model [8] as extended to the DOS prediction problem. This allows the atomic environment fingerprinting to be adapted to the data and target problem, with the end goal of generalizing to greater types and complexities of environments within a single model.

We use experimentally evaluate our approach for accurately resolving the band energy (calculated from DOS) of aluminum at the melting point. Our overall approach is able to match and even surpass the accuracy of previous LDOS-based approach for aluminum [7], at a fraction of the time. We believe our atom-centered approach also opens the door to resolving DOS for systems containing thousands of atoms or more that are beyond existing DFT capabilities.

2. Related Work. Molecular dynamic simulations depend on accurate determination of the energy of an atomistic system as a function of the atomic positions. Over at least the past decade, there has been an evolving body of work on using data to directly learn interatomic potentials (IAPs) that predict this energy. While different in their choice of method for the regression problem, these ML-based potentials share a need for *fingerprints*, or feature vector representations of localized atomic environments as input. Methods such as [1, 2, 12, 18] rely on hand-crafted descriptors for fingerprinting, while more recently some methods [15, 17] have used neural descriptors to learn the fingerprint end-to-end.

Recently there also have been efforts to use ML to approximate solutions to the fundamental bottleneck of DFT calculations, the Kohn-Sham differential equations [5–7]. Solving these equations involves accurately resolving properties of the electronic structure, such as the electronic density of states (DOS). Existing ML approaches predict this quantity indirectly through spatially localized contributions, centered around 3D grid points [5, 7] or atoms of the system [3, 6, 16]. In the former case, grid points correspond to supervised quantities from DFT calculation (LDOS), providing millions of training samples for a single configuration of atoms. However, this leads to computationally intensive training and inference. Atom-centered contributions are significantly more cost effective for training and inference, but thus far do not have a well-defined formulation for localized supervision. As the only supervision is from the total DOS of the system, DFT calculations must be run for many more configurations of atoms in order to generate adequate training data, an expensive process. The proposed ADOS approach bridges this gap, providing local supervision while avoiding the unnecessary cost of grid-centered LDOS in training and inference.

3. Methods. This section covers key components of our overall machine learning approach for resolving the electronic density of states. Sec. 3.1 discusses the aluminum snapshots used in subsequent experiments, and details of its generation via simulation and DFT. Sec. 3.2 presents a partition-of-unity approach for deriving atom-level supervision for the DOS, as targets for downstream ML. Finally, Sec. 3.3 gives an overview of the proposed neural fingerprinting model for ADOS prediction.

3.1. Dataset. The focus of our ML approach is for aluminum at ambient density (2.699g/cc) and at melting point temperature (933K). Training data was generated by calculating LDOS for atomic configurations using the *Quantum ESPRESSO* electronic structure code [9–11]. The configurations were generated from snapshots of DFT-MD trajectories of 256-atom supercells of aluminum. This resulted in LDOS training data for a total of twenty snapshots of aluminum at melting point: ten snapshots in the crystalline phase, and ten snapshots in the liquid phase. The LDOS is calculated over a finite grid of evenly spaced energy values, with spacing of 0.1 eV ranging from -10 eV to 14.9 eV. The data for each grid

point is then a vector length 250. The process used to generate the LDOS data is described in detail in Ref. [7], and we refer to it for more detailed discussion and justification of the procedures.

3.2. Atom-Decomposed Density of States. In order to reduce the number of predictions that are required in order to evaluate the DOS for a given system, we wish to replace the LDOS $D^L(r, E)$ evaluated at grid points r and energies E with an "Atom-Decomposed Density of States" (ADOS) $D_i^A(E)$ evaluated for atoms i and energies E. There are two important requirements for the ADOS: (1) The DOS is given by a sum of the LDOS over grid points

$$D(E) = \sum_r D^L(r, E). \tag{3.1}$$

Summing the ADOS over atoms should produce the same DOS, i.e.,

$$\sum_i D_i^A(E) = D(E). \tag{3.2}$$

(2) If $D^L(r, E)$ can be accurately approximated as a function of the atomic positions in some local region around grid point r, then $D_i^A(E)$ can be accurately approximated by some function of the atomic positions in some local region around R_i, the position of atom i.

Both of the above properties can be achieved if $D_i^A(E)$ is defined as a weighted sum of $D^L(r, E)$ over grid points, and the weighted sum is a local partition of unity. In particular, if $D(r, E)$ is the LDOS evaluated at grid point r and energy E, we can define the ADOS associated with atom i as

$$D_i^A(E) = \sum_r w_i(r) D^L(r, E) \tag{3.3}$$

for some weighting functions $w_i(r)$. The set of weighting functions $w_i(r)$ is a partition of unity if

$$\sum_i w_i(r) = 1 \ \forall r. \tag{3.4}$$

Likewise, the partition of unity is local if every $w_i(r)$ decays sufficiently rapidly for large $\|r - R_i\|$.

There are many way to define such a partition of unity, but the specific approach that we have chosen is to define

$$w_i(r) = \frac{\exp\left[-\|r - R_i\|^2/2\sigma^2\right]}{\sum_j \exp\left[-\|r - R_j\|^2/2\sigma^2\right]}. \tag{3.5}$$

Given this definition, it is easy to verify that $w_i(r)$ is a partition of unity and that Requirement (1) above is satisfied. For atom positions R_i that are evenly distributed throughout space, $w_i(r)$ decays as a Gaussian tail for large $\|r - R_i\|$, and the partition of unity is local. For systems that involve large regions with no atoms, some of the weighting functions $w_i(r)$ can remain substantial throughout such regions. However, $D^L(r, E)$ is generally small in such regions, at least for energies E that are occupied by electrons, and thus, for practical purposes, we believe that Requirement (2) also holds for such systems.

When σ is much less than the distance between atoms, the partition of unity defined above closely approximates an approach in which the LDOS at each grid point is assigned

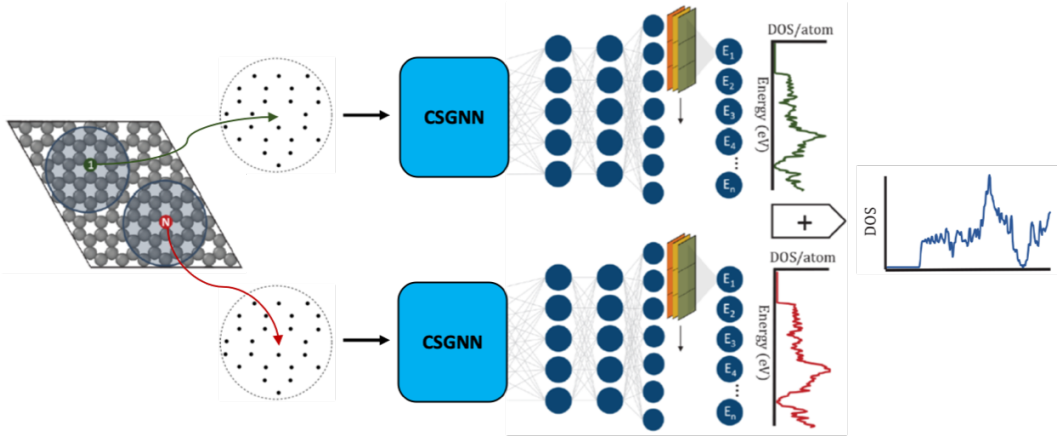


Fig. 3.1: Atom-centered ML workflow: the local atomic environment of each atom, as positions, are input into the learned fingerprinting module (CSGNN). Resulting outputs are mapped through additional neural layers to predict atom-level DOS. These are then summed to obtain the total predicted DOS for the system.

to the nearest atom. In the opposite limit, which σ is comparable to the distance between atoms, the LDOS at each grid point is shared between several atoms. We have picked an intermediate value of $\sigma = 1.3$ Angstroms, compared to an average nearest neighbor distance of around 2.6 Angstroms. Thus, grid points near to an atom will mostly have their LDOS assigned to that atom, while grid points between atoms will have their LDOS shared between the nearby atoms.

Using the above approach, we calculated the ADOS from our previously evaluated LDOS in order to generate training data for a model that predicts the ADOS as a function of the local environment around each atom. This model can then be used to predict the ADOS directly while avoiding the computationally expensive evaluation of the LDOS.

3.3. Concentric Spherical GNN for Atomic Environments. A workflow of the overall ADOS ML approach is illustrated by Fig. 3.1. CSGNN, the proposed model, operates on a concentric spherical spatial sampling of 3D space. Each individual sphere is discretized by the icosahedral grid, resulting in a highly uniform sampling of spherical space. The grid is sub-divided recursively to create higher sampling resolution. The sampling is further extended radially, resulting in concentric spheres about a center, which is defined naturally as an atom for the ADOS problem. We refer to Fig. 3.2 for illustration of the concentric spherical grids. An atom's atomic environment is contained within the concentric spherical sampling, and mapped to an initial description over the sampling. Fig. 3.3 provides an illustration of this mapping.

Two types of convolutions are defined for representation learning over the concentric spherical grid: intra-sphere and inter-sphere convolutions. The former is implemented by graph convolutions [14], with connectivity defined by each vertex's local neighborhood in the icosahedral discretization. Inter-sphere convolutions operate between co-radial vertices, orthogonally to intra-sphere convolutions. The combined use of the two convolution types permits extracting of features volumetrically over the concentric spherical sampling. Furthermore, the intra-sphere convolutions are by design rotationally equivariant to the icosahedral rotation group [19], and approximately equivariant to the general space of 3D rota-

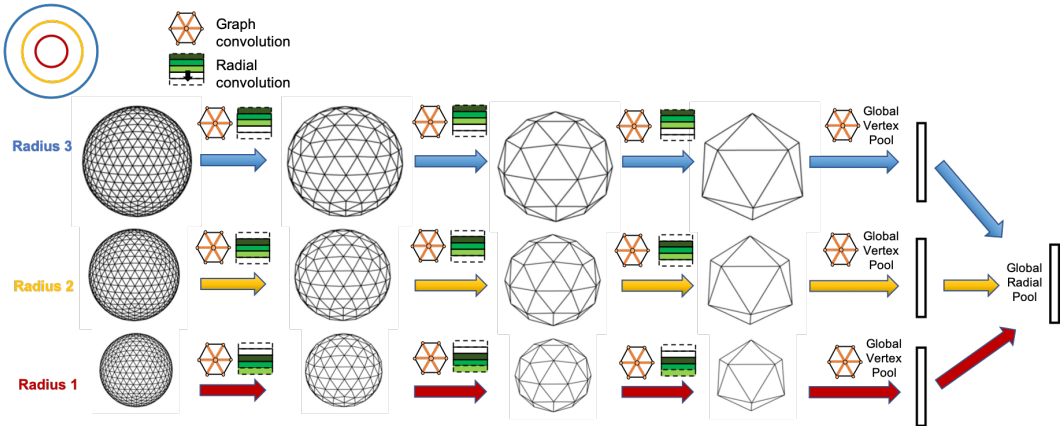


Fig. 3.2: Example CSGNN architecture with $R = 3$ concentric spheres. Graph convolutions are followed by radial convolutions at each density of spherical sampling. Graph convolution is applied within each sphere. 1D convolution is applied between co-radial vertices (3 in this example). Vertex pooling (not shown) and downsampling then coarsens the spherical sampling. Global pooling is applied at the end to obtain the final feature representation.

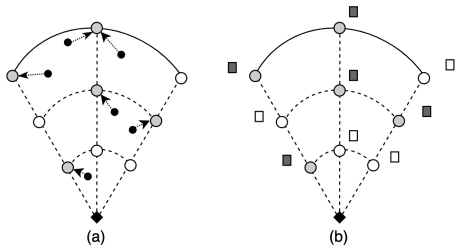


Fig. 3.3: Shown is a 2D cross section of an atomic environment centered at a reference point (black diamond), for an example sector. (a) Each atom (black dot) in the environment a value of $\phi(r)$ to its nearest vertex in 3D space, where r is radial distance from the center and ϕ is a chosen distance mapping (such as the inverse function). (b) Values incident at any given vertex are summed, resulting in a scalar input feature per vertex.

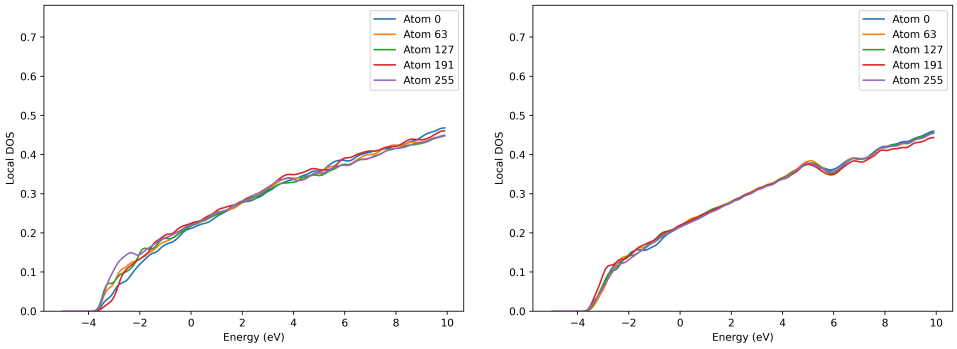
tions. We refer to [8] for more detailed discussion of the concentric spherical convolutions. We combine the proposed convolutions into a hierarchical convolutional architecture, by also utilizing pooling and downsampling over the icosahedral grid. Fig. 3.2 illustrates an example CSGNN architecture. Convolutions at different scales of spherical sampling enables learning representation of the input atomic environment analogously to 2D CNNs for images.

4. Results. In this section we present main results of our atom-centered ML approach for electronic structure calculation, demonstrated for aluminum. Sec. 4.1 shows that the proposed ADOS permits faithful reconstruction of the original DOS, and therefore a sufficient target for atom-centered supervision. Sec. 4.2 presents band energy results using the proposed CSGNN model for learned fingerprinting, combined with ADOS training. Our

proposed approach improves on band energy accuracy over prior LDOS based approach, while requiring orders of magnitude less total samples for both training and inference. Finally, Sec. 4.2 demonstrates how the proposed ADOS approach leads to significant speedup over LDOS in practice for training and inference.

4.1. Reconstruction of DOS from ADOS. For the proposed ADOS to be useful, it must be possible to reconstruct the original DOS derived from LDOS. We experimentally verified that simple summation of the ADOS leads to nearly perfect reconstruction of the original DOS. We further verified that the band energy derived from ADOS matches the original band energy.

Resulting ADOS curves are plotted for sampled atoms from liquid and solid snapshots in Fig. 4.1. Overall, the atom-centered DOS appears much more similar within each snapshot than between liquid and solid snapshots, with the solid snapshots showing prominent wiggles in the 5 to 8 eV range that are remnants of the Van Hove singularities that occur in a perfect crystal. Furthermore, the ADOS within each snapshot tends to reflect the profile of the DOS of their respective snapshots (see Fig. 4.2). These results are to be expected since both solid and liquid aluminum are generally homogeneous systems with each atom in a similar local environment. There are some fluctuations in the local environment, which are reflected in the variations between the ADOS for different atoms within the same phase. The local environment varies more for the liquid than for the solid, and correspondingly, the variation between the ADOS for different atoms is larger in the liquid. However, even in the liquid, these local fluctuations are not as significant as the difference between the solid and liquid phases. This shows that the atom-centered DOS profile is able to resolve the differences between liquid and solid phase aluminum, as well as fluctuations in the local environment of the atoms.



(a) ADOS for liquid aluminum, snapshot 9. (b) ADOS for solid aluminum, snapshot 19.

Fig. 4.1: Atom-centered DOS values resulting from partition-of-unity, for liquid and solid aluminum snapshots at 933K. Shown are DOS from 5 sampled atoms of each snapshot.

4.2. ML Model for Resolving Band Energy of Aluminum. For experiments, we consider a dataset of 20 total snapshots of aluminum at 933K, consisting of 10 liquid and 10 solid phase aluminum snapshots. For each phase, 6 snapshots are used for training, 1 snapshot for validation, and 3 for testing. Band energy is calculated from predicted DOS for each snapshot of the test set, and error from ground-truth is measured by meV per atom. We compare the proposed approach with LDOS-SNAP [7]. Our approach uses atom-based

Method	Training Set	Total Training Samples	Test Set	Band Energy Max Error (meV/atom)	Band Energy Mean Error (meV/atom)
LDOS-SNAP [7]	6 liquid	4.8×10^7	3 liquid	21.3	17.1
	6 solid	4.8×10^7	3 solid	39.3	33.6
ADOS-CSGNN	6 liquid	1.5×10^3	3 liquid	19.9	15.6
	6 solid	1.5×10^3	3 solid	5.3	3.3

Table 4.1: Band energy results, comparing the proposed ADOS-CSGNN approach to prior LDOS-SNAP approach. Band energy error is calculated for the test set, and measured in terms of both max and mean absolute error. Total training samples reflects to actual number of predictions, based on local supervision.

ADOS for supervision, while LDOS-SNAP uses grid-based LDOS. Another key difference, orthogonal to the the form of supervision, is the method of fingerprinting. Whereas LDOS-SNAP used SNAP [18] for fingerprinting, we use a neural fingerprinting approach, CSGNN, to learn atomic environment descriptors end-to-end.

Table 4.1 presents results for the proposed model and comparisons. By using ADOS instead of LDOS, the total number of samples for prediction is reduced by a factor of 32,000 for training. This reduction also extends to inference, although not shown in table for brevity. This is a significant reduction, as the total number of samples direcly reflects the total amount of actual work for the model, all else equal. Importantly, this reduction is achieved without any sacrifice to accuracy.

Compared to LDOS-SNAP , the ADOS-CSGNN model reduces band energy error (mean absolute error) by 9% in the case of liquid phase aluminum, and by 90% the case of solid phase aluminum. For the liquid phase band energy, the ADOS-CSGNN model achieves a slight improvement in accuracy over LDOS-SNAP. However, for the solid phase band energy, the ADOS-CSGNN model achieves nearly 10x improvement, which represents a major advance in predictive power. We surmise that this large reduction in error is due to difference in the learning model–CSGNN learns local environment descriptions end-to-end, which could prove beneficial when using a single model for hybrid dataset. However, this hypothesis remains to be investigated further.

We further plot the DOS predicted by ADOS-CSGNN to the reference DOS from DFT, and show these for example liquid and aluminum snapshots in Fig. 4.2. These plots confirm that the proposed approach is able to produce aluminum DOS closely matching DOS from quantum-mechanical calculation, and that the band energy accuracy is not resulting from some degeneracy.

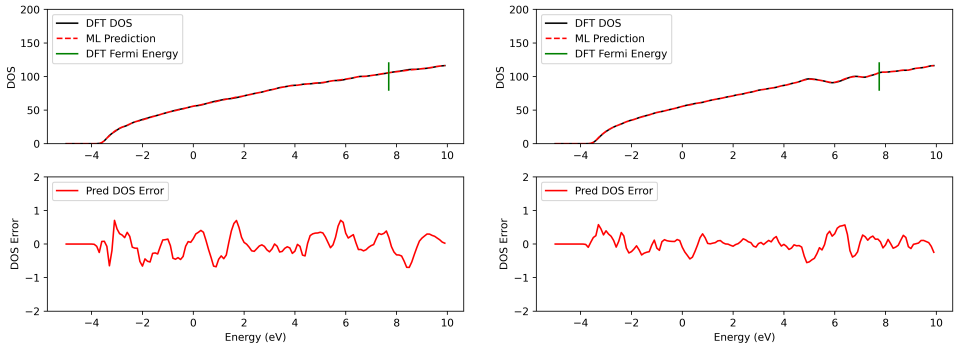
Finally, we list hyperparameter settings for the best-performing ADOS-CSGNN model in Table 4.2. We also used batch normalization [13], which is not counted in the total number of layers. Finally, we also plot training and validation loss for the best-performing model in Fig. 4.3.

4.3. Runtime. In this section we explore how the ADOS-CSGNN approach translates to actual runtime for training and inference, compared to the LDOS-SNAP approach. For training we consider the time for a single epoch (12 training snapshots), and for inference we consider the time to evaluate a single snapshot for its local DOS quantities. Both models are run on a single NVIDIA V100 GPU.

Results are presented in Table 4.3. ADOS-CSGNN provides a 240× speedup in training

Parameter	Value
Concentric spheres	16
Spherical resolution	642
Optimizer	Adam
Batch size	32
Learning rate	0.01
Activation	ReLU
Epochs	200
Layers	18
Total Weights	5.9×10^6

Table 4.2: List of parameter settings for ADOS-CSGNN model used for experiments. Spherical resolution is number of vertices of icosahedral spherical sampling. Number of layers is trainable layers.



(a) DOS of liquid aluminum for snapshot 8. (b) DOS of solid aluminum for snapshot 18.

Fig. 4.2: Density of states for solid and liquid snapshots at 933K. Top row shows DOS curve predicted by ADOS compared to reference curve from DFT. X-axis is energy range from -5 to 10 eV. Units for y-axis is eV. Bottom row plots difference between predicted DOS and the reference DOS of respective snapshots.

Method	Training time (1 epoch)	Inference time (1 snapshot)
LDOS-SNAP	76 minutes	54 seconds
ADOS-CSGNN	19 seconds	1 second

Table 4.3: Runtime comparison for training and inference, run on single V100 GPU. LDOS-SNAP takes grid-centered local descriptors as input to the neural model, but their generation time was not included in this comparison.

per epoch and 54× speedup in inference compared to LDOS-SNAP . While a very significant and practical improvement, the speedups fall short of the factor of reduction $(32,000)$ in the total amount of samples in switching from LDOS to ADOS. This is likely due to the difference in the neural models used in ADOS-CSGNN vs. LDOS-SNAP . Additionally,

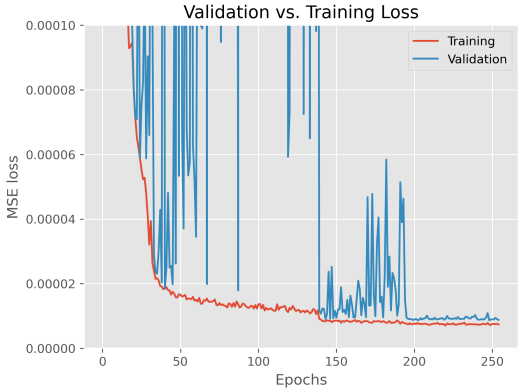


Fig. 4.3: Training and validation loss curves for best-performing version of ADOS-CSGNN used in experiments. Y-axis is mean-squared error loss for ADOS prediction, and x-axis is epoch number.

while fingerprint generation is part of the neural model in the case of ADOS-CSGNN , it is not in the case of LDOS-SNAP and was omitted from time comparison. The speedup of ADOS-CSGNN should therefore be interpreted as a lower bound, especially in the case of inference, as the time to generate fingerprint for input cannot be ignored in practice.

5. Conclusion. In this work we present a machine learning approach for resolving key products of electronic structure calculation, such as the density of states and band energy, at a small fraction of the computational cost of existing LDOS approaches and without sacrificing accuracy. The first key piece of the proposed approach is to create atom-level supervision, ADOS, using a partition-of-unity approach. This reduces the total number of predictions required to resolve DOS compared to LDOS by orders of magnitude, for both training and inference. The second piece of our approach is to incorporate a neural model based on concentric spherical convolutions for learning atomic environment fingerprints end-to-end. We experimentally demonstrate that our overall approach allows resolving DOS and band energy many times faster than with LDOS-based approaches. In combination with our neural model for learned fingerprinting, we match and even outperform LDOS-based approaches in resolving band energy of melting point aluminum. In terms of future work, we believe that our atom-centered approach can be very feasibly extended to systems of size of $O(10^4)$ atoms, which is already well beyond the reach of DFT. Another direction for future work is an extension of atom-centered supervision towards predicting the electron density [4], which together with the DOS would complete a ML-driven solution to electronic structure calculation.

Acknowledgments. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References.

[1] A. P. Bartók, R. Kondor, and G. Csányi, *On representing chemical environments*, Phys. Rev. B, 87 (2013), p. 184115.

[2] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, *Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons*, Phys. Rev. Lett., 104 (2010), p. 136403.

[3] C. Ben Mahmoud, A. Anelli, G. Csányi, and M. Ceriotti, *Learning the electronic density of states in condensed matter*, Physical Review B, 102 (2020), p. 235130.
[4] F. Brockherde, L. Vogt, L. Li, M. E. Tuckerman, K. Burke, and K.-R. Müller, *Bypassing the kohn-sham equations with machine learning*, Nature Communications, 8 (2017), p. 872.
[5] A. Chandrasekaran, D. Kamal, R. Batra, C. Kim, L. Chen, and R. Ramprasad, *Solving the electronic structure problem with machine learning*, npj Computational Materials, 5 (2019), pp. 1–7.
[6] B. G. del Rio, C. Kuenneth, H. D. Tran, and R. Ramprasad, *An efficient deep learning scheme to predict the electronic structure of materials and molecules: The example of graphene-derived allotropes*, The Journal of Physical Chemistry A, 124 (2020), pp. 9496–9502. PMID: 33138367.
[7] J. A. Ellis, L. Fiedler, G. A. Popoola, N. A. Modine, J. A. Stephens, A. P. Thompson, A. Cangi, and S. Rajamanickam, *Accelerating finite-temperature kohn-sham density functional theory with deep neural networks*, Phys. Rev. B, 104 (2021), p. 035120.
[8] J. Fox, B. Zhao, S. Rajamanickam, R. Ramprasad, and L. Song, *Concentric spherical GNN for 3d representation learning*, CoRR, abs/2103.10484 (2021).
[9] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. D. Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. O. de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, *Advanced capabilities for materials modelling with quantum ESPRESSO*, Journal of Physics: Condensed Matter, 29 (2017), p. 465901.
[10] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, *QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials*, Journal of Physics: Condensed Matter, 21 (2009), p. 395502.
[11] P. Giannozzi, O. Baseggio, P. Bonfà, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. F. Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, and S. Baroni, *Quantum espresso toward the exascale.*, The Journal of chemical physics, 152 15 (2020), p. 154105.
[12] T. D. Huan, R. Batra, J. Chapman, S. Krishnan, L. Chen, and R. Ramprasad, *A universal strategy for the creation of machine learning-based atomistic force fields*, npj Computational Materials, 3 (2017), p. 37.
[13] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015, pp. 448–456.
[14] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, International Conference on Learning Representations, (2017).
[15] N. Lubbers, J. S. Smith, and K. Barros, *Hierarchical modeling of molecular energies using a deep neural network*, The Journal of Chemical Physics, 148 (2018), p. 241715.
[16] K. T. Schütt, H. Glawe, F. Brockherde, A. Sanna, K. R. Müller, and E. K. U. Gross, *How to represent crystal structures for machine learning: Towards fast prediction of electronic properties*, Physical Review B, 89 (2014), p. 205118.
[17] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, *Schnet: A continuous-filter convolutional neural network for modeling quantum interactions*, NIPS'17, Red Hook, NY, USA, 2017, Curran Associates Inc., p. 992–1002.
[18] A. Thompson, L. Swiler, C. Trott, S. Foiles, and G. Tucker, *Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials*, Journal of Computational Physics, 285 (2015), pp. 316–330.
[19] Q. Yang, C. Li, W. Dai, J. Zou, G.-J. Qi, and H. Xiong, *Rotation equivariant graph convolutional network for spherical image classification*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4303–4312.

FINE-GRAINED PARALLEL GRAPH PARTITION REFINEMENT

MICHAEL S. GILBERT*, KAMESH MADDURI†, ERIK G. BOMAN‡, AND SIVASANKARAN RAJAMANICKAM§

Abstract. Many graph partitioning algorithms use the multilevel method, where the graph size is progressively reduced and the partitioning problem solved on smaller graphs. Refinement is a key step in the multilevel method. Informally, refinement refers to improving a partition when projecting the solution from a smaller to a larger graph. Several refinement algorithms are known, but most of them are greedy algorithms and not amenable to parallelization. Some coarse-grained parallel refinement algorithms suitable for multicore CPUs exist. In this work, we investigate fine-grained parallel refinement algorithms that are applicable to Graphics Processing Units (GPUs). Our refinement approaches are inspired by the sequential algorithms, as we empirically identify core features of the serial algorithms that are responsible for high-quality partitions on a collection of graphs. We present initial results of the parallel algorithms on an NVIDIA RTX 3090 GPU and a 32-core AMD processor. We find that our parallel algorithm achieves cutsizes within 10% of serial Fiduccia-Mattheyses refinement on 85% of our test graphs, and within 20% of the serial cutsize for all graphs.

1. Introduction. Graph partitioning is the problem of taking an input graph $G = (V, E, W)$, consisting of vertices, edges, and edge weights, and producing a partitioning of V into k parts (disjoint vertex sets), such that the ratio of the largest part's size to the optimal part size ($|V|/k$) is less than some balance constraint λ, while an objective function is optimized. The objective function in partitioning is typically to minimize the sum of edge weights for edges in the cutset (also called the "cutsize" or "edge cut"). The cutset is the set of edges that connect vertices in distinct parts.

Most modern graph partitioners utilize the multilevel method. This is the process of generating a sequence of coarse graphs according to some heuristic, such that the final graph in the sequence is suitably small to run high-precision serial partitioning algorithms on. This sequence is generated recursively, such that $G_{i+1} = \texttt{coarsen}(G_i)$. The initial partition on graph G_c may be denoted P_c, where c is the number of coarse levels. Each P_i can be projected onto the previous graph in the sequence G_{i-1}, to generate a partition P_{i-1}. This can be repeated until P_0 is generated. It is the role of partition refinement algorithms to take a partition P_i immediately after each such projection, and improve it in terms of both balance and cutsize.

In this work, we seek to create high-quality graph-partition refinement techniques that can operate efficiently on GPUs. This requires creating a substantial degree of fine-grained parallelism, whereas all prior work on parallel refinement algorithms has targeted a coarse-grained approach suitable for multicore CPUs. The greedy nature of serial algorithms like Kernighan-Lin (KL) [8] and Fiduccia-Mattheyses (FM) [4] prohibits implementation on GPUs without relaxing the greedy constraints. However, these algorithms have other traits and behaviors that can be emulated by a more parallel algorithm.

The parallel algorithm we develop in this work emulates some qualities of the FM algorithm while diverging from it in many significant ways. We emulate such qualities including moving high gain vertices before low gain ones, and decreasing imbalance by moving vertices in the proper direction when necessary. We diverge from FM in neglecting the globally optimal move in terms of gain. The algorithm requires very little synchronization, needing only to atomically update the gains of neighboring vertices to those swapped. It refines in

*Pennsylvania State University, msg5334@psu.edu
†Pennsylvania State University, madduri@psu.edu
‡Sandia National Laboratories, egboman@sandia.gov
§Sandia National Laboratories, srajama@sandia.gov

iterations, with each iteration requiring a few milliseconds to complete in the worst case for our test graphs.

In our results we will show that this algorithm achieves edge cuts that rival and occasionally outperform that of FM refinement. It greatly outperforms serial FM in terms of time.

2. Background and Related Work. In order to implement a multilevel partitioner, one must choose a coarsening heuristic. The primary role of this heuristic is to determine a mapping from an input graph to a coarse graph. In our past work, we investigated several of these heuristics. In particular, we focused on the heavy-edge coarsening (HEC) heuristic, first used for multilevel methods by Urschel et al. [12]. The HEC heuristic determines the heaviest edge adjacent to each vertex, and coarse aggregates are formed by joining vertices across these edges. A typical implementation will visit each vertex and attempt to create an aggregate with the vertex adjacent on its heaviest edge, or join the aggregate of that vertex if one already exists. If the vertex already has an aggregate when it is visited, its heaviest edge is ignored. Metis [7] uses a different approach based on maximal matchings, wherein coarse vertices are formed from matched pairs. Such matched vertex pairs are usually formed by choosing the heaviest edge of one vertex to determine its partner, or the vertex of heaviest available edge if the first choice is taken. Mt-Metis [10] introduced optimizations to the standard matching approach designed to prevent stalling on skewed-degree graphs. Skewed-degree graphs are those graphs having a large ratio between the maximum degree of any vertex versus the average degree. These optimizations allow matches to be formed between 2-hop neighbors if a vertex has no available 1-hop neighbor to match with.

There are multiple types of partition refinement schemes, but these schemes most commonly use combinatorial methods. A less common approach can be found in partitioners such as Mongoose [1], which uses quadratic programming. A core concept in combinatorial partition refinement is the vertex gain, which reflects the net decrease in cutsize for a vertex v if v were to move to another partition. In a k-way partitioning setting, this gain can be defined for every pair of a vertex $v \in V$ and one of the $k-1$ partitions that v is not currently resident in. For bisection, this gain can be calculated as the weight of all edges adjacent to v in the cutset, minus the weight of all edges adjacent to v not in the cutset. The gain is positive if moving v decreases the cut, while it is negative if moving v increases the cut. This concept is used by algorithms such as FM and the greedy refinement implemented by kMetis and Mt-Metis [9]. Refinement schemes are also used by non-multilevel partitioners, such as PuLP [11], which uses label propagation to refine partitions. Label propagation is the process of assigning a vertex to the partition given by the mode of its neighbors' partitions (weighted by edge-weight). In a bisection setting, this means that a vertex is assigned to the opposite partition if it has a positive gain, whereas it is not moved if it has a negative gain.

The Fiduccia-Mattheyses (FM) refinement algorithm [4] (see algorithm 1) is a greedy heuristic algorithm. It operates by iterations, and within an iteration it can only move each vertex once. It builds a data structure, which is a heap/priority-queue in some implementations, to select the highest gain vertex from the largest partition, and moves it to the smaller partition. It is possible for this vertex to have negative gain, even if positive gain vertices exist in the smaller partition. After moving a vertex, it marks it as moved for the iteration, then updates the gains of neighboring vertices, which also necessitates updating the data structure. A common optimization used by partitioners such as Metis is to restrict consideration only to those vertices on the cut boundary (vertices with an adjacent edge in the cutset). It is typical that this boundary set is a small fraction of the total vertices, which provides substantial computational efficiency. However, this can prevent vertices not on the

boundary from being considered when they might be the optimal move, such as when no positive gain vertices exist in the larger part. This is usually not an issue on regular graphs, but can become an issue on skewed-degree graphs.

Algorithm 1 FM Refinement

Input: $G(V, E, W)$. Partition vector $P_i[1..n]$. $n = |V|$.
Output: $P_{i+1}[1..n]$

```
1: gain_max ← 0
2: cutsize ← cutsize(G, P_i)
3: balance ← imbalance(G, P_i)
4: Gains[1..n] ← gain_values(G, P_i)
5: (Compute maximum possible gain)
6: for u = 1 to n do
7:     E_u ← E adjacent to u
8:     sum ← SUM-EDGE-WEIGHTS(E_u)
9:     if gain_max < sum then
10:        gain_max ← sum
11: (One bucket array for each part, each has 2*gain_max total buckets)
12: B_0, B_1 ← ALLOCATE-BUCKETS(2 * gain_max)
13: for u = 1 to n do
14:    if P_i[u] == 0 then
15:        bucket ← B_0[Gains[u]]
16:    else
17:        bucket ← B_1[Gains[u]]
18:    INSERT_INTO_BUCKET(bucket, u)
19: swap_sequence ← empty_list
20: P_{i+1} ← P_i
21: while B_0 not empty AND B_1 not empty do
22:    if balance > 0 then
23:        swap ← BEST-GAIN-IN(B_0)
24:    else if balance < 0 then
25:        swap ← BEST-GAIN-IN(B_1)
26:    else
27:        swap ← BEST-GAIN-IN-EITHER(B_0, B_1)
28:    (remove swap from data structure, update cutsize and balance)
29:    REMOVE-AND-PERFORM-SWAP(P_1, swap, cutsize, balance)
30:    E_swap ← E adjacent to swap
31:    UPDATE-ADJACENT-GAINS(E_swap, swap)
32:    APPEND-TO-LIST(swap_sequence,{swap,cutsize,balance})
33: (Here we find the optimum cutsize and imbalance combination from swap_sequence. We
    then undo all swaps after that point.)
34: SELECT-BEST-AND-UNDO-REST(swap_sequence, P_1)
```

Mt-Metis provides a coarse-grained CPU parallel refinement algorithm, which is very similar to FM. Each thread owns a subset of the boundary vertices, and for this set each thread builds a priority-queue. Vertices are moved in order of these thread-private priority queues, but the globally optimal heuristic is relaxed. The threads use locks on both the part sizes and neighboring vertices to ensure moving a vertex v will not violate the balance

constraint, and will actually decrease the cutsize.

3. Parallel Refinement Algorithm.

3.1. Parallelism Difficulties with FM. Our parallel refinement algorithm attempts to mimic certain attributes of the FM refinement algorithm. We identify the core attributes of FM refinement in a bipartitioning context:

1. Move vertices from the larger part to the smaller part
2. Move the individual vertex having most positive gain
3. Move each vertex at most once per iteration
4. Terminate iteration after x vertices move without a net cutsize decrease

When considering a direct parallelization of FM, we encounter problems due to attributes 1, 2 and 4. Atrribute 1 limits parallelism because there is a limited number of moves that can be made from the larger part before it becomes the smaller part, especially if the parts are close to balanced. Attribute 1 can be relaxed to allow an algorithm to continue to choose vertices from one part even after it is no longer the largest part. Attribute 2 limits the degree of parallelism, as moving even a singular vertex impacts the gains of each of its neighbors, requiring any data structure that tracks the gains of vertices to be updated before the next highest gain vertex can be determined. Attribute 4 is a smaller limit on parallelism, but we can easily increase x. It is clear that attribute 2 is the primary source of difficulty translating FM into a parallel algorithm, as it has no trivial relaxation.

3.2. Moving Positive Gain Vertices. We relax attribute 2 by allowing our algorithm to select all vertices with positive gain. In order to avoid our selections conflicting with each other, we should select vertices from only one part. To illustrate why we should do this, suppose we select two vertices that share an edge. If both are in the same part, when we swap them they will still be in the same part as each other. In this way, the edge will not be on the cut after the swap, but our calculation of the individual gain values assumed it would be cut. Thus, the gain calculated on a per-vertex basis is a minimum value.

In selecting multiple vertices from the same part, we also relax attribute 1. We usually choose the initially larger part as the source, but we can select the smaller part if the balance constraint is currently satisfied. We found that it can often be advantageous to simply move all vertices of positive gain instead of limiting the number of moves by the imbalance constraint. This can substantially reduce the number of iterations required for convergence, while the next phase we discuss can handle the impact on the imbalance that this causes. This process for finding these vertices is illustrated in algorithm 2.

3.3. Gain Poisoning. We encounter a problem that leads to a limited capability to refine. Suppose we chose to move many vertices from part A to part B, then the gains of vertices in part B would decrease substantially. We denote this as "gain poisoning". Consider M to be the set of vertices we have moved from A to B, and note that while these vertices had positive gain while they resided in side A, they now have negative gain residing in part B. In addition, moving these vertices will decrease the gains of the vertices originally residing in part B. If we move enough things from part A to part B, there will be few to no vertices that individually have positive gain in part B. To put it simply, the more vertices we move from A to B, the harder it is to move vertices from B to A. Gain poisoning is a major problem due to the balance constraint, since the majority of positive gain vertices want to move from A to B (further increasing the imbalance). Gain poisoning can have a positive side effect: after moving enough vertices from one part in early iterations, we can consider positive gain vertices from all parts in later iterations without too many conflicts. This is because most positive gain vertices will exist in one part after a couple refinement iterations.

3.4. Addressing Gain Poisoning. When we violate the balance constraint, we must consider moving vertices with negative gain due to gain poisoning. This allows us to decrease imbalance, so that we can continue to move positive gain vertices in later phases. To accomplish this, we select d vertices (d is the number of vertices that must move to make both parts equal in size; $||A|-|B||$) from the overweight part B that have the least negative gain (including any vertices of positive gain) to move to A. As the number of vertices with negative gain is usually large, we use bucketing to choose the d vertices. Bucket zero contains all positive gain vertices, bucket one contains all zero gain vertices, then buckets two and beyond contain the negative gain vertices. A vertex with negative gain is assigned to the bucket given by $\log_2(|gain|)+2$ (reference algorithm 3 for more detail). Algorithm 4 details how these buckets are populated. In algorithm 5 we demonstrate the process of choosing d vertices from these buckets, which requires selecting all vertices from each bucket in order of ascending bucket id until we have chosen d vertices. In our experiments, we found that this process is the most time-consuming section of our algorithm. This is due to a large number of atomic operations which are performed on a small number of memory locations, necessary to count the size of each bucket and for assigning each vertex a position within its respective bucket. We make efforts to reduce this contention by splitting buckets into "minibuckets", and assigning vertices to a minibucket using a simple hash of the vertex id.

We also use algorithm 5 to help find cutsize decreases when there are no positive gain vertices in either part. To do this, we set the number of buckets to be something small, around two or three (this corresponds to gains ≥ 0 and ≥ -1 respectively). After running this on one of the parts, we look for positive gain vertices in that part again and move them. This process works as the inverse of gain poisoning, by increasing the gains of vertices in the part they are moved from. If this process doesn't produce a net cutsize decrease, we revert it.

3.5. Overall Algorithm. The resulting algorithm (see algorithm 6) has multiple phases, which are traversed in sequence. In the first phase, it rebalances the parts (or as close as it can get). In the second phase, it attempts to move all vertices of positive gain from part 1. In the third phase it tries to move vertices of small negative gain from part 0. In the fourth phase, it moves any vertices of positive gain from part 0. Phases 5, 6, 7, and 8 mirror phases 1, 2, 3, and 4 respectively, but phases 6, 7, and 8 move from the opposite part as their counterparts. These phases are cycled through until a full cycle is passed without decreasing the cutsize. At the end of each iteration, the vertices chosen during the iteration are swapped, and this swap list is used to update the gains of neighboring vertices (using algorithm 7), the total imbalance, and the total cutsize.

3.6. Boundary Vertices. We have a separate version that attempts to optimize the number of vertices that must be processed in each iteration by limiting consideration to those vertices that are on the cut boundary. We accomplish this with a list of all vertices on the boundary. In order to minimize the effort to maintain this list, this list is created once and is reused without modification for each iteration.

3.7. Kokkos Library. To implement the parallel kernels necessary for this algorithm, we use the Kokkos Library [3]. Kokkos can be used to target a variety of execution environments, including Nvidia and AMD GPUs, and multicore CPUs, without writing distinct versions for each target. In this work, we leverage "views", which are memory-managed multi-dimensional arrays, to store data such as vertex gains. We also make extensive use of "parallel_for", "parallel_reduce", and "parallel_scan" routines, which enable efficient processing of these views. Kokkos even provides a wrapper for "compressed sparse row" format matrices, which we use to store our graphs. Additionally, we make use of "hierarchical

parallelism" for processing adjacency lists, which allows the effort of tasks like computing a vertices' gain to be shared among a thread "team".

Algorithm 2 Move Positive Gain

Input: $G(V, E, W)$. P. Gains Y. Source Partition p_{id}
Output: X

```
t_pos ← 0
for u from 1 to |V| in parallel do                    ▷ count positive gain vtx
    g ← Y[u]
    if g > 0 and P[u] = p_id then
        t_pos ← t_pos + 1
X ← nulls(t_pos)
for u from 1 to |V| parallel scan i do                ▷ write positive gain vertex
    g ← Y[u]
    if g > 0 and P[u] = p_id then
        X[i] ← u
        i ← i + 1
```

Algorithm 3 Bucket ID

Input: g.
Output: b

```
b ← 0
if g = 0 then
    b ← 1
if g < 0 then
    b ← floor(log2(−g)) + 2
```

Algorithm 4 Bucket Negative Gains

Input: $G(V, E, W)$. P. Total Buckets b_t. Gains Y. Source Partition p_{id}
Output: X

```
R ← zeros(b_t)
for u from 1 to |V| in parallel do                    ▷ count vtcs by bucket idx
    g ← Y[u]
    b ← bucket_id(g)
    if b < b_t and P[u] = p_id then
        R[b] ← R[b] + 1
B ← exclusive_prefix_sum(R)                           ▷ create offsets for each bucket
X ← nulls(B[b_t])
R ← zeros(b_t)
for u from 1 to |V| in parallel do                    ▷ write vtcs to bucket
    g ← Y[u]
    b ← bucket_id(g)
    if b < b_t and P[u] = p_id then
        i ← R[b] + B[b]
        R[b] ← R[b] + 1
        X[i] ← u
```

Algorithm 5 Move Negative Gain

Input: $G(V, E, W)$. P_i. Gains Y. Desired imb change d. Source Partition p_{id}. Total Buckets b_t
Output: X

```
X ← bucket_negative_gains(G, P_i, b_t, Y, p_id)
P_{i+1} ← P
X ← X[1..min(d, |X|)]    ▷ If using wgted vertices, can use a scan to determine where in X to truncate
```

Algorithm 6 Parallel Refinement

Input: $G(V, E, W)$. P. Max allowed imbalance ratio λ
Output: P_{i+1}

```
Y ← gain_values(G, P)
P_{i+1} ← P
cutmin ← cutsize(G, P)
d ← imbalance(G, P)                    ▷ positive if 0 is larger, negative if 1 is larger
dmax ← (λ − 1)∗total_vertex_wgts(G)    ▷ dmax gives max allowed difference in size
between both parts
s ← 0
c ← 0
while true do
    X ← null
    c ← c + 1
    switch s + 1 do
        case 1 or 5
            p_id ← 0
            if imb < 0 then
                p_id ← 1
            X ← move_negative_gain(G, P, Y, dmax − |d|, p_id, 20)
        case 2
            X ← move_positive_gain(G, P, Y, 1)
        case 3
            X ← move_negative_gain(G, P, Y, dmax + d, 0, 3)
        case 4
            X ← move_positive_gain(G, P, Y, 0)
        case 6
            X ← move_positive_gain(G, P, Y, 0)
        case 7
            X ← move_negative_gain(G, P, Y, dmax + d, 1, 3)
        case 8
            X ← move_positive_gain(G, P, Y, 1)
    P, Y ← move_vtcs_update_gains(G, P, Y, X)
    d ← imbalance(G, P)
    cut ← cutsize(G, P)                ▷ can compute these when computing gain updates
    if cut < cutmin then
        P_{i+1} ← P
        cutmin ← cut
        c ← 0
    s ← s + 1  mod 8
    if c > 7 then break
```

Algorithm 7 Move Vertices and Update Gains

Input: $G(V, E, W)$. P_i. Gains Y. Swap List X
Output: P_{i+1}. Y

```
P_{i+1} ← P
for i from 1 to |X| in parallel do
    u ← X[i]
    P_{i+1}[u] ← opposite_part(P_i[u])
    E_u ← adjacency_list(u, E)
    g_next ← 0
    for j from 1 to |E_u| do                          ▷ process adjacencies
        v ← E_u[j]
        w ← W[u, v]
        if P[u] = P[v] then                           ▷ contributes negatively to gain in next phase
            g_next ← g_next − w
            if v not in X then                        ▷ vertices in swap list will process their own gains
                atomic_add(Y[v], −2w)
        else                                          ▷ contributes positively to gain in next phase
            g_next ← g_next + w
            if v not in X then                        ▷ vertices in swap list will process their own gains
                atomic_add(Y[v], 2w)
        Y[u] ← g_next
```

Algorithm 8 gain_values

Input: $G(V, E, W)$. Partition P.
Output: Y

```
Y ← zeros(|V|)
for i from 1 to |V| do
    u ← V[i]
    E_u ← adjacency_list(u, E)
    for j from 1 to |E_u| do
        v ← E_u[j]
        if P[u] = P[v] then
            Y[i] ← Y[i] − W[u, v]
        else
            Y[i] ← Y[i] + W[u, v]
```

Table 4.1: A collection of graphs used for performance evaluation. The graphs are based on sparse matrices from the SuiteSparse matrix collection [2], and networks from OGB [6]. We preprocess the graphs to extract the largest connected component, relabel vertex identifiers, and transform edges to undirected. The number of edges (m), number of vertices (n), and the ratio of max vertex degree (Δ) to average degree after preprocessing are given. Based on this ratio, we partition graphs into two groups: regular and skewed-degree. Within each group, the graphs are ordered by size ($2m + n$).

Graph	Domain	m	n	$\Delta/(2m/n)$
HV15R	cfd	162 357 569	2 017 169	3.1
rgg24	syn	132 557 200	16 777 215	2.5
nlpkkt160	opt	110 586 256	8 345 600	1.0
europeOsm	road	54 054 660	50 912 018	6.1
CubeCoup	fem	62 520 692	2 164 760	1.2
delaunay24	syn	50 331 601	16 777 216	4.3
Flan1565	fem	57 920 625	1 564 794	1.1
MLGeer	sim	54 687 985	1 504 002	1.0
cage15	bio	47 022 346	5 154 859	2.5
channel050	sim	42 681 372	4 802 000	1.0
ic04	www	149 054 854	7 320 539	6296.9
Orkut	soc	117 185 083	3 072 441	436.7
vasStokes4M	vlsi	97 708 521	4 344 906	25.3
kmerU1a	bio	66 393 629	64 678 340	17.0
kron21	syn	91 040 839	1 543 901	1813.7
products	ecom	61 806 303	2 385 902	337.4
hollywood09	soc	56 306 653	1 069 126	108.9
mycielskian17	syn	50 122 871	98 303	48.2
citation	cit	30 344 439	2 915 301	480.4
ppa	bio	21 231 776	576 039	44.0

4. Experiments and Results. In order to evaluate our refinement, we use a test set of 20 graphs from the SuiteSparse repository [2] (see Table 4.1). In our previous work [5], we evaluated graph coarsening algorithms on this same test set, however we needed to rerun the experiments to collect timing info so the cutsizes are different. The preprocessing of each graph is also the same as that work. We split these graphs into two groups with 10 graphs each. The first ten graphs listed are the "regular" graphs, and the last ten graphs listed are the "skewed-degree" graphs. We compare our parallel refinement to Mt-Metis in terms of cutsize and runtime, as well as the serial FM refinement implementation used in our previous work. Our test system is a 32-core AMD Ryzen Threadripper 3970x CPU with 256 GB of RAM, and an Nvidia RTX 3090 GPU with 24 GB of device memory.

For all experiments, we aggregate data from 21 runs, and report median statistics unless indicated otherwise. For our evaluation of this refinement, we use the HEC coarsening algorithm to provide the coarse graph hierarchy, and we use greedy-graph growing partitioning to generate the initial partition on the coarsest graph. Our coarsening terminates once the coarsest graph has less than 50 vertices, but the coarsest graph may be discarded if it has fewer than 10 vertices. For this reason, our initial partitions are usually outsize the balance constraint, but the serial FM algorithm is usually capable of bringing the partition within the balance constraint after refining the coarsest couple levels. For refinement, we use our new refinement algorithm on the finest 3 levels (Par-Top3), and on all levels (Par-All), in

two separate experiments. This is to determine if our algorithm has any difficulty handling coarser levels of the multilevel hierarchy. In a third experiment, we evaluate our refinement using a basic boundary vertex optimization (Boundary-All), and this is also performed on all levels. We allow our new algorithm and Mt-Metis a maximum imbalance ratio of 1% ($\lambda = 1.01$), whereas the implementation of serial FM we use does not allow for an input imbalance ratio (instead it always returns a 0% imbalance partition). We do not make any claims that our serial FM implementation is optimized for runtime, rather it is optimized for cutsize. We chose to compare to Mt-Metis because it is the most widely used parallel graph partitioner, although its coarsening and initial partitioning differ from ours. We additionally compare to PuLP [11], as our scheme is similar to label propagation.

In Table 4.2, we compare median uncoarsening times for each method. This includes any time spent initializing refinement data structures, projecting the partition vector, and performing the refinement. We use this notion of uncoarsening time as opposed to refinement time, because Mt-Metis reports the time it spends initializing its refinement data structures together with its projection time. This means that the "refinement" time Mt-Metis reports does not include such tasks as calculating the boundary vertices, so we believe total uncoarsening time is a more accurate comparison. Mt-Metis is run with 64 threads on the CPU, serial FM is run with one thread on the CPU, and the three parallel refinement experiments are run on the GPU. In Table 4.3, we compare the total bisection time, including coarsening, initial partitioning, projection, and refinement times.

Our primary purpose of comparison to serial FM is to compare with the cutsizes of our parallel refinement, as our serial FM implementation is integrated into the same codebase as our three parallel experiments. To minimize sources of variation, our serial FM and three parallel experiments share the same coarse graph hierarchy and initial partition in a single run. This enables us to calculate the ratio of the cutsizes achieved by each method. We wanted to do such a comparison using the coarse graph hierarchy and initial partition generated by Mt-Metis, but we could not find a simple way to export these from the application. In Figure 4.1 we report the median of ratios between the FM cutsize and the cutsize of each of the three parallel experiments. We also report the ratio of median cutsizes between FM and Mt-Metis. We exclude comparison to PuLP cutsizes in this figure, as the ratio of PuLP cutsize to FM cutsize is too large to fit inside the bounds for most graphs. Instead we report that the median PuLP cutsizes are 3.57x larger than median FM cutsizes by geometric mean. Interestingly, this ratio is equal for both the regular graphs and irregular graphs. The median PuLP cutsize is only within 1.5x of the median FM cutsize for three regular graphs, and one irregular graph.

We also have CPU execution results for our parallel refinement on all levels, although we do not include these in any figures or tables due to the redundancy of information conveyed. For parallel refinement on all levels, the GPU results have cutsizes 1.02x smaller than the CPU cutsizes by geometric mean. The GPU refinement times for performing parallel refinement on all levels are 3.2x faster than the equivalent CPU refinement times. In a breakdown by graph class, the GPU is 2.6x faster than the CPU on regular graphs, and 3.8x faster on the skew-degree graphs.

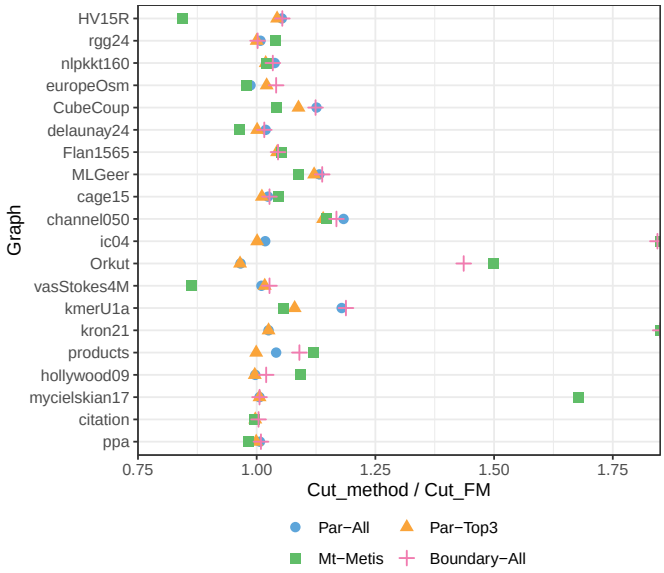


Fig. 4.1: Comparison of cutsize between FM, Parallel Refinement on All Levels, Parallel Refinement on the Top 3 Levels, and Parallel Refinement on All Levels with the Boundary Optimization. Out-of-bounds results are shown on the edges of the figure.

Table 4.2: Median Bisection Refinement Time (s) Comparison

Graph	FM Serial	Par-All	Par-Top3	Boundary-All	Mt-Metis
HV15R	1.1	0.036	0.193	0.034	0.094
rgg24	1.27	0.079	0.095	0.07	0.186
nlpkkt160	1.15	0.071	0.14	0.071	0.131
europeOsm	3.19	0.114	0.203	0.07	0.29
CubeCoup	0.62	0.033	0.081	0.032	0.035
delaunay24	1.07	0.084	0.102	0.054	0.137
Flan1565	0.39	0.027	0.056	0.026	0.027
MLGeer	0.38	0.03	0.044	0.028	0.024
cage15	1.09	0.05	0.195	0.05	0.125
channel050	0.48	0.062	0.081	0.05	0.051
ic04	0.97	0.042	0.102	0.045	0.115
Orkut	3.27	0.056	0.658	0.054	0.385
vasStokes4M	0.8	0.057	0.104	0.038	0.083
kmerU1a	55.66	0.191	1091	0.138	0.791
kron21	9.02	0.106	0.109	0.107	0.49
products	1.77	0.044	0.191	0.041	0.123
hollywood09	1.09	0.025	0.279	0.026	0.142
mycielskian17	2.82	0.042	0.043	0.043	0.353
citation	1.4	0.046	0.173	0.046	0.087
ppa	0.72	0.024	0.143	0.024	0.074

Table 4.3: Median Total Bisection Time (s) Comparison

Graph	FM Serial	Par-All	Par-Top3	Mt-Metis	PuLP
HV15R	1.29	0.23	0.39	0.65	0.17
rgg24	1.63	0.44	0.45	1.07	0.48
nlpkkt160	1.47	0.4	0.47	1.1	0.2
europeOsm	3.6	0.52	0.6	3.39	4.56
CubeCoup	0.79	0.2	0.24	0.26	0.08
delaunay24	1.29	0.3	0.32	0.87	1.51
Flan1565	0.55	0.18	0.21	0.21	0.07
MLGeer	0.51	0.16	0.18	0.16	0.09
cage15	1.29	0.25	0.4	1.66	0.18
channel050	0.65	0.23	0.25	0.37	0.11
ic04	1.18	0.25	0.31	1.84	1.2
Orkut	3.68	0.46	1.06	8.47	1.31
vasStokes4M	1.01	0.27	0.31	1.09	0.29
kmerU1a	56.3	0.83	11.55	9.65	3.86
kron21	9.35	0.43	0.44	15.48	4.1
products	1.95	0.23	0.37	2.04	1.18
hollywood09	1.37	0.3	0.56	2.31	0.5
mycielskian17	3.07	0.29	0.29	0.88	0.31
citation	1.55	0.2	0.33	1.8	0.85
ppa	1.09	0.4	0.51	0.87	0.16

5. Discussion. We note that our parallel refinement on all levels about 2.4x faster by geometric mean than the Mt-Metis refinement, although the former is running on a GPU vs the latter running a 32-core CPU. If we break this down further, we find that it is 1.5x faster on the regular graphs, and 3.6x faster on the skew-degree graphs. If we compare the timing results for our boundary optimization (Table **??** on the regular graphs to Mt-Metis, the geometric mean increases to 2.6x faster (1.8x on regular graphs vs Mt-Metis, 3.9x faster on skew-degree graphs). We must additionally note that our parallel refinement does many more iterations, usually 10-30 depending on the graph, but sometimes as high as 40. This is much larger than the 5 or fewer iterations that Mt-Metis performs. This leads to a large amount of parallel overhead, due to our refinement using 12-15 kernel invocations per iteration. This parallel overhead is particularly notable during refinement of the coarsest graphs, as we note in our experiments that the duration of a single iteration saturates at around 100-120μs, regardless of how few vertices there are. This would indicate a kernel latency of 6-10μs. We ran a separate experiment on the RTX 3090 GPU to determine the latency of launching very small kernels, and found that the kernel latency was 5.9μs.

With respect to quality, we find that the Par-Top3 and Par-All versions of our parallel refinement are similar except on kmerU1a. For this graph, we note that FM outperforms both methods. This seems to indicate that our parallel refinement is not finding certain cutsize decreases that FM can find. This trend is observable on several of the regular graphs as well such as CubeCoup, MLGeer, and channel050. We find that our parallel refinement on all levels doesn't experience any more difficulty on coarser levels than finer levels of the multilevel heirarchy for the majority of test graphs. Our all-level

refinement is neck-and-neck with Mt-Metis or often better, with the exception of HV15R and vasStokes4M. The difference on these two graphs in particular may come down more to coarsening algorithms than refinement. This is often the case for graphs where serial FM and our parallel refinement are superior to Mt-Metis as well. Additionally, the use of a boundary vertex optimization may be responsible for Mt-Metis' worse cuts on several skew-degree graphs. This is corroborated by results for Boundary-All on graphs such as ic04, Orkut, and kron21. Across all graphs, our parallel refinement (Par-All) achieves cutsizes 1.13x smaller by geometric mean than those achieved by Mt-Metis (0.96x smaller on regular graphs, and 1.32x smaller on skew-degree graphs). Comparing to our serial FM implementation, which does not allow any imbalance, our parallel refinement achieves cuts 0.96x smaller by geometric mean. Breaking down further by skewness, the regular graphs have 0.94x smaller cuts with our parallel refinement, versus 0.98x smaller on the skew-degree graphs. Comparing PuLP directly to our parallel refinement, we find our cutsizes to be 3.42x smaller overall, 3.37x smaller for regular graphs, and 3.47x smaller for irregular graphs.

The total bisection time for our GPU partitioner (shown in Table 4.3), greatly outperforms Mt-Metis and is competitive with PuLP, despite PuLP not using the multilevel method. Our Par-All experiment is 1.54x faster than PuLP and 4.33x faster than Mt-Metis (geometric mean) on all graphs. On regular graphs, our partitioner is 0.93x faster than PuLp and 2.38x faster than Mt-Metis. On irregular graphs, our partitioner is 2.51x faster than PuLP and 7.88x faster than Mt-Metis. Our partitioner is the best method tested in terms of both cutsize and runtime for the irregular graphs. Mt-Metis has a slight advantage in cutsize for the regular graphs, while PuLP has a slight advantage in runtime on the regular graphs.

The greatest attribute of this parallel refinement method is its simplicity compared to both FM and greedy refinement methods used by Mt-Metis. When choosing to move any particular vertex, our method never explicitly considers its interactions with neighboring vertices which we might also move in the same iteration. Only once we have chosen all the vertices we desire to move in an iteration, do we then consider how these affect the cutsize and gains in the next cycle. Overall, this refinement is similar to the label propagation scheme used by PuLP, although it differs in how we maintain balance. Additionally, we implement this in a multilevel partitioner, while PuLP is not multilevel.

While developing this algorithm, we considered several related parallelization approaches for refinement, where we attempted to address the issue of interactions between simultaneous moves of adjacent vertices. We investigated using colorings and special constructions of connected components, but these approaches did not work as expected in terms of optimizing cutsize. For instance, we found that using color sets introduces difficulty in selecting vertices to rebalance a partitioning. Selecting vertices from one color at a time makes it more difficult to choose large numbers of vertices with small negative gain. Computing the coloring itself is also an expensive overhead, and some graphs have too many colors to process one at a time. Our connected components construction considered the subgraph induced by the boundary vertices, and formed connected components. This has the benefit of forming independent vertex components for which we calculate the gain, but these clusters often were too large to swap without greatly impacting the imbalance. It was also possible that some vertices in the components detracted from the overall gain of the component. We restricted the boundary by limiting it to only those vertices with positive gain individually, which greatly improved the cutsizes we could obtain, but also made the components redundant. In the context of fine-grained parallelism, we found the

best strategy to address this is to move groups of vertices from the same source part. As mentioned in the "Gain Poisoning" section, even this requirement can be relaxed in certain situations.

6. Conclusion. In this work we present a method for fine-grained parallel graph partition refinement. We develop Kokkos-based implementations of this method. We were able to greatly improve on the speed of our serial implementation of Fiduccia-Mattheyses refinement, and nearly match its performance in terms of bisection quality. Additionally, we demonstrated that our GPU implementation can achieve substantial speedups over CPU partitioners. In future work, we hope to extend this refinement method to k-way partitioning, as well as further optimize time and cutsize results.

References.

[1] T. A. Davis, W. W. Hager, S. P. Kolodziej, and S. N. Yeralan, *Algorithm 1003: Mongoose, a graph coarsening and partitioning library*, ACM Trans. Math. Softw., 46 (2020).
[2] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. on Mathematical Software, 38 (2011).
[3] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202–16.
[4] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in 19th Design Automation Conference, 1982, pp. 175–181.
[5] M. Gilbert, S. Acer, E. G. Boman, K. Madduri, and S. Rajamanickam, *Performance-portable graph coarsening for efficient multilevel graph analysis*, in 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, May 17-21, 2021, IEEE, 2021.
[6] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, *Open graph benchmark: Datasets for machine learning on graphs*, arXiv:2005.00687, (2020).
[7] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998).
[8] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, 49 (1970), pp. 291–307.
[9] D. Lasalle and G. Karypis, *Multi-threaded graph partitioning*, in Proc. IPDPS, 2013, pp. 225–236.
[10] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, *Improving graph partitioning for modern graphs and architectures*, in Proc. Workshop on Irregular Applications: Architectures and Algorithms (IA3), 2015.
[11] G. M. Slota, K. Madduri, and S. Rajamanickam, *PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks*, in Proc. IEEE Int'l. Conf. on Big Data (Big Data), 2014.
[12] J. C. Urschel, J. Xu, X. Hu, and L. T. Zikatanov, *A Cascadic Multigrid Algorithm for computing the Fiedler vector of graph Laplacians*, Journal of Comp. Math., 33 (2015), pp. 209–226.

AN EXPLORATION OF MINIMOD OVERHEAD AND GRANULARITY

DONALD A. KRUSE *, W. PEPPER MARTS †, AND MATTHEW G. F. DOSANJH ‡

Abstract. As we approach Exascale many tools have been proposed to enable future application designs. Some examples include, partitioned communication, multithreaded MPI support through software offloading, and MPI endpoints. MiniMod aims to help designers understand how fine-grained communication affects their application through a modular application structure that allows different communication granularities to be selected at runtime.

MiniMod achieves this by requiring the application kernel to mark each data element as ready when the element is in it's final state for the iteration. There is an overhead related to having small element sizes, as the application must call a 'ready' function for each element. MiniMod processes and transfers this data using logic defined in the Granularity component, with modules such as 'bulk-synchronous', 'fine-grained', and 'bins'. In the bins module, the bin size can be modified to separate a communication payload into multiple payloads. These smaller payloads are sent after they are finished and marked 'ready' by the application. Additionally, small bin sizes have the potential to result in payloads that are smaller than the maximum transmissible unit for the network and increasing message-matching overhead, and thus being bottlenecked by the networks latency rather its bandwidth.

In this study, we measure this overhead by modifying the element size and thereby the number of calls to the 'ready' function. This is done by leveraging kernels that have a configurable element size. Additionally, we explore how different communication granularities impact performance by switching between our three granularity modules.

1. Introduction. MiniMod is a modular proxy application framework created by Marts et al [17] that aims to explore and evaluate new communication access patterns . This framework uses hooks added to a computational kernel to mark data as ready for transfer. This allows the user to select runtime behavior which includes what threading library is being used, which communication interface is being used, and the granularity of communication. One open question about this framework has been how much overhead do these hooks cause and whether data elements should be aggregated before each hook.

To examine the question, we leveraged a five-point stencil kernel that models 2D heat diffusion. This kernel has a parameter that modifies the extent of communication aggregation before calling into MiniMod. By modifying this parameter while using MiniMod to maintain a Bulk Synchronous communication pattern, we can test the overhead of using these hooks.

In this paper we measure and analyze this overhead to determine if there is a granularity beyond which we need to aggregate elements before marking them ready.

The rest of the paper is structured as follows: In Section 2 we present the background of MiniMod as it pertains to this study. In Section 3 we discuss our experimental setup. In Section 4 we present our results and provide an analysis of MiniMod overhead. In Section 5 we present our plans for future expansions to this study. In Section 6 we discuss related work and how it pertains to this study. And finally, in Section 7 we present our conclusions of this work.

2. Background. In recent years a number of proposals for middleware to support fine-grained communication have emerged. These aim to improve application performance through better utilization of network resources. By sending data as soon as it is ready, an application can overlap communication and computation, and spread network utilization out over a larger portion of each iteration. This strategy has the potential to increase the performance of bandwidth bound applications.

*University of New Mexico, krused@unm.edu
†Sandia National Laboratories and University of New Mexico, wmarts@sandia.gov
‡Sandia National Laboratories, mdosanj@sandia.gov

There are several challenges to adapt an application to use fine-grained communication. One can no longer pack data *en masse* at the end of an iteration. Instead one must pack each element sent over the network at the time of completion, but this technique requires careful consideration to preserve the cache performance of the compute loop. In addition, the increased volume of messages can prove costly to communication methodologies such as MPI's two-sided message passing, where the larger message count can increase receive queue depths and thus matching workload. Finally one is incentivized to send smaller payloads over the network, as it allows completion of communicable elements earlier in the computational phase, and thus more overall time in which to achieve communication computation overlap.

MiniMod is a runtime configurable modular application framework. It is designed to explore different levels of granularity in communication and allow for direct comparison between application behaviors. The architecture of MiniMod is divided into three layers arranged hierarchically. At the top is the application layer that consists of the computation kernel and controls over all execution. At the bottom is the interface layer, which executes particular network operations such as sending data via OpenSHMEM or spinning up a work thread with Pthreads. In between is the control layer, which allows the user to select from various granularity methods of completing the requests of application layer with the low level primitives of the interface layer. Each layer is composed of one or more components that further increase the configurability of each layer. Minimod modules implement these components and its overall behavior is determined by configuring which modules to use at runtime. Its granularity component in the control layer provides modules for bulk-synchronous, binned, and element-by-element fine-grained communication.

In order to port an application for use in MiniMod, existing communication and threading calls are replaced with hooks into MiniMod's abstracted interface. These hooks require an initialization function that defines the external needs of each process, i.e., who is sending and receiving what data from what external process, and a ready call that specifies when each data member is ready to be sent. With that high level information, Minimod is able to ensure the requisite data movement is completed by the epoch deadlines defined by a pair of hooks to demarcate the beginning and end of each iteration.

3. Experimental Setup. We applied a strong-scaling method to understand the overhead associated with MiniMod for the different granularity layers at different partition sizes. The number of nodes was fixed at nine with problem sizes 2^{12} to 2^{15} stencil points per side in increments of powers of two, while sweeping through the maximum partition sizes of 1 to the problem size in powers of two.

At each combination of problem size, maximum partition size, and granularity module, 10 trials were completed. The data was the trial total time to complete as well as the total `comm_wait` time. MiniMod was compiled with Intel's `icpc` version 19.1.3.304 with Cray's MPICH 7.7.16 MPI implementation using flags `-g -Wall -O3`. of problem sizes and maximum partition counts on that single set of nine nodes. To ensure that the network configuration remained controlled for the strong scaling configuration, each trial constituted a single set of nine nodes, one of the granularity modules, and the entire set of problem sizes and maximum partition counts.

The heat diffusion kernel uses a five-point stencil halo exchange for each node because it is the minimal configuration for a complete communication pattern that occurs with the middle node.

All experiements were run on the Mutrino testbed at Sandia National Laboritories. The Mutrino cluster is a Cray machine with Intel Haswell CPUs and a Cray Aries dragonfly network. Each node has 32 cores and 128 GB of RAM.

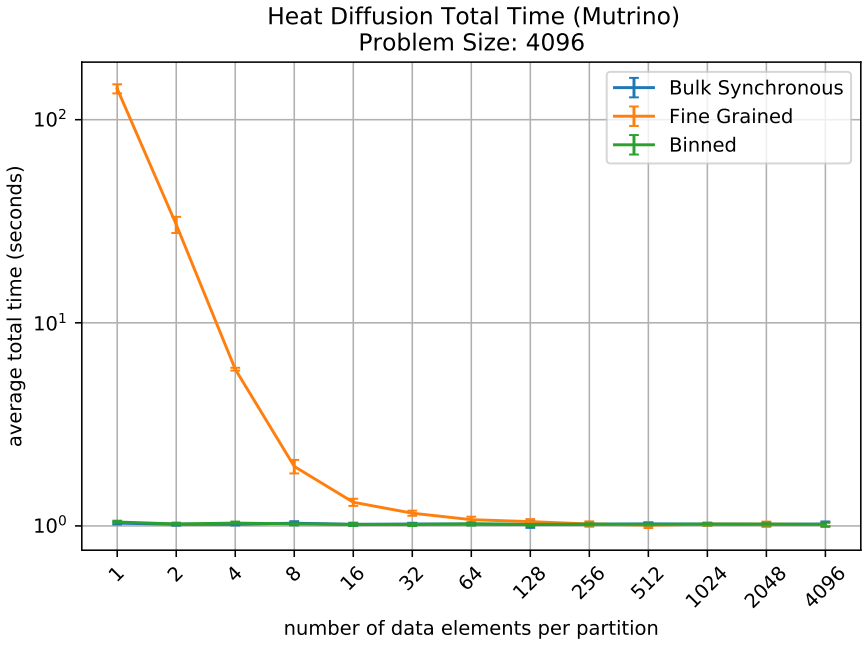


Fig. 4.1: The average total run times of the heat diffusion kernel of size 4096 for all granularity modules.

4. Experimental Results. The bulk-synchronous total run time remained effectively constant for fixed problem size across all partition sizes (see 4.1, 4.2, 4.3, and 4.4) but with increased variance in the total run times for larger problem sizes and is likely due to the fact that for a bulk-synchronous communication, data is only sent all at once when all ranks are ready, so changing the partition size should have negligible effect (see table 4.1).

The `comm_wait` times obeyed a similar pattern. For the smallest two problem sizes, the average total times remained fairly constant with minimal standard deviations. However, for the largest two sizes, the variance was more significant most likely due to large data element sizes used during the communication portions of the kernel. Table 4.2 shows that the largest problem sizes experience greater variance during the communication.

In the case of our heat diffusion kernel, partitioning the communication elements for fine-grained communication had the most significant effect on the total time if the the problem size was large enough. By increasing the the partition count, we are able to take advantage of the large computational overhead and utilize the network when individual data elements are ready. We see that the overall average total time decreases as the number of data elements per partition increases and is most apparent for problem sizes of 16384 and 32768 (see 4.3 and 4.4) and where the performance reaches an optimum at about 16 and 32 elements per partition respectively. Similar trends were noticed for sizes 4096 and 8192 but the effect of fine grained communication is not as pronounced.

The `comm_wait` times trend similarly to that of the total time. This makes sense since we expect the run time to be dependent on the communication time for fine-grained communication.

The binning module admitted similar results as bulk-synchronous. The total time re-

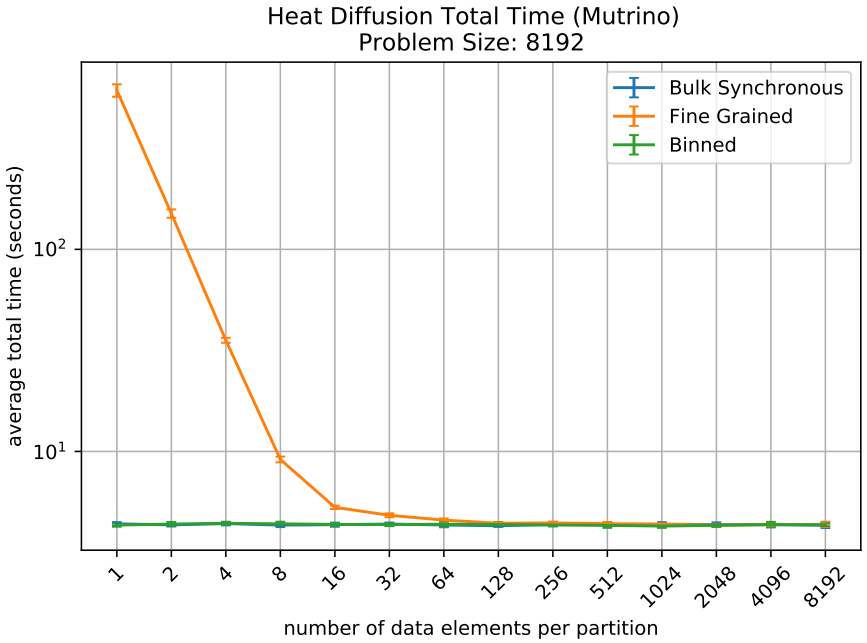


Fig. 4.2: The average total run times of the heat diffusion kernel of size 8192 for all granularity modules.

Total Time Summary (seconds)					
	size	4096	8192	16384	32768
max mean	bulk	1.031656	4.397255	16.52813	64.56762
	fine	142.031000	612.399100	3017.515000	14134.210000
	bins	1.045520	4.408428	16.560920	64.885290
min mean	bulk	1.0166657	4.287672	16.30881	63.88878
	fine	142.031000	612.399100	3017.515000	14134.210000
	bins	1.045520	4.408428	16.560920	64.885290
max σ	bulk	0.038680	0.133902	0.257060	1.197230
	fine	7.327983	43.230068	184.167173	1175.881021
	bins	0.023270	0.151192	0.234688	1.108074
min σ	bulk	0.011931	0.051372	0.074931	0.567370
	fine	7.327983	43.230068	184.167173	1175.881021
	bins	0.023270	0.151192	0.234688	1.108074

Table 4.1: A summary of the extreme values for all the averages of the total run time.

mained about constant despite increasing the maximum partitions (see 4.3 and 4.4). The `comm_wait` time also was semi-constant but showed more fluctuation at the two largest problem sizes. Fine-grained granularity performed similarly to both binning and bulk-synchronous after data transfers reached a size of 1KiB or greater (128 8 byte elements).

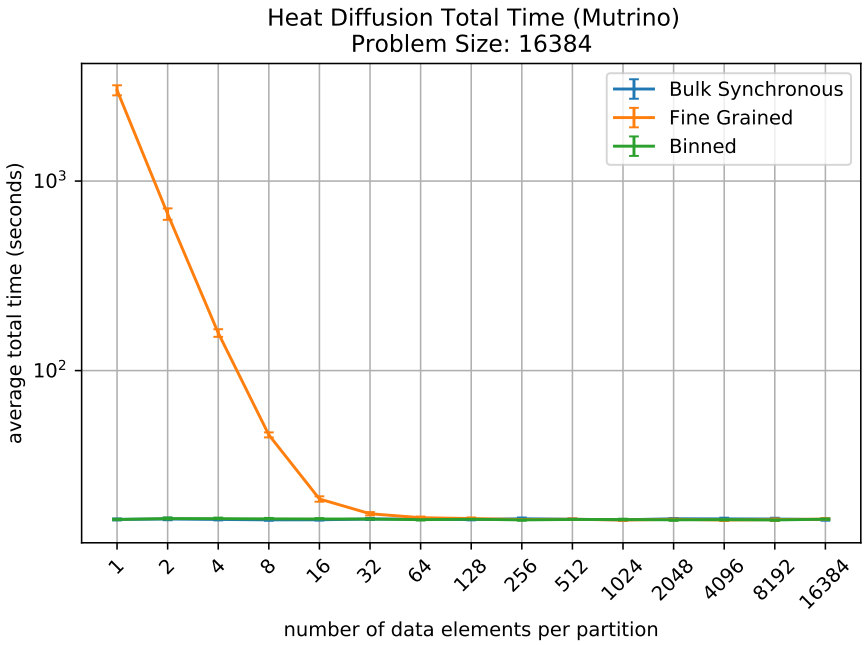


Fig. 4.3: The average total run times of the heat diffusion kernel of size 16384 for all granularity modules.

`comm_wait` Time Summary (seconds)					
	size	4096	8192	16384	32768
max mean	bulk	0.030324	0.069749	0.547271	1.082221
	fine	0.018150	0.020889	0.385387	0.588321
	bins	0.012884	0.021640	0.361266	0.580323
min mean	bulk	0.019197	0.028622	0.028622	0.677152
	fine	0.018150	0.020889	0.385387	0.588321
	bins	0.012884	0.021640	0.361266	0.580323
max σ	bulk	0.013564	0.060339	0.263614	0.410079
	fine	0.007288	0.009593	0.126373	0.180278
	bins	0.005363	0.008116	0.118258	0.202490
min σ	bulk	0.006334	0.011271	0.106094	0.090161
	fine	0.007288	0.009593	0.126373	0.180278
	bins	0.005363	0.008116	0.118258	0.202490

Table 4.2: The summary of all the extreme values for the `comm_wait` time.

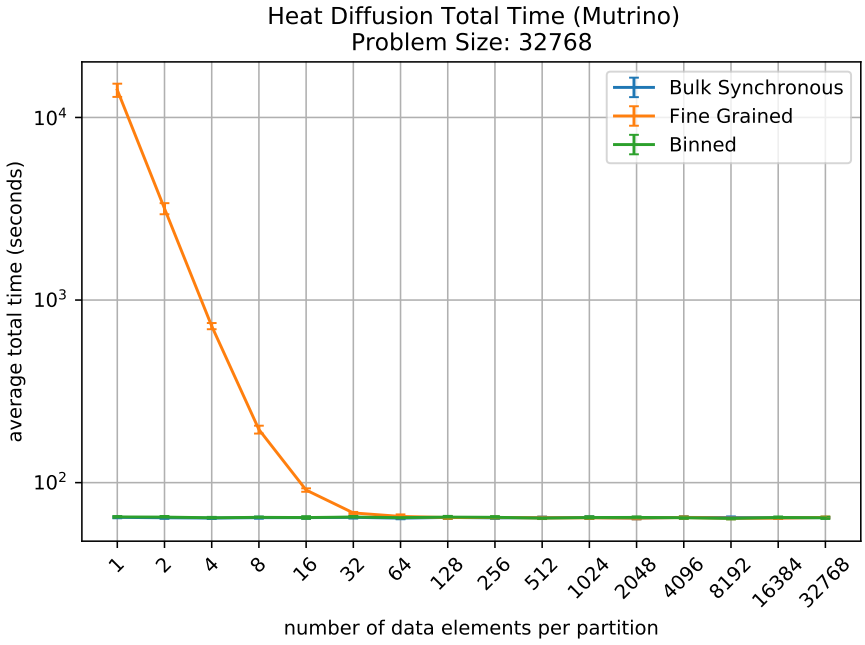


Fig. 4.4: The average total run times of the heat diffusion kernel of size 32768 for all granularity modules.

5. Future Work. There are several limitations to this study that we plan to relax. The context of this study is limited to one computational kernel. While a five-point halo exchange code is used in some cases, it is a limited view of the overall application space. Additionally, its communication to computation ratio shrinks as the problem size scales. If n is the size of the local x and y dimensions of the problem space, communication grows as $O(n)$ while computation grows as $O(n^2)$. To address this we plan to identify and evaluate other computational patterns.

Another avenue of future exploration includes analyzing additional methods to handle granularity. We plan to examine other behaviors for accessing communication, such as creating a dedicated communication thread to reduce communication processing on worker threads. This would allow us to invoke the communication layer from a single thread, which can improve the performance of some communication libraries.

Finally, we plan to extend this study to cover other methods of communication, such as MPI RMA, Persistent Communication, and OpenSHMEM.

6. Related Work. Benchmarking applications can be a challenging endeavor due to their complexity. Therefore, a popular technique is to provide an application proxy, a simple version of the main task of an application that has performance representative of the application itself, enabling the study of the application without a significant burden on the person doing the benchmarking. Proxy application suites like Mantevo [10] are used for this purpose as well as many different versions of singular miniapps like LULESH [13]. For each communication subsystem (e.g., MPI, OpenSHMEM, RDMA) a separate benchmark needs to be written. Therefore there are many different versions of popular benchmarks, each of which needs to be maintained. In addition, new subsystems or system architectures may require new versions of the miniapps, each having to release separate new versions.

There have been many prior works evaluating communication middleware and comparing approaches. MPI has been extensively studied in traditional modes [9, 15] as well as one-sided [3, 11]. MPI multi-threading is currently a hot topic and has been extensively explored recently [4–6, 14, 16, 18]. Many MPI comparisons are limited to studying MPI libraries themselves–some by necessity due to comparisons of new features or proposed features [2, 7, 8, 12, 19]. However, such new approaches would be desirable to compare broadly across many different communication subsystems which this work enables and makes significantly less burdensome. This is a necessity when creating a new communication middleware solution to convince new users that they can achieve higher application performance with the new method. It can be challenging to compare approaches comprehensively as it requires many application proxies using a wide variety of middleware options. It is clear that application developers want to compare communication middleware options as well as concurrency methods, as shown in a survey of US application developers [1].

7. Conclusions. In this work we have demonstrated that MiniMod's ready calls have a low functional overhead via our course grained experimental results. Additionally, we have shown that choosing too few data members per data transfer using fine-grained granularity has serious performance penalties. Finally, we see that for binning granularity there is measurable if small performance gains.

References.

[1] D. E. Bernholdt, S. Boehm, G. Bosilca, M. Venkata, R. E. Grant, T. Naughton, H. Pritchard, and G. Vallee, *A survey of MPI usage in the U.S. Exascale Computing Project*, Concurrency and Computation: Practice and Experience, (2018). DOI: 10.1002/cpe.4851.

[2] J. Dinan, R. E. Grant, P. Balaji, D. Goodell, D. Miller, M. Snir, and R. Thakur, *Enabling communication concurrency through flexible MPI endpoints*, The International Journal of High Performance Computing Applications, 28 (2014), pp. 390–405.

[3] M. G. F. Dosanjh, T. Groves, R. E. Grant, R. Brightwell, and P. G. Bridges, *RMA-MT: a benchmark suite for assessing MPI multi-threaded RMA performance*, in 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2016, pp. 550–559.

[4] K. B. Ferreira, S. Levy, K. Pedretti, and R. E. Grant, *Characterizing MPI matching via trace-based simulation*, in Proceedings of the 24th European MPI Users' Group Meeting, EuroMPI '17, New York, NY, USA, 2017, ACM, pp. 8:1–8:11.

[5] M. Flajslik, J. Dinan, and K. D. Underwood, *Mitigating MPI message matching misery*, in International Conference on High Performance Computing, Springer, 2016, pp. 281–299.

[6] S. M. Ghazimirsaeed, R. E. Grant, and A. Afsahi, *A dynamic, unified design for dedicated message matching engines for collective and point-to-point communications*, Parallel Computing, 89 (2019), p. 102547.

[7] R. Grant, A. Skjellum, and P. V. Bangalore, *Lightweight threading with MPI using persistent communications semantics*, tech. rep., Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2015.

[8] R. E. Grant, M. G. Dosanjh, M. J. Levenhagen, R. Brightwell, and A. Skjellum, *Finepoints: Partitioned multithreaded MPI communication*, in International Conference on High Performance Computing, Springer, 2019, pp. 330–350.

[9] W. Gropp and E. Lusk, *Reproducible measurements of MPI performance characteristics*, in European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting, Springer, 1999, pp. 11–18.

[10] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, *Improving Performance via Mini-applications*, Tech. Rep. SAND2009-5574, Sandia National Laboratories, 2009.

[11] N. Hjelm, M. G. F. Dosanjh, R. E. Grant, T. Groves, P. Bridges, and D. Arnold, *Improving MPI multi-threaded RMA communication performance*, in Proc. of the Int. Conf. on Parallel Processing, 2018, pp. 1–10.

[12] D. Holmes, K. Mohror, R. E. Grant, A. Skjellum, M. Schulz, W. Bland, and J. M. Squyres, *MPI sessions: Leveraging runtime infrastructure to increase scalability of applications at exascale*, in Proceedings of the 23rd European MPI Users' Group Meeting, 2016, pp. 121–129.

[13] I. Karlin, J. Keasler, and J. Neely, *Lulesh 2.0 updates and changes*, tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013.

[14] S. Levy, K. B. Ferreira, W. Schonbein, R. E. Grant, and M. G. Dosanjh, *Using simulation to examine the effect of MPI message matching costs on application performance*, Parallel Computing, 84 (2019), pp. 63–74.

[15] J. Liu, J. Wu, and D. K. Panda, *High performance RDMA-based MPI implementation over infiniband*, International Journal of Parallel Programming, 32 (2004), pp. 167–198.

[16] W. P. Marts, M. G. Dosanjh, W. Schonbein, R. E. Grant, and P. G. Bridges, *MPI tag matching performance on ConnectX and ARM*, in Proceedings of the 26th European MPI Users' Group Meeting, 2019, pp. 1–10.

[17] W. P. Marts, M. G. F. Dosanjh, S. Levy, W. Schonbein, R. E. Grant, and P. Bridges, *Minimod: A modular miniapplication benchmarking framework for hpc"*, in Proceedings of IEEE Cluster, vol. Virtual.

[18] W. Schonbein, M. G. Dosanjh, R. E. Grant, and P. G. Bridges, *Measuring multithreaded message matching misery*, in European Conference on Parallel Processing, Springer, 2018, pp. 480–491.

[19] R. Zambre, A. Chandramowlishwaran, and P. Balaji, *How i learned to stop worrying about user-visible endpoints and love MPI*, arXiv preprint arXiv:2005.00263, (2020).

EFFICIENT COMPUTATION OF HIGHER ORDER JOINT MOMENT TENSOR

ZITONG LI* AND HEMANTH KOLLA†

Abstract. The decomposition of higher-order joint cumulant tensor of spatial temporal data sets is useful in analyzing multivariate non-Gaussian statistics with a wide variety of applications (e.g. anomaly detection, independent component analysis, dimensionality reduction). Computing the cumulant tensor often requires computing the joint-moment tensor of the input data first, which is very expensive using a naïve algorithm. The current state-of-the-art algorithm for computing joint moment tensors takes advantage of the symmetric nature of a moment tensor by dividing it into smaller cubic tensor blocks and only compute the blocks with unique values and thus reducing computation. We propose an improvement over this algorithm by posing its computation as matrix operations, specifically Khatri-Rao products and standard matrix multiplications. Because this approach is much more cache efficient, we expect considerable speedup in single processor. We implemented our algorithm in Julia and in MATLAB and compared against the state-of-the-art approach. The results show a speedup of up to 10x.

1. Introduction. Many scientific applications in the exascale era involve computations/observations of multi-scale multivariate physical phenomena. Higher-order joint statistical moments and cumulants are natural, information-rich, compact statistical representations that can facilitate a wide variety of statistical learning in such high-dimensional data. Such compact representations can also aid prominent analytics including surrogate/reduced order model construction, dimensionality reduction, low-dimensional manifold and subspace identification, uncertainty quantification, data fusion/assimilation from different sources (experiment, simulation, sensors).

Joint cumulants can be defined using joint moments, which are multivariate extensions of the commonly used univariate (marginal) moments. Consider a vector of N random variables $\mathbf{X} \equiv [X_1, X_2, \ldots, X_N]$. The joint moments $\mathfrak{M}$ of orders two, three, and four can be defined using index notation as, respectively,

$$\mathfrak{M}_{i,j} = \mathbb{E}[X_i X_j], \;\; \mathfrak{M}_{i,j,k} = \mathbb{E}[X_i X_j X_k], \;\; \mathfrak{M}_{i,j,k,l} = \mathbb{E}[X_i X_j X_k X_l], \;\; 1 \leq i,j,k,l \leq N, \tag{1.1}$$

where $\mathbb{E}$ is the expectation operator, and the number of indices reflects the order of the moment. The joint cumulants $\mathfrak{C}$ are related to joint moments. The first order cumulant is equal to the first order moment and for the second, third and fourth order cumulants the relations are

$$\mathfrak{C}_{i,j} = \mathfrak{M}_{i,j} - \mathfrak{M}_i \mathfrak{M}_j, \tag{1.2}$$

$$\mathfrak{C}_{i,j,k} = \mathfrak{M}_{i,j,k} - \mathfrak{M}_i \mathfrak{M}_{j,k} - \mathfrak{M}_j \mathfrak{M}_{i,k} - \mathfrak{M}_k \mathfrak{M}_{i,j} + 2\mathfrak{M}_i \mathfrak{M}_j \mathfrak{M}_k, \tag{1.3}$$

$$\begin{aligned} \mathfrak{C}_{i,j,k,l} = {} & \mathfrak{M}_{i,j,k,l} - \mathfrak{M}_i \mathfrak{M}_{j,k,l} - \mathfrak{M}_j \mathfrak{M}_{i,k,l} - \mathfrak{M}_k \mathfrak{M}_{i,j,l} - \mathfrak{M}_l \mathfrak{M}_{i,j,k} - \mathfrak{M}_{i,j} \mathfrak{M}_{k,l} \\ & - \mathfrak{M}_{i,k} \mathfrak{M}_{j,l} - \mathfrak{M}_{i,l} \mathfrak{M}_{j,k} + 2(\mathfrak{M}_i \mathfrak{M}_j \mathfrak{M}_{k,l} + \mathfrak{M}_i \mathfrak{M}_k \mathfrak{M}_{j,l} + \mathfrak{M}_i \mathfrak{M}_l \mathfrak{M}_{j,k} \\ & + \mathfrak{M}_j \mathfrak{M}_k \mathfrak{M}_{i,l} + \mathfrak{M}_j \mathfrak{M}_l \mathfrak{M}_{i,k} + \mathfrak{M}_k \mathfrak{M}_l \mathfrak{M}_{i,j}) - 6\, \mathfrak{M}_i \mathfrak{M}_j \mathfrak{M}_k \mathfrak{M}_l \end{aligned} \tag{1.4}$$

The relations simplify considerably if one considers centered joint moments, i.e. if the random variables comprising $\mathbf{X}$ are centered around their means.

While many analytics techniques are centered on the co-variance and second-order statistics, with a few exceptions such as financial modeling and medical diagnostics, higher-order joint statistics have not been widely adopted for analyses by scientific applications. A

*Wake Forest University, liz20@wfu.edu
†Sandia National Laboratories, hnkolla@sandia.gov

key property of multivariate Gaussian distributions is that all cumulants of order greater than two are zero. However, many scientific phenomena, e.g. turbulence, are associated with non-Gaussian statistics and important statistical information is contained in joint moments of order greater than two. Independent Component Analysis (ICA) [4] identifies a linear transformation of multivariate data to a set of *statistically independent* variables, which is a stronger condition than linearly uncorrelated variables identified by Principal Component Analysis (PCA). ICA is specifically geared towards non-Gaussian statistics and certain classes of ICA algorithms are based on the fourth-order joint cumulant tensor [3, 5]. Higher-order statistical moments have been employed for assessing accuracy of coupled climate models [19], and for unique source identification using ICA when comparing different model predictions to observed data [10, 17]. They have also been used for various analyses in hyperspectral imaging [11, 13, 21] and medical electrodiagnostic techniques, e.g. electroencephalography (EEG), electrocardiography (ECG) and electromyography (EMG) [6, 8, 16, 22]. Based on the principle that outliers are another manifestation of non-Gaussian behavior, higher-order moment tensors have also been tailored for anomaly detection [1, 18].

In applying tensor and multi-linear algebra, software have been typically designed for the scenario where the raw data (field quantities at every mesh point) itself constitutes a higher-order tensor. In contrast, for higher-order joint statistics *the tensor is not the raw data, rather a result of contractions on the raw data.* This is evident from Eq. 1.1, where the moment tensors are a result of an expectation operation over samples spanning space and/or time. For an application with N variables, and joint statistics of order d, the full tensor is dense and of size N^d. The tensor, being super-symmetric, has duplicate elements, with the number of unique elements being equal to $\binom{N+d-1}{d}$. While the tensor size is independent of the mesh size and time steps, each element of the tensor is a result of contractions over samples from a suitable portion of the space-time domain, and the mesh size and time steps are reflected in the cost of forming the tensor. For exascale applications the space-time degrees of freedom can be order of billions, and hence it is vital to develop efficient and scalable algorithms for computing higher-order moment tensors efficiently.

2. Preliminaries.

2.1. Notation.

We are going to use the capital letters in bold Euler script front ($\boldsymbol{\mathcal{X}}$) to denote a tensor. For matrices we use capital letters in bold ($\mathbf{Y}$). For vectors we use the lower case letter in bold ($\boldsymbol{a}$).

$\boldsymbol{\mathcal{M}}^{(k)}$	k-th moment tensor
$\boldsymbol{\mathcal{C}}^{(k)}$	k-th cumulant tensor
$\odot$	*Row-wise* Khatri-Rao product
$\bigodot_i^j \mathbf{X}_i$	$\mathbf{X}_1 \odot ... \odot \mathbf{X}_j$
$\boldsymbol{\mathcal{M}}_{B(i,j,k)}$	The block of $\boldsymbol{\mathcal{M}}$ with index (i, j, k)
$\circ$	Outer product

In this paper, we will assume that the input matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ where $m >> n$.

2.2. Symmetric tensor.

In this paper, a *supersymmetric tensor* refers to a tensor whose elements have the same value for all the permutations of their indices. For example, a 3-dimensional tensor $\boldsymbol{\mathcal{X}}$ is supersymmetric if and only if

$$\boldsymbol{\mathcal{X}}_{i,j,k} = \boldsymbol{\mathcal{X}}_{i,k,j} = \boldsymbol{\mathcal{X}}_{k,i,j} = \boldsymbol{\mathcal{X}}_{k,j,i} = \boldsymbol{\mathcal{X}}_{j,k,i} = \boldsymbol{\mathcal{X}}_{j,i,k}$$

for any i, j, and k. For brevity, we will refer to supersymmetric tensors simply as symmetric tensors. All symmetric tensors are *cubical*, which means each of its dimensions has the same size. The following are two obvious properties of a d-dimensional symmetric tensor of size s (i.e. the size of each dimension is s) :

- The total number of elements is s^d.
- The number of unique elements is at most $\binom{s+d-1}{d}$.

From these two properties we can see that the storage and computation cost of symmetric tensor can be reduced by $\mathcal{O}(d!)$ if we can exploit its symmetric nature.

2.3. Vectorization of a tensor. It is sometimes useful to convert a tensor into a vector. The following function $vec : \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d} \to \mathbb{R}^{I_1 I_2 \ldots I_d}$ is one of the ways to do this:

DEFINITION 2.1. $vec(\mathcal{X})_\alpha = \mathcal{X}_{i_1,i_2,\ldots,i_d}$ *where*

$$\alpha = 1 + \sum_{k=1}^{d} (i_k - 1) \prod_{l=1}^{k-1} I_l$$

2.4. Storing supersymmetric tensor. One of the efficient ways of storing a symmetric tensor is proposed in [20] as Blocked Compact Symmetric Storage (BCSS). We adopt this approach in our implementation. The general idea of BCSS is to partition the d-dimensional tensor into smaller d-dimensional blocks and only store those blocks with unique values using conventional ordering (e.g. column major). Figure 2.1 shows an example where a $4 \times 4 \times 4$ tensor is partitioned into 8 $2 \times 2 \times 2$ tensor blocks. Only 4 of those blocks(e.g. those in the legend) have unique values and are stored. We choose to store only those blocks whose indices are in strictly increasing order because this makes it easy to list out those unique blocks.

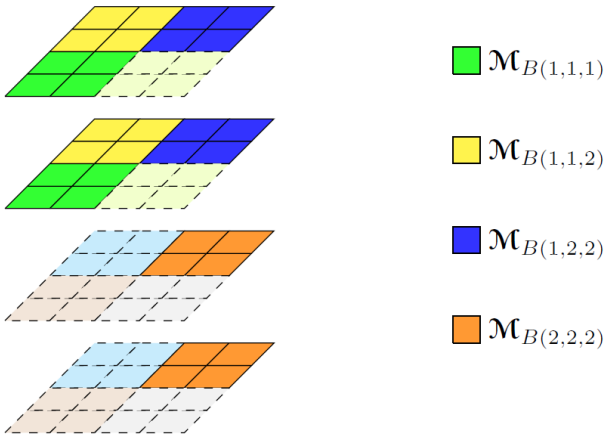


Fig. 2.1: How a $4 \times 4 \times 4$ symmetric tensor is divided into $2 \times 2 \times 2$ blocks. The 4 blocks included in the legend have unique values and only those are stored.

This data structure is not optimal in terms of memory efficiency because the blocks on the super diagonal of a symmetric tensor are symmetric and thus contains redundancy. More specifically, for a d-dimensional supersymmetric tensor $\mathcal{X}$ of size n, let the block size be s. There are $\frac{n}{s}$ blocks on each mode, resulting in $\binom{\frac{n}{s}+d-1}{d}$ blocks where each block has

s^d entries. So the total storage needed with this blocked approach will be

$$s^d \binom{\frac{n}{s} + d - 1}{d}$$

. Whereas the storage needed for only the unique values is

$$\binom{s + d - 1}{d}$$

. This sacrifice of memory is justified since otherwise we have to use alternative ways of ordering the unique values of the tensor in memory, which causes the complexity of indexing them to increase quickly [12].

2.5. Moment tensor. For $\mathbf{X} \in \mathbb{R}^{m \times n}$, its dth moment tensor is a d-dimensional tensor where each of its dimension has the same size. In this paper, we assume that each row of $\mathbf{X}$ contains a sample of n variables. The element-wise expression for the dth moment tensor $\mathcal{M}$ of $\mathbf{X}$ is shown below:

$$\mathcal{M}_{i_1, i_2, \ldots, i_d} = \frac{1}{m} \sum_{a=1}^{m} \prod_{x=1}^{d} X_{a, i_x} \tag{2.1}$$

Computing this moment tensor in this naive way is very expensive. However, it's easy to see that moment tensor of any matrix is supersymmetric. It is natural to think about exploiting the symmetry of the moment tensor by employing the BCSS data structure and only computing the elements in its unique blocks.

2.6. Related work. In [7], Domino *et al.* successfully exploited the symmetric structure of the moment tensor and employed the BCSS to reduced the operations needed to compute the moment tensor. Their performance experiments shows that a block size of 2 is usually the optimal. This means the memory requirement for storing the moment tensor is also greatly reduced. However, their implementation still has room for improvement. Specifically, to compute each element of a d-dimensional moment tensor, they use two nested loops corresponding to the summation and multiplication shown in Equation (2.1). It is well known that this type of nested for loops can usually be vectorized and posed as matrix operations, which is much more cache efficient. In fact, this is what we have done and we consider it to be the main contribution of this work. We will elaborate on our algorithms and the improvements in performance in the following section.

2.7. Row-wise Khatri-Rao product. For the following sections, we need to first introduce the row-wise Khatri-Rao product. Denoted by $\odot$, the row-wise Khatri-Rao product on two matrices with the same number of rows is defined as the following:
With $\mathbf{A} \in \mathbb{R}^{r \times m}$, $\mathbf{B} \in \mathbb{R}^{r \times n}$, $\mathbf{C} \in \mathbb{R}^{r \times mn}$ and $\otimes$ denoting the Kronecker product:

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{A}_{1,:} \otimes \mathbf{B}_{1,:} \\ \mathbf{A}_{2,:} \otimes \mathbf{B}_{2,:} \\ \ldots \\ \mathbf{A}_{r,:} \otimes \mathbf{B}_{r,:} \end{bmatrix}$$

The complexity of the row-wise Khatri-Rao product is $\mathcal{O}(rmn)$. As an example, given

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 9 & 6 & 3 \\ 8 & 5 & 2 \\ 7 & 4 & 1 \end{bmatrix}$$

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} 9 & 6 & 3 & 18 & 12 & 6 & 27 & 18 & 9 \\ 32 & 20 & 8 & 40 & 25 & 10 & 48 & 30 & 12 \\ 49 & 28 & 7 & 56 & 32 & 8 & 63 & 36 & 9 \end{bmatrix}$$

3. Algorithm.

3.1. Computing moment tensor with matrix operations. As we mentioned in Section 2.6, we can speed up the computation of the moment tensor substantially if we can frame the computation as matrix operations instead of nested for loops. In fact, given the input matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, we can compute its moment tensors with two series of Khatri-Rao products and a standard matrix multiplication. The vectorization, defined in section 2.3, of the 2nd moment tensor, which is a matrix, is simply the following:

$$vec(\mathbf{M}^{(2)}) = vec(\frac{1}{m}\mathbf{X}^T\mathbf{X}) \tag{3.1}$$

The vectorization of the 3rd moment tensor can be computed as:

$$vec(\mathbf{M}^{(3)}) = vec(\frac{1}{m}\mathbf{X}^T(\mathbf{X} \odot \mathbf{X})) \tag{3.2}$$

The vectorization of the 4th moment tensor can be computed as:

$$vec(\mathbf{M}^{(4)}) = vec(\frac{1}{m}(\mathbf{X} \odot \mathbf{X})^T(\mathbf{X} \odot \mathbf{X})) \tag{3.3}$$

The vectorization of the dth moment tensor can be computed as:

$$vec(\mathbf{M}^{(d)}) = vec(\frac{1}{m}(\bigodot_{i=1}^{\lceil d/2 \rceil} \mathbf{X})^T(\bigodot_{i=\lceil d/2 \rceil+1}^{d} \mathbf{X})) \tag{3.4}$$

Organizing the computation this way might seem out of the blue but it is in fact very similar to how we reconstruct the tensor from its CP factor matrices. The connection between computing moment tensor and CP decomposition[14] is the following: A CP decomposition can be seen as decomposing a tensor into the sum of a series of rank-one tensors. For a 3-way tensor $\mathcal{X}$ with rank R, its CP decomposition can be written as:

$$\mathcal{X} = \sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r$$

. By definition, a d-dimensional rank-one tensor is the outer product of d vectors. The d-th moment tensor of an input matrix $\mathbf{X}$ can also be seen as the sum of a series of rank-one tensors where the i-th rank-one tensors is the outer product of the i-th row of $\mathbf{X}$. The idea of using Khatri-Rao product to replace the sum of a series of outer product can be found in this survey paper [15].

3.2. Complexity. In this approach using Khatri-Rao products, for d-th order moment tensor, if the input matrix has shape $m \times n$, the complexity for computing the whole moment tensor is:

$$2m(n^2\frac{(n^{d/2-1}-1)}{n-1}) + 2mn^d = \mathcal{O}(2mn^d) \tag{3.5}$$

On the other hand, the complexity for using Equation (2.1) to compute the moment tensor element-wise is:

$$(dm+m)n^d = O(dmn^d) \tag{3.6}$$

We can see that these two are essentially the same when d is small, which is typically the case.

3.3. Blocked version. The previous section introduces the general idea of how to compute the entire moment tensor with matrix operations. To leverage the symmetry of the moment tensor we have to use the blocked data structure, which allows us to skip computing the blocks that are redundant in the symmetry. For simplicity, let's assume that with input matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, the block size s can divide n. The building blocks of this algorithms are the column blocks of $\mathbf{X}$. In the following equations, we denote the ith column block of $\mathbf{X}$ as $\mathbf{X}_i$, which is defined as $\mathbf{X}[:, ib - b + 1 : ib]$ (b is the block size). For the 4th order moment tensor, we can compute vectorization of the block with index (i, j, k, l) as the following:

$$vec(\mathbf{M}_{B(i,j,k,l)}) = vec(\frac{1}{m}(\mathbf{X}_i \odot \mathbf{X}_j)^T(\mathbf{X}_k \odot \mathbf{X}_l)) \tag{3.7}$$

More generally, for the d-th order moment tensor, we can compute the vectorization of the block with index $(i_1, i_2, ..., i_d)$ as the following:

$$vec(\mathbf{M}_{B(i_1,i_2,...,i_d)}) = vec(\frac{1}{m}(\bigodot_{i=1}^{\lceil d/2\rceil} \mathbf{X}_i)^T(\bigodot_{i=\lceil d/2\rceil+1}^{d} \mathbf{X}_i)) \tag{3.8}$$

4. Implementation details. Domino *et al.*[7] implemented their algorithm in Julia, a high-level language tuned for performance. We also implemented our algorithm in Julia to make direct comparisons. We adopted the `symmetricTensor` data structure implemented by Domino *et al.* The pseudocode for our sequential algorithm is shown in Algorithm 1. Here $\mathbf{X} \in \mathbb{R}^{m\times n}$ is the input matrix, $\mathbf{M}$ is the 4th moment tensor of $\mathbf{X}$ and s is the block size.

One thing to note about Algorithm 1 is that it is solely focused on computing the 4th order moment tensor. To compute lower/higher order moment tensors we would have to modify the code to include less/more nested for loops. In contrast, the implementation by Domino *et al.* can compute moment tensor of arbitrary order. Our reasoning for implementing it this way is the following: First, at the moment, we rarely see a need to compute moments higher than 4th order. Second, modifying our code to compute higher order moment tensor is relatively easy. Finally, using this nested loop structure allow some additional saving of computation, which we discussion in Section 4.1.

Algorithm 1 Sequential Algorithm for the 4th Joint Moment Tensor

```
1: function L =4thMomentTensor(M, X, s)
2:     b = ceil(n/s)                          ▷ b is the number of blocks on each dimension
3:     for i = 1:b do
4:         A = X(:, (i − 1) ∗ s + 1 : i ∗ s)
5:         for j = i:b do
6:             B = X(:, (j − 1) ∗ s + 1 : j ∗ s)
7:             E = B ⊙ A
8:             for k = j:b do
9:                 E = X(:, (k − 1) ∗ s + 1 : k ∗ s)
10:                for l = k:b do
11:                    D = X(:, (l − 1) ∗ s + 1 : l ∗ s)
12:                    F = D ⊙ C
13:                    M(i, j, k, l) = (1/m) E^T F
14:                end for
15:            end for
16:        end for
17:    end for
18: end function
```

In this algorithm, there are two computation intensive kernels: the Khatri-Rao product and matrix multiplication. For matrix multiplication, we used the BLAS `gemm` interface implemented by Julia. We implemented our own Khatri-Rao product of $\mathbf{A}$ and $\mathbf{B}$ as broadcast multiplications between each column of $\mathbf{A}$ and the entire $\mathbf{B}$. In addition, one problem we had to solve for using matrix operations is memory allocation/deallocation. Comparing with computing each element of the moment tensor individually, using matrix operations needs more memory allocation/deallocation, which can become a bottleneck in the algorithm if not managed properly. Specifically, in lines 4, 6, 9, and 11 of Algorithm 1, Julia will make copies the submatrices instead of creating a reference. We addressed this issue by using the `@views` macro, which forces those submatrices to be references to the original input matrix. In addition, simply allocating memory for matrices $\mathbf{E}$ and $\mathbf{F}$ when they are computed in line 9 and 10 also result in unnecessary allocation and deallocation in each iteration of the for loops. We address this problem by allocating memory for those matrices outside of the loop and reusing the associated memory when possible.

To rule out the influence of performance difference that results from the language particular features, we also implemented both our algorithm and that proposed by Domino *et al.* in MATLAB and compared the performances. The results are shown in Section 5.3. In this implementation, we used the Khatri-Rao product that implemented in Tensor Toolbox [2] and the standard matrix multiplication implemented by MATLAB.

4.1. Complexity of `4thMomentTensor` implementation. Using nested for loops to iterate through the block indices allows us to reuse the results of some of the Khatri-Rao products. If both Khatri-Rao product is computed in the inner-most for loop, we will need $2\binom{b+3}{4}ms^2$ operations for all the Khatri-Rao products. By moving one Khatri-Rao product to the outer loop we will need $(\binom{b+1}{2} + \binom{b+3}{4})ms^2$ operations, resulting in a saving of $\mathcal{O}(\frac{b^4}{24}ms^2)$ operations. This speed up is not necessarily significant especially considering that b is usually fairly small in applications where the typical shape of the input matrix $\mathbf{X}$ is often have much more rows than columns.

5. Performance results. The following experiments are conducted on a laptop with a Intel Ice Lake i7 CPU running on a base frequency of 2.3 GHz. In all the experiments we

are only using 1 thread on 1 core.

5.1. Julia results. We compared our Julia implementation against that by Domino *et al.* The input for this experiment is a randomly generated matrix. In our case, the typical shape for the input tensor is tall and skinny. The number of rows can range from the thousands to tens of thousands while the number of columns is usually no more than 100. The results of the following two experiments are shown in Figure 5.1. In the first experiment experiment, we fixed the number of rows of the input tensor to 1000 and the block size to 2 and vary the number of columns. The time for the two implementations are recorded in the left figure below. We can see that speed up is consistent at around 5x to 10x.

In this next experiment, we fixed the number of columns of the input tensor to 30 and the block size to 2 and vary the number of rows. The time for the two implementations are recorded in the right figure below. We can see that speed up is consistent again at around 5x to 10x. The reason for the speed up in both these experiments is mostly due to the cache efficiency of matrix operations.

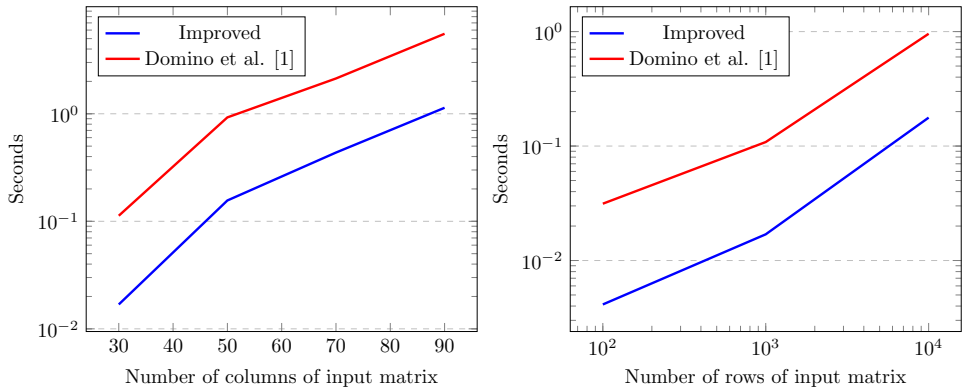


Fig. 5.1: Scaling the number of rows/columns. For the plot on the left, we fix the input $\mathbf{X}$ to 1000 rows and the block size to 2. For the plot on the right, we fix the input $\mathbf{X}$ to 30 rows and the block size to 2.

5.2. Optimal block size. In this experiment, we test the impact of having different block size has on the performance. We fixed the number of columns of the input tensor to 1000 and the number of rows to 30 and varied the block size. The time comparison for the two implementations are recorded in Figure 5.2. We can see that for the implementation from [7], as the block size increase, the performance decreases. However, for our implementation, increasing the block size to an extent will result in increase in performances. Specifically, going from a block size of 2 to a block size of 3 results in a nearly 2x speed up. This is because when the block size increases, the number of computation increases while the number of matrix operations needed decreases. When starting from a small block size such as 2, the benefit of having launching less matrix operations overweighs the cost of the increase in computation. The optimal block size that we recorded is around 20. However, the optimal block size is most likely dependant on the hardware and has to be tuned. More importantly, increasing the block size not only increases the flop count but also increases the memory needed to store the final moment tensor as the redundancy in the diagonal blocks increases. That being said, even with the block size being suboptimal such as 2, we are still seeing a significant speed up of about 5x using our algorithm.

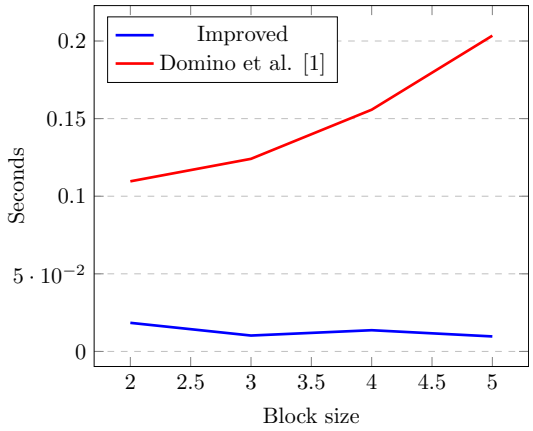


Fig. 5.2: Optimal block size in Julia. In this experiment we fix the input $\mathbf{X}$ to 1000×30

5.3. Matlab results. In this section we show the results of the same experiments in last section ran on our MATLAB implementations. Note although we label the red curves in these figures as Domino *et al.* This is not their implementation. Instead, we have translated their Julia code to MATLAB in order to make this comparison.

From the results shown in Figure 5.3, we can see the MATLAB results largely confirms what we have seen in the previous section. The MATLAB implementation is slower in general than the Julia implementation. This is because we haven't done the extensive performance tuning to either of the approaches as we have done for the Julia version. This results shows that the improvements in performance that our algorithm brings is not particular to one programming language.

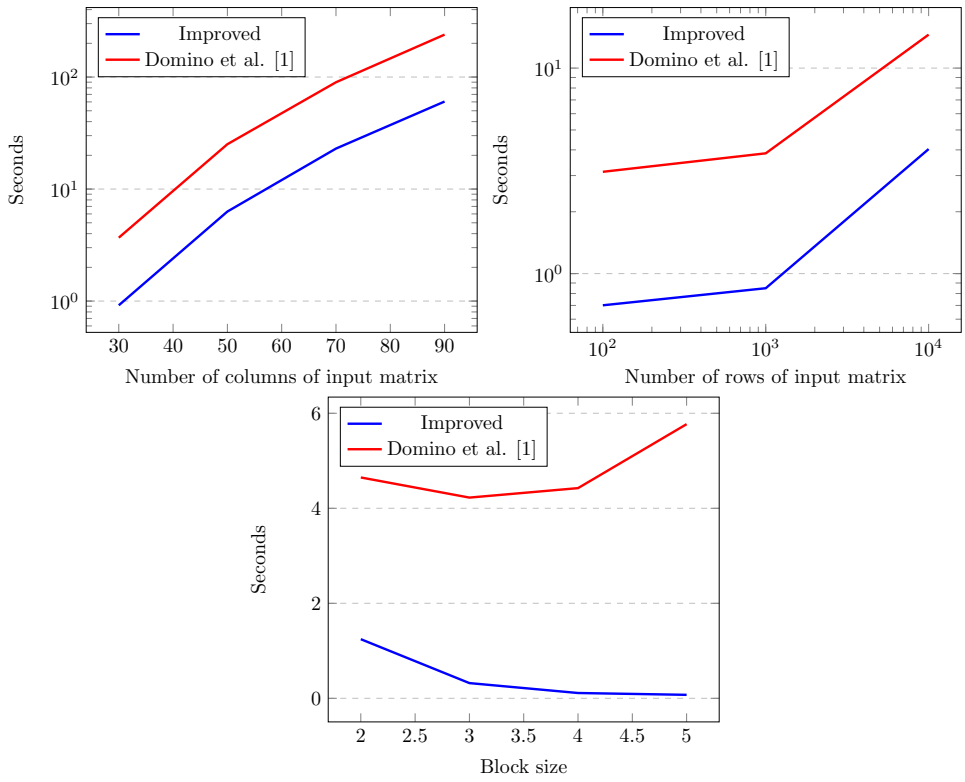


Fig. 5.3: MATLAB experiment results. Similar to the Julia experiments, the baseline input $\mathbf{X}$ is 1000×30 with a block size of 2.

6. Ongoing work. Currently we are implementing this algorithm in parallel on a shared memory system. We aim to make this implementation a performance portable one by using C++ with Kokkos [9]. In [7], Domino *et al.* parallelized this algorithm by dividing the input matrix vertically into several submatrices, each owning a subset of the rows of the input matrix. The sequential algorithm can then be applied to each of the submatrix to compute their moment tensors. Those partial moment tensors then have to be reduced and averaged. This approach is compatible with our new sequential algorithm as well. However, the final reduction creates a bottle neck because it is communication intensive and does not scaled well. Instead, we intend to implement our algorithm in a way such that we can compute the blocks of the moment tensors in parallel, which avoids communication.

Another extension of this work we are investigating is the ways to parallelize the computation of the cumulant tensor. The computing the moment tensors is a substantial portion of the work involved in computing the cumulant tensor. However, after the moment tensors are known, there is currently no good implementations that will compute the cumulant tensor in parallel efficiently.

7. Summary. In this work we present a more efficient way of computing the moment tensor that takes advantage of its symmetry at the same time. The complexity of this evaluated. We also documents the implementation of this algorithm in Julia and MATLAB and compared its performance with the benchmark. From the results, we see that using matrix operations that has higher efficiency to compute a block of the final results can result

in substantial increase in performance compared to more naive approach of computing each values individually. In addition, we discuss the ongoing effort to parallelize this algorithm and ways to expand the results we have now to more efficient computation of higher-order joint cumulant tensors as well.

References.

[1] K. Aditya, H. Kolla, W. P. Kegelmeyer, T. M. Shead, J. Ling, and W. L. Davis, *Anomaly detection in scientific data using joint statistical moments*, Journal of Computational Physics, 387 (2019), pp. 522–538.

[2] B. W. Bader, T. G. Kolda, et al., *Tensor Toolbox for MATLAB, Version 3.2.1*, April 5, 2021.

[3] J. F. Cardoso, *Source separation using higher order moments*, in International Conference on Acoustics, Speech, and Signal Processing, vol. 4, 1989, pp. 2109–2112.

[4] P. Comon, *Independent component analysis, a new concept?*, Signal Processing, 36 (1994), p. 287–314.

[5] P. Comon and J.-F. Cardoso, *Eigenvalue decomposition of cumulant tensor with applications*, in SPIE Conference on Advanced Signal Processing Algorithms, Architectures and Implementations, San Diego, United States, 1990, pp. 361–372.

[6] A. Delorme, T. Sejnowski, and S. Makeig, *Enhanced detection of artifacts in EEG data using higher-order statistics and independent component analysis*, NeuroImage, 34 (2007), pp. 1443 – 1449.

[7] K. Domino, P. Gawron, and L. Pawela, *Efficient Computation of Higher-Order Cumulant Tensors*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1590–A1610. Publisher: Society for Industrial and Applied Mathematics.

[8] M. Domino, K. Domino, and Z. Gajewski, *An application of higher order multivariate cumulants in modelling of myoelectrical activity of porcine uterus during early pregnancy*, Biosystems, 175 (2019), pp. 30 – 38.

[9] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[10] I. K. Fodor and C. Kamath, *Using independent component analysis to separate signals in climate data*, in Independent Component Analyses, Wavelets, and Neural Networks, A. J. Bell, M. V. Wickerhauser, and H. H. Szu, eds., vol. 5102, International Society for Optics and Photonics, SPIE, 2003, pp. 25 – 36.

[11] X. Geng, K. Sun, L. Ji, H. Tang, and Y. Zhao, *Joint skewness and its application in unsupervised band selection for small target detection*, Scientific Reports, 5 (2015).

[12] J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn, *Flame: Formal linear algebra methods environment*, ACM Trans. Math. Softw., 27 (2001), p. 422–455.

[13] P. Głomb, K. Domino, M. Romaszewski, and M. Cholewa, *Band selection with higher order multivariate cumulants for small target detection in hyperspectral images*. arXiv:1808.03513, 2018.

[14] H. A. L. Kiers, *Towards a standardized notation and terminology in multiway analysis*, Journal of Chemometrics, 14 (2000), pp. 105–122.

[15] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), p. 455–500.

[16] Y. Kutlu and D. Kuntalp, *Feature extraction for ECG heartbeats using higher order statistics of WPD coefficients*, Comput. Methods Prog. Biomed., 105 (2012), p. 257–267.

[17] M. A. Middleton, P. H. Whitfield, and D. M. Allen, *Independent component analysis of local-scale temporal variability in sediment-water interface temperature*, Water Resources Research, 51 (2015), pp. 9679–9695.

[18] D. Peña and F. J. Prieto, *Multivariate outlier detection and robust covariance matrix estimation*, Technometrics, 43 (2001), pp. 286–310.

[19] T. Reichler and J. Kim, *How well do coupled models simulate today's climate?*, Bulletin of the American Meteorological Society, 89 (2008), pp. 303–312.

[20] M. D. Schatz, T. M. Low, R. A. van de Geijn, and T. G. Kolda, *Exploiting Symmetry in Tensors for High Performance: Multiplication with Symmetric Tensors*, SIAM Journal on Scientific Computing, 36 (2014), pp. C453–C479.

[21] J. Wang and C.-I. Chang, *Independent component analysis-based dimensionality reduction with applications in hyperspectral image analysis*, IEEE Transactions on Geoscience and Remote Sensing, 44 (2006), pp. 1586–1600.

[22] S.-N. Yu and K.-T. Chou, *Integration of independent component analysis and neural networks for ECG beat classification*, Expert Systems with Applications, 34 (2008), pp. 2841 – 2846.

PMEMCPY: AN EFFICIENT I/O LIBRARY FOR PMEM

LUKE M. LOGAN*, GERALD F. LOFSTEAD†, SCOTT LEVY‡, PATRICK WIDENER§, XIAN-HE SUN¶, AND ANTHONY KOUGKAS‖

Abstract. Persistent memory (PMEM) devices can achieve comparable performance to DRAM while providing significantly more capacity. This has made the technology compelling as an expansion to main memory. Rethinking PMEM as storage devices can offer a high performance buffering layer for HPC applications to temporarily, but safely, store data. However, modern parallel I/O libraries, such as HDF5 and pNetCDF, are complicated and introduce significant software and metadata overheads when persisting data to these storage devices, wasting much of their potential. In this work, we explore the potential of PMEM as storage through pMEMCPY: a simple, lightweight, and portable I/O library for storing data in persistent memory. We demonstrate that our approach is up to 2x faster than other popular parallel I/O libraries under real workloads.

1. Introduction. Scientific applications generate massive amounts of data; however, storage performance lags behind CPU performance resulting in applications being bottlenecked by I/O both with other nodes as well as with storage. One approach to alleviate this problem is to expand the memory capacity of the nodes, enabling more local processing before requiring communication. PMEM (e.g., phase change memory and Intel DC Persistent Memory) offers an excellent solution that is cheaper than DRAM, but offers reasonably similar performance characteristics. This technology has driven considerable work into using main memory (i.e., DRAM) as a working cache for an expanded PMEM main memory [30]. As compelling as this case is, it only addresses the inter-node portion of the bottleneck. For applications where communication with storage is a more serious concern, using that same PMEM technology as fast storage (instead of slower memory) offers a flexible resource that can address multiple kinds of workloads. For example, various works investigate the use of storage hierarchies in order to combat the I/O bottleneck [5, 21, 34]. In these works, storage such as PMEM, NVMe, SSD, and HDD are arranged in a hierarchy based on performance and capacity characteristics. Data is initially buffered in faster storage tiers and then asynchronously flushed to slower mass storage, which helps avoid costly data stalls. While there has been considerable work examining the use of node-hosted storage technology with more favorable performance characteristics than hard drives, the interfaces for PMEM offer another potential performance gain, but only if the software uses the devices with these more efficient interfaces.

Due to the DRAM-like performance of PMEM, software overheads are no longer negligible on the I/O path. For this reason, researchers have started rethinking the design of node-local storage stacks [1, 19, 23, 40, 41], which had previously been designed for slow storage technologies such as hard drives. EXT4/XFS DAX [1] allows applications to directly store data in PMEM without first copying to DRAM using memory mapped I/O. SplitFS [19] aims to improve the performance of DAX by splitting metadata and storage operations between kernel space and user space respectively, allowing the majority of I/O operations to avoid the kernel entirely. NOVA [40] is a log-structured filesystem that aims to exploit the parallelism and random access properties of PMEM by storing logs per-inode as opposed to a global log. These works avoid many of the overheads introduced by the

*Illinois Institute of Technology, llogan@hawk.iit.edu
†Sandia National Laboratories, gflofst@sandia.gov
‡Sandia National Laboratories, sllevy@sandia.gov
§Sandia National Laboratories, pwidene@sandia.gov
¶Illinois Institute of Technology, sun@iit.edu
‖Illinois Institute of Technology, akougkas@iit.edu

Linux kernel, such as context switching, splitting/merging, lock contention, and request reordering. However, improving node-local storage stacks is not enough. HPC applications typically use parallel I/O (PIO) libraries on top of node-local storage stacks to persist data. We assert that fundamental changes in the design of PIO libraries must be made to gain the full benefits of PMEM for I/O.

Various PIO libraries exist, such as ADIOS [13, 29], HDF5 [22], and pNetCDF [24]. However, these libraries introduce significant programming burden, software overhead, and complex configuration spaces. In order to maximize the performance of these libraries and reduce the user's burden, researchers have investigated the use of auto-tuning to identify optimal parameters specific to the characteristics of applications and parallel filesystems [3, 4, 6, 7], with approaches such as genetic algorithms and Bayesian optimization. However, at a fundamental level, existing PIO libraries do not interact with PMEM efficiently, regardless of how well they are tuned. For example, all production-ready work depends on the use of MPI-IO and POSIX, which causes unnecessary networking communication and data copies that degrade the performance of I/O to PMEM [20]. Furthermore, PIO libraries tend to have complicated APIs, requiring many lines of code to store simple data structures, such as arrays. A simple *memcpy* interface is more desirable. PIO libraries should be designed with awareness of the underlying device characteristics in mind in addition to being more user-friendly.

In this work, we present pMEMCPY: a simple, lightweight, and portable I/O library for storing data in persistent memory. Using the Persistent Memory Development Kit (PMDK) [32], applications have direct access to PMEM while maintaining consistency guarantees. Users can store data structures with a simple key-value store interface that adds the minimal metadata necessary to deserialize the data structures in addition to avoiding costly network communications and data copies that other PIO libraries introduce.

Our contribution offers an optimized approach for parallel I/O library design that can store application data structures in node-local PMEM directly with minimal overhead using a simple key-value store interface similar to *memcpy*. Through this style of I/O library, users can achieve the best possible PMEM performance for their storage operations and enjoy an API much closer to memcpy.

The rest of this paper is organized as follows. First in Section 2 is a deeper discussion of background and related work. Next in Section 3 we detail the reasoning and design decisions for our demonstration. Section 4 presents a collection of evaluations comparing this approach against alternatives. Finally, in Section 5 we summarize the work.

2. Background & Related Work. There are various existing parallel I/O libraries, including HDF5, ADIOS, and pNetCDF. Furthermore, there are various libraries and APIs that exist to efficiently interact with PMEM. However, there has been no published approach, to our knowledge, that demonstrates how to optimize the I/O library for PMEM interfaces and simplify the API to a most basic memcpy-like approach.

2.1. Parallel I/O (PIO) Libraries. HDF5 [22] is a popular PIO library, and is used as the foundations for other popular PIO libraries, such as NetCDF4 [31]. HDF5 exposes a hierarchical namespace to users, where H5Groups are analogous to directories. HDF5 can store primitive types (ints, floats, doubles, etc.), compound data types (structures), and arrays (H5Datasets) of those types. Subsets of datasets can be taken using the Hyperslab APIs. HDF5 can store datasets using various data layout policies: contiguous, chunked, and compact. The contiguous layout stores arrays as a 1-D sequence of data, and is the default layout for HDF5. The chunked mode divides the array into fixed-size sub-arrays (i.e., chunks) where the dimensions of the sub-arrays are user-defined. In chunked mode, HDF5 also allows for the definition of filters, which are operations to perform on individual

chunks, such as compression [10, 11]. Lastly, if the dataset is less than 64KB, the compact mode stores the dataset in its corresponding metadata entry. In order to persist data to storage, HDF5 allows multiple approaches: MPI Independent I/O, MPI Collective I/O, and POSIX I/O. In each of these cases, the final output of HDF5 is a single binary file. Furthermore, HDF5 introduced a multi-tiered buffer management system, Hermes [21], that allows users to manage the complexity of heterogeneous, multi-tiered storage environments without changing application code. While HDF5 is a feature-rich library that has specific functionality for buffering and prefetching, it has many limitations. The Neuroscience community, for example, has noted multiple flaws in the user-friendliness of this library [12]. HDF5 stores data in a single binary file, where metadata is not human readable. This also makes version control systems less efficient. Furthermore, HDF5 compound types do not support the nesting of compound types or dynamically sized arrays. Furthermore, MPI-IO relies on the underlying filesystem (for Linux, read/write) APIs in order to store data. However, read/write perform data copies which introduces unnecessary overhead [20].

An alternative to HDF5 and NetCDF4 (a PIO library depending on HDF5) is pNetCDF [24]. This was developed around the same time as NetCDF4 as an effective demonstration on how to maintain the NetCDF3 compatibility as much as possible while extending for 64-bit support. While the two libraries "compete", the reality is that they co-exist peacefully and are widely supported as a pair rather than individually. For exmaple, the NCAR PIO library [9] offers a single API that can switch to use either NetCDF4 or pNetCDF underneath. As with HDF5, pNetCDF is designed with MPI-IO as the primary IO interface for parallel IO and optimized for slow storage devices through additional work to prepare data to more efficiently be moved into storage. However, the performance gains of PMEM shifts the bottleneck of the storage device that required such optimizations to the I/O library itself. NVMe devices have had a similar effect [2], but PMEM offers additional performance exaggerating the performance overhead the software layer imposes.

ADIOS is an alternative PIO library to HDF5, NetCDF and pNetCDF. ADIOS aims to encompass various I/O transport mechanisms (e.g., MPI-IO, POSIX, HDF5, and NetCDF) under a simplified interface that is easily configurable and requires little change to application code to change which implementation is used. ADIOS is designed to reduce the code complexity of HDF5 and acknowledge that some of the performance optimizations employed by HDF5 and other PIO libraries do not scale for reading and writing as well as hoped [28]. One approach ADIOS uses to address the performance gap is to use its own Binary Packed (BP) format whenever possible. BP offers delayed consistency, lightweight data characterization, and data resilience. Unlike HDF5, ADIOS stores data in the same format as it was produced on a process-by-process basis rather than constructing a global linearization of complex datastructures. For example, a 3D domain decomposition is stored as a single item in HDF5 with all three dimensions across all processes being linearized through a data rearrangement phase prior to hitting storage. This has the advantage of eliminating any potential artifacts from unusual process decompositions. ADIOS has each process write the data it owns with no coordination with other processes. This eliminates the data rearrangement phase, which can improving performance greatly. In particular, large 3D domain decompositions see radical performance improvements for both writing and reading [28]. ADIOS also supports transparent and custom operators, similar to HDF5. In addition, ADIOS2 [13] is an update to ADIOS that provides a C++ interface that is more simplistic and extensible than that of the original ADIOS. The recent revision includes a key-value store API for storing data. However, ADIOS2 suffers from the same drawbacks as the original when it comes to PMEM as it is storage device agnostic.

A more recent effort, Proactive Data Containers [33] offers a similar key-value store approach for data management. However, it is designed for hard drives and solid state drives,

offering no optimizations for PMEM. This approach still suffers from the same limitations as the other aformentioned approaches.

One system recognizing the need for a different interface to non-volatile memory is DStore [14]. However, DStore is intended as a way to store a log for an in-DRAM key-value store. Unlike other attempts to optimize key-value stores with PMEM, such as MongoDB-PMEM [17] and PMEM-RocksDB [39], DStore uses PMEM to store the logs rather than as the main store, offering greater performance while still offering predictable consistency.

In all cases, effectively using PMEM using efficient interfaces is a relatively new endeavour that popular HPC I/O libraries have yet to embrace. While some progress has been made in the scale-out space, the recent DStore paper demonstrates that a simple "switch to the PMDK interface" may not be the most efficient nor optimal approach for achieving both performance and price/performance.

2.2. Accessing PMEM. PMEM can be exposed like any other storage device. Application developers can use traditional filesystem APIs such as POSIX, stdlib, and iostream in order to store data in PMEM. However, these interfaces introduce significant software overheads. For example, these interfaces will cause unnecessary data copies and memory allocations to occur. To avoid this, applications can access PMEM directly using memory mapped I/O (MMIO) and DAX. However, managing memory-mapped regions requires application developers to provide their own memory allocation functions and concurrency control mechanisms, which can cause data consistency and reliability concerns.

The Persistent Memory Development Kit (PMDK) [32] is a collection of libraries and tools for managing PMEM devices. It provides low-level primitives for interacting with PMEM and a transactional object store that utilizes memory mapping in order to interface with PMEM devices. What this really means is that PMEM is mapped directly in the memory space for a process enabling direct access. Unlike MPI-IO and POSIX I/O, this approach allows applications direct, zero-copy access to PMEM while providing consistency guarantees. PMDK provides optimized memory allocation functions, persistent locks, basic data structures (e.g., thread-safe lists), and transactions. This allows applications to have efficient and safe access to PMEM while reducing the complexity of managing memory-mapped files.

2.3. New Filesystems. In the Introduction, we covered many of the newer generation storage systems written from the ground up to take advantage of solid state, node local storage. However, these have all been written for NVMe devices, at best, and still assume a more traditional device interface. One major exception to this is DAOS [16]. The original design of DAOS [27] was to offer a new storage architecture, but still assuming non-PMEM storage devices. The current DAOS generations have been reimagined using Intel Optane PMEM devices as a core component. Using these devices, DAOS was able to achieve top marks on the IO500 benchmark at sc19 [18]. More recent conversations with the DAOS team about Optane and DAOS or other storage use recommended at most 1% of the capacity using the PMEM devices as a way to ensure top performance for the most critical operations while keeping costs from spiraling out of control [26]. This makes DAOS a good potential candidate for using PMEM as a storage device, but it does not address the I/O library layer entirely. The plug-ins for HDF5 for speaking directly with DAOS and the DAOS native APIs may offer better support. However, the interfaces are still complex and focused on a container-like structure with POSIX-structures layered on top.

3. Design & Implementation. This work offers pMEMCPY, a simplistic and portable I/O library for managing node-local PMEM. Our design assumes that the compute nodes running the application also contain PMEM. Data structures in memory are stored

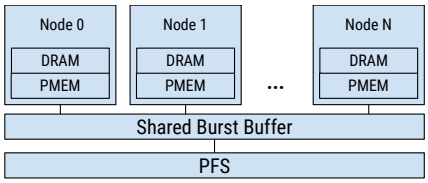


Fig. 3.1: Basic Machine Architecture

directly on PMEM without extra metadata, context switching, or data copies beyond what is necessary to reload the data during a different application run or for an analysis job. Our assumed, basic machine architecture is illustrated in Figure 3.1.

```
#include <pmemcpy/pmemcpy.hpp>
pmemcpy::PMEM pmem;
pmem.mmap(std::string filename, int comm);
pmem.munmap();

pmem.store<T>(std::string id, T &data);
pmem.alloc<T>(std::string id,
  int ndims, size_t *dims);
pmem.alloc<T>(std::string id,
  pmemcpy::Dimensions dims);
pmem.store<T>(std::string id, T *data,
  int ndims, size_t *offsets, size_t *dimspp);

pmem.load<T>(std::string id);
pmem.load<T>(std::string id, T &num);
pmem.load<T>(std::string id, T *data,
  int ndims, size_t *offsets, size_t *dimspp);
pmem.load_dims(std::string id,
  int *ndims, size_t *dim);
```

(a) pMEMCPY API

```
#include <pmemcpy/pmemcpy.h>
int main(int argc, char** argv) {
    int rank, nprocs;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    pmemcpy::PMEM pmem;
    size_t count = 100;
    size_t off = 100*rank;
    size_t dimsf = 100*nprocs;
    char *path = argv[1];

    double data[100] = {0};
    pmem.mmap(path, MPI_COM_WORLD);
    pmem.alloc<double>("A", 1, &dimsf);
    pmem.store<double>("A", data, 1, &off, &count);
    MPI_Finalize();
}
```

(b) pMEMCPY API Usage Example

Fig. 3.2: pMEMCPY API and an example of writing a 1-D array

API: pMEMCPY exposes a key-value interface for storing and loading data from PMEM. Users can store primitive types, structured types, and arrays of these types using the templated load/store APIs. The C++ API is shown in Figure 3.2(a). In Figure 3.2(b), we demonstrate the usage of pMEMCPY for writing a 1-D array of data in parallel. In the example, each process writes 100 doubles to non-overlapping offsets in the array directly to PMEM. *alloc* is used to specify the final dimensions of the array, and *store* is used to persist pieces of the array generated by each process. In Figure 3.3(a), we show the equivalent HDF5 code. HDF5 requires a user to create and free dataspace and dataset objects in addition to subsetting the dataset, and each of these interfaces contain many parameters. The dataspace defines the dimensions of the array, and the dataset represents the array within HDF5. The HDF5 version is 34 lines of code and 248 tokens, whereas our code is 16 lines and 132 tokens, which is a 47% reduction in the number of tokens. Similar to HDF5, NetCDF and pNetCDF requires users to define and allocate the dimensions of the array using special APIs, which adds unnecessary complexity. While ADIOS simplifies this, it still requires the user to store the dimensions of the array separately and then associate those variables with the array. pMEMCPY automatically stores the dimensions of the array and the per-process subarrays in the *store* API by appending "#dims" to the id; dimensions can be queried using *load_dims*. In Figure 3.3(b), we show the equivalent ADIOS code, which is 24 lines and 164 tokens. Overall, we see that pMEMCPY provides a more simplified and compact API than other libraries.

Data Transfer and Serialization: Unlike ADIOS, NetCDF, and pNetCDF which depend on POSIX and MPI-IO, pMEMCPY uses memory mapping and independent I/O

```c
#include <hdf5.h>
int main (int argc, char **argv) {
  int nprocs, rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  hid_t file_id, dset_id;
  hid_t filespace, memspace;
  hsize_t       count = 100;
  hsize_t offset = rank*100;
  hsize_t dimsf = nprocs*100;
  hid_t plist_id;
  char *path = argv[1];
  int data[100];

  plist_id = H5Pcreate(H5P_FILE_ACCESS);
  H5Pset_fapl_mpio(plist_id,
    MPI_COMM_WORLD, MPI_INFO_NULL);
  file_id = H5Fcreate(path,
    H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
  H5Pclose(plist_id);

  filespace = H5Screate_simple(1, &dimsf, NULL);
  dset_id = H5Dcreate(file_id, "dataset",
    H5T_NATIVE_INT, filespace, H5P_DEFAULT,
    H5P_DEFAULT, H5P_DEFAULT);
  H5Sclose(filespace);
  memspace = H5Screate_simple(1, &count, NULL);
  filespace = H5Dget_space(dset_id);
  H5Sselect_hyperslab(filespace,
    H5S_SELECT_SET, &offset,
    NULL, &count, NULL);

  plist_id = H5Pcreate(H5P_DATASET_XFER);
  H5Dwrite(dset_id, H5T_NATIVE_INT,
    memspace, filespace, plist_id, data);

  H5Dclose(dset_id);
  H5Sclose(filespace);
  H5Sclose(memspace);
  H5Pclose(plist_id);
  H5Fclose(file_id);
  MPI_Finalize();
  return 0;
}
```

(a) Equivalent HDF5 Example

```c
#include <adios.h>
int main(int argc, char **argv) {
    int rank, nprocs;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    char *path = argv[1];
    char *config = argv[2];
    double data[100];
    int64_t adios_handle;
    size_t count = 100;
    size_t offset = 100*rank;
    size_t dimsf = 100*nprocs;

    adios_init(config, MPI_COMM_WORLD);
    adios_open (&adios_handle, "dataset",
      path, "w", MPI_COMM_WORLD);
    adios_write (adios_handle, "count", &count);
    adios_write (adios_handle, "dimsf", &dimsf);
    adios_write (adios_handle, "offset", &offset);
    adios_write (adios_handle, "A", data);
    adios_close (adios_handle);
    adios_finalize (rank);
    MPI_Finalize ();
    return 0;
}
```

(b) Equivalent ADIOS Example.

Fig. 3.3: HDF5 and ADIOS example of writing a 1-D array

to store data in the node-local PMEM, which avoids unnecessary data copies, network/inter-process communications, and kernel interventions. When storing a data structure in PMEM, pMEMCPY serializes the data using well-known, portable serialization libraries, such as BP4 [13], CapnProto [36], and cereal [38]. By default, the BP4 serialization (same as ADIOS) is used; however, other serialization tools can be added, and serialization can be completely disabled. Unlike similar work which serializes data structures into an in-memory buffer and then copies to PMEM, pMEMCPY can serialize the data directly into PMEM without first placing it in DRAM, avoiding a significant data copying cost. Furthermore, we allow users to configure whether or not the MAP_SYNC flag is enabled when storing serialized data structures in a region of PMEM. The MAP_SYNC flag guarantees that, after a crash, a block that has been mapped into memory with write permissions will still be at the same offset within the file [8]. While this improves crash consistency, this can introduce significant latency penalties that severely degrade performance, as shown in our evaluations. This is because MAP_SYNC will cause a flurry of modified file metadata to be flushed on every I/O operation [8]. After serialization, a burst buffer, such as DataWarp [15], will then be triggered to asynchronously flush the buffered data to mass storage. The data will be stored in the same format as it was produced, similar to ADIOS, which avoids the network and inter-process communication required to restructure the data.

Data Layout: By default, pMEMCPY stores all application data in a single file similar to ADIOS, NetCDF, and pNetCDF. However, pMEMCPY uses the PMDK [32] to manage PMEM, which provides direct access to PMEM in addition to data consistency guarantees, concurrency control, and memory allocation policies. Metadata is stored in a flat namespace using a hashtable with chaining. This utilizes the high parallelism and random access char-

acteristics of PMEM. Alternatively, unlike ADIOS, NetCDF, and pNetCDF, pMEMCPY can layout data hierarchically using the PMEM's filesystem. In this approach, instead of writing to a single file, pMEMCPY stores the data structures in a directory and creates a file for each variable. Whenever a "/" is used in the id of the variable, a directory is created if it didn't already exist.

4. Evaluations. Testbed: All tests were conducted in Chameleon Cloud using a Compute Skylake node. Compute Skylake nodes come with 192GB of RAM and 2x Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz, for a total of 24 cores/48 threads. The OS used was Ubuntu 20.04 with kernel 5.4.0-70-generic. We used openmpi 3.1.6.

Emulating PMEM: Since we do not have access to PMEM, we emulate it using the approach presented in the Strata PMEM filesystem paper [23]. We utilize Linux's PMEM emulator to treat 80GB of DRAM as PMEM and format the resulting PMEM device using EXT4 with DAX enabled. We assume that PMEM has a read latency of 300ns, write latency of 125ns, read bandwidth of 30GB/s, and write bandwidth of 8GB/s [35]. We benchmarked DRAM bandwidth and latency using Intel's Memory Latency Checker (MLC) [37] and use nanosecond-accurate monotonic timers to add the additional latency and bandwidth constraints.

4.1. Real-App Evaluation. In this test, we demonstrate the performance impact of pMEMCPY over other popular PIO libraries using real workloads. In this evaluation, we use two workloads that were obtained through the help of scientists [28]. The first workload is a write-only 3-D domain decomposition problem where each process writes a rectangular region of data to storage. The second workload is a read-only workload that reads the regions from storage. For both tests, we use between 8 and 48 processes. This model represents a large memory regular stencil code common in compute models today. One example is the S3D combustion code [25] that was the inspiration for this configuration. This model has been previously used [28] to demonstrate potential I/O performance. In the write-only case, we generate 10 3-D rectangles. For each test, a total of 40GB of data is generated and the 40GB is divided equally among the processes. Each element in the rectangle is a double precision floating point value (8 bytes). The read workload is completely symmetrical to the write workload, where each process reads the same data that had been written. We measure the wall-clock time from the point at which the file is opened/mmapped to when the it is closed. We perform the I/O using ADIOS, NetCDF-4, pNetCDF, and pMEMCPY and compare the runtime between the different approaches. For NetCDF-4, we make sure to call *nc_def_var_fill()* with NC_NOFILL in order to prevent it from initializing variables with a default value, which causes significant overhead for write workloads. For pMEMCPY, we use BP4 serialization with the PMDK hashtable layout. We run each experiment 3 times and take the average of the runs.

The results of the experiment are shown in Figures 4.1 and 4.2. From these figures, we see the effects of concurrency due to the CPU and PMEM wear off after 24 cores in the write case and for most of the reads, with the exception of PMCPY-B and NetCDF4. This makes sense considering the node has 24 physical CPU cores in total. For NetCDF, the performance differences were largely due to differences in the dimensions of the cube being read for the different process counts. For PMCPY-B, this was because the metadata updates were parallelized, which caused fewer stalls. Overall, we see that pMEMCPY outperforms ADIOS, NetCDF, and pNetCDF in both workloads when MAP_SYNC is disabled. This is because pMEMCPY avoids unnecessary communications and data copies that other PIO libraries introduce. In the case of writes, all other PIO libraries first generate the cube in DRAM, serialize the cube into another DRAM buffer, and then copy the serialized cube to the PMEM whereas pMEMCPY generates the cube in DRAM and then serializes the

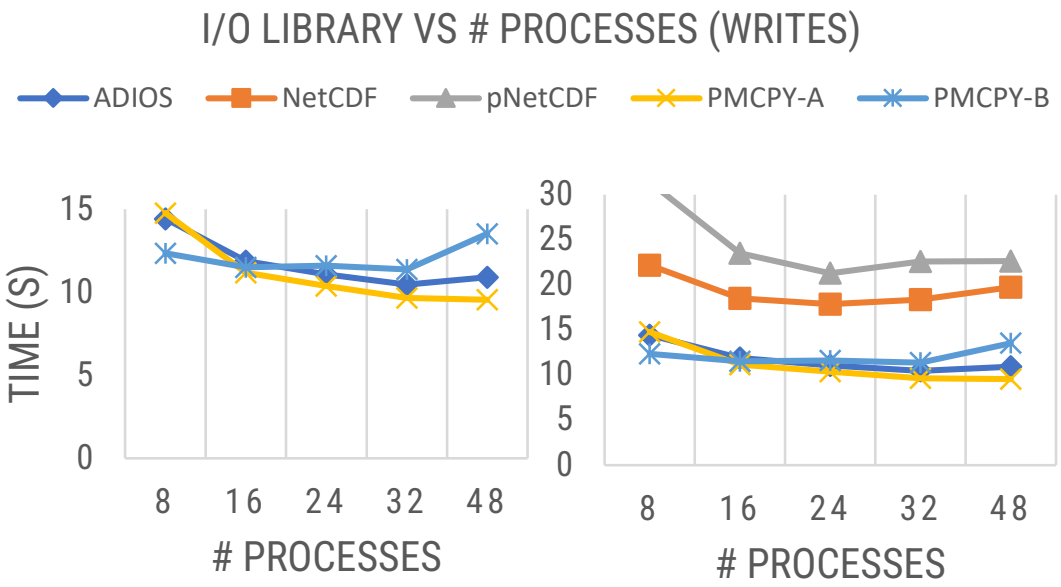


Fig. 4.1: Performance of writing a 40GB 3-D domain to PMEM for a varying number of processes. PMCPY-A has MAP_SYNC disabled, whereas PMCPY-B has it enabled. Each process writes an equal amount of data. pMEMCPY is 2.5x faster than pNetCDF and NetCDF by avoiding network communications and data copying costs. At 24 cores, pMEMCPY is faster than ADIOS by 15% when MAP_SYNC is disabled, and slightly slower when MAP_SYNC is enabled. Note, the left figure is a zoomed in version of the right figure.

cube directly into the PMEM, avoiding an entire copy of the cube. From these figures, we see ADIOS performs far better than NetCDF and pNetCDF in both read and write performance. This is because, similar to pMEMCPY, ADIOS stores data in the same format as it was produced, which avoids costly network communications and data copies during the write phase. Furthermore, since the workload is symmetrical, ADIOS does not need to realign any data, which mitigates data shuffling costs in the read phase. However, pNetCDF and NetCDF store data contiguously, which requires data to be shuffled during both reads and writes, incurring significant overhead. While ADIOS performs much better than pNetCDF and NetCDF, it still introduces data copying overheads that pMEMCPY avoids, causing its performance to be suboptimal. For example, in the case of reads, ADIOS requires the serialized data to be copied from PMEM into DRAM and then deserialized into another DRAM buffer. pMEMCPY deserializes the data directly from PMEM, avoiding the initial copy from PMEM to DRAM. Within pMEMCPY, we see that the choice of flags has a significant impact on performance. When MAP_SYNC is enabled, the performance benefit of serializing/deserializing directly from PMEM is completely lost, and can even cause performance to be worse than simply using POSIX read()/write(). This is because MAP_SYNC causes a flurry of modified file metadata to be flushed on every I/O operation, introducing significant latency penalties. Overall, we see that pMEMCPY can perform at least 15% better for writes and 2x better for reads depending on the level of safety the user requires.

4.2. Discussion. While standard I/O libraries offer a familiar interface, that can come at a cost. ADIOS, with the design break from the previous generation demonstrates better

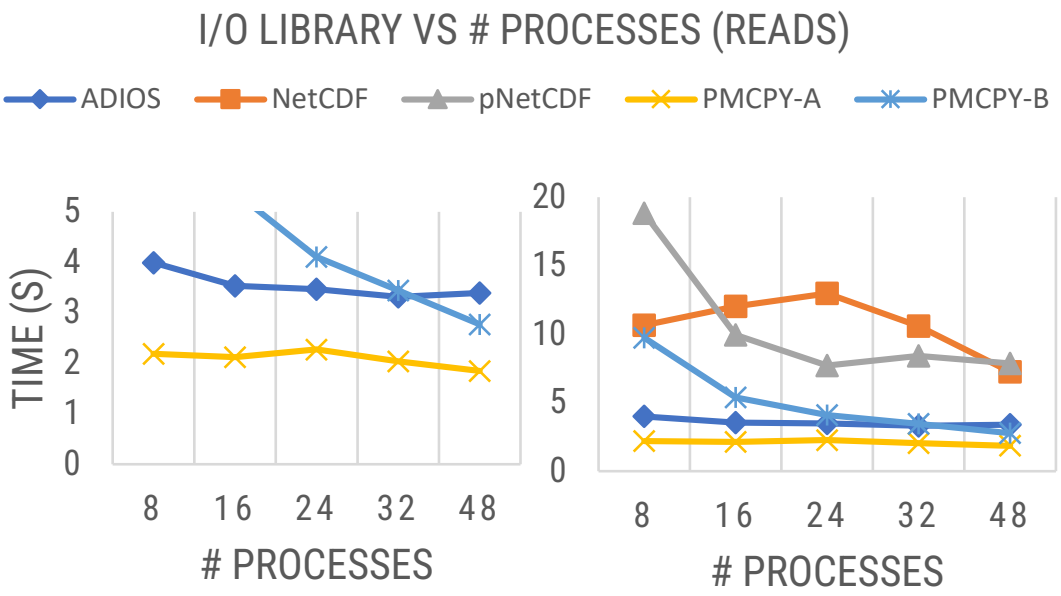


Fig. 4.2: Performance of reading a 40GB 3-D domain from PMEM for a varying number of processes. PMCPY-A has MAP_SYNC disabled, whereas PMCPY-B has it enabled. Each process reads an equal amount of data. pMEMCPY is 5x faster than pNetCDF and NetCDF by avoiding network communications and data copying costs. pMEMCPY is 2x faster than ADIOS whe [8]n MAP_SYNC is disabled. When enabled, pMEMCPY performs no better than ADIOS. Note, the left figure is a zoomed in version of the right figure.

performance, but is still not optimal by a margin of 15% - 100%. Only by using an approach such as the one we demonstrate in pMEMCPY can the full potential of PMEM as a storage device be achieved.

5. Conclusions. Persistent memory (PMEM) is an extraordinarily fast persistent storage device typically thought of as an extension of DRAM main memory. However, using PMEM for storage requires revisiting the design of parallel I/O (PIO) libraries. With PMEM being integrated into compute nodes, PIO libraries should take full advantage of the characteristics of these devices. However, popular libraries, such as HDF5, ADIOS, and pNetCDF, introduce significant overheads when applications store and load data. Furthermore, they introduce complex interfaces and parameters that add unnecessary burden on programmers. In this paper, we introduced pMEMCPY: a simple, lightweight, and portable I/O library for storing data in persistent memory. We compared our design with ADIOS, NetCDF-4, and pNetCDF, and found that write speeds improved at least 15% and reads improved up to 2x.

References.

[1] *Direct access for files*, 2014. https://www.kernel.org/doc/Documentation/filesystems/dax.txt.

[2] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, *File systems unfit as distributed storage backends: lessons from 10 years of ceph evolution*, in Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 353–369.

[3] B. Behzad, S. Byna, and M. Snir, *Optimizing i/o performance of hpc applications with autotuning*, ACM Transactions on Parallel Computing (TOPC), 5 (2019), pp. 1–27.

[4] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, M. Snir, et al., *Taming parallel i/o complexity with auto-tuning*, in SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2013, pp. 1–12.

[5] S. Byna, Q. Koziol, V. Vishwanath, J. Soumagne, H. Tang, J. Mu, B. Dong, R. A. Warren, F. Tessier, T. Wang, et al., *Proactive data containers (pdc): An object-centric data store for large-scale computing systems*, in AGU Fall Meeting Abstracts, vol. 2018, 2018, pp. IN34B–09.

[6] Z. Cao, *A Practical, Real-Time Auto-Tuning Framework for Storage Systems*, PhD thesis, State University of New York at Stony Brook, 2019.

[7] Z. Cao, V. Tarasov, S. Tiwari, and E. Zadok, *Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems*, in 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), 2018, pp. 893–907.

[8] J. Corbet, *Two more approaches to persistent-memory writes*, 2017. https://lwn.net/Articles/731706/.

[9] J. M. Dennis, J. Edwards, R. Loy, R. Jacob, A. A. Mirin, A. P. Craig, and M. Vertenstein, *An application-level parallel i/o library for earth system models*, The International Journal of High Performance Computing Applications, 26 (2012), pp. 43–53.

[10] H. Devarajan, A. Kougkas, L. Logan, and X.-H. Sun, *Hcompress: Hierarchical data compression for multi-tiered storage environments*, in 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2020, pp. 557–566.

[11] H. Devarajan, A. Kougkas, and X.-H. Sun, *An intelligent, adaptive, and flexible data compression framework*, in 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), IEEE, 2019, pp. 82–91.

[12] S.-A. Dragly, M. Hobbi Mobarhan, M. E. Lepperød, S. Tennøe, M. Fyhn, T. Hafting, and A. Malthe-Sørenssen, *Experimental directory structure (exdir): An alternative to hdf5 without introducing a new file format*, Frontiers in neuroinformatics, 12 (2018), p. 16.

[13] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, et al., *Adios 2: The adaptable input output system. a framework for high-performance data management*, SoftwareX, 12 (2020), p. 100561.

[14] S. Gugnani and X. Lu, *Dstore: A fast, tailless, and quiescent-free object store for pmem*, in Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '21, New York, NY, USA, 2020, Association for Computing Machinery, p. 31–43. https://doi.org/10.1145/3431379.3460649.

[15] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, and N. J. Wright, *Architecture and design of cray datawarp*, Cray User Group CUG, (2016).

[16] Intel, *Daos: Revolutionizing high-performance storage with intel optane technology*. https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/high-performance-storage-brief.pdf.

[17] ———, *Mongodb persistent memory storage engine*. Github, July 2021. https://github.com/pmem/pmse.

[18] io500, *10 node challenge, i0500-sc19*, November. https://www.vi4io.org/io500/list/19-11/10node?fields=information__system,information__institution,information__storage_vendor,information__filesystem_type,information__client_nodes,information__client_total_procs,io500__score,io500__bw,io500__md,information__data,information__list_id&equation=&sort_asc=false&sort_by=io500__score&radarmax=6&query=.

[19] R. Kadekodi, S. K. Lee, S. Kashyap, T. Kim, A. Kolli, and V. Chidambaram, *Splitfs: Reducing software overhead in file systems for persistent memory*, in Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 494–508.

[20] J. Kim, Y. J. Soh, J. Izraelevitz, J. Zhao, and S. Swanson, *Subzero: zero-copy io for persistent main memory file systems*, in Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems, 2020, pp. 1–8.

[21] A. Kougkas, H. Devarajan, and X.-H. Sun, *Hermes: a heterogeneous-aware multi-tiered distributed i/o buffering system*, in Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, 2018, pp. 219–230.

[22] Q. Koziol, D. Robinson, et al., *Hdf5*, tech. rep., Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2018.

[23] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson, *Strata: A cross media file system*, in Proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 460–477.

[24] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, *Parallel netcdf: A high-performance scientific i/o interface*, in SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, IEEE, 2003, pp. 39–39.

[25] D. Lignell, C. Yoo, J. Chen, R. Sankaran, and M. Fahey, *S3d: Petascale combustion science,*

performance, and optimization, in Proceedings of the Cray Scaling Workshop, Oak Ridge National Laboratory, TN, 2007.

[26] J. Lofstead, *Pmem for storage conversation in io500 general slack channel*. Slack, February 2021. https://io500workspace.slack.com/archives/C01BMTNT56K/p1612291357026500.

[27] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, and E. Barton, *Daos and friends: a proposal for an exascale storage system*, in SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2016, pp. 585–596.

[28] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, *Six degrees of scientific data: Reading patterns for extreme scale science io*, in Proceedings of the 20th international symposium on High performance distributed computing, 2011, pp. 49–60.

[29] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, *Flexible io and integration for scientific codes through the adaptable io system (adios)*, in Proceedings of the 6th international workshop on Challenges of large applications in distributed environments, 2008, pp. 15–24.

[30] J. Ren, J. Luo, I. Peng, K. Wu, and D. Li, *Optimizing large-scale plasma simulations on persistent memory-based heterogeneous memory with effective data placement across memory hierarchy*, in Proceedings of the ACM International Conference on Supercomputing, 2021, pp. 203–214.

[31] R. Rew and G. Davis, *Netcdf: an interface for scientific data access*, IEEE computer graphics and applications, 10 (1990), pp. 76–82.

[32] S. Scargall, *Pmdk internals: Important algorithms and data structures*, in Programming Persistent Memory, Springer, 2020, pp. 313–331.

[33] J. Soumagne, R. Warren, J. Mu, V. Vishwanath, F. Tessier, S. Byna, Q. Koziol, H. Tang, T. Wang, B. Dong, and J. Liu, *Final technical report - proactive data containers for scientific storage*, (2019). https://www.osti.gov/biblio/1577855.

[34] K. Tang, P. Huang, X. He, T. Lu, S. S. Vazhkudai, and D. Tiwari, *Toward managing hpc burst buffers effectively: Draining strategy to regulate bursty i/o behavior*, in 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2017, pp. 87–98.

[35] A. van Renen, L. Vogel, V. Leis, T. Neumann, and A. Kemper, *Persistent memory i/o primitives*, in Proceedings of the 15th International Workshop on Data Management on New Hardware, 2019, pp. 1–7.

[36] K. Varda, *Capn'n proto cerealization protocol*, 2013. https://capnproto.org/.

[37] T. W. P. L. B. F. S. S. Vish Viswanathan, Karthik Kumar, *Memory latency checker*, 2015. https://software.intel.com/content/www/us/en/develop/articles/intelr-memory-latency-checker.html.

[38] R. Voorhies, *cereal - a c++11 library for serialization*, 2014. https://uscilab.github.io/cereal/.

[39] J. Xu, J. Kim, A. Memaripour, and S. Swanson, *Finding and fixing performance pathologies in persistent memory software stacks*, in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 427–439.

[40] J. Xu and S. Swanson, {*NOVA*}*: A log-structured file system for hybrid volatile/non-volatile main memories*, in 14th {USENIX} Conference on File and Storage Technologies (FAST'16), 2016, pp. 323–338.

[41] S. Zheng, M. Hoseinzadeh, and S. Swanson, *Ziggurat: a tiered file system for non-volatile main memories and disks*, in 17th {USENIX} Conference on File and Storage Technologies (FAST'19), 2019, pp. 207–219.

GEO-SPATIAL VISUALISATIONS

MAXWELL R. LOW* AND ANDREW T. WILSON†

Abstract. Geo-spatial analysis studies geographic landmarks for inventorying, locating instances, discovering patterns and predicting behaviors. Visualizations of geographic data provide readily understandable depictions to communicate locations and patterns of motion. This report focuses on visualizations of current trajectory analysis techniques and showcases the capabilities of Dash for applications in a query by example search engine for geo-spatial intelligence exploitation. The trajectory analysis summaries the Tracktable team's current techniques for feature generation and presents visualizations of trajectories clustered by collections of features. The visualizations of Dash's capabilities affirm that it can be used in the larger geo-intelligence project and provides a tutorial for other team members. These presented examples are building blocks of larger projects and provide a snapshot of the current progress.

1. Introduction. The following introduces two distinct geo-spatial visualization concepts that were developed this summer.

Trajectory data gathered by the OpenSky Network on December 1st of 2016 contains examples of anomalous trajectories that exemplified the available anomaly detection and clustering methods developed by the Tracktable team. Tracktable is an open source project developed by Sandia National Labs for handling trajectory analysis and visualization. [6] The following examples were created using built in methods for density based clustering (tracktable.analysis.dbscan), computing distance geometry (tracktable.analysis.distance_geometry), and calculating boxiness (boxiness), along with methods to read in and create trajectories (tracktable.analysis.assemble_trajectories), display trajectories (tracktable.render.render_trajectories), and various geo-spatial math methods (tracktable.core.geomath). These are used in the creation of other features and conversions to account for the increased distance between two points, due to the earth's curvature.

While all of the features can be analyzed together many of the more interesting examples come from pairs of features. The feature list in 2.1 and parings is not complete, but it provides visualizations of Tracktable's applications to be released with a future update.

The second focus of the project was to establish the building blocks of a visualization tool set for Machine Assisted Geo-spatial Intelligence Exploitation (MAGE). The visualization tools are for a query by example initiative, and will allow the user to interact with and edit graph elements to specify structures, fields, roadways, and other landmarks. Query by example (QBE) is a machine learning task in which the user provides examples from which search parameters are derived. Then the user evaluates individual examples of search results to fine tune the search parameters. The final displayed results are based on the learned parameters improved by having a human in the loop.

Spatial-Query-by-Sketch have been explored previously in related contexts. [1] [2] [5] A user enters a desired shape, by drawing it, to pull it from a geographic database. A user can also specify desired attributes of the shapes inputted to further refine the search. The user is then presented with results that then best match the defined sketches and characteristics.

For example, in the geo-spatial domain, the user may input a series of high schools from which the QBE search determines the results should contain a 2 story building with a moderate parking lot and football field. The user then evaluates search results confirming high schools and canceling examples of colleges, middle schools, or even parks and shopping centers. After many iterations of fine turning the search parameters of the QBE search engine, the results can be evaluated to assess the QBE search engine's accuracy. My specific

*Purdue University Aeronautical and Astronautical Engineering, low11@purdue.edu
†Sandia National Laboratories, atwilso@sandia.gov

contribution is to the visualization tools that will eventually allow for user selection of and editing of graph elements.

2. Examples of Anomalous Trajectory Detection. The trajectory data is gathered from the OpenSky Network, a collection of individuals with Automatic Dependent Surveillance Broadcast(ADS-B) receivers that track equipped aircrafts in range. The receivers capture ADS-B which contains an aircraft's id, the time the broadcast was sent and the position in latitude and longitude of the aircraft. The broadcast may also contain additional information like the aircraft's velocity, altitude, and heading. Using the Python package, Tracktable, the trajectories are read in using the assemble_trajectories method. This matches points together by their ID and sorts them by their timestamp. Trajectories are then filtered based on how straight they are using the total distance and end to end distance ratio, which is equivalent to a depth level of one for distance geometry. Flights with a ratio of 0.97 or greater are then removed. These are flights that are primarily straight and not typically very interesting for study. From there, different combinations of features, derived from the properties of each trajectory are clustered using the DBSCAN module. The remainder of this section will evaluate technique and provide case studies of clustering by different combinations of features.

2.1. All Features. With the addition of more features, each cluster tends to become more exact. The examples use a variety of features, listed below.

- Total Distance
- End to End Distance
- Start Point Latitude (Lat)
- Start Point Longitude (Long)
- End Point Lat
- End Point Long
- Median Velocity
- Distance Geometry
  - Measure of curvature via ratio of end to end distance to total distance
- Convex Hull Centroid Lat
  - A convex hull is a polygon formed by connecting the outer most points
- Convex Hull Centroid Long
- Convex Hull Area
- Convex Hull Aspect Ratio
  - Measure of shape via ratio of the shortest and longest polygon axes
- Convex Hull Perimeter
- Total Turn Angle
- Total Turn Winding
- Boxiness

In using all 19 features, the clusters tend to be nearly identical flights with only slight discrepancies. In figure 2.1, 12 distinct clusters are shown with near identical patterns in each cluster.

2.2. Distance Geometry. Distance geometry is a way of creating a feature vector that represents a trajectory's geometry independent of rigid transformations. It is used to cluster trajectories according to their shape. [3] Distance geometry is the ratio of the end to end distance to the total distance over a segment of a trajectory. It creates values in the range of (0,1]. The depth level of distance geometry represents the amount equal segments a trajectory is broken up into before the ratio is determined over each segment. When a depth level is computed it also contains all the prior depth levels as well. A depth level of 4

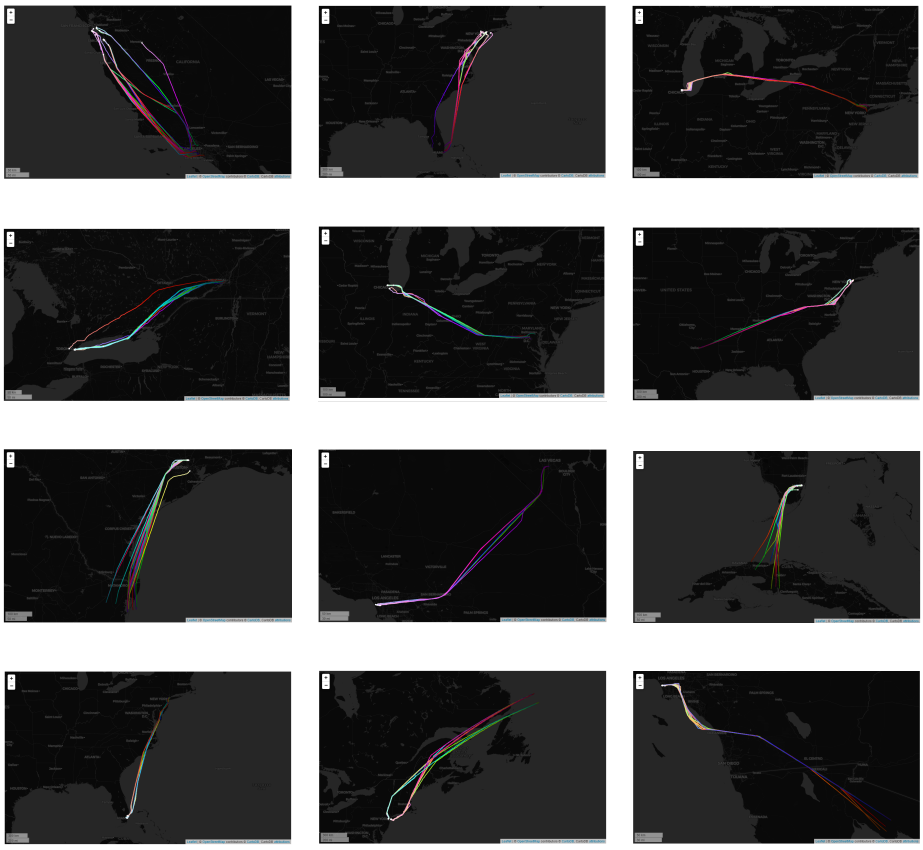


Fig. 2.1: Depictions of Clusters 1-12 of ADS-B Gathered Flight Trajectories Clustered by All 19 Features

contains all four ratios of the trajectory being split into four equal pieces and also contains the three ratios from splitting the trajectory into three equal parts and so on for a total of ten ratios at depth level 4. The following figures are using clustered results from a depth level of 4.

In figure 2.2 the trajectories are brought together by having ratios close to 0 for a depth of 1 and the 2nd segment of the depth of 3. For all the other measurements and depths the ratios are close to 1. One means that all of the travel distance is used in getting to the destination so the flight is straight. Zero means that the flight didn't travel anywhere overall; it ended where it started. None of the traveled distance went towards reaching the destination, it was already there.

2.3. Convex Hull Area and Perimeter. A convex hull is formed by connecting all the points on the periphery of a given set. Unlike simple polygons it doesn't necessarily contain all the points, as any internal points are ignored. Consider a convex hull as the results of stretching a rubber band around the trajectories points. Metrics describing the convex hull can be useful in describing geometry about a set of points in an efficient manner. The following considers clustering trajectories grouped by the area and the perimeter of the convex hull polygon. The selected clusters in figure 2.3 and in figure 2.4 show a

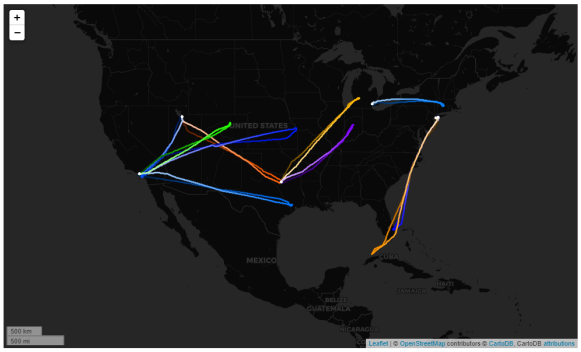


Fig. 2.2: Distance Geometry Clustered Trajectories: Low Area Loops

large amount of encircled area with minimal and maximal perimeters which leads to the trajectories having a triangular elbow shape and others that complete the entire loop.

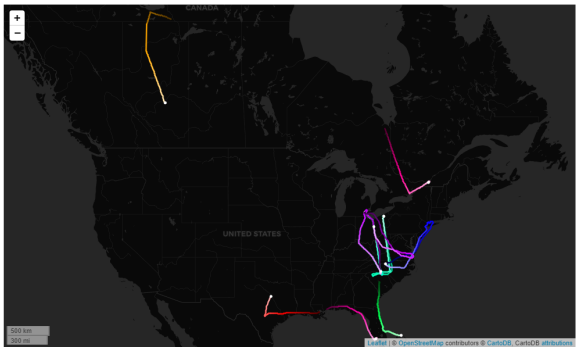


Fig. 2.3: Convex Hull Clustered Trajectories: High Encircled Area, Low Perimeter Distance

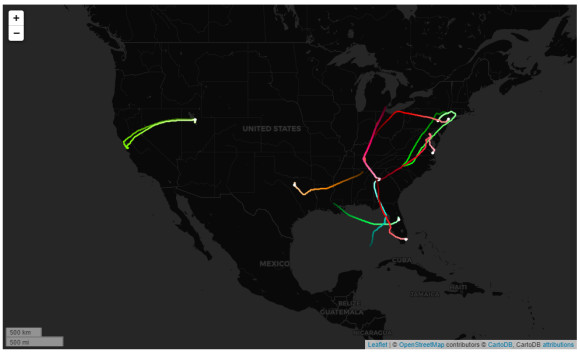


Fig. 2.4: Convex Hull Clustered Trajectories: High Encircled Area, High Perimeter Distance

2.4. Start and End Point Clustering. Figure 2.5 shows the outlier group from clustering by start and end point. All of the flights here do not have other flights that have

flow similar routes, or at least not 5 other flights to define a cluster.

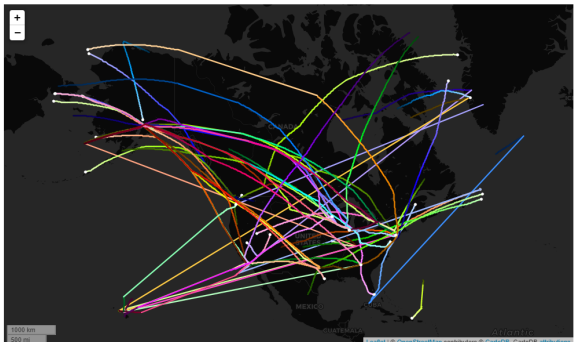


Fig. 2.5: Outlier Cluster from Trajectories Clustered their by Start and End Points

2.5. Oddities. From exploring the various combinations of features, each experiment produced several hundred outliers. Below are some of the most visually unusual that stood out and a brief explanation as to potentially why.

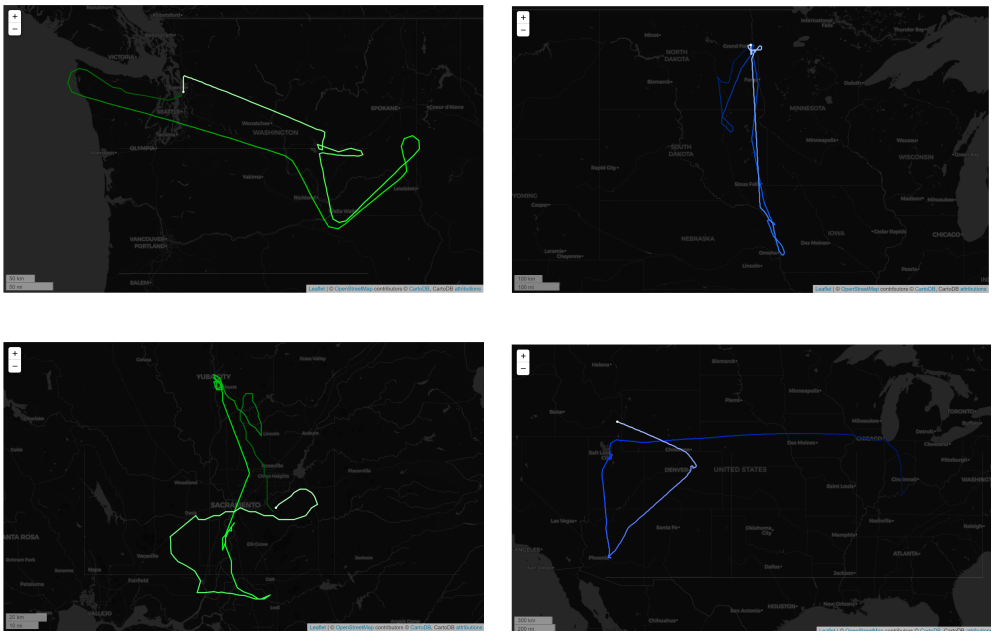


Fig. 2.6: Trajectory Anomalies: Upper Left - BOE378, Upper Right - N330PE, Lower Left - N877BR, Lower Right - SWA872

Flights BOE378 and N877BR were pulled out for their complexity of shape. The flights may be scenic flights or avoiding particular regions of air space, which may account for the trajectories' complexity of shape. Flight N330PE stood out for its extended travel distance while returning to the same location. Flight SWA872 stood out as an outlier by having a high boxiness score. Boxiness is a measure of how much of a trajectory is made up of equal

length sides separated by 90 degree angles. A histogram of the trajectory's heading at each point is created. The expected result for a square trajectory would have 4 spikes, one for each of the trajectory's headings, each of roughly equal counts. Each spike would contain one of the four sides of the square. The boxiness score is a ratio representing how well a trajectory aligns with the profile of a square trajectory's histogram.

3. Query by Example Visualization Tools. The following examples of QBE visualization tools were created using Dash. Dash is a Python package which provides the developer with the ability to create interactive web apps without needing to know the intricacies of HTML coding and hosting apps. [4] The descriptions partially serve as a work summary but also as a tutorial to Dash. The progression of my work started with generating a geographic map, adding static mapping elements, and adding editable map elements. The project will continue in the future and potential next steps are explored in the conclusion.

3.1. Generation of Geographic Map. The map is generated via the scttermapbox module. The background geographic terrain of the map is added to a graph plot. Changing the style argument inside of mapbox will create the background. The following examples use stamen-terrain as a background. The map's initial center and zoom is set but scattermapbox passively allows for users to zoom and pan the map image.

3.2. Adding Static Map Elements. To add static elements, edits are made to the scattermapbox arguments. This is most easily done via update_layout which can reformat and add layers to an existing map or can be specified with scattermapbox when the map is originally created.

3.2.1. Point. Points are specified by specifying marker as the mode and proving two lists, one of longitude and one of latitude for the lon lat arguments. The markers can also be edited via providing a dictionary to the marker argument. The markers have a default of black dots of size 20. Figure 3.1 shows a singular static dot near Albuquerque, NM on a geographic map. Figure 3.2 builds upon the singular point example and displays 4 dots in Southeastern Canada and the Northeastern United States.

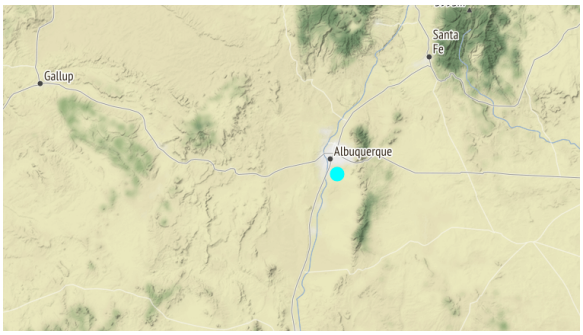


Fig. 3.1: Geographic Map Rendering with Single Static Point

3.2.2. Simple Polygon. Polygons are specified by providing a new map layer to mapbox containing a FeatureCollection with a Multipolygon feature containing a list of coordinates. Figure 3.3 provides an example of 3 static map points creating a shaded area in the Northeast United States.

To create additional polygons, specify an additional feature under features for the FeatureCollection. Figure 3.4 gives an example of two figures created on the same map. The

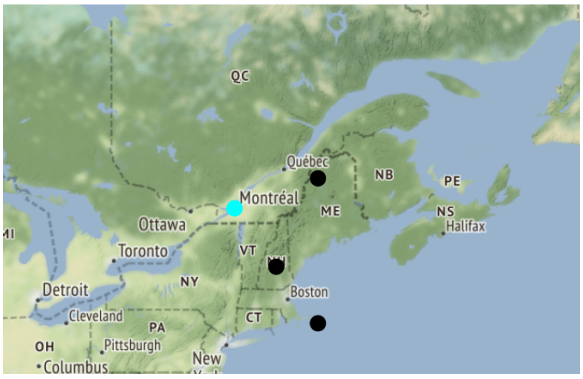


Fig. 3.2: Geographic Map Rendering with Multiple Static Points

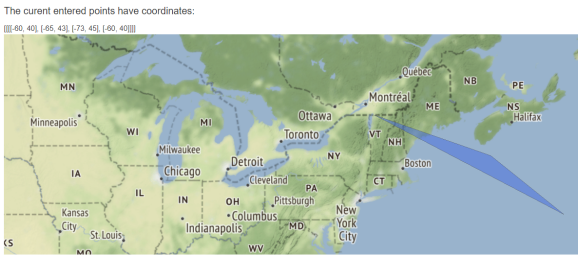


Fig. 3.3: Geographic Map Rendering with Single Static Area

concept is expandable to any numbers of figures.

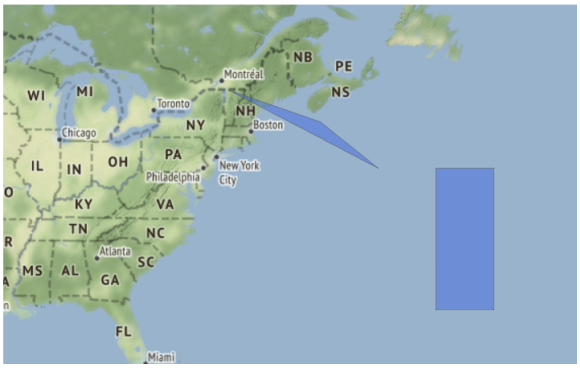


Fig. 3.4: Geographic Map Rendering with Multiple Static Areas

3.3. Making Dynamic Elements. Graph elements can be changed in real time by the user operating the app if callbacks are set up. A callback is a special function within a Dash app that allows for the changing of values in real time and updates the displayed elements based on user input. A callback has 3 potential fields that can be specified, an input, an output, and an optional state. Inputs reference interactable app elements which

will trigger the app to update when their corresponding value or other component property changes. An inputs element might be a button, with a value fo how many times it's been clicked. A state is similar to an input in that is stores a value provided to the function under a particular callback. Changing a state variable, does not update the app. This is useful for when the user inputs a series of inputs but submits all of them at the end with a button or other input element. Outputs are the elements or variables that changes when an input is changed. While there can be multiple callbacks, no two callback can update the same outputs. Inputs and states can be reused. The output of one call back can even be the input of the next allowing callbacks to be chained together.

The callback here allows the user to enter the latitude and longitude of a point and then when the update button is clicked the graphic updates. The first figure (Figure 3.5) shows the use of state variables, lat and long, and the apps visual condition prior to update being clicked. The use of state variables in the callback to prevent immediate updating. The second figure shows the results of updating the app. (Figure 3.6)

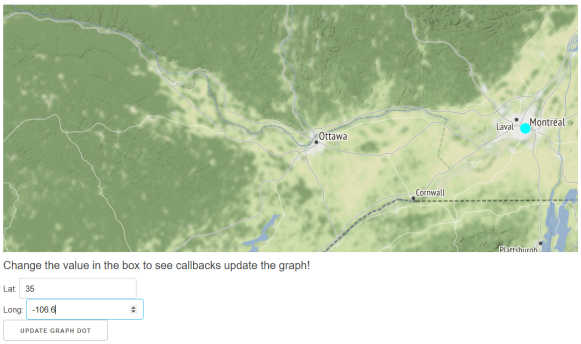


Fig. 3.5: Editable Point, Before Movement

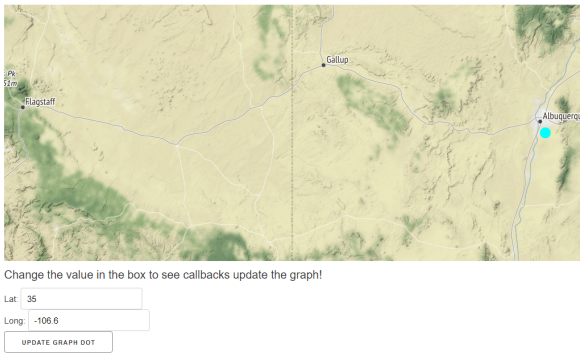


Fig. 3.6: Editable Point, Post Movement

Adding and subtracting points involves keeping a running list of points between callback cycles that is updated every time a point is added or subtracted. The map is also recreated when the points change. The new points list and graphic are then rendered. Figure 3.7 show this application with an editable series of points.

A similar application of keeping track of a points list enables an editable shape. The

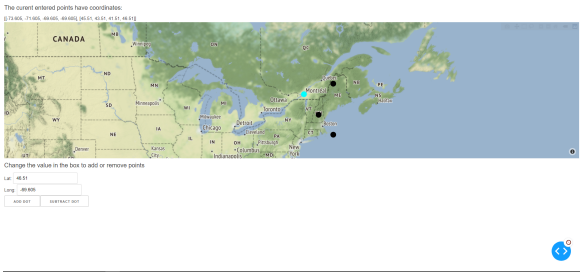


Fig. 3.7: Editable List of Points and Multi-points rendering

first image (Figure 3.8) shows the shape with a new coordinate ready to be added. The second image (Figure 3.9) is of the shape is after the 4th point is added.

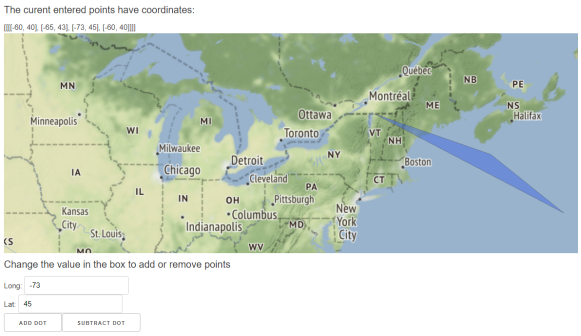


Fig. 3.8: Editable List of Points and Single Area Rendering, Before Point Addition

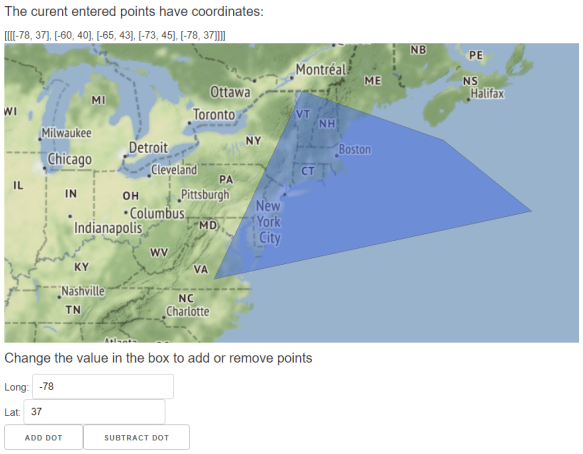


Fig. 3.9: Editable List of Points and Single Area Rendering, Post Point Addition

4. Conclusions. Both geo-spatial visualization projects are continuing, and there are several improvements that can be made to each.

For anomalous trajectories, more rigorous data filtering can be implemented. As a part of determining the features for each trajectory, histograms of the distribution of values for each feature were also created. For every feature the top and bottom 5% of trajectories for each feature could be extracted and identified as outliers as appropriate, in addition to clustering the main bulk of the trajectories. As a precaution the data being assembled into trajectories should also be verified that it is within reasonable ranges. Finally the survey of anomalous trajectories can be expanded to look as additional feature pairs and combinations.

For the QBE visualization tools, there are several next steps. Next, the project involve creating multiple dynamic shape where the user can select a shape by its ID and then edit point by point. This could then be integrated with the dots examples so that multiple user selected dots and shapes are available to the user in one app. Another set of improvements could be to the user experience. A lasso tool can be implemented to make interacting and selecting shapes and points easier. It would also be beneficial to provide points and shapes with the pop-up display showing various properties about the shape/dot on hover with the mouse. The properties could be the object's ID, its location, or even details about its shape. Next the project can look at how to attach the backend to the visualization and render images from a data file using the same techniques from when the user was providing the information. Finally, it would beneficial to be put in an outline of the filters that the user will be able to use to narrowing the data. Most of these steps are preparatory and still only represent a piece in the larger creation of a QBE search engine.

References.

[1] M. Egenhofer, *Query processing in spatial-query-by-sketch*, Journal of Visual Languages and Computing, (1997).
[2] D. P. J. Coram, J. Morrow, *The panther user experience*, OSTI, (2015).
[3] A. W. M. Rintoul, *Trajectory analysis via a geometric feature space approach*, OSTI, (2015).
[4] Plotly, *Plotly graphing libraries*, 2021.
[5] e. a. R. Brost, *Geospatial-temporal semantic graphs for automated wide-area search*, OSTI, (2017).
[6] T. Team, *Tracktable*, 2021.

PERIODIC LOSS FUNCTION FOR GRID CELL ENCODINGS

SARAH LUCA* AND FELIX WANG†

Abstract. Grid cells, located in the medial entorhinal cortex (mEC), have been shown to be involved in the encoding of a mammal's location in space. The periodic firing of these cells with different spatial offsets relative to one another are believed to provide a compact encoding of location and aid in navigation. Of particular interest is how this grid cell activations can be used to encode location from sparse information obtained from the environment. Taking this inspiration from biology, we have developed a simple fully connected neural network to learn the encoding transformation function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \subset \mathbb{R}$ contains elevation information from an elevation map and $\mathcal{Y} \subset [0, 2\pi)^{2N}$ is the grid cell encoding, as part of an encoding/decoding framework. Since the output of this network lies on a flat (Clifford) torus with periodic behavior, a custom loss function was developed. This paper describes the encoding of location using periodic grid cell phases and presents the proposed loss function for training an encoder. We then present results from training a simple fully connected network to learn the encoding using the proposed loss function and compare the results to a network trained with the standard mean-squared error loss function. The results were inconclusive, but this work provides a baseline for comparison with potentially more compatible network architectures for learning the grid cell encoding in future work.

1. Introduction. Grid cells, which are located in the medial entorhinal cortex (mEC), have been implicated in spatial navigation in mammals [1–3, 5]. These cells have hexagonal firing patterns relative to the environment the mammal navigates that are organized in modules within the mEC. The hexagonal firing pattern of grid cells within the same module share the same scale and orientation but a different spatial offset or "phase" relative to each other [1, 2] (see Figure 1.1). These relative spatial phases are preserved across environments and have been shown to help mammals encode their location and help them navigate in novel environments [5]. Additionally, it has been shown theoretically that representing start and goal locations using grid cell activations with modules of different spatial scales can be used to determine distance and direction between locations[1].

Taking this inspiration from biology, we have developed an encoding/decoding framework that promises to reduce the on-board memory demands of navigation in autonomous agents. Rather than store large amounts of environmental data on board, the agent uses sparse information obtained from sensors and transforms the information into a grid cell encoding which can be decoded into an exact location, where the encoding transformation and decoding are learned offline.

The following sections detail the encoding of location using grid cells and how to train an encoder to learn it using a torus loss function. Section 2 explains how an exact location can be encoded with grid cells and in Section 3 we introduce a periodic loss function for training the encoder. Section 4 compares the proposed loss function to the mean-squared error loss function (MSE) by training a fully connected network to encode elevation data and we conclude with results and discussion of future directions in Section 5.

*University of Arizona Department of Mathematics, sarahluca@math.arizona.edu
†Sandia National Laboratories, felwang@sandia.gov

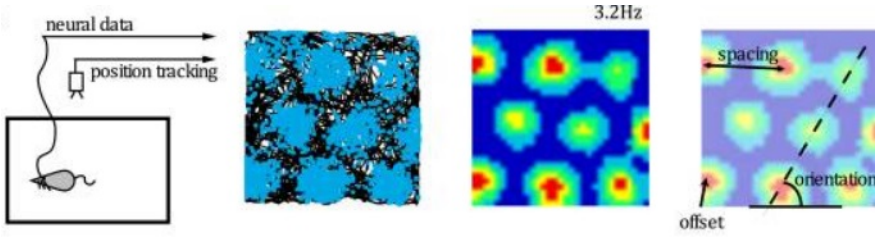


Fig. 1.1: Hexagonal firing pattern of grid cells in the mEC of rats. Leftmost: An electrode measures the activity of grid cells and a position tracker tracks the rat as it moves in its environment. Middle left: raw data from grid cells. The rat's position is recorded in black. The blue indicates the position of the rat as the grid cells fire. Middle right: a firing rate map with high activity for "hot colors". Rightmost: the firing pattern is characterized by the spacing (scale), orientation, and offset (spatial "phase"). Sourced from [1]

2. Grid Cell Encoding of Location. For our application we are interested in navigation of autonomous vehicles and for the environment of interest we are looking at elevation maps as represented by a digital elevation model (DEM) where each location coordinate (x, y) on the map is associated with an elevation (e). For a given elevation map, a hexagonal grid of grid cell activations is generated based on the scale λ, orientation θ, and relative offset ϕ (see Figure 2.1) using equations obtained from [6].

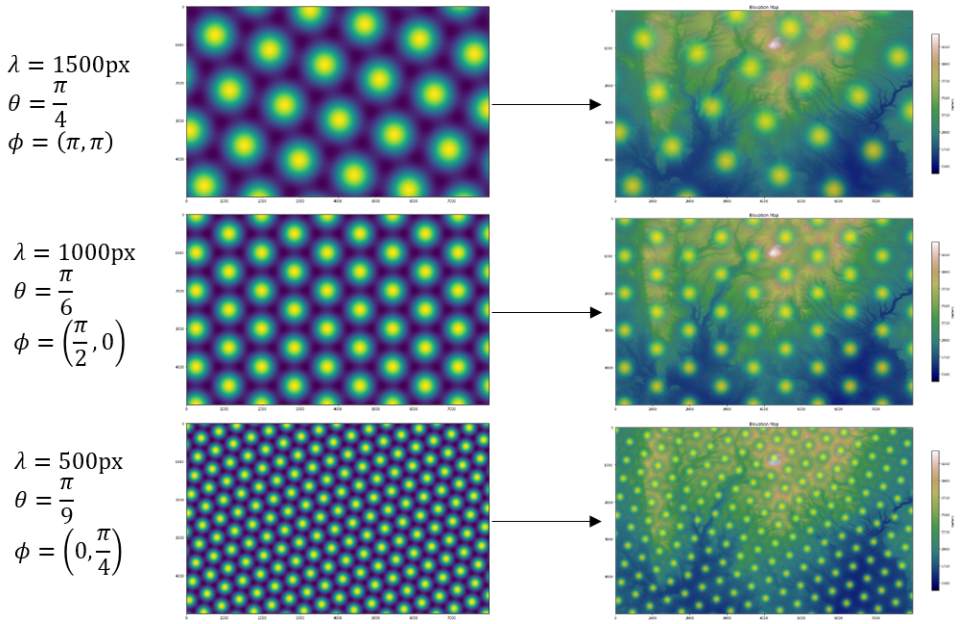


Fig. 2.1: Grid cell activations (left) with different scales λ, orientations θ, and offsets ϕ produced using [6] equations and overlayed on an elevation map (right). The bright green dots correspond to areas of high grid cell activity.

For a specific location on the map, the offset is determined with respect to the reference

point $(\phi_{x_0}, \phi_{y_0}) = (0, 0)$ (see Figure 2.2) and calculated from the phase coding (p_{x_i}, p_{y_i}) of a particular location. Given a location (x_i, y_i), the phase code dimensions are orthogonalized based on the following transformation:

$$(x_i', y_i') = \begin{pmatrix} \cos(\theta) & -\sin(\theta + \frac{\pi}{6}) \\ \sin(\theta) & \cos(\theta + \frac{\pi}{6}) \end{pmatrix} (x_i, y_i)$$

which is then converted to the appropriate phase coding by:

$$(p_{x_i}, p_{y_i}) = \left(\frac{x_i' \text{mod} \lambda}{\lambda} \cdot 2\pi, \frac{y_i' \text{mod} \lambda}{\lambda} \cdot 2\pi \right).$$

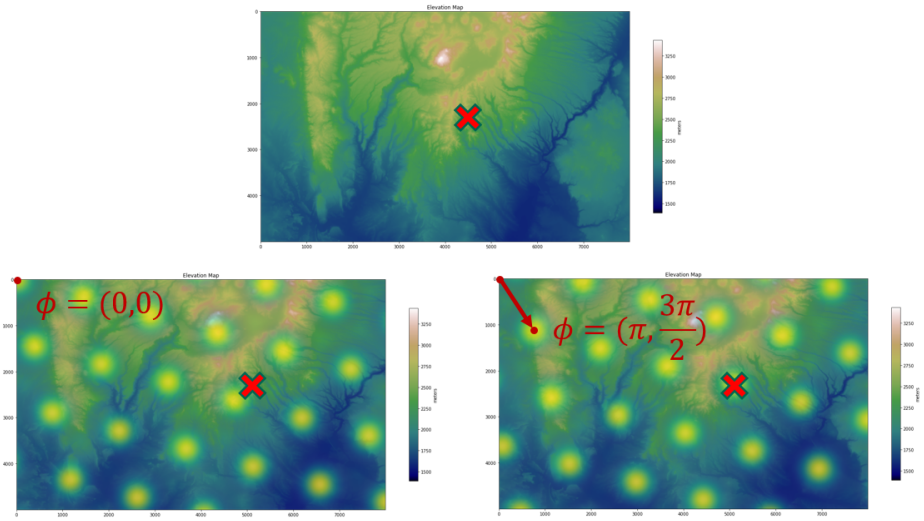


Fig. 2.2: Grid phase offset with respect to reference point (0,0) for a location on an elevation map with grid activations. This represents the phase coding for the location of the red X. Note that every location on the map with a green dot has the same phase coding.

Note that the phase coding is periodic, so many locations on the map share the same phases. Thus, in order to provide a more unique encoding, many grid modules are produced with different scales and orientations (see Figure 2.3). Thus, given N grid modules the location (x_i, y_i) has the following phase encoding:

$$\boldsymbol{p} = (p_{x_i}^1, p_{y_i}^1, p_{x_i}^2, p_{y_i}^2, ..., p_{x_i}^N, p_{y_i}^N).$$

where $p_{x_i}^j, p_{y_i}^j \in [0, 2\pi)$.

2.1. Decoding. In order to convert the grid phase encoding back to exact location coordinates, a coincidence map is calculated. This is done by adding together grid module activations. Increasing the number of modules for the coincidence map produces a more distinguishable location (see Figure 2.4) but also increases the computation time, thus only a subset is needed to reproduce the exact location.

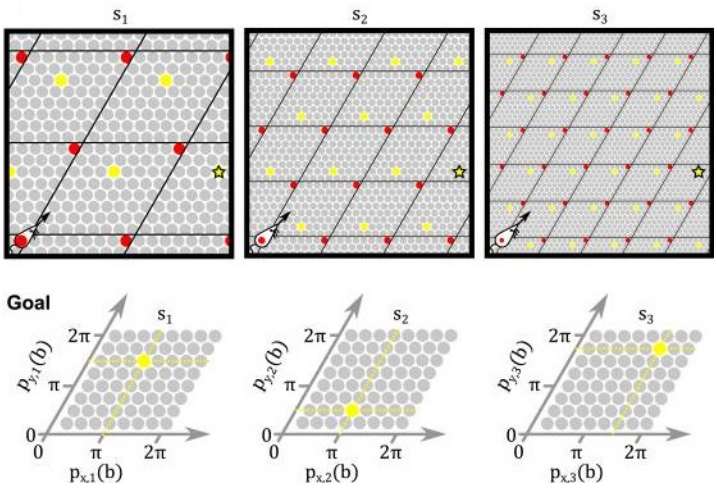


Fig. 2.3: (Top) Each figure (left, right, center) is the environment of a mouse with the location of the grid cell activations (red dots) overlayed on the space. While several locations (yellow dots) in the environment have the same phase coding (Bottom) as the yellow star in each module, multiple modules of different scales (s_1, s_2, s_3) help create a unique encoding of the location. Sourced from [1]

Note that due to the periodicity of the phases, the phase encoding lies on a twisted torus (see Figure 2.5), a manifold which is represented by a parallelogram with opposite sides associated with each other. Normally in supervised learning, the output space of a learning algorithm is either a linear space or a discrete space in the case of a classification algorithm. The next section explores the implementation of learning on a torus and the potential need for a periodic loss function for learning the grid cell encoding.

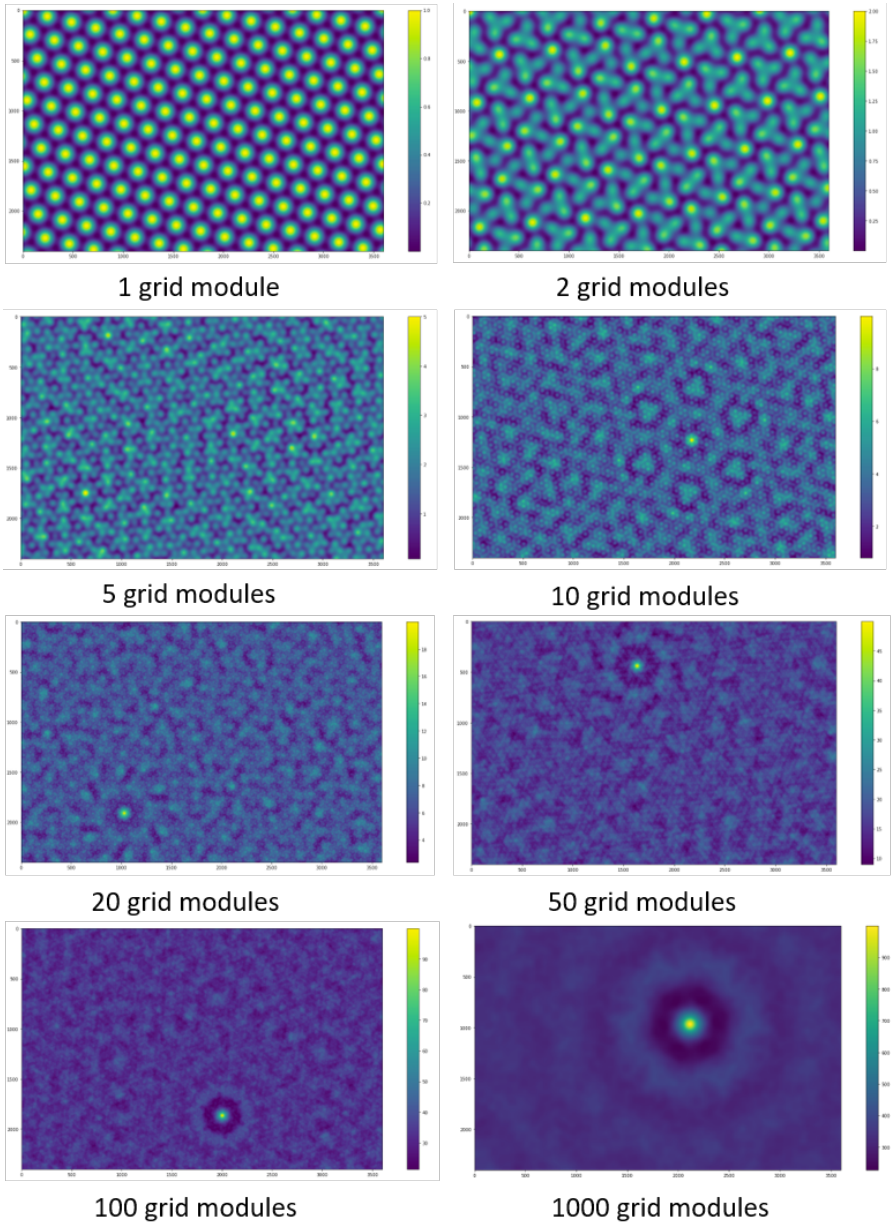


Fig. 2.4: Coincidence maps for different numbers of grid activation modules. As the number of modules used in an encoding increases, the uniqueness of the location becomes more apparent.

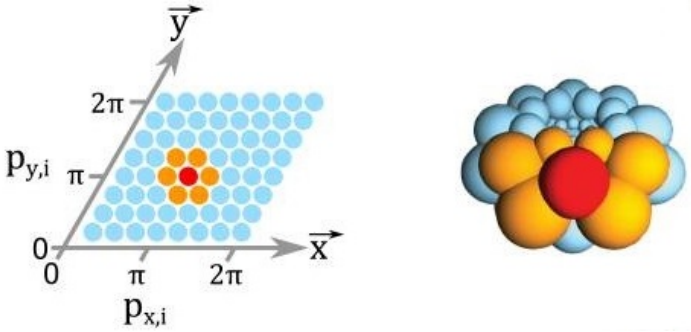


Fig. 2.5: Periodicity of phases lie on twisted torus where opposite sides of a grid tile are joined together. Sourced from [1].

3. Learning on a Torus. Recall that when training a supervised learning algorithm, we seek to find a function $f(x)$ such that $f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space. The learning of this function is achieved by solving the following minimization problem [4]:

$$\text{argmin}_{f:\mathcal{X}\to\mathcal{Y}} \int_{\mathcal{X}\times\mathcal{Y}} \Delta(f(x), y)\rho(x, y)$$

where Δ is a loss function such that $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and ρ is the joint probability distribution for $\mathcal{X} \times \mathcal{Y}$. In this case, $\mathcal{Y} \subset \mathbb{T}^N$, an N dimensional Clifford torus and $\mathcal{X} \subset \mathbb{R}$ is a set of elevations on a map.

3.1. Torus Loss Function. Note that since a torus is a topological manifold it is locally Euclidean. This means that for every point on the torus, there exists a neighborhood around the point that is homeomorphic to $\mathbb{R}^n$. Thus, Euclidean metrics work for measuring the distance between points that lie on a torus. In fact, [4] show how squared geodesic distance is an appropriate loss function for manifolds, which is the equivalent of Euclidean distance when the manifold is $\mathbb{R}^n$. Since the goal of learning is to minimize the distance between the output of the encoder and the expected output, it may be important to consider the periodicity of points that lie on the torus. Consider Figure 3.1. Notice how points are no more than π apart, since opposite sides are associated. Thus the following variation of the mean-squared error loss function was developed to account for the minimum distance between points on $\mathcal{Y}$:

$$\Delta(\hat{\boldsymbol{y}} = f(x), \boldsymbol{y}) = \frac{1}{2N} \sum_{i=1}^{2N} (\min(|\hat{y}_i - y_i|, 2\pi - |\hat{y}_i - y_i|))^2$$

where N is the number of grid cell modules, $\hat{\boldsymbol{y}} \in [0, 2\pi)^{2N}$ is the output of the function $f(x)$ learned by the encoder and $\boldsymbol{y} \in [0, 2\pi)^{2N}$ is the ground truth grid encoding for the location corresponding to the sensor input x.

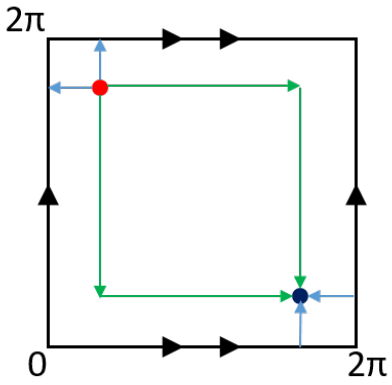


Fig. 3.1: Minimum distance on a torus. The black arrows on the sides indicate which sides are connected, so the top is connected to the bottom and the left connected to the right. The minimum distance in each direction between the red point and the dark blue point is along the blue trajectories.

4. Training Experiment. To evaluate the performance of the torus loss function, a fully connected network was implemented (see Figure 4.1 for network architecture) to learn the transformation of an elevation to a grid cell encoding. In order to train the fully connected network, 10000 data points were generated where each data point contains an elevation $e_i \in \mathcal{X} \subset \mathbb{R}$ from a location (x_i, y_i) on the elevation map. This location has a phase coding $\boldsymbol{p}_i \in \mathcal{Y} \subset [0, 2\pi)^{2N}$ calculated from the transformations detailed in Section 2. For this experiment, 1000 grid modules were used for the phase coding ($N = 1000$).

The fully connected network was trained using two loss functions: the torus loss function proposed in Section 3.1, and the standard mean squared error (MSE) loss function, given by:

$$\Delta(\hat{\boldsymbol{y}} = f(x), \boldsymbol{y}) = \frac{1}{2N} \sum_{i=1}^{2N} (\hat{y}_i - y_i)^2.$$

After training, the location coordinates were estimated by calculating a coincidence map for a subset of 60 grid modules from the phase encoding. The displacement error (D) between the estimated location and the ground truth location was then calculated using Euclidean distance:

$$D = \sum_{i=1}^{10000} \sqrt{(x_{gt_i} - x_{est_i})^2 + (y_{gt_i} - y_{est_i})^2}$$

where (x_{gt_i}, y_{gt_i}) is the ground truth location and (x_{est_i}, y_{est_i}) is the estimated location for each data point i.

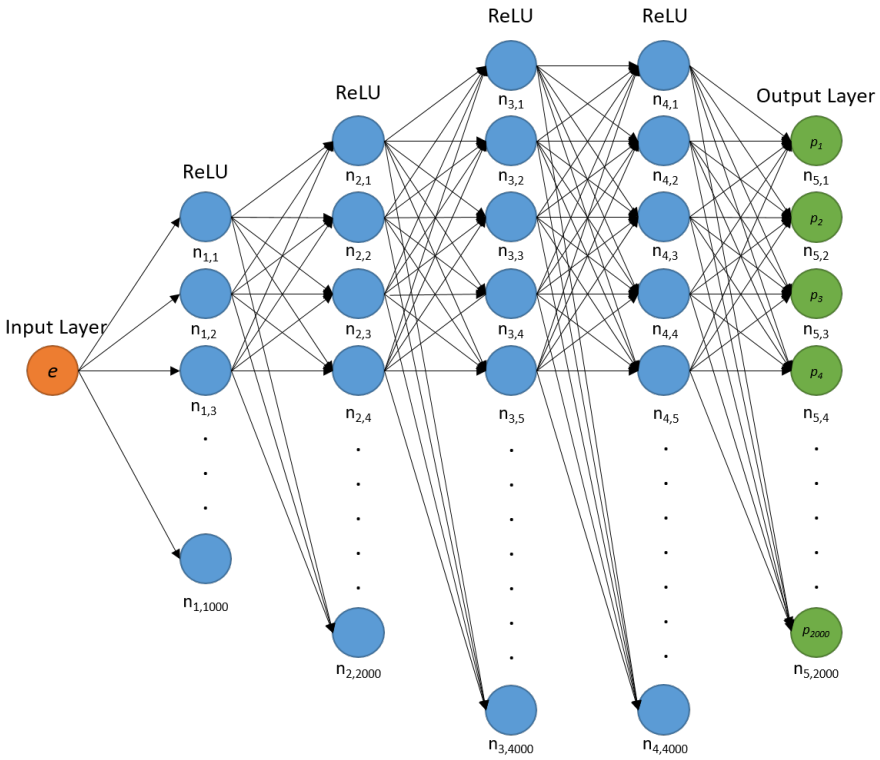


Fig. 4.1: Fully connected network architecture.

5. Results and Discussion. While the MSE loss function had a lower displacement error and loss compared to the torus loss, the fully connected network produced a large displacement error and loss for both functions (see Table 5.1). This could be partially attributed to the chosen network architecture implemented, as the fully connected network was chosen as a base case for comparison to potentially more compatible networks in the future such as an LSTM. It is also possible that the number of grid cell modules chosen to decode the location was not enough to accurately capture the location, but additional grid cell modules significantly increases the computation time so determining the optimal number for the highest accuracy will be explored in the future. Additionally, future training methods would employ more information as input to the network, such as a sequence of elevations and heading information rather than a single elevation. It is likely that multiple locations on the elevation map have the same elevation, so one elevation does not provide much context to the location the encoder is learning.

Given the high displacement and loss for both loss functions, we were not able to conclude that one loss function was significantly better than the other for training the encoder. In the future, an LSTM network with sequential elevation data as input may learn the grid cell encoding better and provide a clearer picture in terms of the effectiveness of the torus loss function. Additionally, a more in depth theoretical analysis might be helpful for determining the usefulness of a periodic loss function or if MSE can be reasonably implemented despite the periodicity of the output space.

	Displacement Error	Loss
Torus Loss	784.8769	90924.4609
MSE Loss	769.1613	81002.75

Table 5.1: Displacement error and loss for the fully connected network for torus loss and MSE loss functions.

References.

[1] D. Bush, C. Barry, D. Manson, and N. Burgess, *Using Grid Cells for Navigation*, Neuron, 87 (2015), pp. 507–520.

[2] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, *Microstructure of a spatial map in the entorhinal cortex*, Nature, 436 (2005), pp. 801–806. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 7052 Primary_atype: Research Publisher: Nature Publishing Group.

[3] M.-B. Moser, D. C. Rowland, and E. I. Moser, *Place cells, grid cells, and memory*, Cold Spring Harbor Perspectives in Biology, 7 (2015), p. a021808.

[4] A. Rudi, C. Ciliberto, G. M. Marconi, and L. Rosasco, *Manifold Structured Prediction*, (2018).

[5] F. Sargolini, M. Fyhn, T. Hafting, B. L. McNaughton, M. P. Witter, M.-B. Moser, and E. I. Moser, *Conjunctive representation of position, direction, and velocity in entorhinal cortex*, Science (New York, N.Y.), 312 (2006), pp. 758–762.

[6] T. Solstad, E. I. Moser, and G. T. Einevoll, *From grid cells to place cells: a mathematical model*, Hippocampus, 16 (2006), pp. 1026–1031.

INTEGRATING PGAS AND MPI-BASED GRAPH ANALYSIS

TREVOR M. MCCRARY*, KAREN D. DEVINE†, AND ANDREW J. YOUNGE ‡

Abstract. This project demonstrates that Chapel programs can interface with MPI-based libraries written in C++ without storing multiple copies of shared data. Chapel is a language for productive parallel computing using global address spaces (PGAS). We identified two approaches to interface Chapel code with the MPI-based Grafiki and Trilinos libraries. The first uses a single Chapel executable to call a C function that interacts with the C++ libraries. The second uses the `mmap` function to allow separate executables to read and write to the same block of memory on a node. We also encapsulated the second approach in Docker/Singularity containers to maximize ease of use. Comparisons of the two approaches using shared and distributed memory installations of Chapel show that both approaches provide similar scalability and performance.

1. Introduction. We developed two methods to interface partitioned global address space programs in Chapel with MPI-based parallel algorithms written in C++, allowing data to be shared, rather than copied, between them. We demonstrate our methods using applications in graph analysis: a simple graph connected-component algorithm in Chapel and a graph hitting-times algorithm in the graph toolkit Grafiki. The general capability to integrate Chapel and MPI-based libraries is valuable in many applications, as it combines the simplicity of Chapel programming with the speed and efficiency of existing high-performance MPI-based algorithms.

Chapel [3, 7] is a partitioned global address space (PGAS) language that is built for productive parallel programming at scale. It is the foundation for the Arkouda [10, 13] NumPy-like parallel computing toolkit. Chapel simplifies parallel programming by providing thread-based parallel loops and managing the layout of arrays within a parallel computer's memory. Although Chapel arrays can be distributed across processors, Chapel users can access any array entry on any processor, without knowing on which processor the data is stored; Chapel manages the data movement or communication required to retrieve data entries. Thus, Chapel provides a very easy-to-use programming environment for parallel algorithm development; algorithms such as parallel graph connected-component labeling can be implemented in just a few lines of code.

In MPI-based libraries, data is also distributed across the memory spaces of parallel processors. The distribution, however, is determined by the programmer. Moreover, each processor can access only the data in its memory. Off-processor data must be explicitly communicated in the MPI program via message passing. This explicit control of data distribution and movement allows highly efficient parallel execution, but requires a great deal more effort on the part of the programmer. For example, Grafiki (the successor of TriData) [14] is an MPI-based library of high-performance parallel graph analysis algorithms with linear solvers from the Trilinos [6,12] toolkit; Grafiki's graph manipulations are done via Trilinos' matrix-vector operations, with inter-processor communication managed explicitly via MPI send/receive operations in Trilinos.

Integrating Chapel and Grafiki allows graph analysts to easily filter and manipulate graph data via Chapel, and then call high-performance algorithms in Grafiki to analyze the resulting data. The main challenges in the integration, however, are avoiding duplication of data between Chapel and Grafiki and insulating graph analysts from the complications of using external MPI-based libraries.

*Mississippi State University, tm2036@msstate.edu
†Sandia National Laboratories, kddevin@sandia.gov
‡Sandia National Laboratories, ajyoung@sandia.gov

Sharing, rather than copying, data between Chapel and MPI-based libraries is crucial to success of the integration. Doubling the amount of memory required to couple two algorithms is often infeasible. In our demonstration, for example, Chapel users wish to analyze the largest connected component of graphs that fill much of their computer's memory; the available memory is insufficient to copy this component in memory for analysis in Grafiki. Our approaches rely on library interfaces with sufficient data abstraction to accommodate Chapel data structures without requiring data to be copied and/or reorganized for Grafiki.

Chapel users should also be insulated as much as possible from the complexities of building external MPI-libraries. One of our approaches addresses this issue by demonstrating how Chapel and Grafiki can share data through memory-mapped regions. Grafiki analysis can then be run as a separate executable or within a Docker container provided to the Chapel users.

In this work, we present two approaches for integrating Chapel programs with MPI-based C++ libraries. The first approach uses Chapel's interface to C-language functions to share data between Chapel and the MPI-libraries. For the second approach, we developed a new Chapel data distribution Domain that uses memory-mapped regions to share data with external processes; to our knowledge, this use of memory-mapped regions in Chapel Domains has not been done before. We demonstrate our approaches using a simple graph connected-components algorithm in Chapel and the high-performance graph hitting-times algorithm in Grafiki. We run on two different Chapel environments: a single-node shared memory environment and a multi-node distributed memory environment.

2. Background. Chapel uses *locales*, which are analogous to MPI ranks. Specifically, a locale is "subset of the target architecture that can be used to control and reason about affinity for the sake of performance and scalability" [7]. Chapel's global address space allows locales to access and manipulate data that are stored on other locales without explicit communication by the Chapel user. However, there is still an underlying communication cost that causes data stored on a different locale to be more expensive to access than data stored on the same locale. In our work, we rely on a one-to-one mapping between Chapel locales and MPI ranks.

Chapel distributed arrays are the main data structured used in this project. We store lists of graph edges and vertices in Chapel distributed arrays. A distributed array is a collection of arrays, with one local array stored on each locale. The Chapel distributed array manages the global address space indexing that allows access of any array entry from any locale. Each locale's local array shares the same indices as its slice of the global array. Chapel's Domain maps describe and manage the distribution of arrays to processes. The `Block` Domain provides a commonly used distribution; in it, the indices are "partitioned evenly across the target locales" [7] so that the first locale has the first contiguous chunk of indices, the second locale has the next chunk, and so on. An example `Block` array with 64 elements distributed across four locales in depicted in Figure 2.1.

A feature of Chapel distributed arrays is the ability for users to create their own distributions to fit their requirements. To create a custom distribution, users create custom Domain map classes that implement the Domain map Standard Interface (DSI). A high-level overview of the DSI can be found at [1], with details on how to build a custom Domain map at [2]. For our second approach below, we create a custom Domain map by modifying Chapel's `Block` distribution to use memory that can be shared among processes.

Chapel supports interoperability with C code. This feature allows a user to access C libraries, variables, functions, structures, and constants using the `extern` keyword. Chapel programs can call C functions, and addresses of Chapel arrays can be passed to C functions to allow Chapel data to be shared with the C functions. We use this capability in our first

Fig. 2.1: Example of a `Block` distribution across four locales of a Chapel distributed array with 64 elements; elements are distributed evenly across locales, with a contiguous chunk of indices assigned to each locale.

approach below.

Chapel also has a library containing definitions for C datatypes (e.g., `long long int`). We use these datatypes for consistency between our Chapel and C code.

The Unix `mmap` function is a critical tool used in this project; it can be used to create memory-mapped regions that can be shared by independent Unix processes. With this function, two separate processes can read and write to the same block of memory by using the Unix `MAP_SHARED` flag. The `mmap` function works with the function `shm_open`, which takes a backing file name (C string) for `mmap`. This backing file name connects `mmap` calls on separate processes to the same block of memory. Chapel has its own `mmap` function, `sys_mmap`, which takes the same arguments as the Unix function, invokes the system `mmap` function and returns an error code. Because we used the Unix `mmap` function in `sys/mman.h>` in our early explorations, we continued using it with Chapel, specifying the `extern` keyword to refer to the Unix `mmap` function. However, we expect Chapel's `sys_mmap` would work identically. In our second approach below, we use `mmap` capability to share edge and vertex lists in mapped memory between separate Chapel and C++ processes or containers.

Chapel can be built with an MPI back-end to do its underlying interprocess communication. When Chapel initializes MPI (via Chapel's `use MPI` directive), Chapel locales and MPI ranks line up; that is, Chapel locale p and MPI rank p see the same local data. Thus, within locale and rank p, we can share data between Chapel and MPI-based libraries without the need for additional communication. We exploit Chapel's alignment between locales and MPI ranks in both of our approaches.

Grafiki (formerly called TriData) [14] is a library of high-performance graph and hypergraph analysis algorithms written in C++. It contains algorithms for computing hitting times, spectral clustering, and eigenvector centrality. For our demonstrations, we use Grafiki's hitting time algorithm, which operates on a square, symmetric matrix that may be distributed across processors. The symmetric matrix represents the adjacency matrix of a connected graph (i.e., the graph has a single connected component). Each edge (i, j) in the Chapel data corresponds to a nonzero a_{ij} in the adjacency matrix A; each vertex corresponds to a row and column of the matrix.

Grafiki's algorithms rely on linear and eigen-solvers; for these solvers as well as for abstractions of matrix and vector operations, Grafiki uses the open-source Trilinos [6, 12] framework. Trilinos has been developed and optimized for both distributed memory, shared memory, and GPU parallel performance, especially in the realm of physics-based scientific simulations. Trilinos also can operate with arbitrary data distributions, including two-dimensional matrix distributions favored for reducing communication in graph analysis applications. In this work, we pass the edge and vertex lists to Grafiki with the same distribution as specified by the user in Chapel; thus, the distribution of nonzeros to processors is arbitrary and matches that of the Chapel edge list distribution.

Trilinos provides an efficient compressed sparse row matrix (`CrsMatrix`) data structure, but creation of a `CrsMatrix` from Chapel data would require creating a copy of the data in CRS format – an unacceptable requirement for this project. However, Trilinos also

provides a `RowMatrix` abstraction that allows users to implement matrix operations on their data using their own underlying data structures. The `RowMatrix` abstraction supports any distribution of sparse matrix entries across processors. It does not require that matrix entries be sorted in any particular manner. Performance of a user's `RowMatrix` strongly depends on the user's implementation and data distribution. In our work, we sacrifice some computational performance by allowing our `RowMatrix` to use the Chapel edge lists directly, without any reordering or reorganization, thus avoiding an additional copy of Chapel users' data.

3. Methodology. The general use case that we wish to support follows. Graph analysts load graph data into Chapel edge lists. They then perform some type of filtering of the graph data to identify vertices and edges of interest. The resulting subgraph is shared with an MPI-based library for further analysis, and results are shared back to Chapel. Our goal is to accomplish this workflow without copying and/or reformatting data that is to be shared between Chapel and the MPI-based library.

Our demonstration follows the pattern of this general use case. We load a graph from MatrixMarket-formatted files, which contain coordinate pairs $(source_k, target_k)$ and, optionally, a weight w_k for all edges k in the graph (i.e., nonzeros in an adjacency matrix). The edge are stored in Chapel arrays *source* and *target* distributed across locales according to Chapel's `Block` distribution. We then filter the graph by identifying its largest connected component via a parallel label-propagation algorithm written in just a few lines of Chapel code. We then share the edge lists, along with an array identifying vertices in the largest connected component, with the MPI-based Grafiki library. Grafiki computes vertex hitting times on the largest connected component. In our implementation, Grafiki writes hitting-time results to a file, but it would be straightforward using the mechanisms below to return an array of hitting-time values back to Chapel for further use.

We have developed two approaches to allow Chapel to interface with Grafiki. Both methods create a Trilinos `RowMatrix` (in C++) to describe the matrix that is passed to Grafiki. The `RowMatrix` class directly uses our shared, distributed edge lists to answer queries about the matrix and perform matrix operations. For our demonstration, we implemented only the ten (out of 23) methods of `RowMatrix` that were needed by Grafiki. The most important method we implemented was sparse-matrix vector multiplication (SpMV): $y = \alpha Ax + \beta y$ for matrix A, vectors x and y, and constants α and β. In parallel with distributed data, SpMV requires communication of off-processor vector entries x_j to be used in multiplication with local matrix entries a_{ij}, and of subproducts $a_{ij}x_j$ to be accumulated into vector entries y_i. Fortunately, the communication operations required for SpMV with our `RowMatrix` are identical to those in Tpetra's `CrsMatrix` implementation, and we were able to copy them into our `RowMatrix`. However, the `localApply` — the on-processor SpMV operation — needed to be rewritten to use the coordinate-formatted edge lists from Chapel rather than Trilinos' compressed-sparse row format. We implemented `localApply` with a straightforward loop over the edge lists, multiplying each edge value $(source_k, target_k)$ with the appropriate, possibly communicated, x vector entry x_{target_k}. For our proof-of-concept demonstration, we did not attempt to optimize this operation; several possible optimizations are discussed in Section 5.

3.1. Implementation One: Direct Chapel-to-C calls. For our first implementation, we use a straightforward approach in which a single Chapel executable reads the graph data file, performs the connected-component label propagation, and then calls directly to a C function that then calls a C++ function that calls Grafiki. A high-level flow chart of its execution is in Figure 3.1.

This implementation takes advantage of Chapel's C interoperability to directly call C

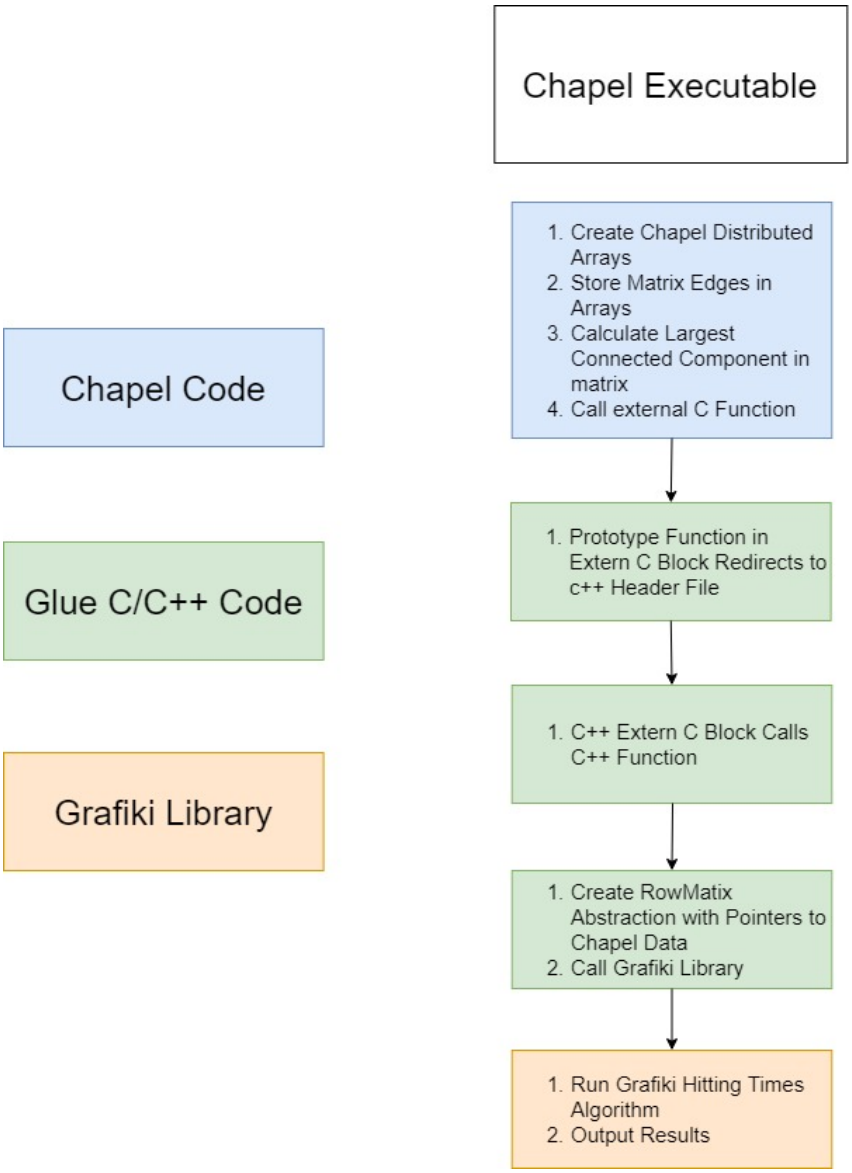


Fig. 3.1: Implementation One: A single Chapel executable calls Grafiki hitting times

code. An overview of this pathway is shown in Figure 3.2. Chapel can call C code directly, but we need to eventually call C++ code in the Grafiki library. So we start by having Chapel call a short C function `glueC` wrapped in `extern "C"` controls to allow it to be compiled by a C++ compiler. The C function then calls a C++ function `glueChapelGrafiki` with the same arguments. Function `glueChapelGrafiki` instantiates a `RowMatrix` object using the Chapel edge lists, and calls Grafiki.

The C and C++ code are in a file separate from the Chapel code. The file is compiled by a C++ compiler and its object file is linked with the Chapel object file during build time.

Each Chapel locale must call the C function in parallel, using Chapel's `coforall`

```
//Chapel main
...
require "glue.h";
extern proc glueC(nEdges, *src, *tgt, ...);

coforall loc in Locales
{
  on loc
  {
    var mySrc = source.localSubdomain();
    var myTgt = target.localSubdomain();
    glueC(mySrc.size:size_t, c_ptrTo(source[mySrc.low]),
                             c_ptrTo(target[myTgt.low]), ...);
    ...
  }
}

```

```
//glue.h file
extern "C"
{
  int glueC(nEdge, *src, *tgt, ...);
}

```

```
//glue C++ file
int glueChapelGrafiki(nEdge, *src, *tgt, ...);
{
  Build RowMatrix(nEdge, src, tgt, ...);
  return Grafiki_hittingTimesDriver(RowMatrix, ...);
}
extern "C"
{
  int glueC(nEdge, *src, *tgt, ...);
  {
    return glueChapelGrafiki(nEdge, src, tgt, ...);
  }
}

```

Fig. 3.2: Chapel's path to call C++ code

parallel-for loop and its built-in *Locales* array as in Figure 3.2. The `coforall` loop creates a parallel task *loc* for each locale, and the subsequent `on` *loc* directive ensures that each locale calls `glueC` independently. This parallelism is needed to prevent Grafiki from hanging in collective MPI communications; all locales (i.e., all ranks) must participate in the call to `glueC`.

The short C function passes the locale's data to C++ function. In our case, we pass arguments that are needed for Grafiki hitting times. Each locale provides the number of edges and vertices stored in the locale, pointers to its local *source* and *target* arrays, along with other data such as the local vertex component-label arrays and an identifier for the largest connected component (for filtering out edges not in the component). The appropriate way to get the pointer to the local arrays is shown in Figure 3.2: using Chapel's `c_ptrTo` function to obtain the C pointer to the lowest-indexed value of the array in the locale.

This approach provides a simple proof-of-concept pathway for integrating an MPI-based C++ library with a Chapel application, but it has several drawbacks. It is intrusive to the Chapel algorithm development. In order for Chapel to directly call to Grafiki, Chapel users must understand how to call C functions from Chapel. They must ensure that their Chapel installation, Grafiki library, and Trilinos library were built all with compatible compilers and MPI libraries. They also need to determine how to link the Grafiki and Trilinos libraries with their Chapel executable. Our next approach reduces this burden for Chapel users.

3.2. Implementation Two: Separate Chapel and C++ Library Processes. Our second implementation has a Chapel process interact with an independent C++ process running Grafiki. A high level flow chart of its execution is in Figure 3.3. We launch the C++ program before the Chapel program; it waits for a semaphore to be posted by Chapel before starting execution. The Chapel program reads edges into arrays that use a new `shareBlock` Domain that allows them to be shared with the C++ program through a `mmap` memory region. It performs its analysis (in this case, connected-component labeling). It then writes some metadata about the edge lists to a file and sets the semaphore indicating the C++ program can begin. The C++ program reads the metadata file to get information on accessing the shared data, creates a `RowMatrix` with the shared data, and calls Grafiki. Upon completion, it resets the semaphore, indicating to the Chapel program that the Grafiki result is ready. Details of these steps follow.

To enable the Chapel data to be shared with the C++ program, we created a modified version of Chapel's `Block` Domain in which the local arrays are built using `mmap` memory. We refer to this modified Domain as a `shareBlock` Domain.

A comparison of constructing Chapel's `Block` Domain and our `shareBlock` Domain is shown in Figure 3.4. The Chapel `Block` Domain stores the local arrays in a variable named `myElems`. This variable is typically assigned with a call to Chapel's `domain.buildArray()`. When a distributed array is initialized, the code that initializes `myElems` is called number of locales plus one times. The "plus one" time requests a domain of size zero. For the `shareBlock` Domain, we test the size of the domain, and use Chapel's code when the domain size is zero. When the domain size is not zero, we we replace the call to `domain.buildArray()` with a new function `createMmap()` that calls the Unix `mmap` function to allocate memory of the same size size (the element's data type size times the domain size).

Each locale's `mmap` local array uses a separate backing file. The backing file name is generated by function `getBackingFile()`; it is a string consisting of the locale's ID and a counter to indicate which `shareBlock` is stored there. An example of these backing file names is shown in Figure 3.7. The files `/share.bak0-1`, `/share.bak1-1`, `/share.bak2-1`, and `/share.bak3-2` store the store the *source* vertices of the edge lists on locales 0, 1, 2, and 3, respectively; the *source* array was the first `shareBlock` array allocated. Similar backing files are defined for the *target* vertices and the *component* flags indicating the connected component owning each vertex.

To use the created `mmap` region for the local array, we pass a pointer to it to modified versions of Chapel's `makeArrayFromPtr` and `makeArrayFromExternArray`. A pseudocode overview of the changes is displayed in Figures 3.5 and 3.6. Function `makeArrayFromPtr` is a short function that calls `makeArrayFromExternArray`. Function `makeArrayFromExternArray` creates a Domain for the new array and returns a call to `_newArray()`, returning a new Chapel local array. (Incidentally, `_newArray()` is also returned by `domain.buildArray()`.) We modified `makeArrayFromPtr` and `makeArrayFromExternArray` to accept a Domain as an extra argument. The Domain was needed because the original functions created a new Domain from 0 to the size of the

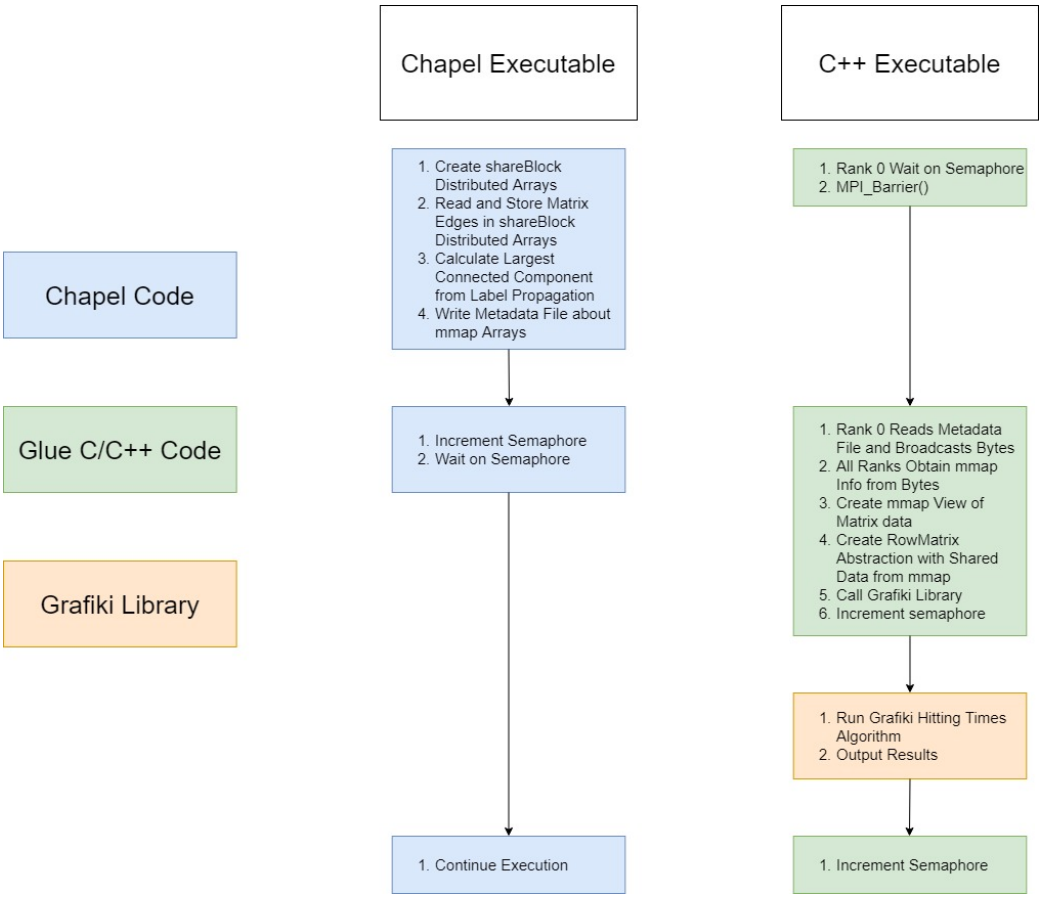


Fig. 3.3: Implementation Two: Separate Chapel and C++ executables share graph data in `mmap` memory.

array minus one, while the `Block` Domain expected indices matching the global indices of the array. The change in Domain resulted in Chapel making a copy of the `mmap` region to the `myElems` array, rather than using the `mmap` region directly. Our modified `shareMakeArrayFromExternArray` uses the same Domain as `myElems`, so we avoid making a copy, and `myElems` refers directly to the `mmap` memory.

After all of the locales' local arrays are allocated with `mmap`, we increment a counter *Status* that is used to differentiate between separate `shareBlock` arrays' backing files.

This `shareBlock` implementation makes some assumptions about Chapel and the user's array. Primarily, some of the Chapel functions above are in a module that is not yet ready for stabilization and may change in the future. Our testing was performed on Chapel 1.24.0; our code may need to change to address changes in future Chapel releases. In addition, the `shareBlock` Domain is not easily mutable. With standard `Block` Domains, a user can redefine the domain of a distributed array, and it will grow or shrink to match the new domain. With our `mmap` arrays, attempts to resize the array will result in a segmentation fault. This limitation required us to pad our *source* and *target* arrays in instances where the number of edges is unknown. For example, when reading a symmetric

```
//Chapel's Block's LocBlockArr
class LocBlockArr {
proc init() {
    ...
    this.myElems = this.locDom.myBlock
    .buildArray(...);


}
```

```
//New shareBlock's LocBlockArr
class LocBlockArr {
proc init() {
    ...
    if(this.locDom.myBlock.size)
    {
      var myPtr = createMmap(eltType,
    this.locDom.myBlock.size:uint);
      this.myElems =
    shareMakeArrayFromPtr(myPtr, this.
    locDom.myBlock.size:uint, this.
    locDom.myBlock);
      allLocalesBarrier.barrier();
      if(here.id == 0) then
        Status += 1;
    }
    else
    {
      this.myElems = this.locDom.
    myBlock.buildArray(...);
    }

```

```
//Chapel functions to create mmap array
proc createMmap(type eltType, size:uint):c_ptr
{
    var myBytes = (size * c_sizeof(eltType)):uint;
    var fd = shm_open(getBackingFile(), ...);
    ftruncate(...);
    var region;
    region = mmap(nil, myBytes:size_t, PROT_READ | PROT_WRITE, MAP_SHARED, fd
    :fd_t, 0:off_t);
    return region:c_ptr(eltType);
}
proc getBackingFile():c_string
{
    return ("/share.bak" + here.id:string + "\-" + Status:string).c_str();
}


```

Fig. 3.4: Chapel's path to building `mmap` arrays for local arrays in `shareBlock`. If the size of the domain is nonzero, an `mmap` array is created to fit the size requirements of the Domain and datatype. We keep track of backing files by incrementing *Status* after all local arrays have been allocated.

Matrix Market file (in which only lower-triangular entries are stored), we allocated an array for twice the number of nonzeros in the file, to store both edges (i, j) and (j, i). Self-edges (i, i), however, are not reflected, leading to an overestimate in the allocated memory size.

For the C++ program to access the `mmap` regions, it needs to know some metadata about the arrays using `shareBlock` Domains. Specifically, it needs the names of the backing files associated with each locale's local arrays, the size of those arrays, and the command line arguments passed to the Chapel program. The Chapel program writes these fields to a metadata file, as in Figure 3.7. While we used a regular file for the metadata, one could easily use a `mmap` memory space to share the metadata; beacuse the file is very

```
//Chapel's makeArrayFromPtr
proc makeArrayFromPtr(value: c_ptr,
    num_elts: uint)
{
  var data =
    chpl_make_external_array_ptr(value
    , num_elts);
  return makeArrayFromExternArray(data
    , value.eltType);
}
```

```
//New shareMakeArrayFromPtr
proc shareMakeArrayFromPtr(value:c_ptr
    , num_elts:uint, dom:domain)
{
  var data =
    chpl_make_external_array_ptr(value
    , num_elts);
  return shareMakeArrayFromExternArray
    (data, value.eltType, dom);
}
```

Fig. 3.5: Chapel's original (left) vs. our new (right) `makeArrayFromPtr`: in the new version, a `shareBlock` Domain is passed as an argument that is then passed to `shareMakeArrayFromExternArray`.

```
//Chapel's makeArrayFromExternArray
proc makeArrayFromExternArray(value:
    chpl_external_array, type eltType)
{
  var dom = 0..number_elements\-1;
  var arr = new unmanaged
    DefaultRectangularArr(dom=dom,
    ...);
  dom.add_arr(arr, locking = false);
  return _newArray(arr);
}
```

```
//New shareMakeArrayFromExternArray
proc shareMakeArrayFromExternArray(
    value:chpl_external_array, type
    eltType, dom:domain)
{
    var arr = new unmanaged
    DefaultRectangularArr(dom=dom.
    _value, ...);
    dom.add_arr(arr, locking=false);
    return _newArray(arr);
}
```

Fig. 3.6: Chapel's original (left) vs.our New (right) `makeArrayFromExternArray`: in the new version, a `shareBlock` Domain is passed to Chapel's `DefaultRectangularArr()` to create the local array.

small, using a regular file is feasible and straightforward. This example file shows a matrix with 25 vertices and 105 edges distributed across four locales, with backing arrays for the *source* and *target* edge lists and the *component* labels.

After the Chapel program posts the semaphore to flag the C++ program to begin, rank 0 of the C++ program reads the entire metadata file into a buffer and broadcasts it to the other ranks. Each rank then individually parses the buffer to extract the information (backing file names and data sizes) associated with its rank. The MPI ranks then set up pointers to their `mmap` data (*source*, *target*, and *component*), collectively create a distributed `RowMatrix` with the data, and call Grafiki hitting times. This implementation writes the results to a file, but it would be straightforward to create another `shareBlock` array to share results back to the Chapel program.

The key advantage of this implementation is that it is less intrusive to Chapel developers. A developer can declare a distributed array using `shareBlock`, and the `shareBlock` class handles creating the `mmap` arrays to fit the required size. In addition, this method allows a C++ process to be built and executed separately from the Chapel program, eliminating the need for Chapel programmers to link their Chapel programs with Grafiki, Trilinos and MPI

```
source /share.bak0-1 /share.bak1-1 /share.bak2-1 /share.bak3-1
target /share.bak0-2 /share.bak1-2 /share.bak2-2 /share.bak3-2
component /share.bak0-3 /share.bak1-3 /share.bak2-3 /share.bak3-3
nEdge 27 26 26 26
nVtx 7 6 6 6
useThisComp 1
argc 3
argv ./chapelExec ... ...

```

Fig. 3.7: Chapel shares matrix data with the separate C++ executable that calls Grafiki via a small metadata file containing per-locale backing file names and data sizes.

libraries. The disadvantage of this method, for now, is that it is restricted to a block-style distribution. Extensions to other Chapel distributions are still needed. Also, currently, we cannot reshape a `shareBlock` Domain after it is initialized.

3.3. Containerization of the C++ Library. To make Grafiki even easier to use by Chapel programmers, we have containerized our C++ glue program with Grafiki, and shown that Chapel and the Grafiki container can share memory through the same `mmap` array mechanism. This containerization allows Grafiki developers to provide tools that are fully encapsulated, sparing Chapel users from have to compile the Grafiki executable. Our Docker container encapsulates our C++ glue program, Grafiki and all of libraries on which Grafiki depends (Trilinos, Kokkos, MPI, BLAS). We use Singularity [9], a container system designed for high-performance parallel computing, to instantiate our container with MPI parallelism. Our workflow then proceeds as in Implementation Two (Figure 3.3), with separate Chapel and Singularity containers accessing the same `mmap` regions.

The primary challenge we faced with integrating Chapel with containers was the use of semaphores. While `mmap` memory works across containers, POSIX semaphores do not. Semaphores are created on the parent process' stack. They are unusable in the container after namespace creation, which copies the semaphore rather than addressing it. Thus, the semaphore never unlocks in the C++ program. Our solution is to implement a "pseudo-semaphore" signaling mechanism. Since `mmap` works across containers, we use a single integer in `mmap` memory to mimic semaphore functionality. The value of this flag indicates which process should be working and which should be idling. Currently, we use a spin lock for this functionality.

With containerization, the Chapel user is spared the chore of building Grafiki and everything on which it depends. This approach delivers the highest ease of use to graph analysts.

4. Experiments and Results. We demonstrate our approaches with a simple Chapel graph analysis application that shares the graph with the C++ library Grafiki. Specifically, our Chapel program reads a matrix from a Matrix Market file, symmetrizing the matrix if necessary, and constructs lists of edges corresponding to the matrix nonzeros. It then identifies the largest connected component of the graph using a simple, commonly used, label propagation algorithm. For label propagation, each vertex's label is initialized to its vertex number. Then the propagation algorithm loops over edges, giving each vertex of an edge the lower-valued label of the edge's two vertices. Iteration over edges is done in parallel with respect to locales (see Figure 4.1) and continues until no vertex labels change. The Chapel code then counts the number of vertices with each label and identifies the label with

```
do {
  changed = 0;
  coforall loc in Locales with (+ reduce changed) {
    on loc {
      var src: c_longlong;
      var tgt: c_longlong;
      var srcVal: c_longlong;
      var tgtVal: c_longlong;

      for i in sourceArray.localSubdomain() {
        src = sourceArray[i];
        srcVal = vtxValue[src];
        tgt = destinationArray[i];
        tgtVal = vtxValue[tgt];
        if(srcVal != tgtVal) then {
          vtxValue[sourceArray[i]] = min(srcVal, tgtVal);
          vtxValue[destinationArray[i]] = min(srcVal, tgtVal);
          changed = 1;
        }
      }
    }
  }
} while (changed != 0);

```

Fig. 4.1: Simple label propagation in Chapel: parallelism is done with respect to locales, with the *changed* flag set with a reduction across locales.

the most vertices. The label of the largest component is passed, along with the edge lists and vertex component labels, to Grafiki, which computes hitting times within the largest component. While vertices that are not in the largest component remain in the edge lists passed to Grafiki, they are ignored in the `RowMatrix` sparse matrix-vector multiplication operation.

We tested our approaches on Sandia's Kahuna high-performance data analytics (HPDA) research cluster. We used Kahuna's Dual Socket Intel E5-2683v3 2.00GHz CPU with 28 cores and 256 GB of memory. Kahuna uses a shared-memory version of Chapel 1.23; all experiments on Kahuna used a single multi-core node. Thus, for Implementation One, in which Chapel calls Grafiki directly, only one MPI rank (and, thus, one locale) can be used. Implementation Two, in which separate processes are used for Grafiki and Chapel, allows more MPI parallelism, as the MPI parallelism is not tied to Chapel's shared-memory implementation; we can run Grafiki and Chapel with up to 28 ranks and locales, respectively.

We also tested on Sandia's Mutrino Cray computing system with 100 Intel Haswell nodes with 96 GB of memory per node. Mutrino has a distributed memory version of Chapel 1.24 with an MPI backend. Thus, we could run both approaches with multiple ranks and locales. We assigned one rank/locale per node.

We tested the performance of our methods with small and large sized graphs. We obtained test case files from the SuiteSparse matrix collection [4]. The small graph is *bcsstk29.mtx*, which contains 28 strongly connected components and a largest connected component with 13.8K vertices and 620K edges. The large graph is *GAP-kron.mtx*, which contains 78M strongly connected components (many of them singleton vertices), and a largest connected component with 63M vertices and 4.2B edges. The goal of the large test case is to use over half of the node's memory for the source and target arrays, making the

Platform	Number of Locales	Chapel LabelPropagation Time per iteration (s) One	Chapel LabelPropagation Time per iteration (s) Two	Grafiki Hitting Times Time per iteration (s) One	Grafiki Hitting Times Time per iteration (s) Two
Kahuna	1	1.24	1.22	0.0285	0.0287
	2	NA	1.91	NA	0.0162
	4	NA	1.56	NA	0.0087
	8	NA	1.00	NA	0.0045
	16	NA	0.65	NA	0.0051
Mutrino	1	0.82	0.89	0.0215	0.0230
	2	0.65	0.77	0.0113	0.0120
	4	0.44	0.47	0.0058	0.0062
	8	0.22	0.29	0.0031	0.0033
	16	0.11	0.15	0.0018	0.0018

Table 4.1: Runtime of Chapel connected-component label propagation and Grafiki hitting times on Kahuna and Mutrino using Implementation One (Direct calls from Chapel to Grafiki, Section 3.1) and Implementation Two (Separate Chapel and Grafiki executables, Section 3.2) with small graph *bcsstk29.mtx*

graph too large to be copied; data sharing between Chapel and Grafiki is the only way to solve the problem.

In Table 4.1, we show the execution times for label propagation in our connected-component algorithm written in Chapel, and the linear solve time in Grafiki's hitting time algorithm using both integration methods and the small graph *bcsstk29.mtx*. Results are shown for both the Kahuna and Mutrino systems. Implementation One uses direct calls from Chapel to C with shared data in regular memory as described in Section 3.1. Implementation Two uses separate Chapel and C++ processes with shared data in memory-mapped regions as described in Section 3.2. Execution times for both implementations are comparable, with no significant loss of performance caused by using mapped memory in Implementation Two.

On both platforms, we see that adding more MPI ranks accelerates the linear solve in Grafiki, with reasonable scaling even for this small graph. Adding more locales also accelerates each iteration of label propagation in the Chapel-based connected component algorithm. Adding locales can increase the number of iterations required for connected component labeling, as each locale operates on a subset of the edges, slowing propagation across the full graph. This effect was seen in this small graph, in which the ordering (sorted by target vertex) of the input graph was optimal for label propagation with one locale; increasing the number of locales from one to 16 increased the number of propagation iterations from two to nine in this graph.

We ran the same experiments with the large *GAP-Kron.mtx* graph; execution times are shown in Table 4.2. Again, we see that Grafiki's linear solve time scales reasonably well with the number of MPI ranks, as does the Chapel label propagation time with the number of locales. On Mutrino, we see a difference in the label propagation time for our two approaches, with per-iteration time significantly longer using Implementation Two. This time difference is not seen for the Grafiki linear solve; indeed, Grafiki linear solve times for both approaches are nearly identical. Thus, the degredation in Chapel's label propagation must be due to our implementation of `shareBlock` rather than some inherent mapped-memory access issue on Mutrino. We do not know yet the cause of this increase, although we suspect it arises due to differences between Chapel versions 1.23 and 1.24. Our `shareBlock` Domain was built

Platform	Number of Locales	Chapel LabelPropagation Time per iteration (s) One	Two	Grafiki Hitting Times Time per iteration (s) One	Two
Kahuna	1	8626	7476	1187	1235
	4	NA	9103	NA	364
	16	NA	3749	NA	188
Mutrino	4	OOM	10333	OOM	371
	16	1274	3079	103	105
	64	320	824	30	26

Table 4.2: Runtime of Chapel connected-component label propagation and Grafiki hitting times on Kahuna and Mutrino using Implementation One (Direct calls from Chapel to Grafiki, Section 3.1) and Implementation Two (Separate Chapel and Grafiki executables, Section 3.2) with large graph *GAP-Kron.mtx*

from the `Block` Domain in Chapel 1.23, which may not exploit performance enhancements in Chapel version 1.24. More investigation and, perhaps, a transfer of our memory-mapped approach to newer versions of Chapel are needed.

5. Conclusions and future work. We have shown a proof-of-concept for integrating Chapel algorithms with MPI-based algorithms. It enables Chapel users to take advantage of existing, optimized parallel algorithms while maintaining the simplicity of Chapel programming. Our approaches enabled integration of Chapel algorithms with MPI-based libraries without requiring additional copies of the user data, allowing users to solve problems that fill the computer's memory. We have demonstrated our approaches with shared-memory and distributed-memory implementations of Chapel, as well as with Docker/Singularity containers.

This project raises many opportunities for future work.

Arkouda [10, 13] is often used for large-scale graph analysis because of its elegant NumPy-like interface. Arkouda is built on Chapel, so providing interfaces from Arkouda to Grafiki using the technology from this project would be a natural (and, we expect, straightforward) extension to this work.

We plan to refine the software engineering of our demonstration code. The glue code that served as the interface between Chapel and Grafiki should, ideally, be distributed as part of Grafiki, rather than exist as a separate layer between Chapel and Grafiki. The management of semaphores should move into this glue interface as well, rather than reside in the Chapel code. These changes are structural only, and do not represent algorithmic changes to our approach.

In our experiments, we saw that our `RowMatrix` sparse matrix-vector multiplication was slower than Trilinos' implementation using its `Tpetra::CrsMatrix` class. Several reasons exist for this difference. The `Tpetra::CrsMatrix` has matrix data organized into an efficient compressed-sparse row data structure to improve memory accesses during matrix-vector multiplication. To use `Tpetra::CrsMatrix` with Chapel data, one could copy and reorganize the graph data; this copying, however, is often infeasible for extreme-scale graphs in limited memories. Alternatively, one could write the Chapel data to a file and read it back into a `Tpetra::CrsMatrix` in Grafiki; this approach puts arguably the least scalable part of graph computation – file input and output – directly in the execution path. By using graph data exactly as it is provided by the Chapel user, our approaches avoid both data copies and file I/O, at the expense of somewhat slower computation in Grafiki. In

future work, we can investigate ways to speed-up our matrix-vector multiplication through the use of thread parallelism.

Distributing the graph's edge lists differently to increase locality and reduce off-processor data dependencies would also speed the matrix-vector multiplication (and, likely, other operations of interest in both Chapel and Grafiki. For example, load balancing algorithms such as those in Zoltan [5] or ParMETIS [8] balance computational work while attempting to minimize off-processor data dependencies. Even simple sorting of edge lists has been shown to reduce cache misses and speed execution [11]. Our current work relies on the users' data layout, so users would need to pre-process their data to increase locality.

This work allows Chapel users to take advantage of years of effort in MPI-based parallel computing for physics-based and graph-based applications by using libraries like Trilinos and Grafiki. Comparisons between Chapel performance and MPI performance are beyond the scope of this work, but are important for evaluating the productivity versus performance trade-off in using Chapel.

6. Acknowledgments. We thank several people for enabling this work. Jon Berry provided motivation and use cases for this project. Samuel Knight, Omar Aaziz and Connor Brown provided system support on Sandia's parallel computers. Chapel developers Elliot Ronaghan, Brad Chamberlain, Michael Ferguson, and Lydia Duncan in the Chapel Discourse Group provided technical advice on use of Chapel and got us over several hurdles. Danny Dunlavy provided valuable feedback on this paper.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References.

[1] B. Chamberlain, S.-E. Choi, S. Deitz, D. Iten, and V. Litvinov, *User-defined distributions and layouts in Chapel: Philosophy and framework*, in 2nd USENIX Workshop on Hot Topics in Parallelism, Cray Inc, 2010, pp. 1–12.

[2] B. Chamberlain, S. Deitz, D. Iten, and S.-E. Choi, *Authoring user-defined domain maps in Chapel*, in Chapel User's Group 2011, Cray Inc, 2011, pp. 1–6.

[3] B. L. Chamberlain, E. Ronaghan, B. Albrecht, L. Duncan, M. Ferguson, B. Harshbarger, D. Iten, D. Keaton, V. Litvinov, and P. Sahabu, *Chapel comes of age: Making scalable programming productive*, in Cray User Group, 2018.

[4] T. Davis, *SuiteSparse matrix collection*. https://sparse.tamu.edu/.

[5] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek, *Parallel hypergraph partitioning for scientific computing*, in Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06), IEEE, 2006.

[6] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, et al., *An overview of the Trilinos project*, ACM Transactions on Mathematical Software (TOMS), 31, pp. 397–423.

[7] Hewlett Packard Enterprise Development LP, *Chapel*. https://chapel-lang.org/docs/.

[8] G. Karypis and V. Kumar, *ParMETIS: Parallel graph partitioning and sparse matrix ordering library*, Tech. Rep. 97-060, Dept. Computer Science, University of Minnesota, 1997.

[9] G. M. Kurtzer, V. Sochat, and M. W. Bauer, *Singularity: Scientific containers for mobility of compute*, PLOS ONE, 12 (2017), pp. 1–20.

[10] M. Merrill, W. Reus, and T. Neumann, *Arkouda: Interactive data exploration backed by Chapel*, in Proceedings of the ACM SIGPLAN 6th on Chapel Implementers and Users Workshop, 2019, p. 28.

[11] E. T. Phipps and T. G. Kolda, *Software for sparse tensor decomposition on emerging computing architectures*, SIAM Journal on Scientific Computing, 41 (2019), pp. C269–C290.

[12] The Trilinos Project Team, *The Trilinos Project Website*. https://trilinos.github.io/.

[13] U.S. Government, *Arkouda: NumPy-like arrays at massive scale backed by Chapel*. https://github.com/Bears-R-Us/arkouda.

[14] M. Wolf, D. Dunlavy, R. B. Lehoucq, J. W. Berry, and D. Bourgeois, *TriData: High performance linear algebra-based data analytics*, tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

TOWARDS VIABLE USE OF MACHINED LEARNED MODELS IN PHYSICAL SIMULATION

C. PEREYRA * AND M. A. WOOD †

Abstract. Within modeling and simulation efforts, data science driven methods are quickly becoming guides for the material selection for next generation fusion reactors. By utilizing high-accuracy quantum electronic structure (QM) calculations as training we are able to bridge the gap between QM and large scale (10^8 particles) molecular dynamics (MD) simulations for material science endeavors. However, training data is limited to a small number of atoms ($< 10^3$ particles) and testing the validity and stability of these ML models are a current challenge within MD. In this study we demonstrate methods of testing different classes of ML models within realistic production LAMMPS-MD simulations. Here we (i) perform benchmark speed tests of the MLIAP/SNAP packages, (ii) integrator stability tests, (iii) mechanical property prediction, and (iv) uncertainty quantification of these material properties in tungsten. Given these results we are able to provide methods of diagnosing reliable ML/NN models for future materials beyond accuracy represented in the training.

1. Introduction. Currently there are a number of computational methods available to scientists, differentiated by their various length and time scale capabilities. Though quantum mechanical (QM) calculations are the most accurate methods for acquiring activation energies, predicting spectra, producing kinetics, and determining correct geometric structures [3], these predictions come at great computational expense. Classical molecular dynamics (MD) simulations have many use cases ranging from biophysical function to solid-state mechanical property simulations [9, 15, 16]), while preserving linear computational scaling $O(N)$ [13] in the number of atoms, N.

With state of the art direct inversion solvers, density functional (DFT) algorithms are still limited to quadratic $O(N_e^2)$ or cubic scaling $O(N_e^3)$ depending on the system size [11,14]. Where N_e is the number of electrons. A question of whether we can merge linear scaling and high QM accuracy is a question that Bartok *et. al.* Gaussian Approximation Potentials (GAP) largely provides a great answer for [6]. This work among others ten years ago laid the foundation for machine learned interatomic potentials [7].

GAP potentials allow for an automatic reconstruction of a potential energy surface (PES) by fitting an atomically local model to many-bodied quantum mechanical energy and forces [6]. This atomic structure is transcribed into local neighborhood descriptors, which are the basis of machine learned potentials. These representations of local atom environments contain minimal invariant operations, called Bispectrum components.

An advancement on GAP is the spectral neighbor analysis potential (SNAP), that differs by assuming a linear relationship between atom energy and bispectrum components [17]. Both models obey permutational, reflectional, translational, and rotational invariance in each environment. This facet is absolutely necessary for a good PES to achieve the same ensemble properties for any structure by utilizing invariant representations of the interatomic potential. Energy should not be dependent upon arbitrary origin nor on permutational order of atom indices. Discerning differences between exact or similar structures is one of the main features of SNAP that enables highly accurate predictions of energy and forces.

Machine learned interatomic potentials have the capability to preserve the accuracy of $O(N^3)$ methods while maintaining their ideal linear scaling. This accuracy hinges on the search for good descriptors that bridge the gap between quantum and molecular dynamics and as such have yielded a wide array of interatomic potentials. Promising flavours include GAP, SNAP, MTP, ACE, etc [20]. Each of these ML descriptor models contain different

*University of California Davis, Mechanical and Aerospace Engineering, czpereyra@ucdavis.edu
†Sandia National Laboratories, mitwood@sandia.gov

choice of basis expansions (also known as descriptors or features) in order to map ab-initio calculations to MD simulation. Given this data driven method we are constructing, it begs the question does the model still represent physical properties of interest? **The focus of our efforts here are to provide meaningful evaluation metrics for SNAP and Neural Network interatomic potentials**. We demonstrate physical modeling capabilities for a previously published tungsten SNAP potential [19], and for an emerging neural network potential that is currently in development.

2. Interatomic Potentials. Testing and validating the models in this article focuses on two classes of machine learned potentials, (i) lasso regressed models and (ii) neural network trained potentials [1]. Many of the studies here compare the neural network (NN) functionals trained on unbiased training datasets that sample a maximal amount of atomic configuration space. Building on the work of Karabin *et. al.*[10] a unique training set that maximizes the information entropy in feature space was generated specifically to train complex NN models.

The motivation is that most improvements in scalar (RMSE, MAE, etc.) metrics of model performance do not account for the complexity of the training set, this entropy maximized set is ideal for comparing trained models in this regard. Additionally, these entropy maximized training sets can be generated in a highly automated fashion with very little user input, making them ideal for data-driven materials modeling.

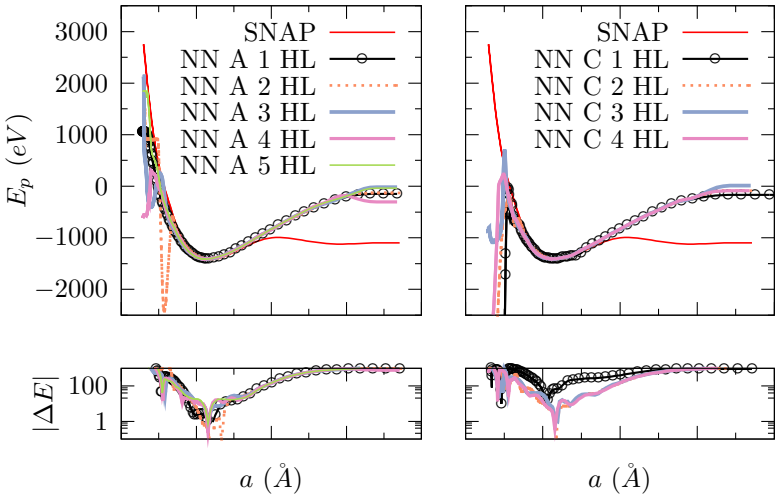


Fig. 2.1: Expansion and compression of a perfect BCC crystal for assessment of machine learned interatomic potentials as a function of lattice parameter a ($\mathring{A}$). Top panel represents the total energy of the system, while the bottom panel compares the total energy difference to linear SNAP. The NN shown on the left and right panels represent linearly increasing and expanding contracting networks respectively.

A highly optimized linear SNAP model is used as reference to compare NN results, since this simple ML model has been shown to match equilibrium material properties in tungsten [19]. Shown in Fig. 2.1 are slices through PES as a function of lattice parameter of a body-centered cubic(BCC) crystal. Linear SNAP models are taken as the ground truth when comparing the family of Neural Network models.

Qualitatively, the SNAP model demonstrates a local minimum approximately near the experimentally observed tungsten lattice constant $3.1652\mathring{A}$ signaling this potential's structural accuracy with respect to experiments. The details on the combined linear SNAP & ZBL model (see section 2.1) is deployed in LAMMPS MD code for physical simulation and characterization of integration stability in efforts to demonstrate use case limitations in MD.

Currently in development are robust neural network trained models. Fig. **??** shows a few NN potentials diverging under extrapolation conditions near the repulsion barrier. Despite this the NN models differ, $|\Delta E| = |E_{SNAP} - E_{NN}|$, very little from the linear model near the local minimum around $3.2\mathring{A}$. While accurate for crystalline and other simple geometries, linear models lack capability to capture subtle, non-linear features of the PES, so we turn to NN to acquire these features.

2.1. The SNAP Potential. Here we take a closer look at the SNAP model form to setup our later analysis. We begin by defining the local atom environments with the local atom density for a given central atom (2.1), at position $\boldsymbol{r}$ with respect to the domain's reference frame. The local environment is limited to atoms within a cut-off radius R_{cut} where the sum of density contributions to the central atom i extends to N_{cut} or the number of atoms within the radius R_{cut}. To emphasize a greater or lesser importance for a particular species on a neighboring atom i' weight $w_{i'}$ is added to the expression in equation (2.1), while f_c ensures that the density goes to zero smoothly at the cut off distance. This smoothing function is absolutely necessary for the numerical stability of these classical simulations, a point that will be discussed further in later sections herein.

$$\rho_i(\boldsymbol{r}) = \delta(\boldsymbol{r}) + \sum_{i'}^{N_{cut}} f_c(\boldsymbol{r}_{ii'}) w_{i'} \delta(\boldsymbol{r} - \boldsymbol{r}_{ii'}) \tag{2.1}$$

The density of atoms is transformed from real space to a space defined by expanding with spherical harmonics (2.2). Now the previous cartesian space is represented on a 3D sphere, where $\hat{\boldsymbol{r}}$ is on the unit sphere pointed in the direction of the original coordinates $\boldsymbol{r}$.

$$\rho_i(\boldsymbol{r}) = \delta(\boldsymbol{r}) + \sum_{i=0}^{\infty} \sum_{m=-l}^{l} c_{lm} Y_{lm}(\hat{\boldsymbol{r}}) \tag{2.2}$$

The basis coefficients c_{lm} are given by,

$$c_{lm} = \sum_{i} Y_{lm}(\hat{\boldsymbol{r}}_i)$$

Bartok *et al.* demonstrates this yields translational, rotational, and permutational invariance [4]. Furthermore, this representation can be translated into a four dimensional hypersphere. Coordinates r_0, θ, ϕ and now θ_0 represent the four dimensional space, where θ_0 is a measure of radial distance. The surface of the hypersphere's surface is defined as the magnitude of s_1, s_2, s_3, and s_4.

$$\begin{aligned} s_1 &= r_0 \cos(\theta_0) \\ s_2 &= r_0 \sin(\theta_0) \cos(\theta) \\ s_3 &= r_0 \sin(\theta_0) \sin(\theta) \cos(\phi) \\ s_4 &= r_0 \sin(\theta_0) \sin(\theta) \sin(\phi) \end{aligned}$$

It is important to note that at $r_0 = r_{cut}$ there is no angular component (for $\theta_0 \approx |\boldsymbol{r}|/r_0$), since the hypersphere is at the radial cutoff, this is essentially when we are at the south pole.

It is best to choose $r_0 > r_{cut}$ so there is greater representation in the radial coordinate. The atom density on the four dimensional hypersphere is defined using hyper-spherical harmonic functions $U^j_{mm'}(\phi, \theta, \theta_0)$ [12].

$$\rho = \sum_{j=0}^{\infty} \left(\sum_{m,m'=-j}^{j} c^j_{m'm} U^j_{m'm} \right) \tag{2.3}$$

Through a series of rotational operations on atom density using analogous four dimensional Wigner matrices the produced bispectrum array, equation (2.4) is a tensor product of three matrices, where coupling coefficients H are ordinary Clebsch-Gordan coefficients of SO(4) and * indicates complex conjugation. Inherently the bispectrum components are invariant under rotation, and characterize the strength of density correlations along the 3D sphere.

$$B_{j\ j_1\ j_2} = \sum_{m_1' m_1}^{j_1} c^{j_1}_{m_1' m_1} \sum_{m_2' m_2}^{j_2} c^{j_2}_{m_2' m_2} \times \sum_{m' m}^{j} H^{jmm'}_{j_1 m_1 m_1',\ j_2 m_2 m_2'} \left(c^j_{m'\ m} \right)^* \tag{2.4}$$

Half integer indices j, j_1, j_2 are defined for values $\{0, 1/2, 1, ...\}$ and m, m' are defined for $\{-j, -j+1, ..., j-1, j\}$. Since most combinations of are redundant the half integer indices can be reduced by symmetry. Non-integer combinations of indices $j + j_1 + j_2$ change sign so we must omit indices of this condition [4]. Reducing redundancy is also accomplished by using the "diagonal" elements $j_1 = j_2$, where also these indices are limited to $j, j_1, j_2 \leq J_{max}$. Ultimately the J_{max} factor may be prescribed so that we expand this basis expansion further and thus more accurately describe the atomic neighborhood [5]. In practice negative indices are accounted for by symmetry and their corresponding components are scaled or given correct signage to reduce computational complexity [17].

The SNAP potential energy (2.5) for atom i is a linear combination of coefficients β and bispectrum components B. More succinctly, the functional form of total SNAP potential E_{SNAP} is expressed as a weighted sum of linear coefficients $\beta \in \mathbb{R}^N$ and their corresponding bispectrum components (rows in bispectrum tensor) $B \in \mathbb{R}^{N \times k}$ shown in equation (2.5). For any choice of J_{max} we have k descriptors which reduces the $B_{j\ j_1\ j_2} \rightarrow B_k$ formalism.

$$E_{snap}(\boldsymbol{r}) = \sum_{i=1}^{N} \beta_0^{\mu_i} + \sum_{i=1}^{N} \sum_{k=1}^{K} \beta_k B_k^{\mu_i} \tag{2.5}$$

$$= N\beta_0^{\mu_i} + \beta \cdot \sum_{i=1}^{N} B^{\mu_i} \tag{2.6}$$

In terms of per-atom energies a more granular description of the potential is defined as the vector-matrix product of the linear coefficients β and the bispectrum array $B_k^{\mu_i}$ shown in equation (2.7). Then by applying the negative gradient of interatiomic distances, r_j, to the bispectrum components of equation (2.7) yields the per atom force [17]. These atomic forces are precisely what is needed for molecular dynamics simulations. Shifting the bispectrum components by B^i_{k0} makes sense so that by setting $\beta_0 = 0$ this constrains isolated atoms to

zero potential energy [18].

$$E^i_{snap} = \beta^{\mu_i}_0 + \sum_{k=1}^{K} \beta_k \left(B^{\mu_i}_k - B^{\mu_i}_{k0}\right) \tag{2.7}$$

$$F^i_{snap} = -\nabla_j \sum_{k=1}^{K} E^i_{SNAP} \tag{2.8}$$

$$= -\beta \cdot \sum_{k=1}^{K} \frac{\partial B^{\mu_i}}{\partial r_j} \tag{2.9}$$

More details for obtaining linear β_k coefficients are found in section 2.2.

In general SNAP can be trained for certain relative positions and cutoff distances. Additionally, these potential energy surfaces can be overlapped with empirical *reference* potentials shown in equation (2.10). This is necessary for capturing nearby neighbor repulsion for the SNAP tungsten potential used here [19]. In our case the ZBL potential is used to define the strongly repulsive terms since training data in this range was limited in displaying behavior at these distances. To prevent poor model extrapolation, we incorporate reference repulsion phenomenon.

$$E(\boldsymbol{r}) = E_{ref}(\boldsymbol{r}) + E_{snap}(\boldsymbol{r}) \tag{2.10}$$

In addition to the linear SNAP model we can use non-linear functionals in the form of a neural network that similarly uses bispectrum components as feature inputs. Part of this work is to compare linear SNAP to more complex model forms.

2.2. Machine Learning Optimization. In order to extract quantum mechanical accuracy attributes from electronic structure calculations, data science techniques have proved to be extremely useful for constructing IAPs for atomistic MD simulations.

The SNAP technique provides an optimal set of β_k coefficients that minimize the difference between predicted energy and forces (found in D) and the training set of energy and forces T. Note the training set contains energy and forces obtained from ab-initio calculations - in our case these are DFT computations carried out in VASP. D also contains descriptors of energy and force as determined by snap model energy and force equations (2.7)-(2.9).

Ridge Regression (Tikhonov [2]) method with a norm two loss function is used to fit the training set to corresponding coefficients β shown in equation (2.11).

$$\text{argmin}_\beta \left(\left\| \underset{n\times n}{\sigma} \cdot \underset{n\times 1}{T} - \underset{n\times n}{\sigma} \cdot \underset{n\times k}{\boldsymbol{D}} \beta) \right\| + \gamma_l \left\| \underset{k\times 1}{\beta} \right\|^l \right) \tag{2.11}$$

The exact composition of the descriptor matrix $D \in \{B; \partial B/\partial r\}$ is n training points deep, and k bispectrum components wide (multiplied by the number of element species). In practice, the top portions correspond to energies(bispectrum components), while latter rows contain forces(derivatives of descriptors). Since there are far more training points than descriptors, D is typically "tall and slim" which protects against over fitting.

Sources of error leading from training are minimized during the linear regression step (2.11). The norm is taken to be two and specific values of l enforce efficient sparse fits ($l = 1$) and over fitting protection ($l = 2$). The impact of imposing greater regularization does tend to increase RMSE, see Fig. 2.2 where $l = 2$.

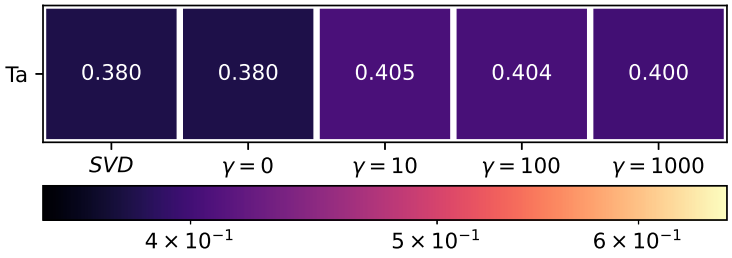


Fig. 2.2: RMSEs in atomic energy (eV/atom) of trained Tantalum potentials in FitSNAP for a regression penalty of order two and penalty weight γ is varied.

Determining β is essential for the linear SNAP model and ultimately for MD simulation use. In some cases certain descriptors are of greater importance than initially determined by the training set, so matrix σ of weighted scalars along the diagonal are used to increase the importance of certain training set elements and their corresponding rows along the descriptor matrix, as shown in equation (2.11).

2.3. Neural Network Models. The details of this class of ML potentials are not the main focus of the work presented in this article. However a high level description of the model forms used in this study are described in brief herein. The first architecture type is a step down architecture, while the second is an expanding and contracting architecture denoted 'A' and 'C', respectively. Additionally, each architecture are given a different number of hidden layers.

Each of these increasingly complex NNs were generated in order to capture a highly diverse training set of atomic configurations. As with polynomial SNAP models, training data for NNs is taken from density functional theory where the energies and forces of small($<$ 100 atoms) configurations are used as ground truth values. However, the complexity of NNs requires much more training data than other model forms. As part of another study [10] an *entropy maximized* training set deliberately found new configurations of unique bispectrum components to add to the training set. This is what the NN models are trained on, while a comparative linear SNAP model uses physically motivated configurations as training. Performance measures of LAMMPS use internal timers in the code and are measured for fixed and variable problem sizes on Sandia computing clusters. Numerical stability and physical modeling viability are demonstrated in sections 3.3 and 3.2.

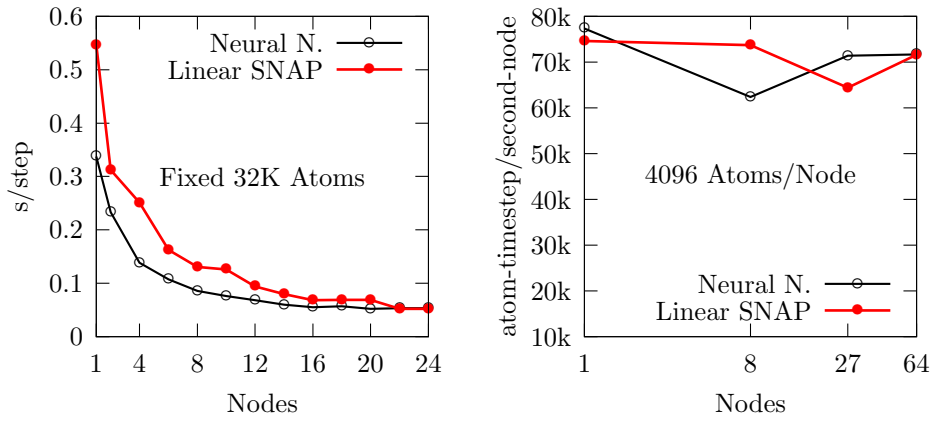
In particular we explore the top down architecture (A) which utilizes network nodes in descending fashion to define hidden layers. Studied and compared are their effects on physical modeling and overall numerical stability.

3. Results and Discussion. Recent updates in our open source parallel MD code base LAMMPS, allows for streamlined workflows for deploying custom trained potentials through the ML-IAP package. Additionally, supervised learning SNAP potentials are equally streamlined for distributed computing resources when deployed using the ML-SNAP package. See code base FitSNAP for generating SNAP potentials.

Two criteria that can be used to evaluate the computational effectiveness of a new potential are its' speed and scalability, while also evaluating its stability in a well known

time integrator. Proof that the ML-IAP and ML-SNAP package satisfy these criteria will be demonstrated in performative benchmarking section 3.1, and in a numerical stability section 3.2. Focusing on the connection that these potentials have in the field of computational materials science will be demonstrated in section 3.3 and 3.4. Finally, we consider the sensitivity of predictions made with respect to changes in ML trained models. For users looking to run any material science simulation, their choice of model and simulation method should consider each of these criteria.

3.1. Computational Performance. Benchmarking scalability of a simulation package is an essential practice for demonstrating parallel capabilities on large supercomputing resources. Historically the trend of allowable computational cost of a particular manybody potential has been shown to increase on an annual basis similarly to Moore's Law [13]. The trade off between computational cost and accuracy has enabled models to incorporate more intricate bonded interactions. This goal is naturally embedded in NN and SNAP models via the extendable basis expansion in bispectrum components. Choosing a more accurate form with higher computational cost can be remedied by parallel scaling capabilities as shown below.



(a) Strong scaling

(b) Weak scaling

Fig. 3.1: Performance benchmarking gathered for two different methodologies. (a) Fixed problem size, and (b) variable problem size, which is proportion to the resources used. SNAP $2j_{max} = 8$ and NN $2j_{max} = 6$ thus SNAP is roughly 1.25 times more expensive than NN.

For weak-scaling behavior we demonstrate parallel efficiency graphically by linearly increasing both atom and node count proportionally. Both the ML-IAP (Neural Network) and ML-SNAP (Linear SNAP) packages in LAMMPS show near horizontal lines, representing near 100% parallel efficiency. Downward trends represent slow down in the overall number of time-steps per second as a function of increased overhead due to increased nodal communication limitations.

Each of these cases were benchmarked on a Intel Broadwell E5 Chipset, as a CPU scalability test, these results are shown in Figures 3.1(a) and 3.1(b). We ran NVE(constant number of particles, volume and total energy) MD simulations for 100 time steps, each initialized by a BCC crystalline lattice configuration for homogeneous distribution of atoms in each allocated computational domain. These simulations are long enough to assess the

cost of occasional tasks such as neighbor-list building and internal modifications that are required by bispectrum calculations.

Fig. 3.1(b), a relatively small atom per node count (4k atom/node) was chosen yet amply demonstrates the communication overhead around 8 and 27 nodes for either models and is sensitive to the communication pattern of neighbor lists within LAMMPS. This figure demonstrates that the measure for timestep per second remains relatively stable as a function of proportional increases in atom per node.

3.2. Integration Stability. In addition to the finite size limitations of classical MD, the ability to simulate long-time scale phenomena is a critical consideration when choosing a model form. While a larger timestep interval can always be forced into a simulation, problems arise due to numerical stability and concerns over thermodynamic sampling of the simulation trajectory. The model form plays a critical role in both of these stability concerns, as is explored herein.

Again NVE simulations were setup to determine the expected stability of NN potentials compared with the stability of linear SNAP potentials for tungsten. Conservation of total energy in a system is a question of the numerical integration scheme, however layered neural networks represent an almost recursive functional form. So with greater complexity there may exist higher frequency modes of the PES that may yield greater integration instabilities. For example, consider the integration timestep, Δt, needed to resolve a high-frequency carbon-hydrogen bond versus a much lower frequency carbon-carbon bond. This is a serious concern for simulations of plasma facing materials where high energy particles collide with cold material, necessitating an adaptive time stepping method to preserve numerical stability. Here we investigate where an approximate threshold of stability exists for various timestep.

$$\sigma_{E_p} = \sqrt{\langle E_p^2 \rangle - \langle E_p \rangle^2} \tag{3.1}$$

The discrete integrator for the NVE ensemble is of the Verlet type which is second order accurate in kinetic measure from the central difference approximation dt [8]. Thus error translates into ensemble properties (ie. potential energy, temperature, enthalpy, etc.). The potential energy dependence upon timestep is demonstrated by Figure 3.2, where in all cases the behavior of average potential energy follows second order accuracy.

Each simulation in Figure 3.2 were run for 50 pico-seconds (and an initial equilibrium period of 30% of the total run time), the number of simulation steps are related to $N =$ sim. time$/\Delta t$. Simulations were initialized with an ideal BCC unit cell replicated along each Cartesian axis eight times equaling 1024 atoms generated within a 3D periodic boundary domain.

These results demonstrate the stability of each model form, albeit with differences in the fluctuation of the simulation based on the timestep used. This is an important consideration for end users as the amount of simulated time for a fixed resource budget is directly proportional to the timestep size.

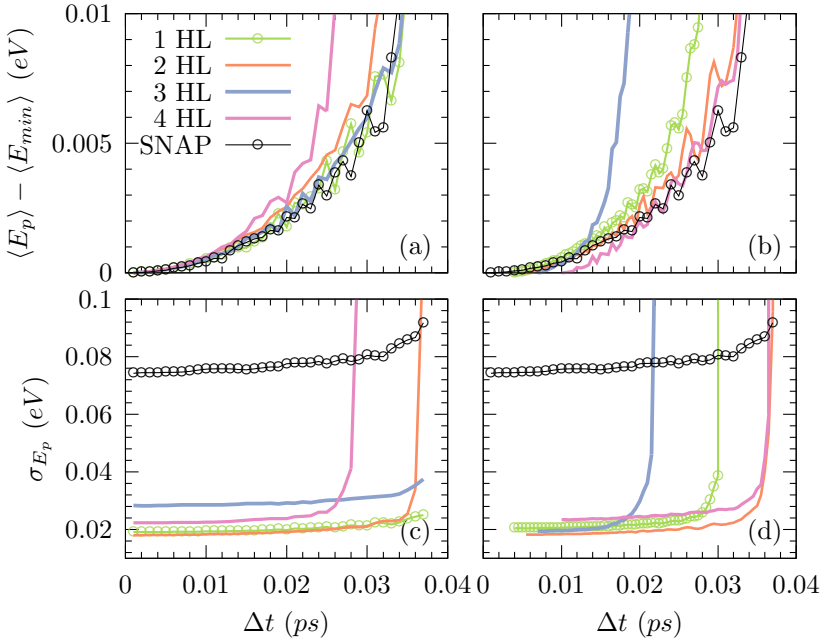


Fig. 3.2: Panels (a) and (c) represent the comparison between the linear NN architecture and SNAP; while, (b) and (d) represent the expansion and contraction NN architecture and SNAP. The panels above demonstrate $O(N^{\,2})$ behavior, while the fluctuation shown below, calculated by equation (3.1) are higher for SNAP.

3.3. Materials Modeling. Ultimately, practitioners of MD want to make predictions that can be corroborated or validated against experimental results, to that end much of the consideration of the accuracy of the model boils down to properties that can be simultaneously calculated at the nanoscale and macroscale. An example of such a property is the mechanical behaviour and failure of a material under tensile(volume increasing) stress loading. In this section we give brief descriptions and figures from two tungsten nanowire tensile-test simulations using the same linear SNAP potential shown in previous sections. These simulations were performed on large scale computing machines to model a 1.93 nm diameter nanowire of length 15.5 nm deforming with intermediate periods of thermal equilibrium. In total, 11k atoms integrated with a timestep size of $\Delta t = 0.002$.

Fig. 3.3 demonstrate the search for optimal strain rate values $e_{rate} = d\varepsilon/dt$, and frictional parameters α to use in the langevin integrator. Two strain rates $d\varepsilon/dt = 5\text{e}{-2}$ and $d\varepsilon/dt = 5\text{e}{-3}$ were tested to find an appropriate scale to match physical reality. Tungsten is known to be a brittle material at room temperature thus a more abrupt fracture(sudden decrease in stress) is better represented by Fig. **??**. This plot shows that while computationally more expensive, a much slower strain rate works well.

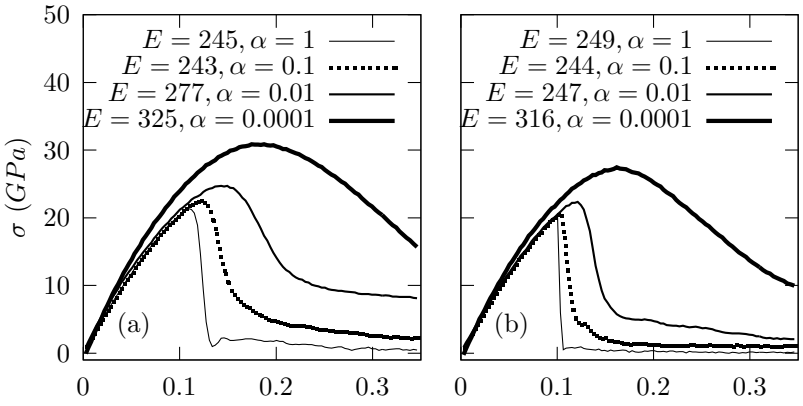


Fig. 3.3: Multiple tungsten tensile test NVT simulation for diagnosing unstable GJF-2GJ thermostat damping coefficients α. The strain rate dependence of the yield strain and stress are characterized in panels (a) and (b).

These simulations were performed at room temperature (T=300 K) with wait periods for equilibration in between each discrete step in the wire strain. The total number of simulation steps is given by the total simulation time divided by timestep width $N_{steps} = (N_{wait}\varepsilon/e_{rate})\, 1/\Delta t$.

Interestingly, linear SNAP can be used to perform predictive modeling for something it may not necessarily have been trained for as no configurations representing the brittle failure mode were present at training. Meaning the potential energy surface of SNAP is well mapped to experiments and DFT as demonstrated by the results above. This would lead us to believe SNAP can be used to predict any number of material properties beyond what was studied here.

These results open up a larger avenue for material scientists going forward to make new predictions of differing properties, materials, etc. and will bring insightful questions into how the model form accuracy varies from property to property. Finally in the future of applying this to exascale modeling, larger domains will only serve to help improve our predictions on the macroscopic level.

3.4. Uncertainty Quantification. While the advent of ever larger supercomputers has quelled many concerns about the finite-size and finite-time approximations in our MD simulations, questions about uncertainties in our PES models will persist. In particular, these data-driven models have brought new questions regarding uncertainty such as *how different would the model be if new training was added/subtracted?* To approximate these effects, we can study a perturbation added to regression coefficients. The impact of adding uncertainty to the linear coefficients is a question that we strive to answer in this section. In a sense we are attempting to approximate the training set perturbations made while constructing a linear SNAP model . By adding some uncertainty($\Lambda_k(t)$) to the β_k coefficients we are gleaning into how sensitive this tensile test model is to changes in the linear coefficients.

$$E^i_{snap} = \beta^{\mu_i}_0 + \sum_{k=1}^{K} \left(\beta_k + \Lambda_k(t)\right) \left(B^{\mu_i}_k - B^{\mu_i}_{k0}\right) \tag{3.2}$$

In terms of quantifying the robustness of physical results from nanowire tensile tests, we perform confidence interval calculations. This is one approach to quantifying the fluctuation in quantities of interest (i.e. mechanical stress and ultimate tensile strength) as we increase noise on the linear coefficients. Fig. 3.4 demonstrates two 95% confidence intervals (CI) around the stress-strain curves (using the NVT ensemble). The first is a test with one standard-deviation of Gaussian distributed noise to β_k, while the second represents two standard-deviations of Gaussian distributed noise to β_k.

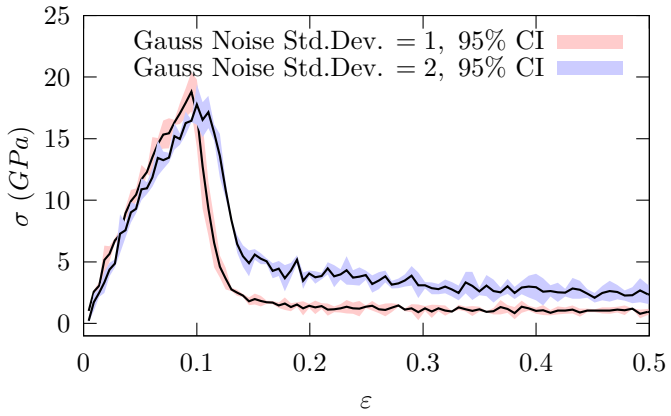


Fig. 3.4: Tensile test performed at strain rate $d\varepsilon/dt = 5 \times 10^{-2}$ ps yielding an elastic modulus of 280 GPa. Applied noise to linear SNAP coefficients results in differences in the stress-strain behavior. Confidence interval represents here a standard deviation in the predicted results sampled over an emulated larger pool of samples.

Computing the CI relies upon re-sampling techniques, in particular we used the 'bootstraping' method. An uncertainty in the mean values for each stress-strain curve can simply be computed by sampling a statistical distribution generated by numerous different tensile tests. Randomly drawing a stress from the modest collection of stress calculations (random selection is dictated by uniform distribution) at each strain is a repeatable process that helps shape the uncertainty. Specifically one-hundred replicates are generated.

Each tensile test is performed with a unique-random Gaussian distributed value $\Lambda(t)$ added to the linear SNAP coefficients. The exact initialization of these simulations is a BCC crystalline lattice with the [100] orientation as the loading direction.

At first adding one standard-deviation of Gaussian distributed noise to β_k did not alter the yield stress or yield strain values much. However, by adding two standard-deviations of Gaussian distributed noise seems to have changed the physical model quite a bit.

4. Conclusion. The potential energy surface of SNAP is well mapped to experiments and DFT as demonstrated by the results above. Similar to SNAP, the Neural Network potentials also demonstrate stability for long time integration evidenced by data. Given the promise of exascale computing resources, simulating larger domains with these highly accurate models will only serve to help improve our predictions and connections to the macroscopic materials of interest.

To achieve more physically accurate representation matching at the macroscale, it is important to setup larger more experimentally representative simulations. Capturing phenomenon from grain boundaries introduces more active slip systems to the mechanical re-

sponse and offers a more detailed view of actual mechanical properties.

The multi-step process laid out here of contrasting PES models and their viability in MD simulations should become standard practice for end users to justify the use of large-scale computational resources.

Acknowledgements. I sincerely want to thank Mitch Wood for mentorship and meeting virtually during the pandemic. My introduction to many different software tools and material science concepts would not have been possible without this mentorship collaboration. I also want to thank Aidan Thompson, David Montes, Mary Alice Cusentino, and Julien Tranchida for their valuable insight and support on this project.

References.

[1] *Entropy maximized nn training set. these models are to be made available via personal communication. details of training and model form are to be described in a forth coming paper.*

[2] *Numerical methods for the solution of ill-posed problems*, Mathematics and its applications ; v. 328, Kluwer Academic Publishers, Dordrecht ;, 1995 - 1995.

[3] *Chemical applications of density-functional theory*, ACS symposium series, 629, American Chemical Society, Washington, DC, 1996.

[4] A. P. Bartók, R. Kondor, and G. Csányi, *On representing chemical environments*, Phys. Rev. B, 87 (2013), p. 184115.

[5] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, *Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons*, Phys. Rev. Lett., 104 (2010), p. 136403.

[6] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, *Gaussian approximation potentials: the accuracy of quantum mechanics, without the electrons*, Physical review letters, 104 (2010), pp. 136403–136403.

[7] J. Behler and M. Parrinello, *Generalized neural-network representation of high-dimensional potential-energy surfaces*, Phys. Rev. Lett., 98 (2007), p. 146401.

[8] N. Grønbech-Jensen and O. Farago, *A simple and effective verlet-type algorithm for simulating langevin dynamics*, Molecular Physics, 111 (2013), pp. 983–991.

[9] M. F. Horstemeyer, M. I. Baskes, and S. J. Plimpton, *Computational nanoscale plasticity simulations using embedded atom potentials*, Theoretical and Applied Fracture Mechanics, 37 (2001), pp. 49–98.

[10] M. Karabin and D. Perez, *An entropy-maximization approach to automated training set generation for interatomic potentials*, The Journal of Chemical Physics, 153 (2020), p. 094110.

[11] G. Kresse and J. Furthmüller, *Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set*, Phys. Rev. B, 54 (1996), pp. 11169–11186.

[12] A. V. Meremianin, *Multipole expansions in four-dimensional hyperspherical harmonics*, Journal of Physics A: Mathematical and General, 39 (2006), pp. 3099–3112.

[13] S. J. Plimpton and A. P. Thompson, *Computational aspects of many-body potentials*, MRS bulletin, 37 (2012), pp. 513–521.

[14] G. E. Scuseria, *Linear scaling density functional calculations with gaussian orbitals*, The Journal of Physical Chemistry A, 103 (1999), pp. 4782–4790.

[15] R. Service, *Google's deepmind aces protein folding*, Science (American Association for the Advancement of Science), (2018).

[16] M. J. Stevens, J. H. Hoh, and T. B. Woolf, *Insights into the molecular mechanism of membrane fusion from simulation: evidence for the association of splayed tails*, Physical review letters, 91 (2003), pp. 188102–188102.

[17] A. Thompson, L. Swiler, C. Trott, S. Foiles, and G. Tucker, *Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials*, Journal of Computational Physics, 285 (2015), pp. 316–330.

[18] M. A. Wood, M. A. Cusentino, B. D. Wirth, and A. P. Thompson, *Data-driven material models for atomistic simulation*, Phys. Rev. B, 99 (2019), p. 184305.

[19] M. A. Wood and A. P. Thompson, *Quantum-accurate molecular dynamics potential for tungsten*, 2017.

[20] Y. Zuo, C. Chen, X. Li, Z. Deng, Y. Chen, J. Behler, G. Csányi, A. V. Shapeev, A. P. Thompson, M. A. Wood, and S. P. Ong, *A performance and cost assessment of machine learning interatomic potentials*, (2019).

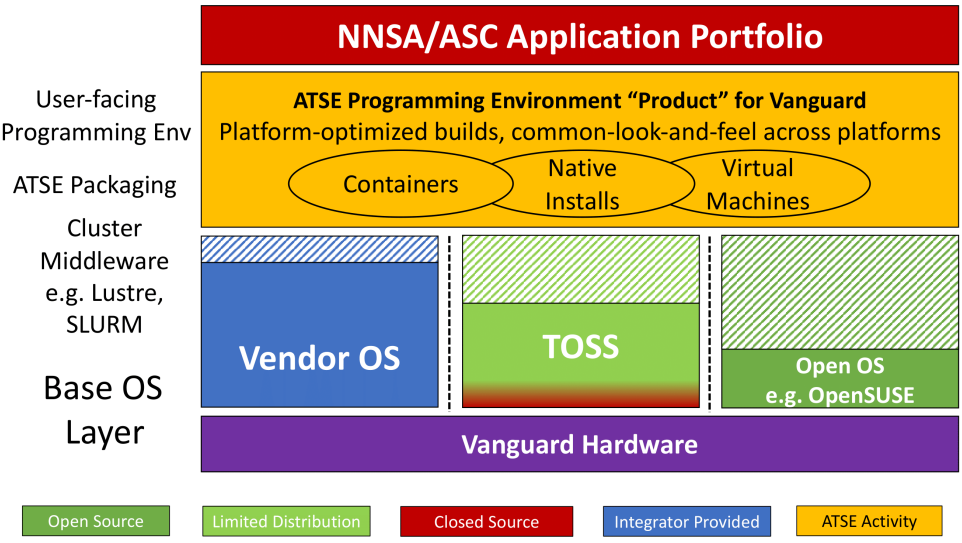
VERIFICATION AND PERFORMANCE TESTING OF THE ADVANCED TRI-LAB SOFTWARE ENVIRONMENT

CARSON WOODS* AND MATTHEW L. CURRY†

Abstract. The Advanced Tri-lab Software Environment (ATSE) is a complex software stack, with a variety of compilers, optimizations, and package inter-dependencies [4]. Previous efforts to make the ATSE environment portable allow ATSE to run on a variety of HPC architectures (e.g., AArch64 with and without NVIDIA GPUs, Fujitsu A64FX, Intel processors supporting AVX-512), which makes verifying this environment challenging and time-consuming. Using ReFrame, a unit-testing framework for HPC environments, we have constructed a series of tests that verify a broad range of the programs and libraries that are provided by ATSE. Coupled with work to port the ATSE environment to Spack (SpATSE), we aim to create a software environment that can be quickly installed and verified on new systems with architecture- and accelerator-specific optimizations, easing the path to validation with applications.

1. Introduction. The Advanced Tri-Lab Software Environment (ATSE) is a software stack built for Vanguard HPC platforms [4]. To support a range of needs from a variety of users, ATSE includes a combination of compiler toolchains, libraries, utilities, and other tools. In addition to the wide range of tools that are provided, they are often provided for several MPIs and compilers, with many libraries being built as often as six times. The ATSE software stack runs on top of vendor and OS-provided software. Once installed, user applications can be built against and use all of the provided libraries, tools, and compilers that ATSE provides. A diagram of this software stack architecture is shown in Figure 1.1.

Fig. 1.1: Position of ATSE environment within a full software stack [4].



Creating, maintaining, and modifying such a complex environment is a time-consuming task. Previously, we created a portable Spack implementation of this environment (SpATSE) to simplify the installation and maintenance of the ATSE environment on multiple sys-

*The University of Tennessee at Chattanooga: Sim Center, carson-woods@utc.edu
†Sandia National Laboratories, mlcurry@sandia.gov

tems [7]. This implementation of the environment is functional and provides near-feature parity with the more traditional RPM implementation of the ATSE environment.

However, despite our changes, maintaining these environments can still be a time-consuming process. Provided software has to be trusted by users to be functional, performant, and correct. Spack introduces additional complexities and ambiguities that can work against these goals. For example, Spack is capable of automatically determining dependent packages and building compatible versions, but these derived dependencies often do not match explicitly specified versions. This creates duplicate packages within a single installation. These duplicates may be missing microarchitecture-specific optimizations, support for specific hardware, or package options (i.e., Spack variants) that ensure correctness or compatibility. Further, some packages may build correctly but are ultimately nonfunctional because of a specification problem. SpATSE also presents a much broader surface for problems to be introduced. Previously, ATSE was built using an RPM-based build farm that heavily leveraged OS-provided packages. In contrast, Spack, and thus SpATSE, builds many of these packages independently. These issues motivate a need for environment verification.

Software verification is more than a loosely defined functionality test. Verification, along with validation, comprise the term "V&V" which is defined as "a collection of analysis and testing activities across the full life cycle" of a piece of software [6]. In a report on verification and validation, Wallace and Fujii define verification as: "evaluating software during each lifecycle phase to ensure that it meets the requirements" [6]. Our testing pipeline aims to verify an installation of ATSE to ensure that we are meeting specified requirements in each installation. Validation is completed as a separate step, by building and running user applications like SPARC [5] and EMPIRE [1] against SpATSE.

We are developing a regression-testing framework to verify the ATSE environment. This will allow for rapidly testing selective components of the environment to improve the speed of making changes, as well as allowing for a quick first qualification when setting up completely new instances of ATSE/SpATSE on systems with new hardware architectures.

2. Methods. Many methods can be used to thoroughly test software environments. We considered manually writing a collection of scripts that would do the testing we needed. We also considered the package testing functionality within Spack, which allows for testing Spack-installed packages [2]. We decided against using Spack for the testing, due to Spack's tests being limited to whatever the package supported. Writing and maintaining parallel package recipes with custom test infrastructure is an alternative; however, it would place an extended burden on maintaining SpATSE and the testing pipeline. In the end, we selected Reframe, a unit testing framework designed for the verification of HPC environments [3].

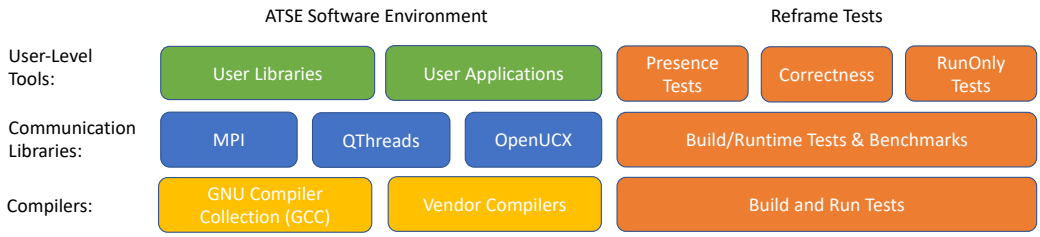
Reframe is a full-featured unit-testing framework. Reframe tests are written as Python classes and are customizable [3]. While Reframe offers many different types of tests by default (BuildOnly, RunOnly, Build and Run, etc.), any of these tests can be modified to meet unique requirements. Reframe simplified the development process of our testing pipeline. Out of the box, Reframe offered substantial functionality for testing various compilers and many common build systems easily. It also supports differentiating between systems and hardware partitions, so that tests that required certain hardware requirements would only be run on compatible systems.

Reframe also integrates with common module systems, and allows for easily loading specific modules before a test. Not only does it provide lmod support, but it can directly load packages that are installed via Spack. This is vital due to the way that Spack generates module files. When a package is installed via Spack, it generates a unique hash from the various build attributes for that package (compiler, variants, version, package name, etc.) [2]. These

hashes can be used to differentiate between two packages with slight differences. Module files may incorporate Spack's package hashes to differentiate between different installations of the same package. Unfortunately, if changes are made to the environment or the environment is installed on another system, these hashes will change. As such, they are currently not viable to use for loading packages for tests. Because of this, Reframe's direct Spack integration is useful for loading the specific package needed for a test in a more portable way.

Another significant feature of Reframe that we wanted to leverage was performance testing. In addition to functionality testing, performance testing is a vital part of the environment verification process. There are many possible points of performance problems that can be uncovered via testing: missing microarchitecture optimizations, enabling options for networking interconnects failing silently, etc. These performance issues are often more insidious than traditional errors, as they are difficult to detect and appear only when an environment is needed for a job where performance is necessary. As a result, it was important for us to include performance testing when building our test pipeline.

Fig. 2.1: General mapping of ATSE components to planned Reframe tests.



Long term, we aim for our testing pipeline to cover most aspects of the ATSE environment. Figure 2.1 showcases the eventual goal of the testing pipeline. The figure offers a general mapping of what types of tests will eventually cover what categories of application. While the scale and scope of tests that we are developing can be extensive, we are focusing our initial efforts on basic functionality and software-presence tests. Specifically, we are making sure that every package can be used in some basic way. For libraries, this is compiling against the library on a variety of compilers and testing functionality of a program. Applications such as CMake are tested by downloading and building another package that implements CMake as its build system. For performance testing, we are using a range of benchmarks (Intel MPI Benchmarks, OSU Micro Benchmarks, etc.) to validate the performance of MPI's collective operations across multiple compute nodes. Writing performance tests in Reframe did not involve creating new benchmarks, but instead leveraged and collated commonly used benchmarks into easily launched tests, where results could be made available in a standardized way.

These tests were developed on Stria and were also run on Mayer to prove the portability of tests. Since both the Stria and Mayer compute platforms share many hardware similarities and run ATSE, this was a basic test of portability. In the future, tests that rely on specific hardware features such as CUDA support could be written to only run on certain machines that have a specific type of partition in the pre-defined Reframe settings.

Reframe's `settings.py` file is a mechanism provided by Reframe to customize what type of tests can be run on what machine. It also allows for complete customization of supported compilers, what information is logged, and which hardware resources can be exposed to tests.

2.1. Reframe's System Structure. Before building tests of any type, creating a detailed Reframe settings file is a necessary prerequisite. Besides controlling many of the more advanced features of Reframe such as logging customization, the `settings.py` file's primary function is to define systems in the context of Reframe and the types of tests that can be run on those systems. They are structured in such a way that we can have a single settings file for all of our HPC platforms. In the context of Reframe settings, there are three major components to consider: systems, partitions, and programming environments. A system is the highest level of these components. A Reframe "system" refers directly to a hardware platform, and using this platform, specific hostnames can be specified, a module system can be selected, and partitions can be defined. If a system is hostname limited (does not use "*"), then tests for that system will only be launched on a system that matches hostnames. Partitions are the next component of a Reframe test. Partitions define specific portions of a system to be run on. Specifically, they allow tests to differentiate between launching locally, through various job schedulers, and via various parallel launch commands (i.e. mpirun, etc.). Additionally, the number of max jobs can be defined, and additional hardware resources (job time, nodes, etc.) can be specified. Finally, programming environments for that partition can be selected. Programming environments are the smallest component of a Reframe test. They define a list of compilers that can be used, the path to their executables, and (if necessary) the modules that must be loaded to use them.

2.2. Anatomy of a Test. There are many different types of Reframe tests so an illustrative example is included to describe a subset of the available features. The test that is being described can be found in Appendix 4. The test uses a custom Python decorator `@rfm.simple_test` to indicate that it is a test to be run. The `valid_systems` variable is set to any system using a "*" and a ":mpi" specifies that the system must have an MPI partition. The `valid_prog_environs` variable specifies that only programming environments named "openmpi3" and "openmpi" should be selected. The final setup step is to make the program aware of the correct source file through the `sourcepath` variable.

The next major component of a Reframe test is a pipeline hook function. Reframe tests are broken into predefined stages, and additional steps can be inserted into the test process via pipeline hooks. For each stage of a test, there can be a "run_before" or "run_after" hook that will run custom test code immediately preceding or following that stage of the test.

Continuing through the example test, we have a "run_before" run function which passes `-n 4` to the launcher command for the job before the test is run. Finally, there is a "run_before" sanity function which defines the sanity pattern for the test using a combination of Reframe provided functions and regex parsing.

Reframe tests, both regression and performance, rely on "sanity patterns" to determine the pass/fail state of a test [3]. In theory, this means that nearly any metric can be either captured or evaluated so long as it follows a standard format when being output. In practice, many of the sanity expressions we wrote were simple output checks to ensure that the expected output was or wasn't present depending on the type of test being run. Performance tests followed a similar pattern, instead of timing anything themselves, they rely on self-reported metrics from whatever is being tested, or user-implemented metrics if necessary. Performance tests simply capture and log metrics rather than using them to determine if there is a pass/fail status. Using performance tests, our goal is to collate the important performance metrics so that they can be run and gathered easily rather than running a

wide range of benchmarks by hand.

3. Discussion. The testing framework, while still incomplete, is already proving to be useful. As the environment changes, it is easy to launch new tests that can identify possible problems. It is not possible to determine how long it takes to run the entire test suite, as it is often machine dependent and can be affected by compute node queue times, but the tests take less than an hour to run on Stria when there are no blocking processes inhibiting runtime.

3.1. Challenges. Reframe is a significant improvement to writing similar tests entirely from hand, however, it is not without some trade-offs. One major performance bottleneck is parallelism between tests. Currently, Reframe will launch a test in parallel only when a test is being run on multiple programming environments within a partition [3]. For example, if a simple C compiler test is being run on both Slurm and non-Slurm partitions, jobs will run on one partition before launching on the other. Tests within a partition (such as testing on different compilers) can be launched in parallel if supported, but this limits parallelism despite larger quantities of system resources being available for use. As mentioned previously, the runtime of the current tests is not prohibitively long, and the testing pipeline offers a massive increase in time saved validating environments. However, as the test pipeline continues to grow to cover larger amounts of the environment and package functionality, maintaining performance is desired. One mechanism that could be implemented to correct this is manually assuming control of launching tests. It would be possible with custom scripting to launch jobs across multiple tests in parallel however, this has not been thoroughly explored as it is not yet needed.

It is also challenging to gather performance data from tests with dynamic output. Benchmarks such as the Intel MPI Benchmarks have performance metrics that scale to the physical hardware capabilities of the machine. As a result, collecting the proper metrics on a specific machine while also maintaining portability for yet untested platforms requires significant work. It is certainly still possible and can be done, however, it is more complex than collecting data from tests with deterministic output.

3.2. Early Results. As mentioned, the effort to build a comprehensive and exhaustive testing pipeline is not yet complete. Despite this, some early results reinforce pre-existing beliefs. We successfully validated our various compiler toolchains and can ensure that they can be reliably used to build necessary software. Additionally, we were able to validate many of the auxiliary libraries and applications that are included in ATSE. A major component of this is ensuring the functionality of OpenMPI across hardware partitions and underlying compilers.

One unexpected result was the significant differences between the ARM compiler and the GNU compilers. When building applications that linked against third-party libraries, build processes that worked for the GNU compilers often failed on the ARM compilers. This introduced additional complexity in building tests which supported both compilers and their separate build processes.

4. Conclusions. Building a programming environment testing pipeline offers not only a significant quality-of-life improvement when developing ATSE and SpATSE, but it encourages and enforces a requirement of correctness that was not previously feasible. Facilitating testing and providing a framework for continued and future tests to be written will be a key focus point in environment development moving forward. Creating new tests in line with new features and software that could be added to ATSE would be a quick process and could easily keep the test pipeline in line with changing and expanding capabilities of ATSE in the future. This testing capability will also facilitate the long-term goal of being able to

provide a portable and reproducible software environment that can be widely relied upon as a verified computing platform [7]. The testing framework developed by this work will allow for faster iterations on the ATSE computing environment, will help to ensure correctness for end-users, and ease the burden of environment maintenance across multiple HPC platforms.

Appendix: Reframe Test. The following is an example Reframe test that could be used to test if MPI can be initialized.

```
@rfm.simple_test
class Init_MPI(rfm.RegressionTest):
    """
    Test if MPI can be initialized by running
    a MPI_Init and hello world program on openmpi3 and openmpi4
    """
    valid_systems = ['*:mpi']
    valid_prog_environs = ['openmpi3', 'openmpi4']
    sourcepath = 'mpi_hello_world.c'

    @run_before('run')
    def set_core_opts(self):
        self.job.launcher.options = ['-n', '4']

    @run_before('sanity')
    def set_sanity_patterns(self):
        num_messages = sn.len(sn.findall(r'Hello from processor \d+ of \d+', self.
    stdout))
        self.sanity_patterns = sn.assert_eq(num_messages, 4)
```

Listing 1: MPI_Init Reframe Test

References.

[1] M. Bettencourt, K. Cartwright, and G. Laity, *EMPIRE: A revolutionary modeling tool for agile design*, 2021.

[2] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral, *The Spack package manager: Bringing order to HPC software chaos*, in SC15: International Conference for High-Performance Computing, Networking, Storage and Analysis, Los Alamitos, CA, USA, nov 2015, IEEE Computer Society, pp. 1–12.

[3] V. Karakasis, T. Manitaras, V. H. Rusu, R. Sarmiento-Pérez, C. Bignamini, M. Kraushaar, A. Jocksch, S. Omlin, G. Peretti-Pezzi, J. P. S. C. Augusto, B. Friesen, Y. He, L. Gerhardt, B. Cook, Z.-Q. You, S. Khuvis, and K. Tomko, *Enabling continuous testing of HPC systems using ReFrame*, in Tools and Techniques for High Performance Computing, G. Juckeland and S. Chandrasekaran, eds., Cham, 2020, Springer International Publishing, pp. 49–68.

[4] J. H. Laros, K. Pedretti, S. D. Hammond, M. J. Aguilar, M. L. Curry, R. Grant, R. J. Hoekstra, R. A. Klundt, S. T. Monk, J. B. Ogden, S. L. Olivier, R. D. Scott, H. L. Ward, and A. J. Younge, *FY18 L2 milestone #8759 report: Vanguard Astra and ATSE – an ARM-based advanced architecture prototype system and software environment*, (2018).

[5] J. Smith and J. Snyder, *Sandia parallel aerodynamics reentry code (SPARC): the future of production and research aerodynamics*, 2021.

[6] . Wallace, Dolores R and R. U. Fujii, *Software verification and validation: An overview*, IEEE Software, 6 (1989), pp. 10–17.

[7] C. Woods and M. L. Curry, *Implementing a common HPC environment in a multi-user Spack instance*, SC '19 HPCSYSPROS', Nov. 2019.

III. Applications

Articles in this section discuss the application of computational techniques to simulate physical systems.

1. *Alsup* and *Catanach* employ **Bayesian Optimal Experimental Design** to optimize the configuration of a network of sensors, maximizing the expected information gain.
2. *Callahan* and *Catanach* deploy an importance sampling method to approximate the **Expected Information Gain** used in the optimal placement of sensors.
3. *Clements, Geraci,* and *Olson* use a non-deterministic sampling approach for uncertainty quantification for **Monte Carlo Radiation Transport Solvers** through variance deconvolution strategies.
4. *Frink, Campbell, Smith, Baczewski,* and *Albash* analyze the accuracy and efficiency of a variety of neural networks for simulating the ground states of small molecules with applications in **Quantum Chemistry**.
5. *Gaiewski, Sockwell, Connors,* and *Bochev* present a simplified model of the ocean-atmosphere system based on the **DOE E3SM Model**. This new model enables rapid testing and prototyping of various coupling methods for the ocean-atmosphere system.
6. *Gerot* and *DiPietro* determine necessary preprocessing steps needed to use **Precipitation Data** in optimal transport models, and provide initial results for an optimal transport based model for precipitation impact.
7. *Hoy, Tezaur,* and *Mota* develop a novel simulation method using the Schwarz alternating method for simulating **Contact within Mechanical Systems**.
8. *Lennon* and *Rajamanickam* leverage modern machine learning techniques in an effort to produce surrogate models that transfer and adapt quickly for **Molecular Dynamics** applications.
9. *Torchinsky* and *Taylor* examine four modifications to the piecewise-parabolic method used in the **Atmospheric Dynamical Core** component of the Energy Exascale Earth System Model. These modifications target the treatment of domain boundaries and aim to eliminate noise.
10. *Valaitis* and *Maupin* implement a neural network to predict **Single Particle Motion in Plasmas** in an effort to provide a computationally efficient alternative to the Boris-Bunemann algorithm.

J.D. Smith
E. Galvan

November 1, 2021

UNCERTAINTY QUANTIFICATION FOR EXPECTED INFORMATION GAIN ESTIMATES

TERRENCE A. ALSUP* AND THOMAS A. CATANACH†

Abstract. Sensor networks are a ubiquitous tool for detecting events in many applications. Here a network of sensors is deployed to gather data for estimating an event's properties. Bayesian optimal experimental design (OED) provides a framework for optimizing the configuration of the network of sensors. In particular, the sensors are configured to maximize the expected information gain (EIG). For a given sensor configuration, the EIG takes the form of a nested expectation and therefore is computationally intensive to accurately estimate by sampling. For the Bayesian optimization of the network we use a sub-division method to estimate the uncertainty in the EIG estimate. We find that, in practice, the proposed sub-division method provides a close approximation to the true mean-squared error. In this work we will consider both the general problem of robust EIG estimation and its specific application to Bayesian OED for seismic monitoring networks.

1. Introduction. Seismic monitoring networks are an important tool for detecting seismic events such as earthquakes and nuclear tests. An array of seismic sensors is deployed to gather data that is then used to estimate an event's properties such as origin location and magnitude. Different configurations of the sensor network will determine the data that is observed for the same event. Since in practical applications the events are unknown ahead of time, it is prudent to configure the sensors to perform well for typical events by maximizing the expected information gain (EIG), which intuitively, is the average amount of additional information obtained after observing an event for a given sensor configuration. A Bayesian optimization procedure is then used to determine the optimal sensor network configuration. For this technique, one must know, or be able to estimate, the EIG. Moreover, it is advantageous to also quantify the uncertainty in the estimate itself. Sensor placements that yield high-variance estimators indicate regions where additional sensors would be beneficial to improve estimates. This work proceeds as follows. First we outline the Bayesian approach to optimal experimental design in the context of seismic monitoring with sensor networks. Second, we describe a procedure to estimate the EIG and its uncertainty, quantified by the mean-squared error of the estimator. Finally, we present numerical results on how our uncertainty estimation procedure performs.

2. Bayesian optimal experimental design for seismic monitoring. In this section we start by describing the underlying Bayesian inference problem of inferring an event's properties in Section 2.1. In Section 2.2 we describe optimal experimental design problem, where we want to choose the configuration of the network of sensors.

2.1. Bayesian inverse problem. Sensor networks can monitor seismic events through a variety of measurements, such as arrival time and amplitude of a seismic phase, at each individual sensor's location. Let $\theta \in \Theta$ denote the parameters or properties of the seismic event that we are interested in inferring, such as the location of the event's origin and its magnitude, with $\Theta \subset \mathbb{R}^d$ being the parameter domain or space of possible events. In this work we consider a $d = 4$ dimensional parameter defining the event

$$\theta = (\text{longitude},\ \text{latitude},\ \text{depth},\ \text{magnitude})\,.$$

The event origin time could be an additional variable, but we instead treat it implicitly in the likelihood by marginalizing it out in order to reduced the dimension of the parameter

*Courant Institute of Mathematical Sciences, alsup@cims.nyu.edu
†Sandia National Laboratories, tacatan@sandia.gov

space. Figure 2.1 shows an actual physical domain in Utah where a dense network of seismic sensors were temporarily installed as part of the 15 year rolling Transportable Array (TA) deployment that eventually covered the entire Continental United States.

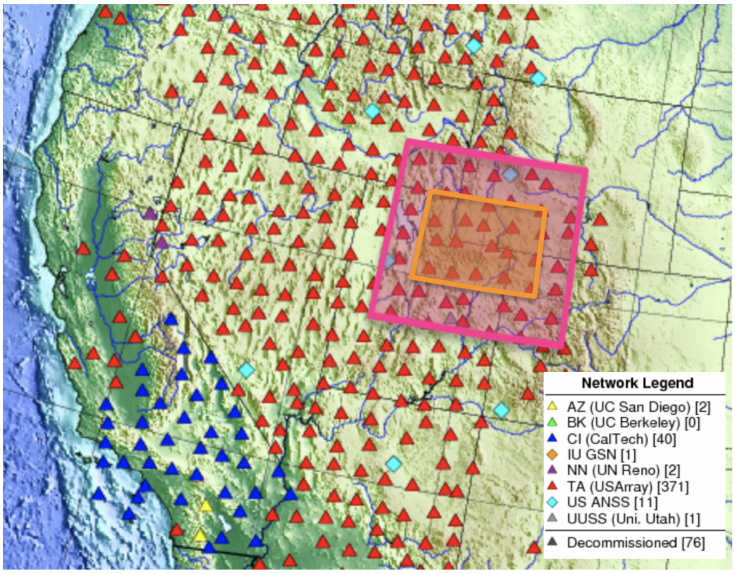


Fig. 2.1: The array of sensors over Utah used for seismic monitoring. The pink region indicates the region used to gather sensor data for the parameters of the likelihood models. The orange region is the domain in which we monitor seismic events.

A physics-based model for how seismic events propagate through the earth can be combined with a model for the sensors themselves and the uncertainty of the measurements in order to derive a likelihood model $p(\mathcal{D} \mid \theta)$, which is the probability of observing some data $\mathcal{D} \in \mathbb{D}$ conditioned on a particular event $\theta \in \Theta$. In this work we consider a likelihood model comprised of two components: a detection model for seismic phases and an arrival time model. The seismic detection model is a logistic regression model similar to that used by NET-VISA [1]. The arrival time model is largely based on the IASP91 Earth velocity model. An uncertainty model for the arrival time is constructed using the Crust1.0 [5] shallow Earth model and the TauP [3] travel time prediction software package. Correlations between travel times to different sensors can also be incorporated in the model. We refer to [2] for a detailed description of the likelihood model.

The likelihood model also presents a way to generate synthetic data from a given event. To infer θ, one method would be to maximize the likelihood to determine the most likely event. However, oftentimes we have additional prior information about where the seismic events are likely to occur before-hand, typically from historical data. This can be incorporated through the use of a prior distribution $p(\theta)$ over Θ. Thus, through Bayes' formula, we can formulate the posterior $p(\theta \mid \mathcal{D})$ over events given the observed data $\mathcal{D}$, which depends on the sensor network configuration, as illustrated in Figure 2.2. Constructing the posterior distribution also gives us uncertainty about the parameter estimate which is important for high consequence decision making.

2.2. Bayesian optimal experimental design. In Bayesian optimal experimental design one would like to configure the network of sensors to be as informative as possible. The

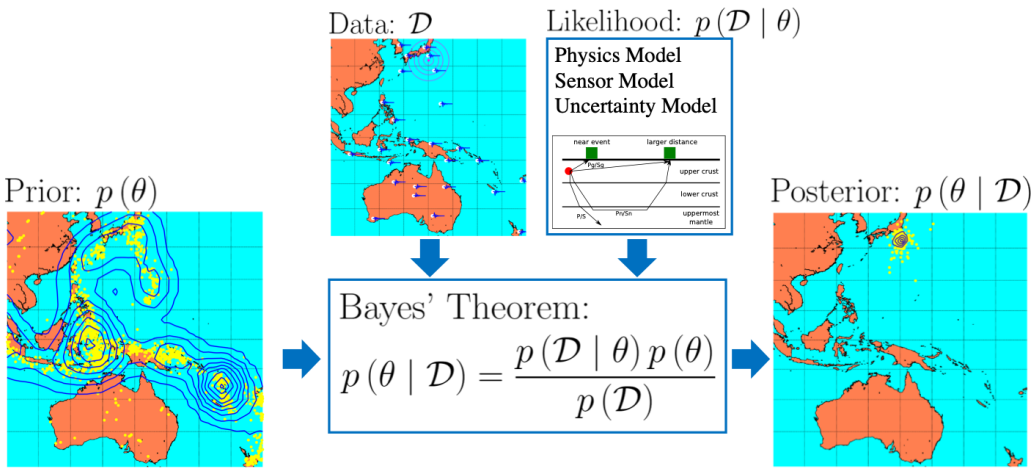


Fig. 2.2: The Bayesian approach to seismic monitoring combines prior information with a likelihood model to update the distribution over possible events.

posterior $p(\theta \mid \mathcal{D})$, which depends implicitly on the sensors through the observed data $\mathcal{D}$, encapsulates the information obtained by sensors' measurements. To quantify the performance of the sensors for given observed data $\mathcal{D}$ we use information gain or the Kullback-Leibler (KL) divergence from prior $p(\theta)$ to the the posterior $p(\theta \mid \mathcal{D})$

$$\mathrm{KL}\left[p(\theta \mid \mathcal{D}) \mid\mid p(\theta)\right] = \int_{\Theta} p(\theta \mid \mathcal{D}) \log\left(\frac{p(\theta \mid \mathcal{D})}{p(\theta)}\right) \mathrm{d}\theta\,. \tag{2.1}$$

The KL divergence is a useful measure for determining how informative some observed data is. Ideally, the data will be such that the posterior is very different from the prior and the information gain (i.e. KL divergence) will be large. Because the events are random, and hence the observed data $\mathcal{D}$ is as well, one must look at many different hypothetical events to see what the typical information gain is for a fixed network of sensors. This is made precise by the expected information gain

$$\mathcal{I}(\mathcal{S}) = \mathbb{E}_{\mathcal{D}}\left[\mathrm{KL}\left[p(\theta \mid \mathcal{D}) \mid\mid p(\theta)\right]\right]\,. \tag{2.2}$$

On the right-hand side of Equation (2.2) the dependence on the sensors $\mathcal{S}$ is again implicit through the data $\mathcal{D}$. The optimal experimental design problem is to maximize the expected information gain over a set $\mathbb{S}$ of all possible sensor configurations to consider

$$\mathcal{S}^* = \operatorname*{argmax}_{\mathcal{S}\in\mathbb{S}} \mathcal{I}(\mathcal{S})\,. \tag{2.3}$$

This approach of using expected information gain as a design criterion was first proposed in [6] and has been widely used since [4, 8, 10]. Note that the optimization problem (2.3) is different from the problem of estimating the expected information gain, and has been approached from various directions. In general one would like to use a derivative-free optimization algorithm such as Bayesian optimization [2, 10], but stochastic approximation

algorithms which approximate derivatives have also been used [4]. We approach solving the optimization (2.3) via Bayesian optimization where the objective function $\mathcal{I}(\mathcal{S})$ is evaluated to build a surrogate model of the optimization surface, for example, via a Gaussian process. The points at which to evaluate the objective function requires some measure of uncertainty over the parameter space. The purpose of this work is to provide accurate estimates of the uncertainty in the evaluation of the objective function. In other words, we want to determine the accuracy of the expected information gain calculation.

3. Uncertainty quantification for expected information gain estimates. In this section we describe a procedure to estimate the expected information gain (2.2) for a fixed sensor configuration $\mathcal{S}$ as well as a method for estimating the uncertainty of the estimate itself. Section 3.1 describes an implementable algorithm for estimating the expected information gain, while Sections 3.2 and 3.3 are focused on estimating the uncertainty.

3.1. Estimating expected information gain. The expected information gain (2.2) is computationally challenging to compute as it requires first solving many inference problems for the posteriors $p(\theta \mid \mathcal{D})$, then computing the KL divergence, and finally computing an expectation over all possible observed data. Recall that the data $\mathcal{D}$ can be sampled by first sampling a hypothetical event $\theta' \sim p(\theta')$ according to the prior and then using the likelihood model to simulate the data. Therefore, the expected information gain can be alternatively represented as

$$\mathcal{I}(\mathcal{S}) = \int_\Theta p(\theta') \int_{\mathbb{D}} p(\mathcal{D} \mid \theta') \int_\Theta p(\theta \mid \mathcal{D}) \log\left(\frac{p(\theta \mid \mathcal{D})}{p(\theta)}\right) \mathrm{d}\theta \; \mathrm{d}\mathcal{D} \; \mathrm{d}\theta' \,. \tag{3.1}$$

We note here that, in practice, there is an implicit assumption that the likelihood model is a good approximation of the true underlying process of seismic events.

Virtually all approaches to estimating the expected information gain follow from the nested expectation form of (3.1). However, there is some variation in estimating the information gain for given data such as importance sampling [8], Markov chain Monte Carlo [8], polynomial chaos expansions [4], or simplified analytic expressions with Gaussian posteriors [10]. Depending on the method for inference one can obtain different estimates and therefore methods for quantifying uncertainty of these estimates will also necessarily be different. Here we follow the approach introduced in [2] which presents a method for computing the expected information gain by sampling many hypothetical data observations and computing the information gain for each of the corresponding posteriors (see Algorithm 9 below). The expected information gain is the sample mean of these individual information gains.

From looking at Algorithm 9 there are two main complications that arise in the computation. First, we cannot exactly compute the outer expectation with respect to the observed data $\mathcal{D}$, or equivalently with respect to the prior and then using the likelihood model as in Equation (3.1). Therefore, we need to draw some finite number m samples of the data $\{\mathcal{D}_j\}_{j=1}^m \subset \mathbb{D}$ and then compute the sample mean of the corresponding information gain values. The second complication that arises is that we also cannot compute the information gain values exactly. Instead we use a discrete approximation $p_n(\theta \mid \mathcal{D})$ to the posterior for which we can evaluate the KL divergence exactly. The discrete approximation is constructed by selecting a grid of n points $\{\theta_i\}_{i=1}^n \subset \Theta$ and then evaluating the true un-normalized posterior $\tilde{p}(\theta \mid \mathcal{D})$ on the grid and re-normalizing, so that

$$p_n(\theta \mid \mathcal{D}) = \frac{1}{\sum_{i=1}^n \tilde{p}(\theta_i \mid \mathcal{D})} \sum_{i=1}^n \tilde{p}(\theta_i \mid \mathcal{D}) \delta_{\theta_i}(\theta) \,, \tag{3.2}$$

with δ_{θ_i} denoting the delta measure at point θ_i. The prior $p(\theta)$ is discretized similarly to obtain $p_n(\theta)$. An illustration of the discretization is shown in Figure 3.1. Because the

Algorithm 9 Expected Information Gain (EIG) Calculation

Input: Sensor configuration $\mathcal{S}$

Construct the set of plausible events $\theta' \in \Theta$ with parameters: $\theta' \sim p(\theta')$;

for each plausible event θ' **do**
  Use the likelihood model to draw many samples of data $\mathcal{D} \sim p(\mathcal{D} \mid \theta')$;
  for each simulated data set, $\mathcal{D}$ **do**
    Compute the likelihood $p(\mathcal{D} \mid \theta)$ for each event $\theta \in \Theta$;
    Compute the posterior probability of each event from the likelihood, $p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)\, p(\theta)$;
    Compute the KL divergence i.e. information gain for this realization $\mathrm{KL}\left[p(\theta \mid \mathcal{D}) \mid\mid p(\theta)\right]$;
Compute the expected information gain $\mathcal{I}(\mathcal{S})$ as the sample mean across all event hypotheses and simulated data (Equation (3.1));
return Expected information gain $\mathcal{I}(\mathcal{S})$

approximations to the posterior and prior are discrete distributions the KL divergence can be computed exactly

$$\mathrm{KL}\left[p_n(\theta \mid \mathcal{D}) \mid\mid p_n(\theta)\right] = \sum_{i=1}^{n} p_n(\theta_i \mid \mathcal{D}) \log\left(\frac{p_n(\theta_i \mid \mathcal{D})}{p_n(\theta_i)}\right), \tag{3.3}$$

and is well defined because the prior and posterior are discretized on the same grid of points. Moreover, (3.3) converges to the continuous KL divergence (2.1) as the number of grid points $n \to \infty$. One computational advantage of this approach, besides its simplicity, is that it can easily be vectorized so that the posteriors corresponding to different observed data can all be evaluated on the same grid points. Using the discrete approximations $p_n(\theta \mid \mathcal{D})$ as well as finitely many samples of the observed data leads to the following estimator for the expected information gain

$$\hat{\mathcal{I}}_n(\mathcal{S}) = \frac{1}{m} \sum_{j=1}^{m} \mathrm{KL}\left[p_n(\theta \mid \mathcal{D}_j) \mid\mid p_n(\theta)\right]. \tag{3.4}$$

Although it remains computationally expensive to compute, this estimator is implementable since the data can be simulated and the information gain computed exactly now. In the next section, we present a method to determine the error of this estimate.

3.2. Error estimation for expected information gain estimates. In order to successfully apply the Bayesian optimization as described in Section 2.2 we need some estimate of the uncertainty in the expected information gain estimator (3.4). We quantify the uncertainty through the mean-squared error (MSE) defined as

$$\mathrm{MSE} = \mathbb{E}\left[\left(\hat{\mathcal{I}}_n(\mathcal{S}) - \mathcal{I}(\mathcal{S})\right)^2\right]. \tag{3.5}$$

The discrete approximation $p_n(\theta \mid \mathcal{D})$ to the posterior introduces a bias into the estimator $\hat{\mathcal{I}}_n(\mathcal{S})$, while the finite number of observed data samples used introduces a variance. Indeed, the classical bias-variance decomposition can be applied to the mean-squared error (3.5) in order to see each of these individual effects on the estimator. First define

$$\mathcal{I}_n(\mathcal{S}) = \mathbb{E}_{\mathcal{D}}\left[\mathrm{KL}\left[p_n(\theta \mid \mathcal{D}) \mid\mid p_n(\theta)\right]\right], \tag{3.6}$$

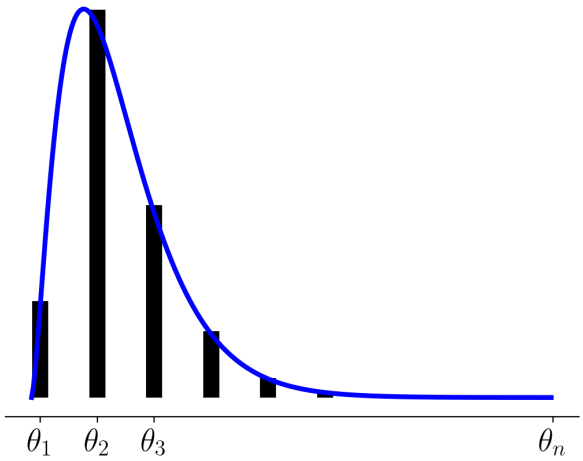


Fig. 3.1: The discrete approximation to the posterior.

which is just the expected information gain (2.2) but with the posterior and prior replaced by their discrete approximations using n grid points chosen according to some pre-specified sequence (e.g. the Sobol sequence). Then, the bias-variance decomposition can easily be derived.

$$\begin{aligned}
&\text{MSE} \\
&= \mathbb{E}\left[\left(\hat{\mathcal{I}}_n(\mathcal{S}) - \mathcal{I}(\mathcal{S})\right)^2\right] \\
&= \mathbb{E}\left[\left(\hat{\mathcal{I}}_n(\mathcal{S}) - \mathcal{I}_n(\mathcal{S}) + \mathcal{I}_n(\mathcal{S}) - \mathcal{I}(\mathcal{S})\right)^2\right] \\
&= \mathbb{E}\left[\left(\hat{\mathcal{I}}_n(\mathcal{S}) - \mathcal{I}_n(\mathcal{S})\right)^2 + 2\left(\hat{\mathcal{I}}_n(\mathcal{S}) - \mathcal{I}_n(\mathcal{S})\right)\left(\mathcal{I}_n(\mathcal{S}) - \mathcal{I}(\mathcal{S})\right) + \left(\mathcal{I}_n(\mathcal{S}) - \mathcal{I}(\mathcal{S})\right)^2\right] \\
&= \text{Var}\left[\hat{\mathcal{I}}_n(\mathcal{S})\right] + \left(\mathcal{I}_n(\mathcal{S}) - \mathcal{I}(\mathcal{S})\right)^2
\end{aligned} \tag{3.7}$$

Note that in the last line of Equation (3.7) we have used the fact that $\hat{I}_n(\mathcal{S})$ is an unbiased estimator for $\mathcal{I}_n(\mathcal{S})$ and so the middle term vanishes.

In light of the bias-variance decomposition (3.7), we will estimate the mean-squared error by estimating the bias and the variance terms separately and then add them together. We will start by estimating the variance, which is straight-forward because we can consider the different observed data as i.i.d. and hence the corresponding information gain values $\text{KL}\left[p_n(\theta \mid \mathcal{D}_j) \mid\mid p_n(\theta)\right]$ for $j = 1, \ldots, m$ are i.i.d. as well. Thus, the estimator $\hat{\mathcal{I}}_n(\mathcal{S})$ is really just a sample mean of i.i.d. samples from the distribution of information gains, see Figure 3.2. A simple method to estimate the variance is to then take the sample variance

of the information gains and divide by the number of samples m

$$\widehat{\mathrm{Var}}\left[\hat{\mathcal{I}}_n(\mathcal{S})\right] = \frac{1}{m(m-1)} \sum_{j=1}^{m} \left(\mathrm{KL}\left[p_n(\theta \mid \mathcal{D}_j) \mid\mid p_n(\theta)\right] - \frac{1}{m} \sum_{i=1}^{m} \mathrm{KL}\left[p_n(\theta \mid \mathcal{D}_i) \mid\mid p_n(\theta)\right] \right)^2, \tag{3.8}$$

which simply comes from the fact that the variance of a sample mean is the population variance divided by the number of samples m. Another alternative for estimating the variance is to use bootstrap resampling by resampling the computed information gain values and constructing many replicates of the sample mean, which is the expected information gain estimate. Then an estimate of the variance of the estimator is given by the variance of the bootstrap replicates. Either method for estimating the variance tends to work well in practice and in fact we will see in the next section that estimating the bias is both more challenging and more important.

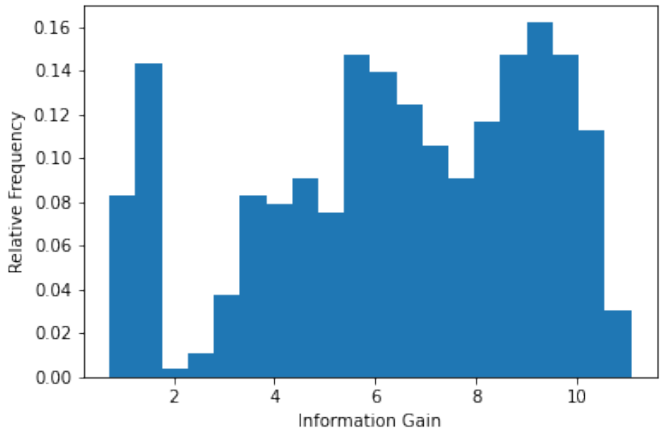


Fig. 3.2: A histogram of the information gain values with 10 uniformly randomly placed sensors. The bootstrap method re-draws samples from this empirical distribution.

3.3. Estimating the bias. The bias arises from computing the information gain using a discrete approximation to the posterior, which is effectively a discretization of the integral for the information gain of the original posterior. Thus, one can bound the bias by bounding the integration error by using the grid points $\theta_1, \ldots, \theta_n$. There is some theory in this direction. Of course, in one dimension and for higher dimensions with regularly-spaced grid points there are bounds on the integration error. Moreover, if the grid points are randomly sampled from a uniform distribution or according to some quasi-Monte Carlo scheme, then one must rely on an upper bound of the integration error, which depends on two factors:

1. The variation in the sense of Hardy and Krause of the function being integrated, which is the posterior and log-likelihood in this case.
2. The star discrepancy of the grid points, which for uniformly-distributed random points is $\mathcal{O}(n^{-1/2})$ and for quasi-Monte Carlo, or low-discrepancy, sequences it is $\mathcal{O}(n^{-1})$.

One approach would be to estimate these two factors separately and although the discrepancy is known, the variation of the integrand is intractable to compute because it requires the computation of many derivatives. Moreover, even if these factors were known exactly, there is still no guarantee that the upper bound on the integration error would be useful. Indeed, in many cases the bound does not become tight until many grid points have been chosen. See [7] for more details. Thus, we cannot rely on analytical methods to estimate the bias.

An even simpler approach to estimating the bias is to evaluate the information gain for each event using a very fine grid of $N \gg n$ points and then comparing the two information gain estimates

$$\widetilde{\text{bias}} = \frac{1}{m}\sum_{j=1}^{m} \left(\text{KL}\left[p_n(\theta \mid \mathcal{D}_j) \;||\; p_n(\theta)\right] - \text{KL}\left[p_N(\theta \mid \mathcal{D}_j) \;||\; p_N(\theta)\right]\right). \tag{3.9}$$

While this method is simple, the most immediate drawback is that it requires computing the information gain on a much finer grid than is necessary, and indeed if we have to compute it on a much finer grid, then we may as well have just used the more accurate information gain values to estimate the expected information gain in the first place.

Instead of resorting to a finer grid, which is computationally inefficient, we propose a sub-division method to re-use the n grid points where the posterior has already been evaluated. The key idea is to observe the bias of smaller sub-grids (subsets of the original grid points) relative to the current grid and then extrapolate to the current grid size. One starts by dividing the current grid into roughly equal sets of training grid points and holdout grid points. In particular, the points should be divided so that each subset populates the parameter domain Θ. For example, if the grid points are generated by the Sobol sequence $\theta_1, \theta_2, \ldots, \theta_n$, then an appropriate choice of the training and holdout grid points are $\Theta_{\text{train}} = \{\theta_1, \ldots, \theta_{\lceil n/2 \rceil}\}$ and $\Theta_{\text{holdout}} = \{\theta_{\lceil n/2 \rceil + 1}, \ldots, \theta_n\}$, respectively. If the grid points are uniformly distributed, then any random partition of the points is also viable. Next, a reference expected information gain estimate is computed using the holdout points. The training grid is further sub-divided into smaller grids of size approximately $n/2^k$, where k is the number of sub-divisions as illustrated in Figure 3.3. Next the expected information gain is estimated on each of these sub-grids and the estimates are compared to the reference value from the holdout set to obtain estimates for the bias of the smaller grid sizes. Finally, a curve is fit between the estimated bias and the smaller grid sizes and we extrapolate to the current grid size n to obtain an estimate of the bias. The proposed method is outlined in Algorithm 10 and is similar to a method discussed in [9].

One problem with this sub-division method, at least empirically, is that it frequently under-estimates the information gain as well as the bias. Figure 3.4 demonstrates this where the information gain values computed on a sub-grid and a reference grid for different events are compared. An easy way to interpret this plot is that points above the dashed black curve indicates that the sub-grid is over-estimating the information gain (relative to some accurate reference value), whereas those underneath are underestimating. Because we are using a discrete approximation to the posterior, the largest information gain possible on a grid of size n is $\log(n)$ (shown in red). In fact we can see that many points actually lie on this upper bound, which is an artifact of using a small discrete grid of points.

Although it is expensive to evaluate the posterior on a fine grid of points, and should be avoided, we can still know the grid points themselves ahead-of-time. Thus, a better approach that circumvents this artificial upper bound is to use the sub-grid and interpolate to a fine grid, which is relatively cheap. For this purpose, we make a simplifying assumption that the posterior can be approximated by a Gaussian. For large information gain values

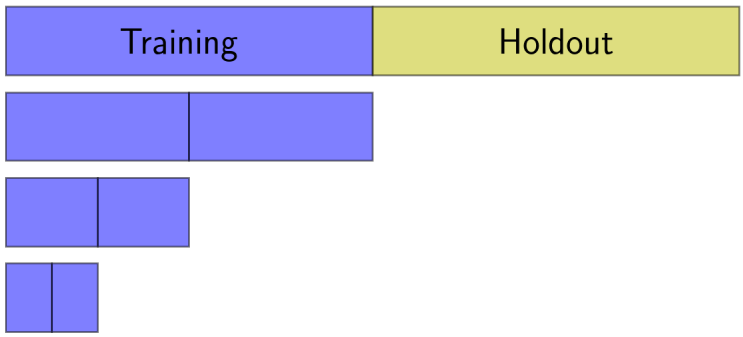


Fig. 3.3: The sub-division procedure used to estimate the bias by splitting into a training and holdout set of grid points and then further sub-dividing the training grid.

Algorithm 10 Bias estimation via the sub-division method

Input: Current grid points $\theta_1, \ldots, \theta_n$, with $n = 2^k$ for some $k \in \mathbb{N}$, and the corresponding posterior density values $p(\theta_i \mid \mathcal{D}_j)$ for $j = 1, \ldots, m$

1: Split the grid points into a training set $\Theta_{\text{train}} = \{\theta_i^{(t)}\}_{i=1}^{n/2}$ and a holdout set $\Theta_{\text{holdout}} = \{\theta_i^{(h)}\}_{i=1}^{n/2}$ with $\Theta_{\text{train}} \cap \Theta_{\text{holdout}} = \emptyset$
2: Compute an estimate of the expected information gain using only the holdout grid points and their corresponding posterior density values
3: Set the current training sub-grid to $\Theta_{\text{subgrid}} = \Theta_{\text{train}}$
4: **for** $k = 1, \ldots, \log_2(n) - 1$ **do**
5: Split the current training sub-grid into two sets and set Θ_{subgrid} to be the first set
6: **for** each simulated data $\mathcal{D}_j$ with $j = 1, \ldots, m$ **do**
7: Compute the information gain using the current training sub-grid
8: Compute an estimate of the expected information gain by taking the sample mean over all information gains
9: Estimate the bias of the current training sub-grid by taking the absolute value of the difference between the estimated expected information gain value and the reference value from the holdout set.
10: Fit a curve for the estimated bias as a function of grid size
11: Extrapolate by evaluating the curve at n to estimate the bias, $\widehat{\text{bias}}_n$ for the original grid
return Estimated bias on the current grid of n points, $\widehat{\text{bias}}_n$

and with enough sensors the posterior will frequently be very concentrated in one region, giving some empirical evidence to support this assumption. The Gaussian interpolant is fitted by computing a weighted mean $\hat{\mu}$ and covariance $\hat{\Sigma}$ using the current grid of points as

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} p_n(\theta_i \mid \mathcal{D}) \theta_i, \qquad \hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} p_n(\theta_i \mid \mathcal{D})(\theta_i - \hat{\mu})(\theta_i - \hat{\mu})^T . \tag{3.10}$$

Here the weights correspond to the posterior approximation's probabilities and are necessary because the grid points are uniformly distributed. Thus, these estimates can be thought of as discrete approximations to the expectations for the mean and covariance. We can then evaluate the density of the fitted Gaussian to interpolate onto the new grid of points. These interpolated values are then used to compute the information gains needed in Line 7 of

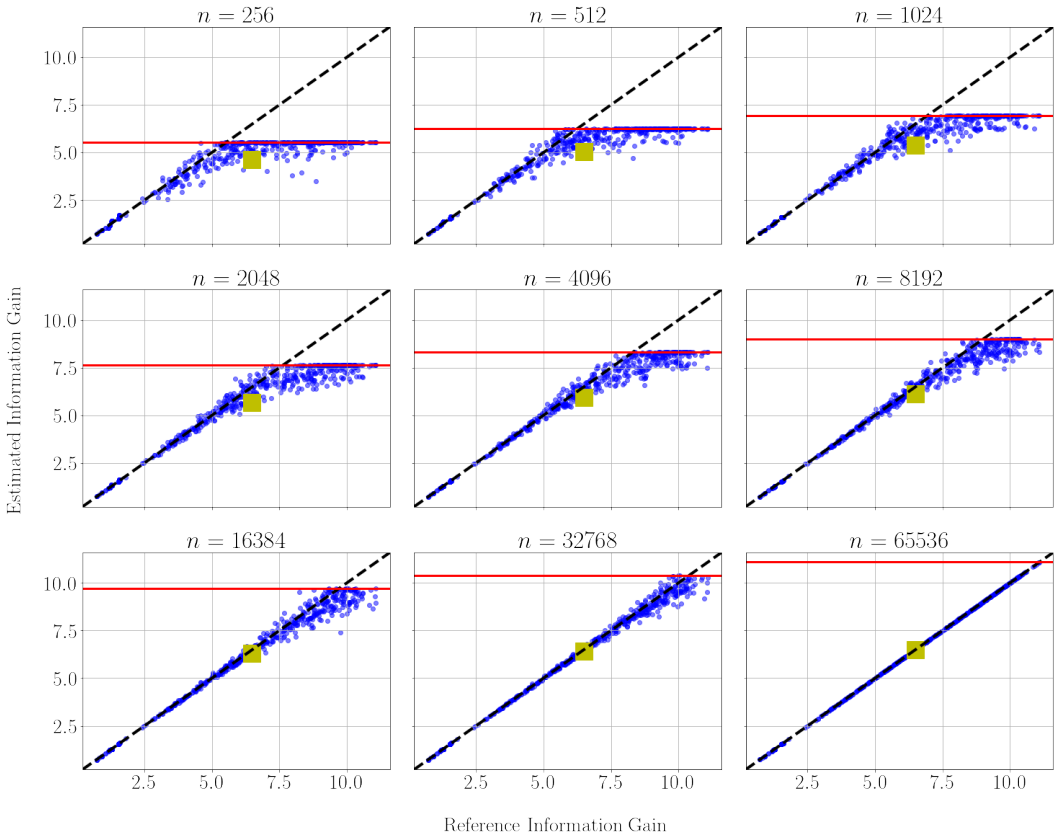


Fig. 3.4: A scatter plot of the information gain estimates for each data $\mathcal{D}_j$ for $j = 1, \dots, m$ and for different grid sizes $n = 2^8, \dots, 2^{16}$. The x-coordinate of each point is the information gain computed on a very fine grid of $N = 2^{16}$ points and the y-coordinate is the information gain computed on a smaller sub-grid of size n. The dashed black curve is the line $y = x$. The red horizontal line is the value $\log(n)$. The yellow square is located at the mean information gain (EIG) for both the fine reference and the corresponding sub-grid.

Algorithm 10. Some care has to be taken when fitting the covariance matrix to ensure that it is positive definite. When the posterior is very concentrated for large information gain values, almost all of the mass is placed on the most likely grid point. Thus, the covariance matrix becomes singular. A simple fix is to simply add a small multiple of the identity matrix, and an even better fix empirically is to add a small multiple of the identity that goes to zero as $n \to \infty$, such as n^{-1}. Figure 3.5 shows that using a Gaussian interpolant does alleviate the problem of underestimating the information gain. In the next section we will see that it provides a better estimate of the bias as well.

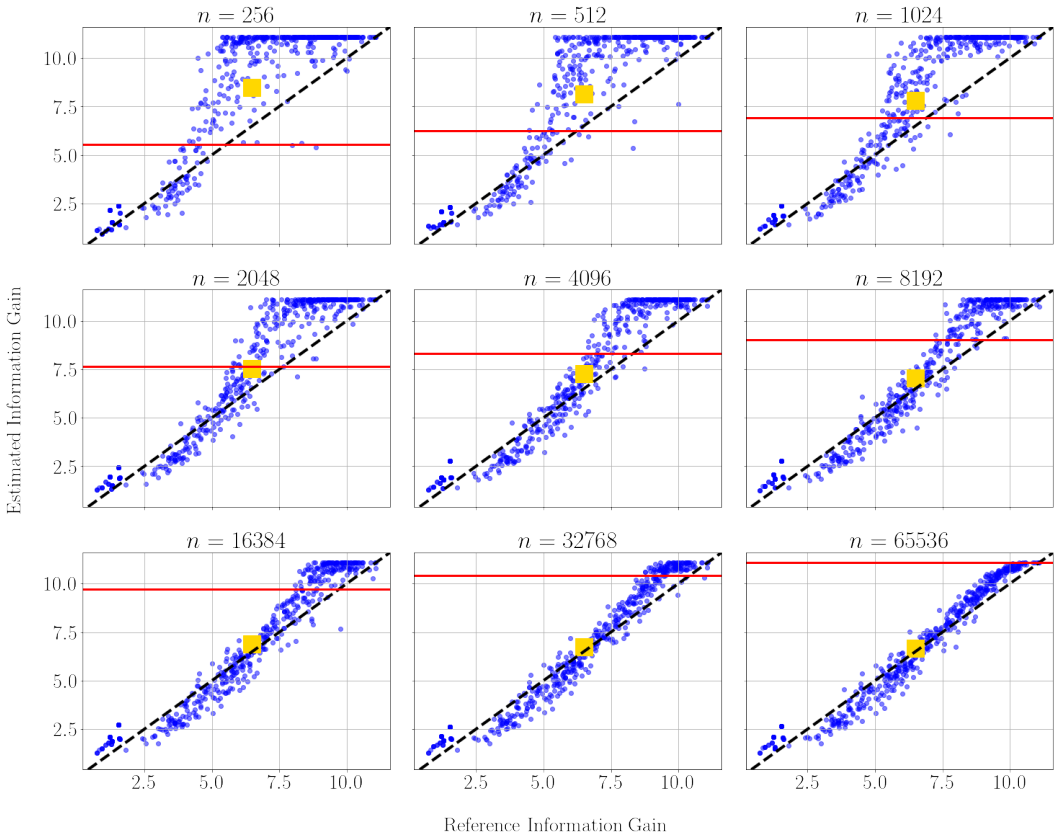


Fig. 3.5: A scatter plot of the information gain estimates using the Gaussian interpolant for each data $\mathcal{D}_j$ for $j = 1, \ldots, m$ and for different grid sizes $n = 2^8, \ldots, 2^{16}$. The x-coordinate of each point is the information gain computed on a very fine grid of $N = 2^{16}$ points and the y-coordinate is the information gain computed on a smaller sub-grid of size n. The dashed black curve is the line $y = x$. The red horizontal line is the value $\log(n)$. The yellow square is located at the mean information gain (EIG) for both the fine reference and the corresponding sub-grid.

4. Numerical results. In this section we briefly present some numerical results for the performance of our sub-division method described in Algorithm 10 with and without the Gaussian interpolant on a problem with 10 uniformly randomly placed sensors with grid points chosen uniformly randomly as well. Figure 4.1 shows a reference for the mean-squared error using a reference value computed on a grid of $N = 2^{17}$ points and with $m = 2048$ sampled data observations. We see that while the original sub-division method greatly underestimates the bias, adding the Gaussian interpolant provides a much more accurate approximation to the reference mean-squared error.

Recall that, following from Equation (3.7), we estimate the mean-squared error by estimating the bias and variance separately and then add them together. Figure 4.2 shows the break-down of the separate bias and variance estimates. Here the importance of correctly estimating the bias is revealed. We see that, by far, the bias is the dominant term for the mean-squared error. This is due to the relatively large number of sensors. Nearly any

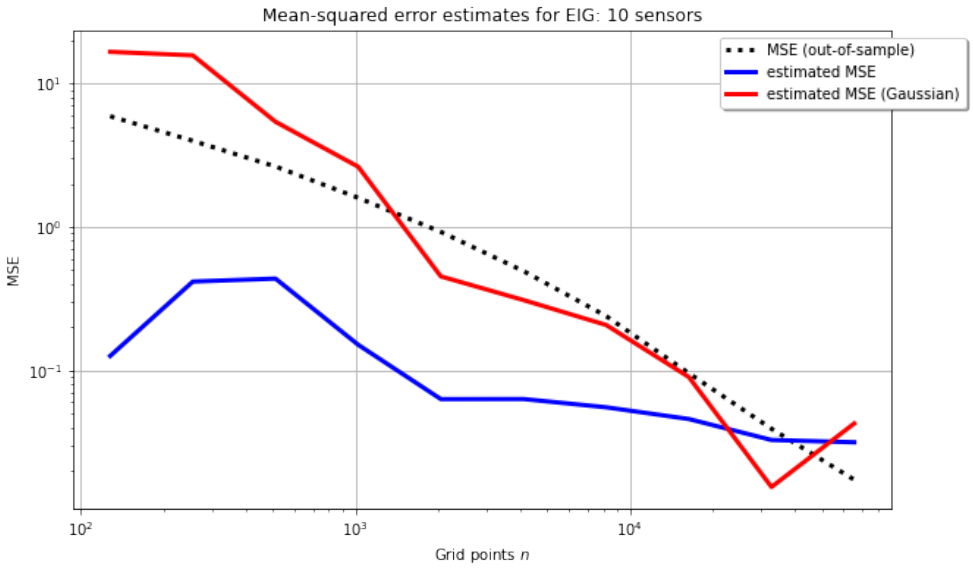


Fig. 4.1: The mean-squared error of the expected information gain estimates as the number of grid points n increases. The black curve is computed using a high-fidelity reference estimate. The blue curve shows the estimated MSE using the sub-division procedure outlined in Algorithm 10 and the red curve is the MSE estimate using a Gaussian interpolant as well.

observed data will result in a large information gain and a concentrated posterior. Because the posterior is so concentrated, the variance will be extremely small. For fewer sensors, and more diffused posteriors, the role of the variance becomes more prominent. Overall, using a Gaussian interpolant gives a good approximation to the bias as shown by the magenta and yellow curves.

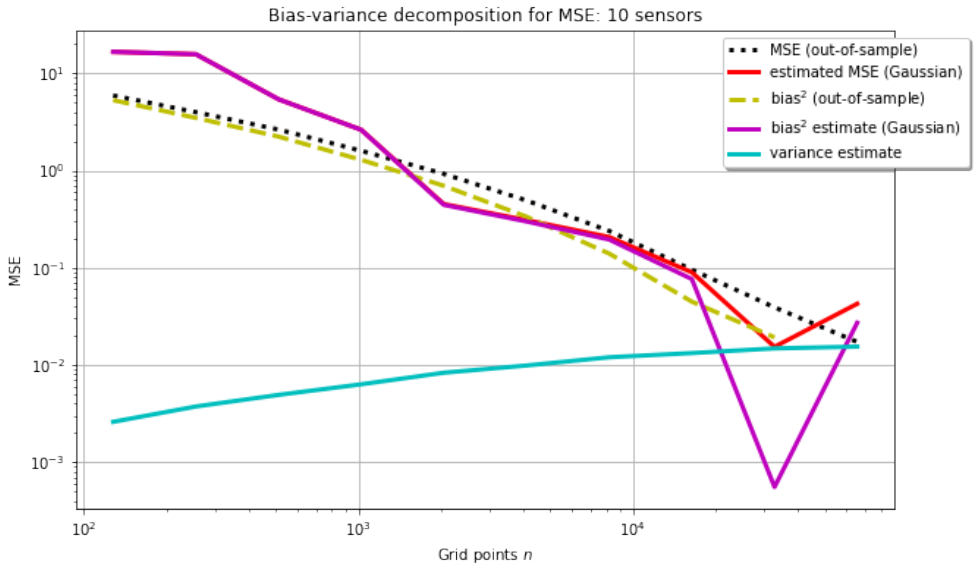


Fig. 4.2: The bias-variance decomposition for the mean-squared error. The red and black curves are the same as in Figure 4.1. The yellow curve is an out-of-sample estimate of the bias using a very fine grid as in Equation (3.9). The magenta curve shows the estimated bias using the sub-division procedure with the Gaussian interpolant and the cyan curve shows the variance estimate by using the bootstrap method with 100 bootstrap replicates.

5. Conclusions. Expected information gain estimates are needed to approximate the objective function in the Bayesian optimal experimental design (2.3) for determining sensor configuration. Adding uncertainty estimates, in terms of the mean-squared error, can help construct better surrogate models, such as a Gaussian processes, which can serve as a model for errors of estimators obtained from different sensor configurations. These surrogate models can help determine new sensor configurations to evaluate the objective function at, for example, by placing more sensors in regions where estimators currently have high variance. With this application in mind, the mean-squared error can be estimated by estimating both the bias and the variance of the expected information gain estimator separately. The variance is easily estimated using the bootstrap method and the bias can be estimated by continually sub-dividing the grid of points and interpolating from the sub-grid to a finer grid by fitting a Gaussian.

In addition to these mean-squared error estimates being useful for the Bayesian optimization, they can also be used to determine the number of grid points needed to control the error to within some tolerance and how to trade-off the number of grid points and sample data observed by trading-off the bias and variance. Future work will explore more efficient methods for estimating the bias as well as scenarios in which the Gaussian interpolant is and is not a valid assumption. Of particular interest are scenarios with few sensors and when the posterior is noticeably non-Gaussian.

Acknowledgments. This research was funded by the U.S. Department of Energy. Sandia National Laboratories is a multimission laboratory manged and operated by Na-

tional Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003535. The views expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government. This research was funded by the National Nuclear Security Administration, Defense Nuclear Nonproliferation Research and Development (NNSA DNN R&D). The authors acknowledge the important interdisciplinary collaboration with scientists and engineers from LANL, LLNL, MSTS, PNNL, and SNL. We would like to specifically thank Jaideep Ray, Teresa Portone, and Chris Young for their help and advice in developing this research and feedback while preparing this manuscript. We would also like to thank CSRI Summer program organizers for their support.

References.

[1] N. Arora, S. Russell, and E. Sudderth, *Net-visa: Network processing vertically integrated seismic analysis*, Bulletin of the Seismological Society of America, 103 (2013), pp. 709–729.

[2] T. Catanach and K. Monague, *Analysis and optimization of seismo-acoustic monitoring networks with Bayesian optimal experimental design*, Sandia National Laboratories. (SNL-CA), Livermore, CA (United States), (No. SAND2021-3972) (2021).

[3] H. Crotwell, T. Owens, and J. Ritsema, *The TauP toolkit: Flexible seismic travel-time and ray-path utilities*, Seismological Research Letters, 70 (1999), pp. 154–160.

[4] X. Huan and Y. Marzouk, *Simulation-based optimal bayesian experimental design for nonlinear systems*, Journal of Computational Physics, 232 (2013), pp. 288–317.

[5] G. Laske, G. Masters, Z. Ma, and M. Pasyanos, *Update on Crust1.0 – a 1-degree global model of earth's crust*, Geophys. Res. Abstr, 15 (2013), p. 2658.

[6] D. Lindley, *On a measure of the information provided by an experiment*, Ann. Math. Statist., 27 (1956), pp. 986–1005.

[7] W. Morokoff and R. Caflisch, *Quasi-random sequences and their discrepancies*, SIAM J. Sci. Comput., 15 (1994), pp. 1251–1279.

[8] K. Ryan, *Estimating expected information gains for experimental designs with application to the random fatigue-limit model*, Journal of Computational and Graphical Statistics, 12 (2003), pp. 585–603.

[9] W. Snyder, *Accuracy estimation for quasi-Monte Carlo simulations*, Mathematics and Computers in Simulation, 54 (2000), pp. 131–143.

[10] Z. Xu and Q. Liao, *Gaussian process based expected information gain computation for Bayesian optimal design*, Entropy (Basel, Switzerland), 22 (2020), p. 258.

IMPORTANCE SAMPLING IN BAYESIAN OED FOR SENSOR PLACEMENT

JAKE P. CALLAHAN* AND THOMAS A. CATANACH†

Abstract. The goal of Bayesian optimal experimental design (OED) is to find experiments that reduce uncertainty in an optimal way. We use Bayesian OED to find optimal sensor configurations for detecting seismic events as part of a seismic monitoring network. It is useful to be able to easily sample from different event priors that represent various situations in which these events are manifest Some such priors are irregular and difficult to sample from. Importance sampling can be used to sample other, easier-to-sample distributions, and use those samples to obtain good approximations of the parameter of interest. In this work we implement an importance sampling method to approximate Expected Information Gain (EIG), examine the effects of importance distribution choice on the quality of EIG approximation, and compare the effect of prior choice on sensor placement.

1. Introduction. Seismo-acoustic monitoring networks are powerful tools for detecting nuclear tests and other events that generate seismo-acoustic energy. One area for improvement in seismo-acoustic monitoring methods is the reduction of estimate uncertainty in quantities of interest (QoIs). In this work we take a Bayesian approach to reducing uncertainty and employ a framework known as Bayesian Optimal Experiment Design (OED). Bayesian OED works to choose an experiment design, in our case a sensor configuration, that minimizes uncertainty by maximizing information gain. This framework allows us to both design effective monitoring networks and understand the quality of QoI estimates they generate. In this work we focus on minimizing uncertainty about event location and magnitude, but this framework is general enough to target other QoIs if desired. An important contributor to estimate quality is the choice of likelihood and prior distribution used to analyze a given dataset. In [2], Catanach and Monogue present both a Bayesian OED algorithm for sensor configurations and a methodology for choosing likelihood models for seismic monitoring data. However, they do not present a method for choosing a prior distribution.

In this work we seek to improve upon that algorithm by implementing the ability to sample from any specified prior distribution whose pdf can be evaluated. We do this by using an importance sampling algorithm that re-weights samples from easy-to-sample distributions in order to approximate quantities of interest. We explore how choice of this easy-to-sample distribution, called the importance distribution, affects the quality of QoI approximation. Finally, using this importance sampling algorithm, we examine the effect that choice of prior has on the final network configuration chosen by the Bayesian OED algorithm.

2. The Bayesian OED Problem.

2.1. Bayesian Inference.

Bayesian inference is a method for using both stated belief and observed data to quantify uncertainty and inform predictions. To perform Bayesian inference, we begin by expressing belief about parameters of interest θ in the form of a probability distribution, $p(\theta)$. In our specific case, we are interested in seismo-acoustic events, so we may state belief about the distribution of magnitude, the distribution of latitude and longitude, or the distribution of depth of a potential earthquake.

We then gather data $\mathcal{D}$ and use some measure of the likelihood of this data given θ, $p(\mathcal{D}|\theta)$, to update our prior belief. In this work we consider data measured using seismic sensors such as arrival time of a seismic phase and seismic source descriptors. Combining our prior and likelihood functions gives us a new distribution on θ called the posterior. The

*Brigham Young University Department of Mathematics, jake@mathematics.byu.edu
†Sandia National Laboratories, tacatan@sandia.gov

posterior is given by

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

In practice, computing this posterior distribution analytically is not usually feasible, so we use approximate computational methods to solve for the posterior distribution. Common methods to represent the posterior distribution, such as Monte Carlo or Markov Chain Monte Carlo methods, generate samples from this posterior and use those samples to estimate QoIs.

2.2. Optimizing Information Gain. Once we have a posterior distribution on our seismic source parameters, we wish to measure how much observed data from a given sensor configuration $\mathcal{S}$ changes our prior belief. To do this we use the Kullback-Leibler (KL) divergence between the posterior and the prior:

$$KL[p(\theta|\mathcal{D})||p(\theta)] = \int p(\theta|\mathcal{D}) log \frac{p(\theta|\mathcal{D})}{p(\theta)} d\theta$$

To compute Expected Information Gain (EIG) for a given sensor configuration, we seek to compute the average KL divergence over all possible realizations of data:

$$\begin{aligned} I(\mathcal{S}) &= E_{\mathcal{D}|\mathcal{S}}[KL[p(\theta|\mathcal{D})||p(\theta)]|\mathcal{D}] \\ &= \int p(\mathcal{D}|\mathcal{S}) \int p(\theta|\mathcal{D},\mathcal{S}) log \frac{p(\theta|\mathcal{D},\mathcal{S})}{p(\theta)} d\theta d\mathcal{D} \end{aligned}$$

We thus seek to maximize $I(\mathcal{S})$. We do this using greedy optimization algorithms which add sensors one at a time for the chosen sensor network, choosing the sensor with the highest information gain at each step. While not the optimal optimization method (such a method would require computing EIG for every possible sensor location at once), greedy optimization works reasonably well under our optimization conditions and offers a much lower computational cost.

In total, the full EIG calculation algorithm [2] is summarized Algorithm 1.

2.3. Problem-specific Setup. In order to use Bayesian OED to place seismic sensors, we must first formulate a problem space. For a visual representation of the problem setup, see Figure 2.1. We describe seismic events (our event of interest θ) in terms of 4 parameters: latitude, longitude, depth, and magnitude. We define all subsequent prior and posterior distributions on these 4 parameters. We note that the event origin time may be an additional parameter, but we use marginalization to remove this variable.

2.3.1. Domain. Unless otherwise stated elsewhere, our experiments examined the latitude range [40,42], the longitude range [-112,108.38], the depth range [0,40] and the magnitude range [.5,3].

2.3.2. Data and Likelihood. Once we have sampled our events θ, we use these events to generate our data $\mathcal{D}$. We describe $\mathcal{D}$ by an arrival time parameter and several P Phase detection parameters. Our models were built based off the USArray Transportable Array experiment catalog [1], which contains data observations from sensor arrays in the latitude/longitutde domain specified in 2.3.1.

The likelihood function for P Phase detection is a Gaussian Process error model with station correlation of 147.5 km. Events with smaller magnitudes have smaller probability of being detected at distant stations.

The likelihood model for arrival time is built using predicted wave travel time μ_p, standard deviation σ_p, correlation matrix Γ_{GP}, and an independent and identically distributed

Algorithm 1 Expected Information Gain (EIG) Calculation

Result: Information gain, $\mathcal{I}(\mathcal{S} \mid \theta)$, for individual events, θ, and total EIG, $\mathcal{I}(\mathcal{S})$, given sensor configuration, $\mathcal{S}$

1: Construct the set of plausible events $\theta' \in \Theta$ with parameters: locations $\mathcal{L}'$, depth x', and magnitudes m' such that $\theta' \sim p(\theta')$:
2: **for** each event hypothesis, θ' **do**
3: Simulate hypothetical datasets of which stations detect an arrival according to the distribution $\mathbb{D} \sim p(\mathbb{D} \mid \mathcal{L}', x', m', \mathcal{S})$
4: For each arrival dataset, simulate the arrival time according to the distribution $\mathbb{A} \sim p(\mathbb{A} \mid \mathcal{L}', x', \mathbb{D}, \mathcal{S})$
5: **for** each simulated data set, $\mathcal{D} = \{\mathbb{A}, \mathbb{D}\}$ **do**
6: Compute the likelihood $p(\mathcal{D} \mid \theta, \mathcal{S})$ of the observation data $\mathcal{D} = \{\mathbb{A}, \mathbb{D}\}$ given each event in the event space $\theta \in \Theta$ using the detection and arrival time likelihood functions
7: Compute the posterior probability of each event from the likelihood, $p(\theta \mid \mathcal{D}, \mathcal{S}) \propto p(\mathcal{D} \mid \theta, \mathcal{S})\, p(\theta)$
8: Compute the KL divergence i.e. information gain for this realization $\mathcal{I}(\mathcal{S} \mid \theta', \mathcal{D}) = \int p(\theta \mid \mathcal{D}, \mathcal{S}) \log \frac{p(\theta|\mathcal{D},\mathcal{S})}{p(\theta)} d\theta$
9: Compute the EIG for the event hypothesis, $\mathcal{I}(\mathcal{S} \mid \theta')$, as the average over KL divergences of the simulated data realizations, $\mathcal{I}(\mathcal{S} \mid \theta', \mathcal{D})$
10: Compute the total EIG, $\mathcal{I}(\mathcal{S})$, as average across all event hypotheses and simulated data

noise term with covariance $\epsilon^2 \mathbb{1}$. The predicted travel times are derived from a set of ray tracing models that consider earth model uncertainty. These terms are defined in [2]. Thus we first sample θ from the prior distribution, and then sample from the likelihood $p(\mathcal{D}|\theta)$ to generate synthetic data observations.

Using this data and likelihood function, we seek to generate a posterior on our θ parameters given our data and a sensor configuration in order to compute EIG and thus find the optimal sensor configuration. The final choice we must make is of the prior distribution on our θ parameters.

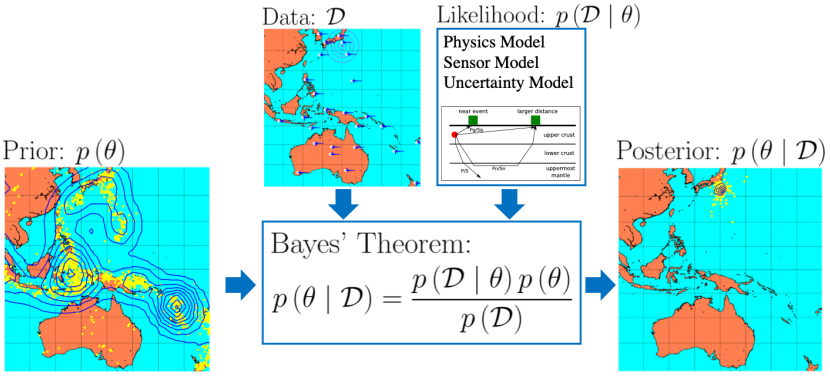


Fig. 2.1: A visual representation of the seismic Bayesian OED problem set up.

3. Importance Sampling. In our application, there are many reasonable choices of prior from which sampling is prohibitively expensive or even impossible. Thus, to be able to fully utilize domain knowledge about areas of interest it is important that we have a way to representatively sample from these challenging priors. Importance sampling is one such way. It is a method where we draw samples from an importance distribution (a distribution different from the prior but one that is possible to sample), weight those samples according to the probability density function (pdf) of our target distribution (prior), and use these weighted samples to approximate our quantities of interest.

3.1. Theoretical Justification. Mathematically, the EIG of a sensor configuration can be expressed as an expected value [2]:

$$I(\mathcal{S}) = \int I(\mathcal{S}|\theta')p(\theta')d\theta' \tag{3.1}$$

where $\theta' \in \Theta$ is the set of possible seismic events such that $\theta' \sim p(\theta')$. Observe that

$$\begin{aligned} I(\mathcal{S}) &= \mathbb{E}[I(\mathcal{S}|\theta'] \\ &= \int I(\mathcal{S}|\theta')p(\theta')d\theta' \\ &= \int I(\mathcal{S}|\theta')\frac{p(\theta')}{q(\theta')}q(\theta')d\theta' \end{aligned}$$

Thus, when $\theta' \sim q(\theta')$, we have that

$$\begin{aligned} \int I(\mathcal{S}|\theta')\frac{p(\theta')}{q(\theta')}q(\theta')d\theta' &= \mathbb{E}[I(S|\theta')\frac{p(\theta'_i)}{q(\theta'_i)}] \\ &\approx \frac{1}{N}\sum_i I(\mathcal{S}|\theta'_i)\frac{p(\theta'_i)}{q(\theta'_i)} \end{aligned}$$

and therefore

$$I(\mathcal{S}) \approx \frac{1}{N}\sum_i I(\mathcal{S}|\theta'_i)\frac{p(\theta'_i)}{q(\theta'_i)} \tag{3.2}$$

Thus we can approximate EIG with samples from a distribution different from the prior distribution. For a detailed discussion of importance sampling see [3].

3.2. Approximation Quality. Of course, choice of importance distribution and number of samples from the importance distribution can affect the quality of approximation. We measure quality of approximation by Effective Sample Size (ESS). After performing the importance sampling process, we can calculate the ESS to determine the "worth" of our samples, or, in other words, how effectively our samples achieve the same result that samples from the actual target distribution would. For example, if we drew 1000 samples from the importance distribution and calculated an ESS of 200, then our 1000 samples from the importance distribution are achieving the same result that 200 samples from the target distribution would. ESS is defined as follows:

$$ESS = \frac{(\sum_i w_i)^2}{\sum_i w_i^2} \tag{3.3}$$

where $w_i = \frac{p(\theta_i)}{q(\theta_i)}$ are the importance weights.

We chose to examine the approximation quality for a mixture prior distribution that resembles a fault line (see Figure 3.1). The seismic events which are sampled from this distribution have four parameters: latitude, longitude, depth, and magnitude. Thus, the random variable X sampled from the prior is a vector of length 4. Each parameter is assumed to be independent of the others. The depth and latitude are assumed to be uniform across the intervals $[0, 40]$ and $[40, 42]$, respectively. The magnitude is assumed to follow an exponential distribution with rate parameter $\lambda = 10$. The longitude is assumed to be sampled from a mixture distribution comprised of a Gaussian $\mathcal{N}(-110.19, .125)$ with weight .98 and a uniform $U(-112, -108.38)$ with weight 0.02. Thus, the probability of the longitude is given by

$$p(long) = .98f(long) + 0.02g(long)$$

where f and g are the PDFs of $\mathcal{N}(41, .125)$ and $U(-112, -108.38)$, respectively. The total probability of the event is thus given by the product of each individual probability.

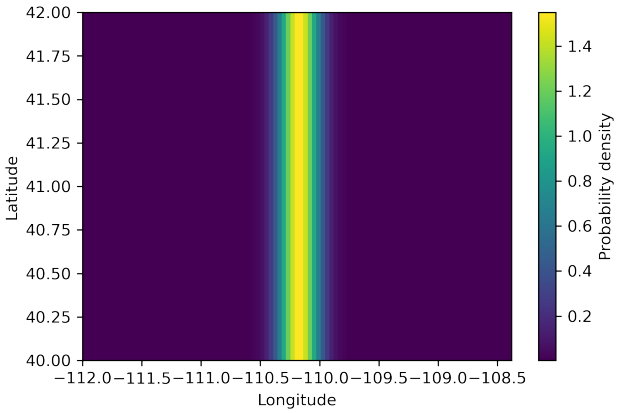


Fig. 3.1: Prior distribution used for examining importance sampling approximation quality

Our importance distribution was chosen to be a standard multivariate normal for the latitude, longitude, and depth with mean $\mu = [41, -110.19, 20]$, representing the center of the area of interest for each parameter. We chose to sample from the true prior for the magnitude, and multiplied the evaluations of the PDF for these two distributions to give us the importance distribution PDF.

To examine how importance distribution affects sample size, we modified the covariance of the multivariate normal and calculated the EIG and ESS for a fixed set of data and sensors. In Figure 3.2 we see that as the covariance of the multivariate normal increases, there is a marked improvement in sample quality that levels off at roughly 3 times the length of the area of interest. We can conclude that with a prior sufficiently wide, we can achieve a high enough ESS for a good approximation.

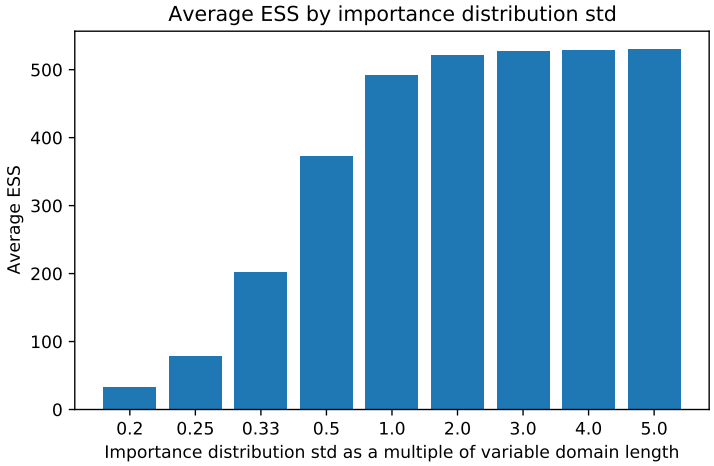


Fig. 3.2: Effective Sample Size as a function of importance distribution width. As the covariance of a multivariate Gaussian importance distribution increases, a sample size of 8192 more closely approximates a set of samples from a uniform prior.

4. Effect of Prior Choice on Sensor Placement. We wish to examine how a non-uniform prior affects the sensor placement and therefore the Expected Information Gain.

4.1. Prior Definition. To sample the latitude and longitude we chose to use a mixture distribution representing a fault line and a point source as our prior distribution. The first mixture component was a bivariate Gaussian centered at (40.25,-109) with covariance matrix $\begin{bmatrix} .125 & 0 \\ 0 & .125 \end{bmatrix}$. The second mixture component was a 1-dimensional Gaussian in the longitude direction with mean -110.19 and standard devation .125 multiplied by a uniform in the latitude direction. The final mixture component was a uniform distribution across both latitude and longitude. These components were given mixture weights .49, .49, and .02 respectively. See Figure 4.1 for a visual representation. The depth was sampled from a uniform distribution and the magnitude was sampled from an exponential distribution with $\lambda = 10$. The total probability for a single event is thus given by the product of the probability for each parameter.

4.2. Sensor Placement. Using this prior distribution, we used the Bayesian OED process to place 8 sensors in addition to an initial sensor grid of 9 evenly-spaced sensors

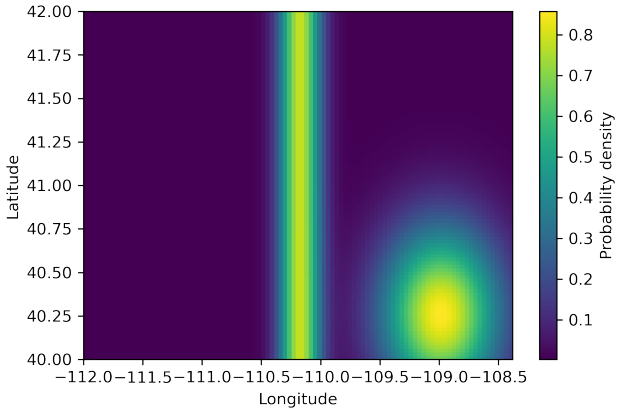


Fig. 4.1: Fault and point source prior distribution on latitude and longitude. It is comprised of 3 mixture components: a bivariate Gaussian representing the point source, a univariate Gaussian in the longitude direction multiplied by a uniform in the latitude direction representing the fault, and a uniform in both directions representing the background probabilities.

to create a 17-sensor network. We then examined the Expected Information Gain surface generated by this sensor configuration at various cross sections of the magnitude and depth, and compared these sensor configurations and EIG surfaces to those generated by using a uniform prior. These results are visualized in Figure 4.2 and Figure 4.3

When using the fault-and-point-source prior, sensor placement follows the shape of the prior on the latitude and longitude. We see sensors placed in a horizontal line that follows the shape of the fault line Gaussian, and which start to drift longitudinally toward the point source as they get closer to it latitudinally. This is what we would expect to see, especially on a event dataset that contains mostly low magnitude event such as ours; if the magnitude of our events is generally low, we would want sensors placed close to the areas of high event probability to better guarantee detection.

It is clear that events that occur away from the high-density regions of the prior provide the most information gain, since these are more "surprising" than events that occur in the high-density regions. As event magnitude increases, EIG becomes stronger. However, EIG seems to be strongest at depths in the middle of the depth range, weaker at the top of the depth range, and weakest at the bottom of the depth range.

Compare these results to sensors placed using a uniform prior distribution (see Figure 4.3). We see that, in contrast to the sensor placement generated by the fault-and-point-source prior, there is no discernible pattern in the sensor configuration placed using the uniform prior. Further, the information gain generated by uniform sensor configuration is much lower than that of the configuration placed on the fault-and-box prior. This is not surprising, given that the uniform distribution is equally dense everywhere. Since our test prior is very dense in certain areas and has almost no density everywhere else, we would expect a posterior with density in these areas of low prior density to be more dissimilar to our test prior than to a uniform distribution.

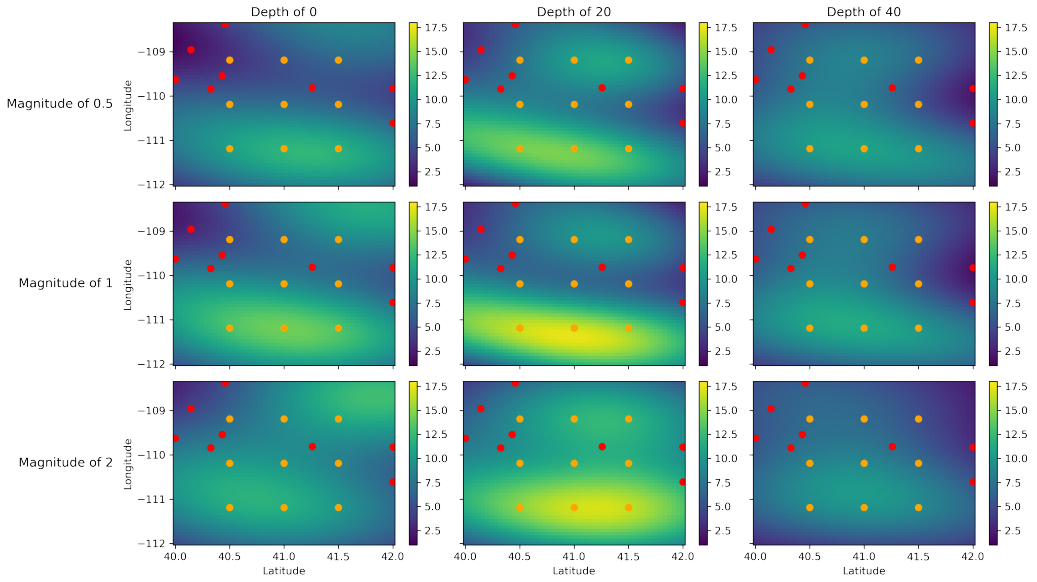


Fig. 4.2: Cross sections of the Expected Information Gain of the sensor configuration produced by the Bayesian OED process using the fault-and-point-source prior. Orange dots are the nine evenly spaced sensors in the initial grid, while the red dots show the sensors placed to maximize EIG. We see sensor placement clustered around the point source Gaussian and along the fault line Gaussian, as expected. The events that happen at locations away from the areas of high density in the prior give the most information gain. Events of higher magnitude are more informative, and events at shallower or deeper depths are less informative than events at middle depths.

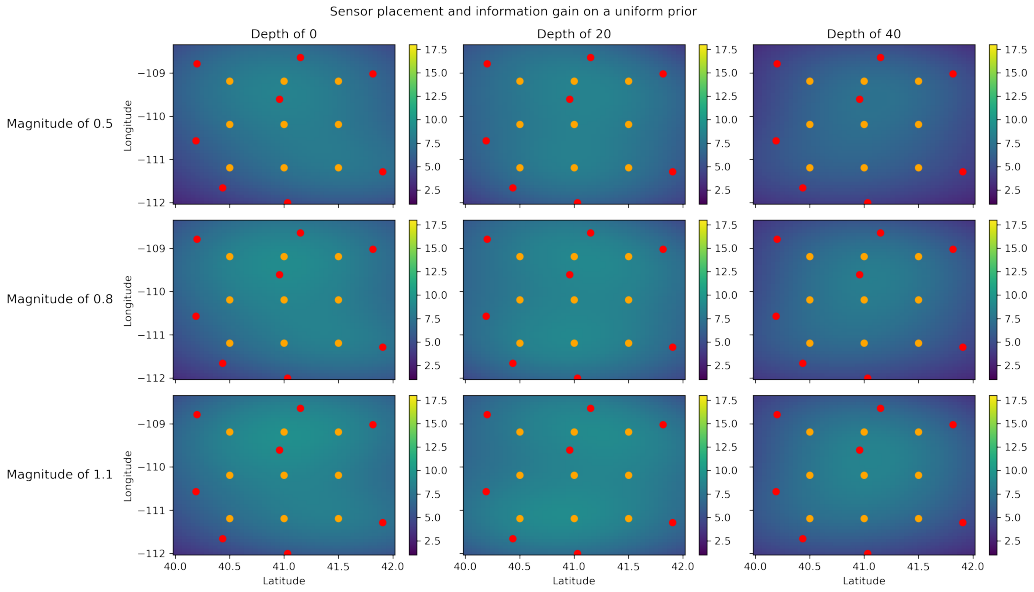


Fig. 4.3: Cross sections of the Expected Information Gain of the sensor configuration produced by the Bayesian OED process using a uniform prior. We see no truly discernible pattern of sensor placement with respect to information gain. The information gain from this set of priors is also much more uniform and far lower than that of the sensor configuration generated by the fault-and-point-source prior.

5. Conclusion. In this work, we reviewed a framework for Bayesian optimal experiment design and implemented an importance sampling method for sampling from various complex prior distributions, allowing us to perform experiments with a variety of new priors we would not otherwise have been able to use. We examined how well EIG was approximated when using a prior distribution sampled with this importance sampling method and found that with a large enough sample size we can achieve reasonably good approximation. Finally, we examined how prior choice affects sensor placement and EIG, finding that the choice of prior greatly affects both the EIG of a given sensor network and the actual placement of sensors. Based on these results, we have identified several next steps for future work:

1. In an attempt to understand exactly how prior choice affects sensor placement, repeat the experiment comparing uniform prior to a new prior, this time using a prior with even higher densities concentrated in two corners of our latitude/longitude domains and at very low magnitude.
2. To better understand the effect of prior choice on Expected Information Gain, compare the EIG generated by the fault-and-point sensor network when evaluated on the uniform prior to the EIG generated by the uniform sensor network when evaluated on the fault-and-point prior.
3. Examine how number of synthetic data events affects sensor placement.

Acknowledgment. This research was funded by the U.S. Department of Energy. Sandia National Laboratories is a multimission laboratory manged and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003535. The views expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government. This research was funded by the National Nuclear Security Administration, Defense Nuclear Nonproliferation Research and Development (NNSA DNN R&D). The authors acknowledge the important interdisciplinary collaboration with scientists and engineers from LANL, LLNL, MSTS, PNNL, and SNL. We would like to specifically thank Jaideep Ray, Jorge Garcia, and Chris Young for their help and advice in developing this research and feedback while preparing this manuscript. We would also like to thank CSRI Summer program organizers for their support.

References.

[1] *USArray Transportable Array monthly catalogs picked by ANF*. https://anf.ucsd.edu/tools/events/. Accessed: 2020-04-30.

[2] T. Catanach and K. Monague, *Analysis and optimization of seismo-acoustic monitoring networks with Bayesian optimal experimental design*, Sandia National Laboratories. (SNL-CA), Livermore, CA (United States), (No. SAND2021-3972) (2021).

[3] A. B. Owen, *Monte Carlo theory, methods and examples*, 2013.

A VARIANCE DECONVOLUTION APPROACH TO SAMPLING UNCERTAINTY QUANTIFICATION FOR MONTE CARLO RADIATION TRANSPORT SOLVERS

KAYLA B. CLEMENTS*, GIANLUCA GERACI†, AND AARON J. OLSON‡

Abstract. Radiation transport computations in realistic systems are affected by the presence of uncertainty sources, *e.g.* nuclear cross section data and variability in geometric arrangement. Therefore, it is of paramount importance to statistically characterize the response of quantities of interest by performing efficient and accurate uncertainty quantification (UQ). Traditionally, UQ is focused on evaluating how the statistics, *e.g.* variance, of quantities of interest from a numerical code response are affected by sources of uncertainty, which can be propagated through several runs of the numerical code. For radiation transport problems, solved using Monte Carlo particle transport methods, one sample (in the parameter space) of the quantity of interest is obtained by averaging various particles' random walks, and, therefore, the non-deterministic nature of the solver itself introduces an additional source of variance. In this contribution, we describe how we can obtain efficient sample variance estimators by taking into account the additional variability introduced by a non-deterministic solver. We provide a rigorous mathematical treatment of these variance contributions and their sampling estimator counterparts. In particular, we present numerical test problems in which this variance deconvolution strategy is deployed with and without scattering, in 1D slabs, and with uncertain material properties. We also demonstrate how our novel estimator is more precise than a related variance estimator recently introduced in [9].

1. Introduction. Uncertainty Quantification (UQ) is the process of propagating sources of uncertainty through numerical codes in order to evaluate the statistics of Quantities of Interest (QoIs). It is nowadays well established that truly predictive numerical simulations can be obtained only when UQ is included in the analysis workflow. Over the years, several UQ strategies have been proposed to address different challenges; however, virtually all of them still face challenges when dealing with a large parameter dimension. Like other computational science applications, radiation transport applications are impacted by a large number of uncertainty sources, which roughly correspond to a number of uncertainty parameters that scale with the complexity of the models. In this contribution, we want to consider radiation transport applications in which a Monte Carlo transport method is used to evaluate the QoI and a sampling strategy, whose convergence is unaffected by the input parameter dimension, is then used for the UQ forward propagation. We note that two sources of uncertainty are often considered in UQ, namely epistemic and aleatory uncertainty. In this contribution we only consider aleatory uncertainties, which are sources of uncertainty that can be characterized and analyzed by using a statistical approach.

Monte Carlo (MC) methods can be used to solve for QoIs in a radiation transport problem as an alternate to deterministic methods, which explicitly solve the Boltzmann transport equation for a complete profile of average particle behavior throughout the phase space of the problem [2]. Instead, MC methods treat the physical system as a statistical process, using nuclear data to construct probability distributions that describe the various ways particles could behave in the system. Individual particles are simulated and their behavior (*e.g.* colliding with other particles, exiting the system, etc.) is recorded in tallies based on what information the user might want. The central limit theorem can then be applied to extrapolate the tallied behavior of the simulated particles to the average behavior of all particles in the system, with some associated uncertainty based on the number of particles simulated. Monte Carlo methods are useful depending on the information needed by the user or the problem space itself; for example, they can be used to handle complex

*Oregon State University, Sandia National Laboratories, clemekay@oregonstate.edu
†Sandia National Laboratories, ggeraci@sandia.gov
‡Sandia National Laboratories, aolson@sandia.gov

geometries more effectively than deterministic solvers [2].

On the other end, UQ can be performed using a variety of strategies such as surrogates [7], sampling-based strategies [11], or hybrid strategies in which a sampling technique is used for the construction of a surrogate [1,4,8]. Sampling-based methods are generally more robust and flexible for applications characterized by noisy QoIs, poor solution regularity and large input dimensions, and, therefore, in this contribution we focus on them since these features are well suited for solving radiation transport problems based on MC transport solvers. Moreover, purely sampling approaches can be incorporated into hybrid approaches for the surrogate construction without extensive modifications, *e.g.* in the case of Polynomial Chaos Expansion (PCE) (see [4,8]). The main sampling method is Monte Carlo, which is a provably convergent method for obtaining statistics of problems with arbitrary dimension and regularity. Notably, MC is also popular due to the possibility of implementing it in an embarrassingly parallel fashion. Another important feature of MC is its ability to provide an estimate of the error in resolving statistics of the QoI, *e.g.* the expected value, and therefore provide a rigorous mathematical framework to assess its convergence. Our goal in this contribution is to show how these properties can be used to study the convergence properties of an MC UQ method applied to an MC radiation transport code. The combination of these strategies can be in fact interpreted as a nesting of two MC estimators for which convergence to the true statistics can be studied as a function of the number of both the UQ samples and radiation transport particles.

The remainder of this paper is organized as it follows. In Section 2, the theoretical contributions of the paper are introduced. In Section 3, the radiation transport problem that we want to solve is described along with the definition of the relevant uncertainty parameters. In Section 4, we present numerical results for several UQ scenarios concerning the 1D radiation transport problem in which we we corroborate our theoretical findings. Concluding remarks close the paper in Section 5.

2. Theory for nested sampling estimators. In this section, we describe the mathematical foundations behind the use of sampling estimators for MC radiation transport (RT) codes. We present the nested estimators in a general context and highlight how these concepts can be applied to other applications in which there is a source of randomness that cannot be explicitly controlled, *i.e.* in the presence of a generic non-deterministic code. The framework that we want to present and analyze here consists of an MC sampling estimator for the UQ that is used to compute statistics (we will focus here on the variance) of a QoI.

We introduce here some definitions and the notation that will be used in the rest of the manuscript. We consider for simplicity a generic scalar QoI Q, which is a function of a vector of input parameters $\xi \in \Xi \subseteq \mathcal{R}^d$, where $d \in \mathcal{N}_0$ is the number of uncertain parameters. As is typical for UQ, we consider a joint probability density function $p(\xi)$ for ξ, such that $\int_\Xi p(\xi)\,\mathrm{d}\xi = 1$.

Moreover, we are interested in computing (central) moments of Q, such as

$$\mathbb{E}[Q] = \int_\Xi Q(\xi)\,p(\xi)\,\mathrm{d}\xi \quad \text{and} \quad \mathbb{V}ar[Q] = \int_\Xi \left(Q(\xi) - \mathbb{E}[Q]\right)^2 p(\xi)\,\mathrm{d}\xi, \tag{2.1}$$

with $\mathbb{E}[Q]$ and $\mathbb{V}ar[Q]$ denoting the mean and the variance of Q, respectively.

The MC approximations for the mean and variance can be written by drawing N_ξ samples for the QoI, each of them corresponding to the solution of a possibly expensive

computational code (the RT in our case), as

$$\begin{aligned}\mathbb{E}\left[Q\right] &\approx \frac{1}{N_\xi}\sum_{i=1}^{N_\xi} Q(\xi^{(i)}) \\ \mathbb{V}ar\left[Q\right] &\approx \frac{1}{N_\xi - 1}\sum_{i=1}^{N_\xi}\left(Q(\xi^{(i)}) - \frac{1}{N_\xi}\sum_{k=1}^{N_\xi} Q(\xi^{(k)})\right)^2 .\end{aligned} \tag{2.2}$$

Both estimators in Eq. (2.2) are unbiased; for more details, the interested reader can refer, for instance, to [11].

In RT applications, as well in all applications involving non-deterministic codes [5], it is common to be interested in QoIs that are obtained as statistics of elementary and observable events. Without a loss of generality, we consider here observing an elementary realization $f(\xi, \eta)$ of each code run. We have introduced a random variable η, possibly a random vector, to notionally represent the code's stochastic behavior. We note here that the knowledge of η is neither implied nor required for the following derivation, however it reflects the fact that multiple realizations of the code will produce event realizations f corresponding to a different realization of η (even for a fixed UQ random input ξ). On the other hand, we assume to be able to sample ξ from $p(\xi)$ and to be able to obtain multiple code evaluations for the same value of ξ. In the RT case, the elementary event f would correspond to a particle history response (*i.e.* tally), ξ would be the set of UQ parameters, and η would be the inaccessible vector of random variables used by the code to generate the particle history and transport events (absorption, scattering, *etc.*) during a particle 'random walk.'

In the following, we consider that $Q(\xi)$ is obtained as an expected value of elementary events f over multiple realizations of η, generated internally in a non-deterministic code, as

$$Q(\xi) \stackrel{\text{def}}{=} \mathbb{E}_\eta\left[f(\xi, \eta)\right], \tag{2.3}$$

where $\mathbb{E}_\eta\left[\cdot\right]$ is a shorthand used to indicate the expected value over realizations drawn with respect to the variable η. For RT applications, this corresponds to the expected value over a number of particle histories. In practice, RT codes can only be queried a finite number of times for a fixed UQ parameter configuration, so the value of $\mathbb{E}_\eta\left[f(\xi, \eta)\right]$ can only be approximated. Our contribution here is to understand the effect of the MC RT approximation of this term and how it propagates and affect the statistics we want to evaluate for the UQ. An MC estimator can be written for the approximation of Eq. (2.3) as

$$Q(\xi) = \mathbb{E}_\eta\left[f(\xi, \eta)\right] \approx \frac{1}{N_\eta}\sum_{j=1}^{N_\eta} f(\xi, \eta^{(j)}) \stackrel{\text{def}}{=} \tilde{Q}(\xi), \tag{2.4}$$

where our approximation $\tilde{Q}(\xi)$ also introduces the notation $\tilde{\cdot}$ to indicate a quantity that embeds the noise/stochasticity introduced by the MC solver, observed with a finite number of instances. If we combine Eqs. (2.2) and (2.4), we can obtain an MC-MC estimator for the expected value of the QoI

$$\mathbb{E}\left[Q\right] \approx \frac{1}{N_\xi}\sum_{i=1}^{N_\xi} Q(\xi^{(i)}) \approx \frac{1}{N_\xi}\sum_{i=1}^{N_\xi} \tilde{Q}(\xi^{(i)}) = \frac{1}{N_\xi}\sum_{i=1}^{N_\xi}\left(\frac{1}{N_\eta}\sum_{j=1}^{N_\eta} f(\xi^{(i)}, \eta^{(j)})\right) \stackrel{\text{def}}{=} \hat{Q}, \tag{2.5}$$

where we introduced the notation $\hat{\cdot}$ to indicate a sample estimator over the UQ parameter space.

Our goal in the next sections is to present several features of this estimator and the variance estimator, under the assumption that the number of histories N_η is constant for each UQ sample. Though not strictly required, this simplifies the derivation while still providing insights into the more general case.

2.1. Statistical properties of the MC-MC estimator. In this section, we discuss the features of the MC-MC estimator. We start by considering the estimator bias

$$\mathbb{E}\left[\hat{Q}\right] = \mathbb{E}\left[\frac{1}{N_\xi}\sum_{i=1}^{N_\xi}\tilde{Q}(\xi^{(i)})\right] = \mathbb{E}\left[\tilde{Q}(\xi)\right] = \mathbb{E}_\xi\left[\mathbb{E}_\eta\left[f(\xi,\eta)\right]\right] = \mathbb{E}_\xi\left[Q(\xi)\right], \tag{2.6}$$

where we explicitly indicate the space over which we compute the expected value[1].

The estimator is unbiased due to the linearity of the expected value operators, so it is important to study its variance to understand the impact of the number of both UQ samples N_ξ and histories N_η. The variance of the MC-MC estimator Eq. (2.5) is

$$\begin{aligned}\mathbb{V}ar\left[\hat{Q}\right] &= \mathbb{V}ar\left[\frac{1}{N_\xi}\sum_{i=1}^{N_\xi}\left(\frac{1}{N_\eta}\sum_{j=1}^{N_\eta}f(\xi,\eta^{(j)})\right)\right] \\ &= \mathbb{V}ar\left[\frac{1}{N_\xi}\sum_{i=1}^{N_\xi}\tilde{Q}(\xi^{(i)})\right] \\ &= \frac{1}{N_\xi}\mathbb{V}ar\left[\tilde{Q}\right].\end{aligned} \tag{2.7}$$

The variance of the quantity $\tilde{Q}$ is indeed a function of the set of samples drawn for both ξ and η. In order to evaluate this term, we can turn to the law of total variance

$$\mathbb{V}ar\left[Z(X,Y)\right] = \mathbb{V}ar_Y\left[\mathbb{E}_X\left[Z\right]\right] + \mathbb{E}_Y\left[\mathbb{V}ar_X\left[Z\right]\right], \tag{2.8}$$

where X, Y and $Z = Z(X,Y)$ are three random variables.

Applied to $\mathbb{V}ar\left[\tilde{Q}\right]$, the law of total variance provides a strategy to decompose this variance into the variance associated with the UQ parameters ξ and the variance associated with the set of particle histories η. It is important to remark here that, due to computational cost limitations, it is not always possible to obtain an approximation of $\tilde{Q}$ with a large number of histories, in the context of a UQ workflow in which multiple $\tilde{Q}$ should be evaluated for several values of the input vector ξ. Therefore, it is important to be able to quantify the additional estimator variance introduced by the MC over η, which can be written as

$$\begin{aligned}\mathbb{V}ar\left[\tilde{Q}\right] &= \mathbb{V}ar_\xi\left[\mathbb{E}_\eta\left[\tilde{Q}\right]\right] + \mathbb{E}_\xi\left[\mathbb{V}ar_\eta\left[\tilde{Q}\right]\right] \\ &= \mathbb{V}ar_\xi\left[\mathbb{E}_\eta\left[\frac{1}{N_\eta}\sum_{j=1}^{N_\eta}f(\xi,\eta^{(j)})\right]\right] + \mathbb{E}_\xi\left[\mathbb{V}ar_\eta\left[\frac{1}{N_\eta}\sum_{j=1}^{N_\eta}f(\xi,\eta^{(j)})\right]\right] \\ &= \mathbb{V}ar_\xi\left[\mathbb{E}_\eta\left[f(\xi,\eta)\right]\right] + \mathbb{E}_\xi\left[\frac{\mathbb{V}ar_\eta\left[f(\xi,\eta)\right]}{N_\eta}\right] \\ &= \mathbb{V}ar_\xi\left[Q\right] + \mathbb{E}_\xi\left[\frac{\sigma_\eta^2}{N_\eta}\right] = \mathbb{V}ar_\xi\left[Q\right] + \mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right],\end{aligned} \tag{2.9}$$

[1]Please note that we do not always write explicitly that space and use $\mathbb{E}[\cdot]$ in order to simplify the notation whenever the integration variables are easy to identify by the context.

where σ_η^2 is defined as the variance of the histories for one fixed UQ parameter, *i.e.* $\sigma_\eta^2 \overset{\text{def}}{=} \mathbb{V}ar_\eta\left[f(\xi,\eta)\right]$, and $\sigma^2_{RT,N_\eta} \overset{\text{def}}{=} \frac{\sigma_\eta^2}{N_\eta}$ is the corresponding MC RT estimator variance. The expression above relates the true variance of the QoI with respect to the UQ parameters, $\mathbb{V}ar_\xi\left[Q\right]$, and the variability introduced by the use of a finite number of particle histories in the RT Monte Carlo solver, σ^2_{RT,N_η}, averaged over the UQ parameter space. Both terms contribute to the polluted variance $\mathbb{V}ar\left[\tilde{Q}\right]$, which is the variance that one could observe by sampling the QoI obtained for each RT calculation using N_η particles. In practice, without realizing that an additional contribution to the variance is introduced by σ^2_{RT,N_η}, a practitioner would overestimate the parameter variance by assuming $\mathbb{V}ar_\xi\left[Q\right] \approx \mathbb{V}ar_\xi\left[\tilde{Q}\right]$. Similarly, the variance of the MC-MC estimator for the QoI is obtained by combining Eqs. (2.7) and (2.9)

$$\mathbb{V}ar\left[\hat{Q}\right] = \frac{\mathbb{V}ar_\xi\left[Q\right] + \mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right]}{N_\xi}. \tag{2.10}$$

The goal of a precise estimator is to obtain statistics with the lowest possible variance for a prescribed computational budget. In the case of this MC-MC estimator with the assumption of constant N_η for all N_ξ UQ runs, the total computational budget is $\mathcal{C} = N_\xi \times N_\eta$. Therefore, it is possible to note how the variance $\mathbb{V}ar\left[\hat{Q}\right]$ is minimized for $N_\eta = 1$, which suggests that the most effective estimator for the mean is obtained by using a single particle history per UQ simulation[2]. The interested reader can refer to [4], where these estimators have been discussed in the context of non-intrusive spectral projection for polynomial chaos computations. In the next section, we discuss the impact of the number of particles N_η in the variance estimation.

2.2. Variance deconvolution and practical implementation. The law of total variance allows for the decomposition of the total variance, namely $\mathbb{V}ar_\xi\left[\tilde{Q}\right]$, into two contributions: the true QoI variance, $\mathbb{V}ar_\xi\left[Q\right]$, and the average over the UQ parameter space of the noise introduced by the limited number of particle histories, $\mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right]$, which also corresponds to the estimator variance of the RT MC solver. A few considerations are in order here. In general, we are interested in characterizing the variance of the QoI $\mathbb{V}ar_\xi\left[Q\right]$, referred to hereafter simply as $\mathbb{V}ar\left[Q\right]$. However, we cannot access this quantity directly. As previously discussed, the QoI Q can be only approximated with $\tilde{Q}$, and this latter quantity can be considered to be polluted by MC noise. On the other hand, the possibility of obtaining several evaluations of $\tilde{Q}$ (for several sample of ξ) makes its variance an accessible quantity, *i.e.* it can be estimated. Similarly, given multiple particle histories per UQ sample, it is possible to estimate the term $\sigma_\eta^2 = \mathbb{V}ar_\eta\left[f\right]$ at each UQ parameter location, and, consequently, σ^2_{RT,N_η}.

It follows that the true variance can be written by removing the noise introduced by the finite number of particles from the polluted variance

$$\mathbb{V}ar_\xi\left[Q\right] = \mathbb{V}ar\left[Q\right] = \mathbb{V}ar\left[\tilde{Q}\right] - \mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right], \tag{2.11}$$

and its sample estimator counterpart can then be written as

$$\mathbb{V}ar\left[Q\right] \approx S^2 = \tilde{S}^2 - \hat{\mu}_{\sigma^2_{RT,N_\eta}}, \tag{2.12}$$

[2]We note that in the case of $N_\eta = 1$, it would not be possible to estimate the term $\mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right]$.

where S^2 and $\tilde{S}^2$ represent the sample estimators for the true (inaccessible) and polluted variances, respectively and $\hat{\mu}_{\sigma^2_{RT,N_\eta}}$ indicates the sample mean of the MC RT transport variance over the UQ space.

If the event associated with each particle history is available, *i.e.* $\left\{f(\xi^{(i)},\eta^{(j)})\right\}$ with $i=1,\cdots,N_\xi$ and $j=1,\cdots,N_\eta$, we can define

$$\begin{aligned}\tilde{S}^2 &= \frac{1}{N_\xi-1}\sum_{i=1}^{N_\xi}\left(\tilde{Q}(\xi^{(i)})-\frac{1}{N_\xi}\sum_{k=1}^{N_\xi}\tilde{Q}(\xi^{(k)})\right)^2\\ \hat{\mu}_{\sigma^2_{RT,N_\eta}} &= \frac{1}{N_\xi}\sum_{i=1}^{N_\xi}\frac{\sigma^2_\eta(\xi^{(i)})}{N_\eta},\end{aligned} \tag{2.13}$$

where the term σ^2_η is approximated, for each UQ sample, with an additional sample variance estimator

$$\sigma^2_\eta(\xi^{(i)}) \approx \hat{\sigma}^2_\eta(\xi^{(i)}) = \frac{1}{N_\eta-1}\sum_{j=1}^{N_\eta}\left(f(\xi^{(i)},\eta^{(j)})-\frac{1}{N_\eta}\sum_{k=1}^{N_\eta}f(\xi^{(i)},\eta^{(k)})\right)^2. \tag{2.14}$$

We note here that a similar variance deconvolution strategy has been also adopted for radiation problems in [9], in which an embedded implementation has been also discussed under the name of Embedded VAriance DEconvolution (EVADE). More recently, the EVADE algorithm has been also used to improve the efficiency of RT computations in the presence of stochastic media in [13]. The primary difference between Eq. (2.12) and the estimator adopted in [9] is that in [9], the total variance is written for the case of a single particle history per UQ sample, and therefore the term $\mathbb{V}ar\left[\tilde{Q}\right]$ is always approximated by using only one particle history. We discuss the properties of both estimators in more detail in the following section.

2.3. Connection with estimator proposed in [9]. We consider here the relationship between the estimator proposed in [9] and the estimator derived in Eq. (2.12). We first note that in the original estimator formulation, only one history per UQ sample was used in the computation of the variance $\mathbb{V}ar\left[\tilde{Q}\right]$, effectively approximating $\tilde{Q}(\xi^{(i)}) = f(\xi^{(i)},\eta^{(1)})$, even in the case where multiple particle histories were used for other calculations[3].

If one particle history per sample is used we can write Eq. (2.12) as

$$\mathbb{V}ar\left[Q\right] \approx S^2_I = \tilde{S}^2_I - \hat{\mu}_{\sigma^2_{RT,N_\eta}}, \tag{2.15}$$

where

$$\tilde{S}^2_I = \frac{1}{N_\xi-1}\sum_{i=1}^{N_\xi}\left(f(\xi^{(i)},\eta^{(1)})-\frac{1}{N_\xi}\sum_{k=1}^{N_\xi}f(\xi^{(k)},\eta^{(1)})\right)^2. \tag{2.16}$$

As a first result, we note that the estimators $\tilde{S}^2$, $\tilde{S}^2_I$ and $\hat{\mu}_{\sigma^2_{RT,N_\eta}}$ are unbiased estimators,

[3]We note here that the notation $f(\xi^{(i)},\eta^{(1)})$ represents an useful conceptual aid, however the analysis would be exactly the same if $\tilde{Q}(\xi^{(i)})$ was approximated with a single particle history randomly picked among the vector $\left\{f(\xi^{(i)},\eta^{(j)})\right\}_{j=1}^{N_\eta}$

i.e.

$$\begin{aligned} \mathbb{E}\left[\tilde{S}^2\right] &= \mathbb{E}\left[\tilde{S}_I^2\right] = \mathbb{V}ar_\xi\left[\tilde{Q}\right] \\ \mathbb{E}\left[\hat{\mu}_{\sigma^2_{RT,N_\eta}}\right] &= \mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right], \end{aligned} \tag{2.17}$$

and, therefore, both estimators, S^2 and S_I^2, are also unbiased estimators of $\mathbb{V}ar\left[Q\right]$.

On the other hand, the variances of the estimators S^2 and S_I^2 can be written as

$$\begin{aligned} \mathbb{V}ar\left[S^2\right] &= \mathbb{V}ar\left[\tilde{S}^2\right] + \mathbb{V}ar\left[\hat{\mu}_{\sigma^2_{RT,N_\eta}}\right] - 2\mathbb{C}ov\left[\tilde{S}^2, \hat{\mu}_{\sigma^2_{RT,N_\eta}}\right] \\ \mathbb{V}ar\left[S_I^2\right] &= \mathbb{V}ar\left[\tilde{S}_I^2\right] + \mathbb{V}ar\left[\hat{\mu}_{\sigma^2_{RT,N_\eta}}\right] - 2\mathbb{C}ov\left[\tilde{S}_I^2, \hat{\mu}_{\sigma^2_{RT,N_\eta}}\right]. \end{aligned} \tag{2.18}$$

Although we are not providing here a closed-form solution for these terms, it is possible to infer that the variances associated to the terms involving $\tilde{S}_I^2$ would be larger than their counterpart for $\tilde{S}^2$ due to the larger number of histories used in the current estimator.

In the remainder of this paper, we provide numerical evidence on how the current estimator can be used for an efficient unbiased estimation of the parameteric variance of a RT QoI and how the present formulation can outperform the estimator based on a coarse approximation of $\mathbb{V}ar\left[\tilde{Q}\right]$ for several UQ scenarios in radiation transport.

3. 1D Radiation Transport Problem. Both the MC-MC estimator developed in Section 2 and the estimator developed in [9] are applied here to an example radiation transport problem such that the inner MC loop is a Monte Carlo radiation transport solver described in further detail in Section 3.2, and the outer MC loop is over the UQ parameter space. Our goal is to provide an estimate of the variance of the QoI, over the UQ parameter space, which for this example problem is transmittance through a slab.

3.1. Problem Description. We solve the stochastic, one-dimensional, neutral-particle, mono-energetic, steady-state radiation transport equation with a normally incident beam source of magnitude one:

$$\mu\frac{\partial\psi(x,\mu,\xi)}{\partial x} + \Sigma_t(x,\xi)\psi(x,\mu,\xi) = \int_{-1}^{1} d\mu'\psi(x,\mu',\xi)\frac{\Sigma_s(x,\mu'\to\mu,\xi)}{2}, \tag{3.1}$$

$$0 \le x \le L; \quad -1 \le \mu \le 1, \tag{3.2}$$

$$\psi(0,\mu,\xi) = 1, \mu > 0; \quad \psi(L,\mu,\xi) = 0, \mu < 0 \tag{3.3}$$

where $\psi(x,\mu,\xi)$ is angular flux, $\Sigma_t(x,\xi)$ is total cross section, $\Sigma_s(x,\mu,\xi)$ is scattering cross section, and x, μ, and ξ respectively denote dependence on space, angle, and the vector of independent uncertain variables. The problem boundaries are fixed, i.e. $x \in [0, L]$, as are the locations between material regions. The problem is solved for two different scenarios: attenuation only, in which $\Sigma_s(x,\mu,\xi) = 0$; and with both attenuation and isotropic scattering. For both scenarios, the total cross section of each material is assumed to be uniformly distributed. In the scenario which also involves scattering, the ratio of scattering to total cross section c is distributed uniformly and independently of Σ_t. We consider a slab with a total of M materials, where for each region m the total cross section is defined as

$$\Sigma_{t,m}(\xi_m) = \bar{\Sigma}_{t,m} + \Sigma^{\Delta}_{t,m}\xi_m \tag{3.4}$$

with $\bar{\Sigma}_{t,m}$ representing the average total cross section and $\Sigma^{\Delta}_{t,m}$ the deviation from the mean value. Furthermore, a random parameter $\xi_m \sim \mathcal{U}[-1,1]$ is used to represent the variability

of $\Sigma_{t,m}(\xi) \sim \mathcal{U}[\bar{\Sigma}_{t,m} - \Sigma^{\Delta}_{t,m}, \bar{\Sigma}_{t,m} + \Sigma^{\Delta}_{t,m}]$. For cases with scattering, the scattering ratio is defined analogously using

$$c_m(\xi_{M+m}) = \bar{c}_m + c^{\Delta}_m \xi_{M+m} \tag{3.5}$$

where $\xi_{M+m} \sim \mathcal{U}[-1, 1]$, with $m = 1, \ldots, M$, is a uniformly distributed random variable. We note that in the attenuation-only case the problem contains a number of uncertain parameters equal to the number of materials, *i.e.* $\xi \in \mathbb{R}^d$ with $d = M$, whereas in the case of both attenuation and scattering $d = 2M$.

3.2. Monte Carlo Particle Transport and Woodcock Delta Tracking. Though Equation 3.1 can be thought of as the governing equation for this example problem, Section 1 points out that an estimate of our quantity of interest is obtained not by analytically or numerically solving the Boltzmann equation, but by averaging the tallied responses of individual particles [2]. In this example problem, the tallied QoI is transmittance; a response is tallied when a particle exits the far end of the slab, and a history is terminated when a particle is either absorbed or scatters to exit back through the beam-incident side of the slab. The series of random samples used to determine when the particle will reach a position of interest and what will happen when it does is described stochastically using η, as outlined in Section 2.

Particles' 'random walks' begin with the initial conditions in Eq. (3.1). They're moved through the system by randomly sampling the distance to their next collision[4], d_c ,

$$d_c = \frac{-ln(\Gamma)}{\Sigma_t}, \ \Gamma \in [0, 1) \tag{3.6}$$

where Γ is a randomly sampled number on [0,1) [2]. Equation 3.6, and therefore the calculated distance to collision, remains accurate as long as Σ_t is constant. For homogeneous media where $\Sigma_{t,m}$ is constant across material m, this is uncomplicated [5]. However, for heterogeneous media modeled as having different material sections separated by boundaries, this can become computationally expensive [12]. If the distance to a boundary is less than the calculated distance to collision and the particle would move from material m to material m', traditional collision-distance tracking methods only move the particle so far as the boundary location, then re-calculate d_c using $\Sigma_{t,m'}$ [12].

Instead, Woodcock-delta tracking calculates d_c using a majorant cross section Σ_M, which is typically taken to be the largest total cross section the particle might encounter along its direction of travel. Using the largest possible cross-section renders checking whether a particle has crossed a material boundary unnecessary; the probability of collision is either correct or overestimated, making the calculated distance to collision either correct or underestimated. This is corrected for by taking the ratio between Σ_t at the collision site and Σ_M to be the probability of whether the collision has actually occurred. For a particle moved to a possible collision site in material i, if for a randomly sampled $\omega \in [0, 1)$,

$$\omega < \frac{\Sigma_{t,i}}{\Sigma_M} \ , \tag{3.7}$$

the collision has actually occurred and is evaluated accordingly. Otherwise the collision is rejected as hypothetical (also referred to as "delta-collisions") and a new distance to collision

[4] Readers interested in the derivation of the distance to collision can see ref [2], and a detailed derivation of Monte Carlo transport methods can be found in Ch. 9 of [3].

[5] Our problem assumes constant $\Sigma_{t,m}$ across material m. However, $\Sigma_{t,m}$ in reality can vary with energy, temperature, density, or changing material composition [6].

is sampled [6]. Eventually, the particle will either exit the system via the right-hand-side and be tallied for transmittance, or exit the system in some other way and not be tallied for this problem. The transmittance is averaged over all histories in the sample, and this is then repeated N_ξ times for UQ analysis.

3.3. Analytic Solution. For a problem in which $\Sigma_s(x) = 0$, i.e. attenuation-only, an analytic solution for uncollided transmittance of a normally incident beam through a slab is

$$T(\xi) = \psi(L, 1, \xi) = \exp\left[\sum_{m=1}^{M} \Sigma_{t,m}(\xi_m)\Delta x_m\right]. \tag{3.8}$$

The pth raw moment for the transmittance, as derived in [10], can be written as

$$\mathbb{E}\left[T^p\right] = \prod_{m=1}^{d} exp\left[-p\bar{\Sigma}_{t,m}\Delta x_m\right] \frac{sinh\left[p\Sigma^{\Delta}_{t,m}\Delta x_m\right]}{p\bar{\Sigma}_{t,m}\Delta x_m}, \tag{3.9}$$

which allows for an exact evaluations of central moments by adopting the well-known transformations from raw to central moments. For instance, for the variance, which is the second central moment, we can write

$$\mathbb{V}ar\left[T\right] = \mathbb{E}\left[T^2\right] - \mathbb{E}\left[T\right]^2. \tag{3.10}$$

Moreover, as it is also possible to compute the variance $\mathbb{E}_\xi\left[\sigma^2_{RT,N_\eta}\right]$ in closed form for this problem, this example is well suited for verification purposes.

4. Numerical Results. In this section we present the performance of the variance estimators described in the previous sections for two UQ analysis scenarios, namely attenuation-only and attenuation with scattering. We consider a 1D slab with 3 material sections[6] and fixed boundaries between them. Each problem is solved using a fixed computational cost of $N_\xi \times N_\eta = 1500$ particle histories, however we consider different combinations of N_ξ and N_η. Moreover, we run 25 000 repetitions of the estimators, which correspond to realizations of all random variables, both ξ and η, to generate estimators' distributions and statistics.

4.1. Scenario 1: Attenuation only. In Table 4.1, we report the right boundary location, average total cross section, and deviation from the mean for the cross section for each of the material sections. In Table 4.2, we report the statistics (mean and variance) for the new estimator S^2 and previous estimator S^2_I of $\mathbb{V}ar\left[T\right]$ where T is the calculated transmittance, obtained over 25 000 repetitions of the estimator and for different combinations of N_ξ and N_η. We also compare the calculated variance to the analytic solution outlined in Section 3.3 using the mean squared error (MSE), which measures both the bias and variance of the estimator distribution, thus providing combined metrics for both the accuracy and precision of our estimator. It should be noted again here that no matter the number of histories run for calculation of other QoI such as transmittance, the previous method [9] specifies the use of only one history (typically the first one) to calculate total polluted variance.

[6] The approach can be extended to higher number of sections without any modifications to the algorithm.

	m = 1	m = 2	m = 3
x_R	2.0	5.0	6.0
$\bar{\Sigma}_{t,m}$	0.90	0.15	0.60
$\Sigma^{\Delta}_{t,m}$	0.70	0.12	0.50

Table 4.1: Stochastic Attenuation Problem Parameters

(N_ξ, N_η)	(100,15)	(300,5)	(500,3)	(750,2)	Exact
$\mathbb{E}\left[S^2\right]$	5.506e-03	5.501e-03	5.519e-03	5.498e-03	5.505e-03
$\mathbb{E}\left[S_I^2\right]$	5.557e-03	5.446e-03	5.522e-03	5.508e-03	5.505e-03
$\mathbb{V}ar\left[S^2\right]$	4.328e-06	5.172e-06	7.636e-06	1.276e-05	-
$\mathbb{V}ar\left[S_I^2\right]$	4.716e-04	1.392e-04	7.270e-05	4.543e-05	-
$\mathbb{V}ar\left[S_I^2\right]/\mathbb{V}ar\left[S^2\right]$	108.96	26.91	9.52	3.56	-
MSE$[S^2]$	4.328e-06	5.172e-06	7.636e-06	1.276e-05	-
MSE$[S_I^2]$	4.716e-04	1.392e-04	7.270e-05	4.542e-05	-

Table 4.2: Parametric variance ($\mathbb{V}ar\left[T\right]$) results for the stochastic attenuation-only problem using 25000 repetitions. Note that the previous estimator [9] specifies the use of one history to calculate total polluted variance, causing the higher variance and mean squared error.

We first note that one would expect the MSE of an unbiased estimator to be equal to the variance, as it is possible to observe in Table 4.2, corroborating the unbiased nature of both estimators. We also observe from these results that for the same computational cost $N_\xi \times N_\eta = 1500$, the variance and MSE of S^2 using the novel estimator increase as the number of histories decreases. We observe the opposite using the previous estimator. Because the previous estimator's lower precision results from the use of only one history to estimate the total variance, as the number of histories gets closer to 1 this effect is minimized. Even with $\mathbb{V}ar\left[S^2\right]$ increasing as $\mathbb{V}ar\left[S_I^2\right]$ decreasing, each of the four cases shows a reduction in $\mathbb{V}ar\left[S^2\right]$ when using the new estimator over the old.

In Figure 4.1, we provide the results of the 25 000 repetitions, considering all tested combinations of N_ξ and N_η, for both estimators and we compare them with the exact solutions which is available for this problems.

The distributions from both methods centering around the analytic solution for $\mathbb{V}ar\left[T\right]$ provides additional evidence that both estimators are unbiased. However, $\mathbb{V}ar\left[T\right]$ calculated using our novel method has a tighter distribution around the analytic variance, corresponding with its higher statistical reliability. This is also evident from looking at the ratio between $\mathbb{V}ar\left[S_I^2\right]$ and $\mathbb{V}ar\left[S^2\right]$ reported in Table 4.2, which for this case varies approximately from 3 to 100. This holds even when comparing the estimators' respective most efficient cases, $\mathbb{V}ar\left[S_I^2\right](100,15)/\mathbb{V}ar\left[S^2\right](750,2) = 10.50$, approximately an order of magnitude of improvement. In a practical application, we would only be able to obtain a single realization for S^2 or S_I^2; therefore, it is preferable to use an estimator with a higher precision, *i.e.* smaller variance thus higher probability of being closer to the true statistics.

4.2. Case 2: Attenuation and scattering. For this test case, in Table 4.3 we report the right boundary location, average total cross section and deviation from its mean, and average scattering ratio $c_{s,m}$ and deviation from its mean for each of the material sections. The results of the estimators are summarized in Table 4.4. It is worth noting that though the mean variance appears to systemically change as a function of N_η, this is not the case; all of the solutions are within the statistical uncertainty, and for this series of repetitions happened to increase as N_η does. All calculated variances are higher than their attenuation-

only counterparts, caused by the inclusion of the second stochastic parameter for each of the materials, the scattering ratio. As with Case 1, $\mathbb{V}ar\left[S^2\right]$ decreases as the number of histories increases, and $\mathbb{V}ar\left[S_I^2\right]$ behaves inversely. Though $\mathbb{V}ar\left[S_I^2\right]/\mathbb{V}ar\left[S^2\right]$ is higher in Case 1 than Case 2, we still observe improvement when using the new estimator over the old, and the ratio of their respective best cases $\mathbb{V}ar\left[S_I^2\right](100,15)/\mathbb{V}ar\left[S^2\right](750,2) = 9.11$, still close to an order of magnitude of difference.

	m = 1	m = 2	m = 3
x_R	2.0	5.0	6.0
$\bar{\Sigma}_{t,m}$	0.90	0.15	0.60
$\Sigma^{\Delta}_{t,m}$	0.70	0.12	0.50
$\bar{c}_{s,m}$	0.50	0.50	0.50
$c^{\Delta}_{s,m}$	0.40	0.40	0.40

Table 4.3: Stochastic Absorption and Scattering Problem Parameters

(N_ξ, N_η)	(100,15)	(300,5)	(500,3)	(750,2)
$\mathbb{E}\left[S^2\right]$	9.689e-03	9.148e-03	8.201e-03	6.587e-03
$\mathbb{E}\left[S_I^2\right]$	9.879e-03	9.098e-03	8.345e-03	6.591e-03
$\mathbb{V}ar\left[S^2\right]$	9.143e-06	9.830e-06	1.158e-05	1.555e-05
$\mathbb{V}ar\left[S_I^2\right]$	5.481e-04	1.611e-04	8.327e-05	4.965e-05
$\mathbb{V}ar\left[S_I^2\right]/\mathbb{V}ar\left[S^2\right]$	59.95	16.39	7.19	3.19

Table 4.4: Parametric variance ($\mathbb{V}ar\left[T\right]$) results for the attenuation and scattering problem using 25000 repetitions. Note that the previous estimator [9] specifies the use of one history to calculate total polluted variance, causing the higher variance.

In Figure 4.2, we show the distributions of S^2 and $\tilde{S}^2$ over 25 000 repetitions for all tested combinations of (N_ξ, N_η). The variance is about an order of magnitude larger compared to the attenuation-only case, reported in Figure 4.1. This is further evidence of the increased variability induced by the second parameter for each material section.

Although there is no analytic solution for comparison, it is evident that the variance distributions are wider when calculated using the previous method compared to those from the novel one, as expected. We also note that the advantage of the novel estimator grows as N_η increases since it can take advantage of the additional particle histories to decrease the variability of the term $\mathbb{V}ar\left[\tilde{S}^2\right]$. Moreover, the contribution of the MC RT estimator variance is better resolved and, therefore, the polluted variance and the parametric variance are closer when the number of particle histories increases.

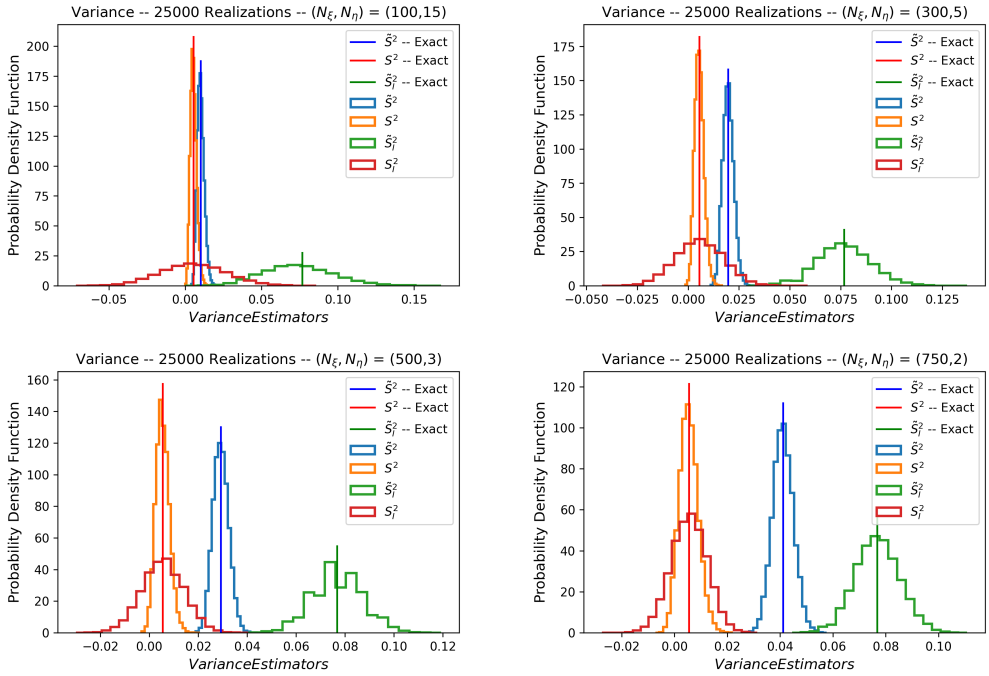


Fig. 4.1: 1D radiation transport problem with absorption only ($d = 3$). 25 000 repetitions for the estimators: the new S^2 and the previous S_I^2, [9], estimators are reported. Exact solutions are reported as vertical lines.

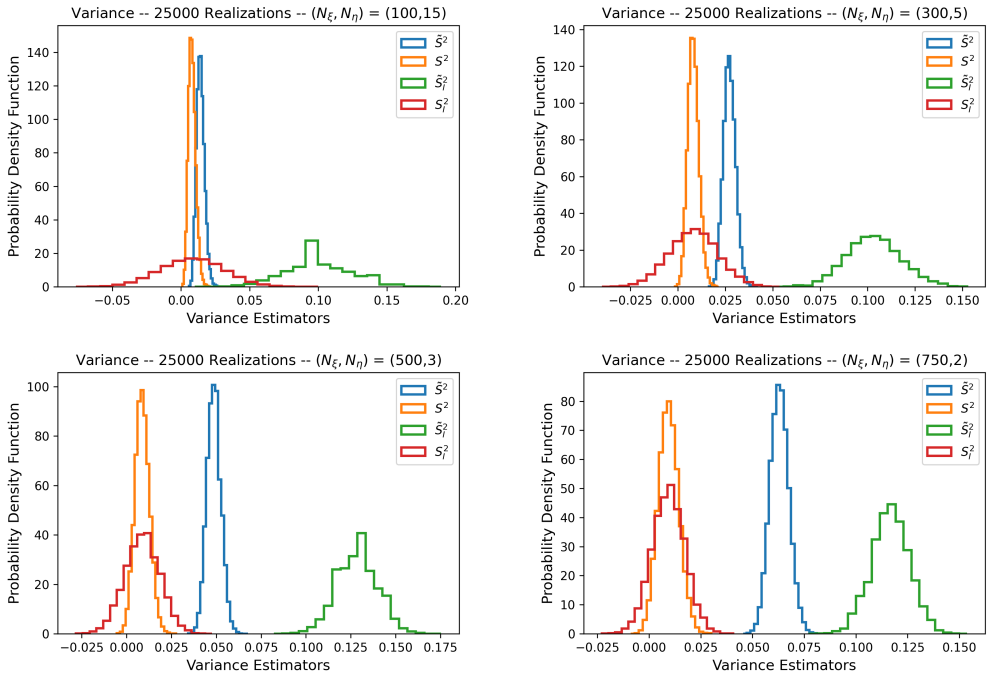


Fig. 4.2: 1D radiation transport problem with absorption and scattering ($d = 6$). 25 000 repetitions for the estimators: the new S^2 and the previous S_I^2, [9], estimators are reported.

5. Conclusions. In this contribution, we focused on the derivation of statistical properties for nested Monte Carlo estimators that can arise in applications involving stochasticity in the solver. In particular, we focused on radiation transport solvers that rely on a Monte Carlo method to obtain quantities of interest, *e.g.* transmittance, by tallying particle histories. We proposed a variance deconvolution approach which has the potential to improve the efficiency of parametric variance computations by taking into account the noise introduced by a limited number of particle histories. First, we demonstrated how the total variance (the variance of a quantity of interest over the space of UQ parameters and particle histories) is affected by the variance of a MC radiation transport solver. Second, we designed an unbiased estimator for the parametric variance based on the variance deconvolution. Moreover, we compared our newly derived estimator with the one previously introduced in [9] and demonstrated, through a series of numerical tests, that the new estimator has the potential to yield higher precision results. For the numerical tests, we considered the transmittance of neutral particles through a 1D slab whose material sections had uncertain properties, i.e. total and scattering cross sections.

Current research directions include the derivation of a closed-form solution for the estimator variance in order to study the role of the number of UQ samples and particle histories, efficient parallelization of the method for GPUs, and the integration of this strategy within larger UQ workflows such as the efficient construction of polynomial chaos expansion surrogates, following [4], and stochastic media UQ algorithms such as Conditional Point Sampling (CoPS) [13].

Acknowledgments. The authors would like to thank the two anonymous reviewers for their comments on an earlier version of this work. This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number DE-NA003967. This work was also supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References.

[1] T. Crestaux, O. L. Maitre, and J.-M. Martinez, *Polynomial chaos expansion for sensitivity analysis*, Reliability Engineering & System Safety, 94 (2009), pp. 1161–1172.
[2] Diagnostics Applications Group, Applied Physics Division, Los Alamos National Lab, *MCNP - A general Monte Carlo n-particle transport code*, 4c ed., March 2000.
[3] J. J. Duderstadt and W. R. Martin, *Transport Theory*, John Wiley & Sons, 1979.
[4] G. Geraci and A. Olson, *Impact of sampling strategies in the polynomial chaos surrogate construction for Monte Carlo transport applications*, 2021.
[5] G. Geraci, L. Swiler, and B. Debusschere, *Multifidelity uq sampling for stochastic simulations*, 16th U.S. National Congress on Computational Mechanics, (2021).
[6] I. Lux and L. Koblinger, *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*, CRC Press, 1991.
[7] O. L. Maitre and O. Knio, *Spectral methods for uncertainty quantification. With applications to computational fluid dynamics*, Springer Netherlands, 2010.
[8] M. Merritt, G. Geraci, M. Eldred, and T. Portone, *Hybrid multilevel Monte Carlo - polynomial chaos method for global sensitivity analysis*, Computer Science Research Institute Summer Proceedings 2020, A.A. Rushdi and M.L. Parks, eds., Technical Report SAND2020-12580R, Sandia National Laboratories, pp. 68–86,, 2020.
[9] A. J. Olson, *Calculation of parametric variance using variance deconvolution*, in Transactions of the American Nuclear Society, vol. 120, 2019.

[10] A. J. Olson, A. K. Prinja, and B. C. Franke, *Error convergence characterization for stochastic transport methods*, in Transactions of the American Nuclear Society, vol. 116, 2017.
[11] A. B. Owen, *Monte Carlo theory, methods and examples*, 2013.
[12] J. S. Rehak, L. M. Kerby, M. D. DeHart, and R. N. Slaybaugh, *Weighted delta-tracking in scattering media*, Nuclear Engineering and Design, 342 (2018).
[13] E. H. Vu and A. J. Olson, *Conditional point sampling: A stochastic media transport algorithm with full geometric sampling memory*, Journal of Quantitative Spectroscopy and Radiative Transfer, 272 (2021), p. 107767.

BENCHMARKING NEURAL NETWORK ANSÄTZE FOR QUANTUM CHEMISTRY APPLICATIONS

COLLIN FRINK*, QUINN CAMPBELL†, CONOR SMITH‡, ANDREW D. BACZEWSKI§, AND TAMEEM ALBASH¶

Abstract. Quantum computers promise to enable the accurate and efficient simulation of certain challenging many-body systems, where classical computers force the user to choose accuracy or efficiency. As quantum computers mature it is increasingly critical to develop benchmarks that distinguish their simulation capabilities relative to classical computers. This requires the identification and analysis of classical algorithms that are in some sense representative of the classical state of the art. Recently, ansätze for many-body wave functions based on neural networks have been shown to be particularly promising. The goal of this article is to analyze the accuracy and efficiency of a variety of neural networks for simulating the ground states of small molecules. Specifically, we compare a relatively simple network architecture (restricted Boltzmann machines) to a more complex network architecture (PauliNet) that has been studied elsewhere in the literature and to full configuration interaction (FCI) exact calculations of the energy for a given basis. For restricted Boltzmann machines, errors and runtimes as functions of the number of hidden nodes and learning rates are reported.

1. Introduction. Among the most promising applications of quantum computing technologies appears to be the simulation of physical systems [5, 11]. Classical algorithms for approximating the energies of physical Hamiltonians tend to force a choice between systematically improvable accuracy and computational efficiency. Simply put, classical algorithms that require polynomial resources in the number of degrees of freedom (e.g., electrons, spins, etc.) deliver accuracies that are difficult (perhaps impossible) to quantify *a priori*, and those that deliver quantifiable and improvable accuracy require resources that scale exponentially. This is a broad and empirical statement and formal results in complexity theory paint a much more nuanced picture [9, 14], but for the simulation of molecular systems where high-accuracy experimental thermochemical data and numerous algorithms are available for comparison this appears to be true. The promise of quantum simulation algorithms is that they obviate this choice, requiring polynomial scaling quantum resources to achieve systematically improvable accuracy. Thus we expect that quantum computers will exceed the capabilities of classical computers at simulating some physical systems at some level of accuracy. This suggests the question, for a target level of accuracy how large of a quantum computer will be needed to surpass the abilities of any realistic classical computer? To answer this question we first need to assess the state-of-the-art in classical algorithms for simulating ground states of quantum many-body systems.

While there is a wide array of algorithms available for this task, in this work we focus specifically on stochastic variational algorithms. Variational Monte Carlo algorithms have a long history of delivering accurate estimates for ground state energies, limited primarily by the expressivity of the variational ansatz. In recent years, ansätze based on neural networks have been shown to be especially promising. While the initial work used restricted Boltzmann machines [16] in the context of spin Hamiltonians [2], more complex architectures (PauliNet [6] and FermiNet [15]) have been studied in the context of the electronic structure problem for small molecules. In this article we examine the performance of restricted Boltzmann machines (RBMs) for electronic structure problems (see also [4]) by using a fermion-to-qubit encoding to reframe the electronic structure problem for small molecules

*University of Wisconsin-Madison, cfrink@wisc.edu
†Sandia National Laboratories, qcampbe@sandia.gov
‡University of New Mexico, cssmith36@unm.edu
§Sandia National Laboratories, adbacze@sandia.gov
¶University of New Mexico, talbash@unm.edu

in terms of a spin Hamiltonian. While this architecture is not necessarily as physically intuitive as PauliNet or FermiNet, it provides a straight forward starting point for comparing the performance of stochastic variational algorithms on classical and quantum hardware.

The contents of this article are arranged as follows. In Section 2, we state the electronic structure problem, review the supporting software, and describe our methods for solving it. Section 3 provides results that illustrate the accuracy of this ansatz for different basis sets and numbers of hidden layers. We conclude with a discussion of the implications of our findings and future work in Section 4.

2. Methods.

2.1. Problem Statement.

We consider the generic electronic structure Hamiltonian of the form

$$\hat{H} = \sum_{pq}^{N_s} h_{pq} \hat{a}_p^\dagger \hat{a}_q + \sum_{pqrs}^{N_s} v_{pqrs} \hat{a}_p^\dagger \hat{a}_q^\dagger \hat{a}_r \hat{a}_s, \tag{2.1}$$

where h_{pq} and v_{pqrs} are real-valued coefficient tensors representing the one- and two-body terms for a given molecular configuration, N_s is the number of spin orbitals, and $\hat{a}_p$ ($\hat{a}_p^\dagger$) is a fermionic annihilation (creation) operator that acts on vectors in the Fock space generated by those spin orbitals. The generation of such a Hamiltonian from a description of a molecular geometry and Gaussian basis set is straight forward [19]. We note that the particular form of the electronic structure Hamiltonian has the total number of fermions as a good quantum number, and we will typically focus on a projection of Eq. 2.1 onto a Hilbert space, $\mathcal{H}_{N_e,N_s}$, with a fixed number of electrons, N_e, occupying the spin orbitals. Specifically, we are interested in finding vectors in that Hilbert space with a high degree of overlap with the eigenvector of $\hat{H}$ with the least eigenvalue.

To do so, we require a variational ansatz that implicitly or explicitly defines a trial vector in that Hilbert space as a function of p parameters,

$$|\psi\rangle = |\psi(\alpha_1, \ldots, \alpha_p)\rangle \in \mathcal{H}_{N_e,N_s}, \tag{2.2}$$

where α_i is the ith parameter indexed by integers from 1 to p and we will henceforth vectorize the parameters as $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p)$. In this article we will consider ansätze generated by an RBM architecture first introduced for quantum many-body problems in Ref. [2]. Given such an ansatz, we are interested in estimating the energy

$$E(\boldsymbol{\alpha}) = \frac{\langle\psi(\boldsymbol{\alpha})|\hat{H}|\psi(\boldsymbol{\alpha})\rangle}{\langle\psi(\boldsymbol{\alpha})|\psi(\boldsymbol{\alpha})\rangle}, \tag{2.3}$$

and specifically finding the minimum value that this functional takes for a particular parametrization,

$$E_0 = \min_{\boldsymbol{\alpha}} \frac{\langle\psi(\boldsymbol{\alpha})|\hat{H}|\psi(\boldsymbol{\alpha})\rangle}{\langle\psi(\boldsymbol{\alpha})|\psi(\boldsymbol{\alpha})\rangle}. \tag{2.4}$$

The dimension of $\mathcal{H}_{N_e,N_s}$ is $N_e!/N_s!(N_s - N_e)!$, so working directly with $|\psi\rangle$ in a basis that spans this space rapidly becomes classically intractable as the number of electrons (i.e., the size of the molecule) or spin orbitals (i.e., the resolution) increases. Fortunately, stochastic methods provide us with an efficient means by which we can estimate E_0 in spite of the combinatorial growth of the Hilbert space dimension.

Before proceeding, we review the form of the RBM ansatz, described in Ref. [2], that we use in this study. The visible layer of the RBM consists of N_s nodes,

each of which is a weighted (a_i) Pauli Z operator ($\sigma_{i,z}$) describing the occupation of a single fermionic mode in the computational basis states encoding $\mathcal{H}_{N_e,N_s} = \text{span}\,(|n_1,\ldots,n_{N_s}\rangle | n_i = (1+\sigma_{i,z})/2 \in \{0,1\})$. The single hidden layer of the RBM consists of a variable number of N_h auxiliary spins that are each weighted (b_i) and coupled to all of the visible nodes (W_{ij}). After tracing out the variables in the hidden layer, the RBM ansatz takes the form

$$|\psi(a_i,b_i,W_{ij})\rangle = \sum_{n_1,\ldots,n_{N_s}} 2\exp\left[\sum_i a_i\sigma_{i,z}\right]\prod_{i=1}^{N_h}\cosh\left[b_i+\sum_{j=1}^{N_s}W_{ij}\sigma_{i,z}\right]|n_1,\ldots,n_{N_s}\rangle, \tag{2.5}$$

where it is evident that $\boldsymbol{\alpha} = (a_i, b_i, W_{ij})$ for $i \in 1,\ldots,N_s$ and $j \in 1,\ldots,N_h$, such that there are $p = N_s + N_h + N_s N_h$ variational parameters in aggregate. To use this ansatz to describe the ground state of the electronic structure Hamiltonian in Eq. 2.1, we will need to compute its coefficients in the Pauli basis so that Eq. 2.3 can be estimated for a particular value of $\boldsymbol{\alpha}$.

2.2. Supporting software. For a given molecular geometry we generate an instance of the second quantized Hamiltonian in Eq. 2.1, and then use the Jordan–Wigner transformation [8] to map it onto an interacting spin model using the OpenFermion software package [12]. The resulting spin Hamiltonian, specified in terms of Pauli operators acting on a qubit Hilbert space, is one of the inputs for a bespoke software package that variationally optimizes the parameters of this ansatz, producing an estimate for E_0 in Eq. 2.4. The stochastic reconfiguration (SR) algorithm [17] is used to do so and described in Section 2.3. We will also present comparisons to results obtained using the PauliNet package, which instead uses artificial neural networks in a Slater-Jastrow framework [6]. Full configuration interaction (FCI) calculations were undertaken using the PySCF software package [18].

2.3. Overview of the solution algorithm. The SR algorithm efficiently finds the value of $\boldsymbol{\alpha}$ that estimates E_0 by iteratively updating $\boldsymbol{\alpha}$ (and the associated wave function) starting from some initial guess. This particular approach follows the gradient of the energy functional in Eq. 2.3, requiring the use of the gradient of the RBM ansatz with respect to its parameters, the specification of which is beyond the scope of this article. Ultimately, a linear system of equations are defined at each iteration of the SR algorithm, the solution of which is used to update the value of $\boldsymbol{\alpha}$. Monte Carlo (MC) sampling is used to efficiently estimate expectation values of the requisite matrix elements and MINRES-QLP [3] is used to iteratively solve the equations. The use of an iterative solver allows us to avoid precomputing and storing the entire SR update matrix, leading to a cost per iteration, in time and memory, that scales linearly with the number of parameters in our RBM ansatz and the number of MC samples used to estimate matrix-vector products. We will examine the impact of the learning rate in the SR algorithm on its performance. Heuristically speaking, the learning rate is a constant of proportionality that determines the extent to which the solution of the SR update equations is mixed with the previous value of $\boldsymbol{\alpha}$ to arrive at a new value after at each iteration [13]. The learning rate is decreased throughout the training and thus we will later refer to the initial and final values of the learning rate. The total time to solution will depend on the number of iterations required to reach convergence, which will itself be a function of the landscape associated with the problem instance, starting guess, and both learning rates. Unlike the cost per iteration, the total time to solution is not necessarily linear in the number of parameters or MC samples. We note that it may be advantageous to ultimately implement an adaptive learning rate solver to avoid manual tuning [1].

3. Results.

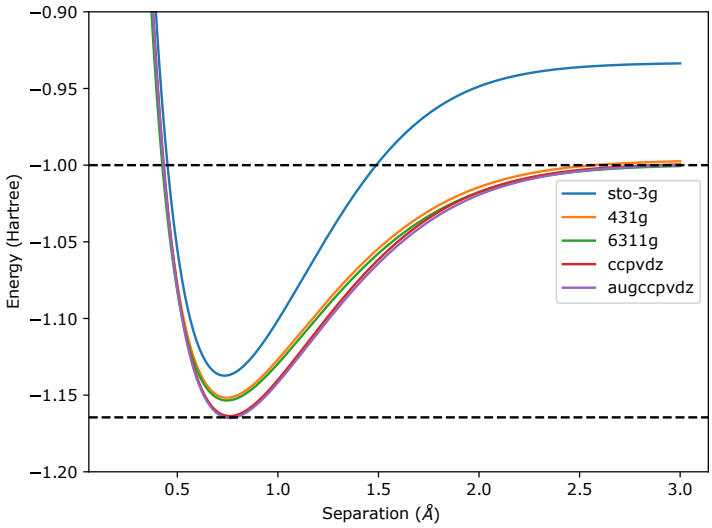


Fig. 3.1: Full configuration interaction calculations of the H_2 dissociation curve. The dashed lines indicate the equilibrium bonding (lower) and fully dissociated (upper) energies for this molecule. We note that the latter is simply twice the energy of two hydrogen atoms.

3.1. H_2 full configuration interaction results. Full Configuration Interaction (FCI) calculations provide exact solutions for a given basis and, thus, a convenient point of reference for qualifying the accuracy of the RBM and PauliNet ansätze. In what follows we will consider the dissociation of molecular hydrogen, H_2, chosen for its simplicity and ubiquity as a benchmark problem. Fig. 3.1 compares FCI results for three different basis sets that will be subsequently investigated using RBMs: STO-3G, 6-311G, and cc-PVDZ. 4-31G and aug-cc-PVDZ are also included, though we leave studying its performance in the RBM ansatz to future work.

Notably, the smallest basis set, STO-3G (4 orbitals), does not produce quantitatively accurate solutions even for such a simple problem. It requires moving to a 6-311G basis (12 orbitals) before we achieve quantitatively accurate results in the dissociated limit. Further increase in the basis set size to cc-PVDZ basis (20 orbitals) gets closer to the experimental dissociation energy [7], but we note that a more formal and precise comparison will require consideration for vibrational, radiative, and relativistic effects [10]. Regardless, the basis sets under consideration will not always capture the chemistry with high fidelity relative to experiment, but we can still benchmark the performance of our RBMs within a given basis accepting that they cannot outperform the FCI result for that basis.

3.2. Initial RBM investigations. Initial investigations into the effectiveness of RBM ansätze involved calculating the dissociation curves of H_2 for some of the basis sets considered in the previous Section. As described in Section 2.2, once a Jordan-Wigner transformation is applied to the H_2 Hamiltonian, the associated Pauli decomposition is given as input to the RBM. The structure of the RBM depends on said operators as the number of visible nodes in the RBM equals the number of qubits needed to span the mapped Hilbert space. The relevant qubit counts are: 4 (STO-3G), 8 (6-31G), 12 (6-311G) and 20 (cc-PVDZ). For these results, initial and final learning rates were set at 0.01 and 0.001 respectively,

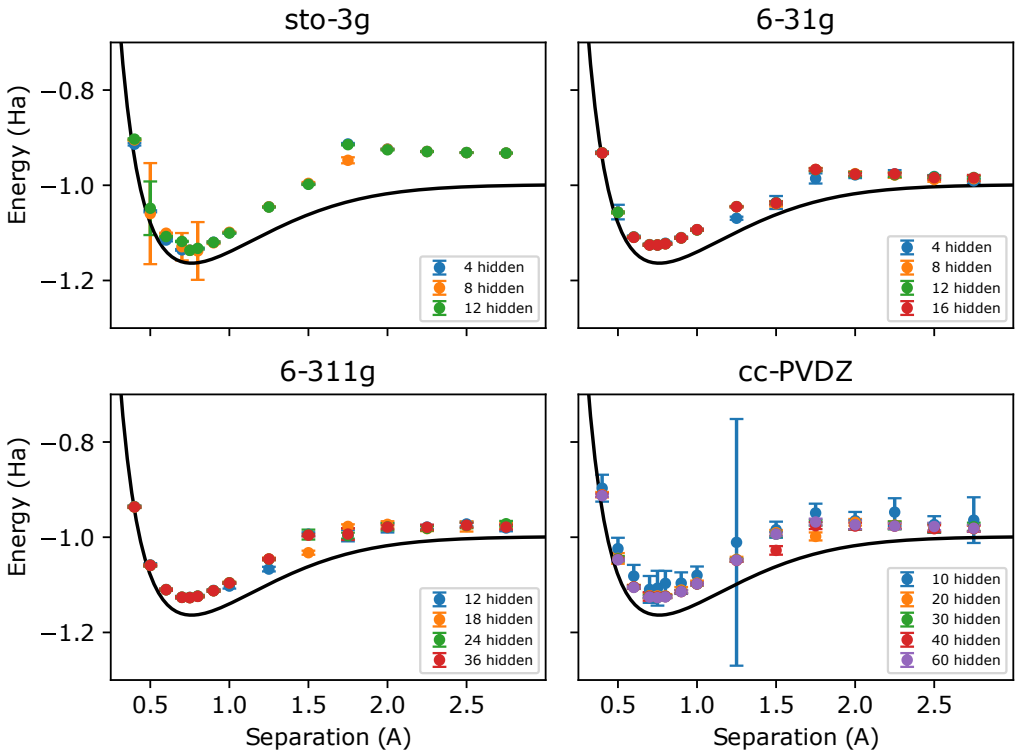


Fig. 3.2: Preliminary H_2 dissociation curves using RBM as functions of the basis set and number of hidden units. FCI curve using the cc-PVDZ basis is included in each plot for comparison to exact results. The initial and final learning rates were 0.01 and 0.001, respectively.

and the number of MC sweeps[1] per sample was set at 10, with a total of 200 samples. Each RBM instance ran for 20,000 iterations. As noted in Section 2.1, we are concerned with finding the ground state energy for a fixed number of electrons. In the Jordan–Wigner transformed picture, computational basis vectors corresponding to the same particle number will correspond to a fixed Hamming weight. However, in this first set of results we do not constrain the Hamming weight to be conserved and our estimated ground state might not have particle number as a good quantum number. Fig. 3.2 illustrates the results of this study, comparing the output of our RBM to the FCI result for the most complete basis set under consideration (cc-PVDZ).

Within each subplot, we vary the ratio of hidden nodes to visible nodes from 0.5 to 3, reasonable values for an RBM network structure. Only one unique RBM was used for each data point, and we report the sample mean of the last 1000 energy values from the training of said RBM. This helps explain some of the roughness of many of the curves, especially for the smaller basis sets, and this is found to be smoothed out by running multiple RBMs per data point in Section 3.5. Note that error bars are the standard deviation of the same sample, and are used as a measure of convergence. With some isolated exceptions, for all

[1] A MC sweep is defined as proposing an update for every visible node once.

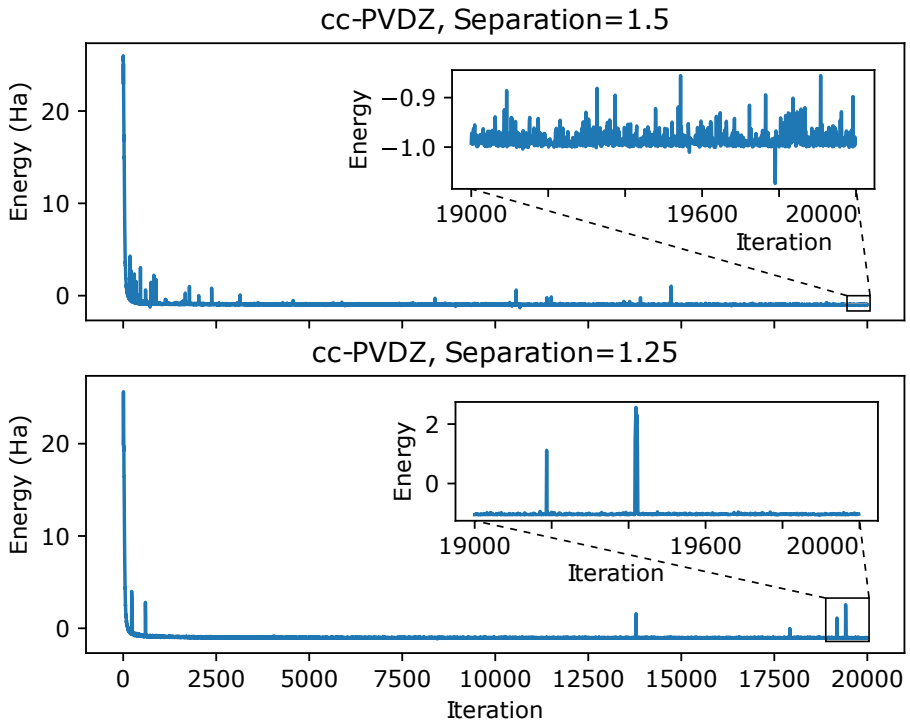


Fig. 3.3: Convergence of the 1.25 Å and 1.5 Å separation distance data points from the cc-PVDZ subplot in Fig. 3.2. The last 1000 values from each plot (of which are averaged to generate the two data points) are included as inset plots.

basis sets, when the number of hidden nodes is greater than the number of visible nodes the RBM converges well. The most notable failure to converge is evident on the cc-PVDZ subplot, at a proton-proton separation of 1.25 Å. Fig. 3.3 contrasts the training convergence of the cc-PVDZ 1.25 Å data point with a better converged cc-PVDZ 1.5 Å data point.

Both estimates appear to converge well before the last 1,000 iterations that are used in Fig. 3.2. However, in both plots there are large spikes that are quickly smoothed out after an iteration or two. Two such spikes occur in the last 1,000 iterations for the 1.5 Å data point, which increase its error bar. In order to obtain smoother results, either these spikes can be filtered out as outliers or more RBMs per data point can be run. One other takeaway from Fig. 3.3 is that the RBMs converge significantly before 20,000 iterations. This is useful for future calculations not just in reducing computational time, but also in estimating potential runtimes for more complicated geometries. Fig. 3.4 is a plot of the RBM wall time associated with the training stage as a function of the basis set and number of hidden nodes for the results from Fig. 3.2. For each basis set, we observe a linear trend in runtime as a function of number of hidden nodes because the number of network connections increases linearly in the RBM with respect to an increase in hidden nodes.

3.3. PauliNet performance. We next compare our RBM ansatz to an ansatz explored elsewhere in the literature, PauliNet. PauliNet takes advantage of a deep neural net-

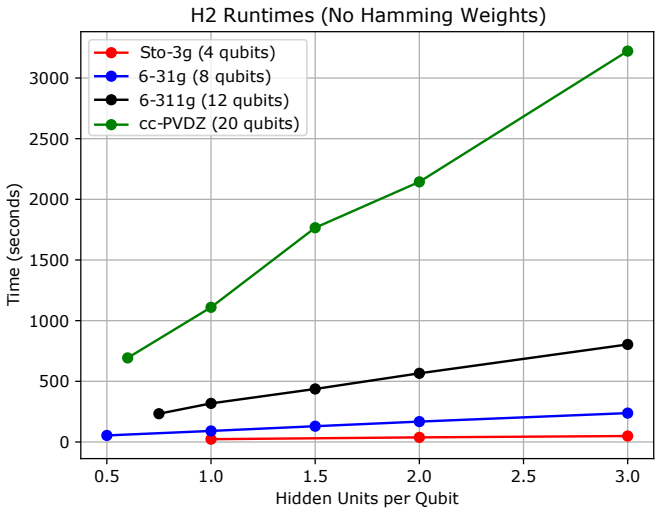


Fig. 3.4: Runtime for performing 20,000 iterations for the H_2 benchmark problem (see Fig. 3.2 for energies), as a function of the ratio of hidden to visible nodes and the basis set. It is approximately linear in the number of parameters defining the RBM.

work architecture that has been shown to achieve highly accurate results in various quantum chemistry applications while maintaining computational efficiency [6]. Results for the H_2 dissociation curve are included in Fig. 3.5. It is evident that PauliNet outperforms our RBM in accuracy for all proton-proton separations. However, per H_2 data point, Paulinet takes 10,800 seconds (3 hours), averaged over all runs, while the slowest RBM tested took just over 3,000 seconds. Considering Fig. 3.3 again, 3,000 seconds is likely an overestimate of the run time needed to achieve the observed accuracy and this suggests that it may be possible to realize a speedup using RBMs if we can improve their accuracy. It may also be worth investigating whether the time for PauliNet results can be similarly decrease by examining convergence plots similar to 3.3. We note that, with a single exception, the PauliNet energies are just below the FCI values for a cc-PVDZ basis set, which match experimental values well.

3.4. Determining ideal parameters for training RBMs. Our initial RBM results, reported in Section 3.2, exhibited irregular convergence and jagged dissociation curves. We subsequently performed a high-throughput search over input parameters for the SR algorithm with the goal of finding the optimal parameters for the H_2 benchmark problems. In Fig. 3.6 we examine how the initial and final learning rates used can influence the results using a cc-PVDZ basis at the equilibrium distance. The optimal learning rate is likely highly specific to the problem being examined. For this problem, we find that the lowest energies and standard deviations are achieved with both initial and final learning rates of 0.1 and 0.01.

In Fig. 3.7, we study the impact of using different numbers of samples and sweeps for each step of the training algorithm. These results are obtained using a 6-311G basis at the dissociation limit where we know the exact energy that should be reached and can therefore report error. As the sample size increases, accuracy increases using both measures

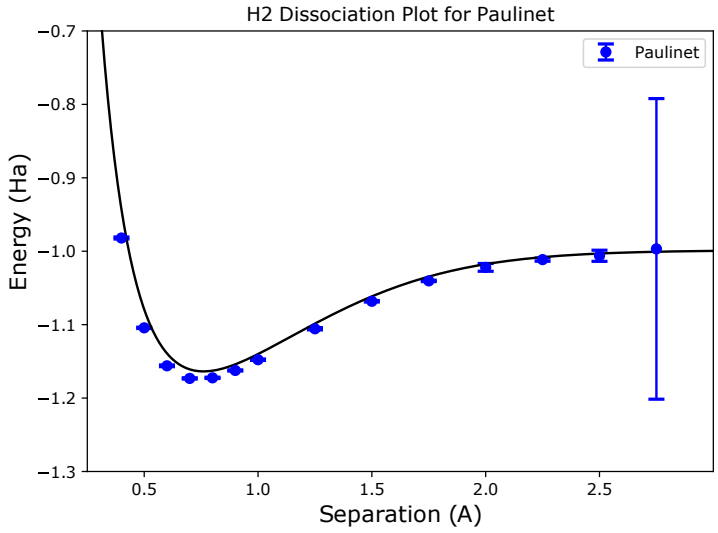


Fig. 3.5: PauliNet calculations of the H_2 dissociation curve. The black line is from an FCI calculation using the cc-PVDZ basis set.

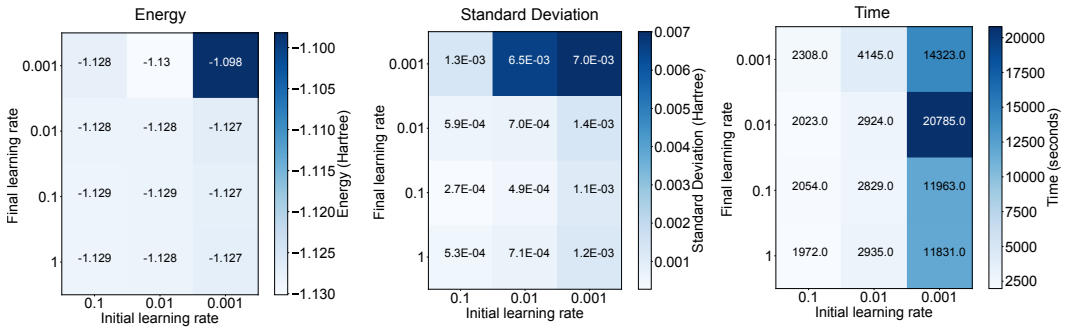


Fig. 3.6: The energy, standard deviation, and time to result as a function of the initial and final learning rates. We use the cc-PVDZ basis (20 qubits) with 40 hidden units running for 8000 steps.

of average training. Altering the number of sweeps does not appear to have a clear impact on the accuracy of results. Unlike our initial results in Section 3.2, we conserve Hamming weights in these calculations as to constrain the number of electrons.

3.5. Updated H_2 dissociation curve. Based on the results in Section 3.4, we selected an initial learning rate of 0.1, a final learning rate of 0.01, a sample size per step of 1,000, and 10 sweeps, and re-examined the problems studied in Section 3.2. The results for this parametrization are presented in Fig. 3.8, which is intended to be a refined version of Fig. 3.4. As discussed in Section 3.4, Hamming weight was constrained to 2 (the number of electrons) in generating Fig. 3.8. Each data point was generated by taking the average of the last 1000 training values for 5 independent RBMs, and is thus an average over 5000 training samples. Given the previous convergence results, the number of training iterations

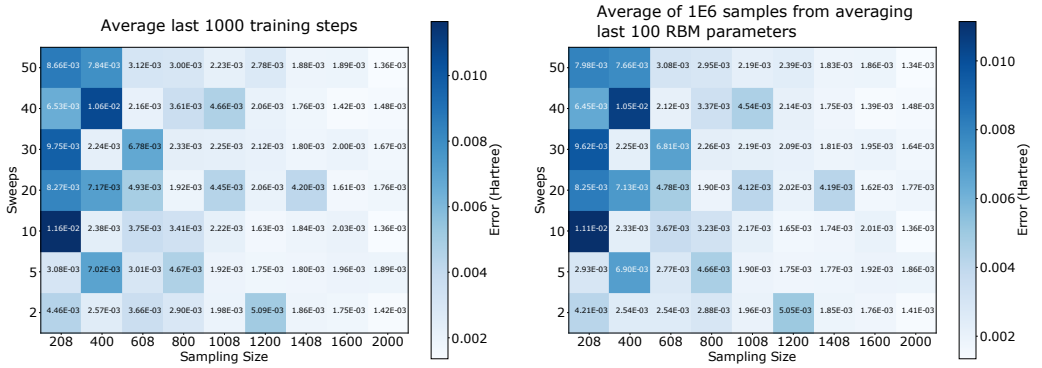


Fig. 3.7: Heatmap of the error in the calculated energy for an H_2 system where the H atoms are separated by 3 Å(i.e. there should be no interaction between the atoms and we know the energy should be -1.0 Hartree). The heatmaps are an average of 5 different RBMs on the same problem. We run the RBMs for 8000 training steps with an initial learning rate of 0.1 and final learning rate of 0.01. We use a basis of 6-311g with 12 qubits and 24 hidden nodes with Hamming weight conservation.

was lowered to 8000.

Compared to Fig. 3.2, the results in Fig. 3.8 are significantly improved by the hyperparameter tuning arrived at in Section 3.4 and running multiple RBMs per data point. This improvement is most evident near full dissociation (i.e. > 2.5 Å) for the basis sets 6-31G, 6-311G, and cc-PVDZ, as the initial curves did not converge to the FCI solution whereas the updated curves do. However, all basis set/hidden node combinations still did not achieve high precision near the minimum, which is significant because this results in inaccurate underestimated dissociation energies. It is also significant to point out that the STO-3G basis still appears to struggle at both the equilibrium and dissociated limits but examination of the FCI results in Fig. 3.1 indicates that the RBM is coming close to the values for that minimal basis set. For all basis sets, however, averaging multiple RBMs per data point resulted in generally smoother curves and smaller standard deviations, despite decreasing the number of training iterations from 20,000 to 8,000. Fig. 3.9, then, is another runtime plot generated for the results in Fig. 3.8. As expected, for all basis sets a linear trend in runtime is observed when increasing the number of hidden units per qubit (visible unit). Compared to Fig. 3.4, the runtimes for a single RBM have significantly decreased as expected, due to both the decreasing in training steps and the constraining of the Hamming weight. We note that preliminary results suggest that increasing the sample size per step may also improve the performance of the RBM ansatz and this is a topic of ongoing research.

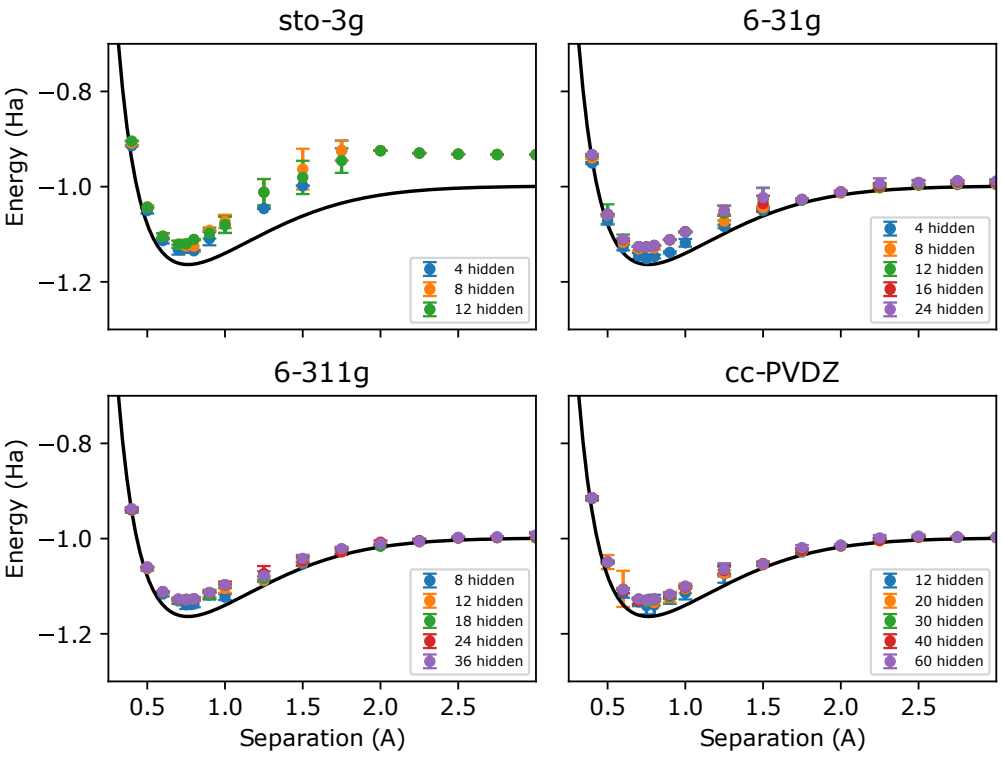


Fig. 3.8: Updated RBM results for a tuned set of parameters for each of the RBMs. Again the FCI calculation for a cc-PVDZ basis are included as a black line.

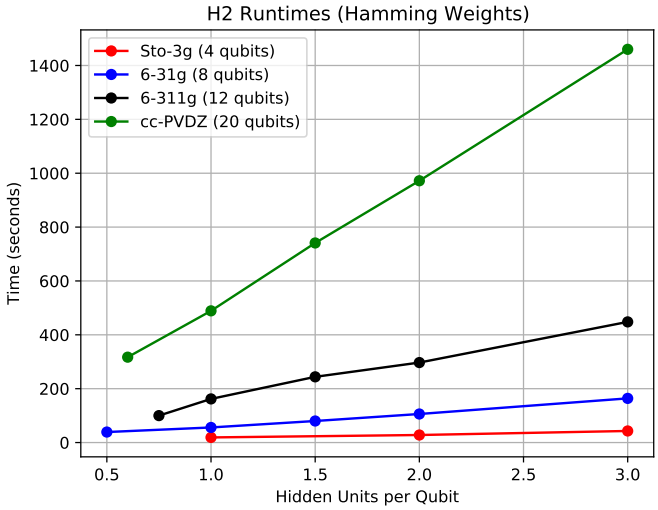


Fig. 3.9: Runtime for performing 10,000 iterations for the H_2 benchmark problem, as a function of the ratio of hidden to visible nodes and the basis set using the updated parameters specified in Sec. 3.5

4. Conclusions. In this work, we study the performance and tuning of an RBM neural network ansatz applied to a simple benchmark problem from quantum chemistry, the dissociation of molecular hydrogen. We find that the accuracy and efficiency of this architecture is highly dependent on the parametrization of the SR algorithm used to find the minimum energy of a given system, particularly the learning rate and sample size. We demonstrate smooth dissociation curves for H_2, showing systematic improvement based on initial results derived from a more naive tuning. We further show that by thoroughly examining the parameter space of optimization, we can improve the accuracy of our results. While we are generally able to find excellent agreement with the total energy when the two hydrogen atoms are highly separated, achieving good agreement with FCI results at the equilibrium bonding distance remains an open problem. While artificial neural networks may prove to be highly accurate tools for electronic structure theory, their performance is highly dependent on tuning the hyperparameters properly. The problem of more systematically and reliably choosing these parameters for larger problem instances will be examined in future work. We will also examine the performance of the RBM variational ansatz relative to ansätze that are amenable to implementation on near-term quantum computers.

Acknowledgements. We gratefully acknowledge Cody Melton and Antonio Russo for valuable feedback. This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories under project 222396 (BibliotheQ). Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for DOE's National Nuclear Security Administration under contract DE-NA0003525. This article describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy of the United States Government.

References.

[1] M. Bukov, M. Schmitt, and M. Dupont, *Learning the ground state of a non-stoquastic quantum Hamiltonian in a rugged neural network landscape*, SciPost Phys., 10 (2021), p. 147.
[2] G. Carleo and M. Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science, 355 (2017), pp. 602–606.
[3] S.-C. T. Choi and M. A. Saunders, *Algorithm 937: Minres-qlp for symmetric and hermitian linear equations and least-squares problems*, ACM Transactions on Mathematical Software (TOMS), 40 (2014), pp. 1–12.
[4] K. Choo, A. Mezzacapo, and G. Carleo, *Fermionic neural-network states for ab-initio electronic structure*, Nature Communications, 11 (2020), p. 2368.
[5] R. Feynman, *Simulating physics with computers*, International Journal of Theoretical Physics, 21 (1982).
[6] J. Hermann, Z. Schätzle, and F. Noé, *Deep-neural-network solution of the electronic schrödinger equation*, Nature Chemistry, 12 (2020), pp. 891–897.
[7] G. Herzberg and A. Monfils, *The dissociation energies of the h2, hd, and d2 molecules*, Journal of Molecular Spectroscopy, 5 (1961), pp. 482–498.
[8] P. Jordan and E. P. Wigner, *Über das paulische äquivalenzverbot*, in The Collected Works of Eugene Paul Wigner, Springer, 1993, pp. 109–129.
[9] J. Kempe, A. Kitaev, and O. Regev, *The complexity of the local hamiltonian problem*, Siam journal on computing, 35 (2006), pp. 1070–1097.
[10] W. Kolos and L. Wolniewicz, *Improved theoretical ground-state energy of the hydrogen molecule*, The Journal of chemical physics, 49 (1968), pp. 404–410.
[11] S. Lloyd, *Universal quantum simulators*, Science, (1996), pp. 1073–1078.
[12] J. R. McClean, N. C. Rubin, K. J. Sung, I. D. Kivlichan, X. Bonet-Monroig, Y. Cao, C. Dai, E. S. Fried, C. Gidney, B. Gimby, et al., *Openfermion: the electronic structure package for quantum computers*, Quantum Science and Technology, 5 (2020), p. 034014.
[13] K. Murphy, *Machine Learning: A Probabilistic Perspective*, Adaptive Computation and Machine Learning series, MIT Press, 2012.
[14] B. O'Gorman, S. Irani, J. Whitfield, and B. Fefferman, *Electronic structure in a fixed basis is qma-complete*, arXiv preprint arXiv:2103.08215, (2021).

[15] D. Pfau, J. S. Spencer, A. G. Matthews, and W. M. C. Foulkes, *Ab initio solution of the many-electron schrödinger equation with deep neural networks*, Physical Review Research, 2 (2020), p. 033429.
[16] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, MIT Press, Cambridge, MA, USA, 1986, pp. 194–281.
[17] S. Sorella, M. Casula, and D. Rocca, *Weak binding between two aromatic rings: Feeling the van der waals attraction by quantum monte carlo methods*, The Journal of chemical physics, 127 (2007), p. 014105.
[18] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, et al., *Pyscf: the python-based simulations of chemistry framework*, Wiley Interdisciplinary Reviews: Computational Molecular Science, 8 (2018), p. e1340.
[19] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, Courier Corporation, 1996.

AIR-SEA LIGHT: COUPLING ATMOSPHERE AND OCEAN MODELS THROUGH THE BULK CONDITION

M. GAIEWSKI[†], K.C. SOCKWELL[‡], J. CONNORS[§], AND P. BOCHEV[¶]

Abstract. The Multilayer Thermal Rotating Shallow Water Equations are often used to model weather and climate [1, 3, 4, 7]. The variables are the velocities, heights, and temperatures (sometimes represented as buoyancy) of vertically "stacked" layers of fluid with varying densities. The layered model can be used to represent the stratified ocean-atmosphere system at climatic scales and their interaction. There is a large discontinuity in the densities at the interface of the atmosphere and ocean, requiring the so called *bulk condition*, a homogenization of the boundary layer, to couple the models in an accurate and stable fashion [2, 9]. In this paper we present a simplified model of the ocean-atmosphere system, based on the DOE E3SM model, which we call *air-sea light*. This model is implemented in MATLAB and is intended to enable rapid testing and prototyping of various coupling methods for the ocean-atmosphere system.

1. Introduction. The Rotating Shallow Water Equations (SWE) are used to model fluids where the horizontal scale is much larger than the vertical scale. This makes the Rotating SWE ideal for modeling the ocean and atmosphere, which possess a large ratio between the horizontal and vertical scales. The equations can be derived from vertically integrating the Naiver-Stokes Equations, and applying the hydrostatic pressure principle as well as a mass balance equation. Several variations of the SWE have been developed to model more complex phenomena in the climate. Some of these include the Multilayer Rotating SWE, which model stratified fluids, and the Single Layer Thermal Rotating SWE which include thermal effects [1, 3]. The various varieties of SWE can be viewed as an approximation to the primitive equations used in global circulation models for the ocean and atmosphere [1, 3, 4, 7]. The existence of the Hamiltonian framework in the SWE makes the derivations, and combinations of variations of the various models a rather straight-forward task. In this report, we will derive a Multilayer Thermal Rotating Shallow Water model for the ocean and atmosphere. Our goal is to develop and verify in MATLAB a simplified ocean-atmosphere model, which we will call the *air-sea light* model, representative of a simplified global circulation model using data from the Southern Ocean Mesoscale Activity, (SOMA) test case [3, 8]. For this task, the coupling at the interface is of central interest and the so called *bulk condition* is utilized in this coupled ocean-atmosphere model [2, 9].

The coupled air-sea light model is being developed for future use in rapidly prototyping newly developed coupling methods under the CANGA project (*Coupling Approaches for Next Generation Architectures*). The model is a stepping stone between coupled advection diffusion and a full climate model like E3SM (Energy Exa-scale Earth System Model), with respect to complexity. The MATLAB implementation makes the rapid prototyping of various coupling methods in the future more viable than using a large code like E3SM, while the underlying model in air-sea light is still representative of the physics in a coupled ocean atmosphere model. Therefore the air-sea light model provides a sweet spot in complexity just right for prototyping methods and proof of concept beyond simple advection-diffusion models. The code base will also be used to investigate couplings between data-based models (Reduced order Models - ROMS, Koopman operator based models, Machine learned models, etc.) with conventional models in a climate and weather prediction context. Once again, the simplicity of the code base lends itself to rapid development and testing before testing on larger code based like E3SM.

[†]University of Connecticut Department of Mathematics, michael.gaiewski@uconn.edu
[‡]Sandia National Laboratories, kcsockw@sandia.gov
[§]University of Connecticut Department of Mathematics, jeffrey.connors@uconn.edu
[¶]Sandia National Laboratories, pbboche@sandia.gov

2. Derivation of Equations.

2.1. Hamiltonian Framework. Three key elements required to define a Hamiltonian system are the following: the variables, the total energy or Hamiltonian (H), and the Poisson tensor (J). Consider the total energy $H(h, \boldsymbol{u}, t)$ of a system as a function of fluid thickness or "height" h, velocity $\boldsymbol{u}$, and time t [1, 3]. The Hamiltonian can be used to derive the equations of motion for variables h and $\boldsymbol{u}$ in the following way. Let $v = [h, \boldsymbol{u}]$ so that $H(h, \boldsymbol{u}, t) = H(v, t)$. Letting $(\cdot, \cdot)_{L^2(\Omega)}$ be the L^2 inner product, the time evolution of the Hamiltonian is given by the chain rule in the sense of Gateaux derivative

$$\frac{dH(v,t)}{dt} = \left(\frac{\partial V}{\partial t}, \frac{\partial H}{\partial v}\right)_{L^2(\Omega)} = 0 \tag{2.1}$$

noting that the evolution is zero by conservation of energy, and that these derivatives are known as the *functional* derivatives from variational calculus. The pair of h and $\boldsymbol{u}$ that satisfy Equation 2.1 must also satisfy the following equation:

$$\frac{\partial v}{\partial t} = J\frac{\partial H}{\partial v} \tag{2.2}$$

for some unknown operator J, where J is a skew-symmetric matrix [1,3]. J must also satisfy the Jacobi identity in order for Equation 2.2 to describe a Hamiltonian system [1, 5]. Of course, Equation 2.2 is the equation of motion, but it can also be interpreted as a condition on the solutions to Equation 2.1. Solutions which satisfy Equation 2.2 naturally lead to energy conservation by skew-symmetry of the operator J,

$$\left(\frac{\partial H}{\partial v}, J\frac{\partial H}{\partial t}\right)_{L^2(\Omega)} = 0. \tag{2.3}$$

We will use this idea of the Hamiltonian framework to derive the Multilayer Thermal Rotating SWE [1,3].

2.2. Multilayer Rotating Shallow Water Equations. The Hamiltonian for the Multilayer Rotating SWE is

$$H(h, \boldsymbol{u}, t) = \sum_{j=1}^{N} \rho_j \int_\Omega \left(h_j \frac{\boldsymbol{u}_j^2}{2} + g h_j \left(b + \sum_{i=j+1}^{N} h_i + \frac{h_j}{2} \right) \right) d\boldsymbol{x} \tag{2.4}$$

where N is the number of layers, $g = 9.81$ m/s^2 is the gravity acceleration, b is the bathymetry which contains the bottom topography of the model, $\boldsymbol{k}$ is the unit normal vector in the Cartesian z direction, f is the Coriolis parameter, and ρ_j is the density at layer j [1,3]. The Multilayer Rotating SWE can be derived through the Hamiltonian framework, as in [5]. The derivation requires the functional derivatives and the Hamiltonian; then the operator J can be inferred. Typically J is inferred using a-priori knowledge of the form of the desired equation. However, this is especially useful when deriving new models where the Hamiltonian changes but the J operator remains the same. The functional derivative of the Hamiltonian with respect to h_j is

$$\frac{\partial H}{\partial h_j} = \rho_j \frac{\boldsymbol{u}_j^2}{2} + g \left(\sum_{i=1}^{j-1} \rho_i h_i + \rho_j \left(b + \sum_{i=j}^{N} h_j \right) \right). \tag{2.5}$$

Let for notation

$$K_j = \frac{\boldsymbol{u}_j^2}{2} \text{ (kinetic energy)} \tag{2.6}$$

$$p_j = \left(\sum_{i=1}^{j-1} \rho_i h_i + \rho_j \left(b + \sum_{i=j}^{N} h_j \right) \right) \text{ (pressure).} \tag{2.7}$$

Then

$$\frac{\partial H}{\partial h_j} = \rho_j K_j + g p_j. \tag{2.8}$$

The functional derivative of the Hamiltonian with respect to $\boldsymbol{u}_j$ is

$$\frac{\partial H}{\partial \boldsymbol{u}_j} = \rho_j h_j \boldsymbol{u}_j. \tag{2.9}$$

The operator J can now be inferred given that the Multilayer Rotating SWE, [1,3,4,7], are

$$\begin{pmatrix} \frac{\partial h_j}{\partial t} \\ \\ \frac{\partial \boldsymbol{u}_j}{\partial t} \end{pmatrix} = \begin{pmatrix} -\operatorname{grad} \cdot h_j \boldsymbol{u}_j \\ \\ -\operatorname{grad}(K_j + \frac{g}{\rho_j} p_j) - q(h_j, \boldsymbol{u}_j) \boldsymbol{k} \times h_j \boldsymbol{u_j} \end{pmatrix} \tag{2.10}$$

where $q(h, \boldsymbol{u}) = (\boldsymbol{k} \cdot \operatorname{grad} \times \boldsymbol{u} + f)/h$ is the potential vorticity. The potential vorticity is not simplified to the total vorticity to achieve the correct form for the Hamiltonian framework. This leads to the operator J_j having the form

$$J_j = \frac{1}{\rho_j} \begin{pmatrix} 0 & -\operatorname{grad} \cdot () \\ \\ -\operatorname{grad}() & -q(h_j, \boldsymbol{u}_j) \boldsymbol{k} \times () \end{pmatrix}. \tag{2.11}$$

It can be verified that each J_j satisfies the Jacobi identity [1, 5]. The J_{ML} operator is the composite operator for the multilayer equations, and is a direct sum of the blocks J_j, therefore the the multilayer equations are also a Hamiltonian system since the direct sum J_{ML} also satisfies the Jacobi identity. This will also be the case for the subsequent multilayer versions of each Hamiltonian models presented in this work. We will use a similar process to derive the Single Layer and Multilayer Thermal Rotating SWE.

2.3. Single Layer Thermal Rotating Shallow Water Equations. To account for thermal effects in the ocean and atmosphere, a third equation representing buoyancy effects can be included in the Hamiltonian framework for the Rotating SWE. Consider the two following variables: $s = g\rho$ and $\sigma = sh = g\rho h$ which are the same variables as buoyancy (s) and mass-weighted buoyancy (S) as seen in [1]; however, both have been multiplied by the density $\bar{\rho}$, which comes from the Boussinesq approximation, where we will let $\bar{\rho}$ be the initial density [7]. Observe that σ is a pressure. From [1], the Hamiltonian for the Single Layer Thermal Rotating SWE *multiplied* by $\bar{\rho}$ so we can use the newly defined variables is

$$H(h, \boldsymbol{u}, t) = \int_\Omega \frac{\sigma h}{2} + \sigma b + \bar{\rho} h \frac{\boldsymbol{u}^2}{2} d\boldsymbol{x}. \tag{2.12}$$

When solving for J, multiplying by a factor of $1/\bar{\rho}$ makes this derivation mathematically equivalent to the derivation in [1], where [1] uses the variables $s = g(\rho/\bar{\rho})$ and $S = gh(\rho/\bar{\rho})$

and does not multiply J by this scalar. This method is now consistent with that of Subsection 2.2 and will be consistent with the derivation of the multilayer case described in Subsection 2.4. From [1] and using the new variables, the Single Layer Thermal Rotating SWE are then

$$\begin{pmatrix} \frac{\partial h}{\partial t} \\ \frac{\partial \boldsymbol{u}}{\partial t} \\ \frac{\partial \sigma}{\partial t} \end{pmatrix} = \begin{pmatrix} -\operatorname{grad}\cdot h\boldsymbol{u} \\ -\operatorname{grad} K - \frac{1}{\bar{\rho}}\operatorname{grad}\left(\frac{\sigma}{2}\right) - \frac{\sigma}{\bar{\rho}h}\operatorname{grad}\left(\frac{h}{2}+b\right) - q(h,\boldsymbol{u})\boldsymbol{k}\times h\boldsymbol{u} \\ -\operatorname{grad}\cdot\sigma\boldsymbol{u} \end{pmatrix} \tag{2.13}$$

for $K = \frac{\boldsymbol{u}^2}{2}$. Using that

$$\frac{\partial V}{\partial t} = J \operatorname{grad}_v H \tag{2.14}$$

and that

$$\frac{\partial H}{\partial h} = \frac{\sigma}{2} + \bar{\rho}\frac{\boldsymbol{u}^2}{2}, \quad \frac{\partial H}{\partial \boldsymbol{u}} = \bar{\rho}h\boldsymbol{u}, \quad \frac{\partial H}{\partial \sigma} = \frac{h}{2} + b \tag{2.15}$$

then we must find a skew-symmetric operator J such that

$$\begin{pmatrix} \frac{\partial h}{\partial t} \\ \frac{\partial \boldsymbol{u}}{\partial t} \\ \frac{\partial \sigma}{\partial t} \end{pmatrix} = J \begin{pmatrix} \frac{\partial H}{\partial h} \\ \frac{\partial H}{\partial \boldsymbol{u}} \\ \frac{\partial H}{\partial \sigma} \end{pmatrix} = J \begin{pmatrix} \frac{\sigma}{2} + \frac{\boldsymbol{u}^2}{2} \\ h\boldsymbol{u} \\ \frac{h}{2} + b \end{pmatrix} = \begin{pmatrix} -\operatorname{grad}\cdot h\boldsymbol{u} \\ -\operatorname{grad} K - \frac{1}{\bar{\rho}}\operatorname{grad}\left(\frac{\sigma}{2}\right) - \frac{\sigma}{\bar{\rho}h}\operatorname{grad}\left(\frac{h}{2}+b\right) - q(h,\boldsymbol{u})\boldsymbol{k}\times h\boldsymbol{u} \\ -\operatorname{grad}\cdot\sigma\boldsymbol{u} \end{pmatrix}. \tag{2.16}$$

Let

$$J = \frac{1}{\bar{\rho}} \begin{pmatrix} 0 & -\operatorname{grad}\cdot() & 0 \\ -\operatorname{grad}() & -q(h,\boldsymbol{u})\boldsymbol{k}\times() & -s\operatorname{grad}() \\ 0 & -\operatorname{grad}\cdot(s\ \ ) & 0 \end{pmatrix}, \tag{2.17}$$

which can be verified to be skew-symmetric with respect to the L^2 inner product and that J satisfies the Jacobi identity [1, 5]. We will use a similar J to construct the Multilayer Thermal Rotating SWE.

2.4. Multilayer Thermal Rotating Shallow Water Equations. To derive the Multilayer Thermal Rotating SWE, we extend the Single Layer Thermal Rotating Shallow Water Hamiltonian to having multiple layers as follows:

$$H(h,\boldsymbol{u},t) = \sum_{j=1}^{N} \int_\Omega \left(\bar{\rho}_j h_j \frac{\boldsymbol{u}_j^2}{2} + \sigma_j \left(b + \sum_{i=j+1}^{N} h_i + \frac{h_j}{2} \right) \right) d\boldsymbol{x}. \tag{2.18}$$

However, instead of dividing by a constant density $\bar{\rho}$, we will divide each layer by $\bar{\rho}_j$, its corresponding *initial* density when we formulate the J_j matrix for each layer. Since each σ_j and therefore ρ_j is changing, we use the Boussinesq approximation on *each* layer. Observe

that we did not need to use the Boussinesq approximation in Subsection 2.2 as none of the densities were changing, so $\bar{\rho_j}$ always would equal ρ_j. Observe that

$$\frac{\partial H}{\partial h_j} = \bar{\rho_j}\frac{\boldsymbol{u}_j^2}{2} + \frac{\sigma_j}{2} + \sum_{i=1}^{j-1}\sigma_i, \quad \frac{\partial H}{\partial \boldsymbol{u}_j} = \bar{\rho_j} h_j \boldsymbol{u}_j, \quad \frac{\partial H}{\partial \sigma_j} = b + \sum_{i=j+1}^{N} h_i + \frac{h_j}{2}. \tag{2.19}$$

Using a J_j similar to the J from Equation 2.17, let

$$J_j = \frac{1}{\bar{\rho_j}}\begin{pmatrix} 0 & -\operatorname{grad}\cdot() & 0 \\ -\operatorname{grad}() & -q(h_j, \boldsymbol{u_j})\boldsymbol{k}\times() & -s_j \operatorname{grad}() \\ 0 & -\operatorname{grad}\cdot(s_j \quad) & 0 \end{pmatrix}. \tag{2.20}$$

It can be verified that each J_j satisfies the Jacobi identity [1, 5]. We can define the J_{ML} operator similar to how way we did in Subsection 2.2, and will satisfy the Jacobi identity for the same reason. Then we can derive the Multilayer Thermal Rotating SWE as follows:

$$\begin{pmatrix} \frac{\partial h_j}{\partial t} \\ \frac{\partial \boldsymbol{u}_j}{\partial t} \\ \frac{\partial \sigma_j}{\partial t} \end{pmatrix} = J_j \begin{pmatrix} \frac{\partial H}{\partial h_j} \\ \frac{\partial H}{\partial \boldsymbol{u}_j} \\ \frac{\partial H}{\partial \sigma_j} \end{pmatrix} = \begin{pmatrix} 0 & -\operatorname{grad}\cdot() & 0 \\ -\operatorname{grad}() & -q(h_j, \boldsymbol{u}_j)\boldsymbol{k}\times() & -s_j \operatorname{grad}() \\ 0 & -\operatorname{grad}\cdot(s_j \quad) & 0 \end{pmatrix} \begin{pmatrix} K_j + \frac{\sigma_j}{2\bar{\rho_j}} + \frac{1}{\bar{\rho_j}}\sum_{i=1}^{j-1}\sigma_j \\ h_j \boldsymbol{u}_j \\ \frac{1}{\bar{\rho_j}}\left(b + \sum_{i=j+1}^{N} h_j + \frac{h_j}{2}\right) \end{pmatrix}$$

$$= \begin{pmatrix} -\operatorname{grad}\cdot h_j \boldsymbol{u}_j \\ -q(h_j, \boldsymbol{u}_j)\boldsymbol{k}\times h_j \boldsymbol{u_j} - \operatorname{grad} K_j - \frac{1}{\bar{\rho_j}}\operatorname{grad}\left(\frac{\sigma_j}{2}\right) - \frac{1}{\bar{\rho_j}}\operatorname{grad}\left(\sum_{i=1}^{j-1}\sigma_j\right) - \frac{\sigma_j}{h_j\bar{\rho_j}}\operatorname{grad}\left(b + \sum_{i=j+1}^{N} h_i + \frac{h_j}{2}\right) \\ -\operatorname{grad}\cdot\sigma_j \boldsymbol{u}_j \end{pmatrix} \tag{2.21}$$

and we are done.

2.5. Temperature Equations. The temperature of each layer T_j, given in units of Kelvin, is a key attribute and informative quantity to output from the model. For the air model, we refer to the Ideal Gas Law $p_j = \rho_j R T_j$, where $p_j = \sum_{i=1}^{j}\sigma_i$ and $R = 287$ J/(kg K) is the air gas constant [7]. Then we solve for T_j as follows:

$$T_j = \frac{p_j}{R\rho_j} = \frac{\sum_{i=1}^{j}\sigma_i}{R\rho_j} = \frac{g h_j \sum_{i=1}^{j}\sigma_i}{R\sigma_j}. \tag{2.22}$$

For the ocean temperature, we use the linear equation of state found in [6]:

$$\rho_j = \rho_0 - \alpha(T_j - T_0) + \beta(S_j - S_0). \tag{2.23}$$

We let $\beta = 0$ as we are not considering the salinity terms S_j and S_0 in this model. Using some of the default values in [6], we let $\alpha = 0.255$ kg/((m^3)(°C)) and $T_0 = 19°$ C. Lastly, we

let $\rho_0 = 1026.5$ kg/m^3 for the scalar density value used in the Boussinesq approximation [7]. This value is close to the average of the three ocean densities that will be used in the model, all of which are similar in value motivating the use of the Boussinesq approximation [7]. Solving for T_j in units of Kelvin, gives

$$T_j = \frac{\rho_0}{\alpha} + T_0 - \frac{\rho_j}{\alpha} + 273.1 = \frac{\rho_0}{\alpha} + T_0 - \frac{\sigma_j}{\alpha g h_j} + 273.1. \tag{2.24}$$

2.6. Modifying Thermal Variable. The bulk condition given in Equations 4.15-4.18, is given defined in terms of temperatures and velocities through the vertical diffusion terms, acting like forcing terms in the interface layers. There are several ways the bulk condition can be incorporated into the model in Equations 2.21: algebraically manipulating the bulk condition into buoyancy-like variables, such as s or σ, or adopting a model that is more like the primitive equations in structure and possesses a temperature tracer equation instead of a buoyancy equation. Using a temperature based tracer equation instead of a buoyancy based tracer equation, leads to a more natural enforcement of the bulk conditions which are defined in terms of temperature. For this reason, we test the coupling by replacing the buoyancy equation with a temperature tracer equation. Although this breaks the Hamiltonian structure, we mainly depended on the Hamiltonian structure for derivation purpose. Additionally, external terms such as forcing, smoothing, and vertical mixing break the Hamiltonian framework, so the Hamiltonian framework has served its purpose at this point. The Multilayer Rotating Thermal SWE with temperature tracer equations are given by

$$\begin{pmatrix} \frac{\partial h_j}{\partial t} \\ \frac{\partial \boldsymbol{u}_j}{\partial t} \\ \frac{\partial (T_j h_j)}{\partial t} \end{pmatrix} = \begin{pmatrix} -\operatorname{grad} \cdot h_j \boldsymbol{u}_j \\ -q(h_j, \boldsymbol{u}_j)\boldsymbol{k} \times h_j \boldsymbol{u_j} - \operatorname{grad} K_j - \frac{1}{\rho_j} \operatorname{grad}\left(\frac{\sigma_j}{2}\right) - \frac{1}{\rho_j} \operatorname{grad}\left(\sum_{i=1}^{j-1} \sigma_j\right) - g \operatorname{grad}\left(b + \sum_{i=j+1}^{N} h_i + \frac{h_j}{2}\right) \\ -\operatorname{grad} \cdot T_j h_j \boldsymbol{u}_j \end{pmatrix}, \tag{2.25}$$

and the system must be closed by an appropriate equation of state such as Equations 2.22 and 2.24. Now $T_j h_j$ are prognostic variables and σ_j are diagnostic variables for $j = 1, 2, 3$.

3. Implementation.

3.1. Description of Model and Input Parameters. Our test models use the domain and data from the SOMA test case as described in [3, 8] with a flat bathymetry in the ocean. The horizontal domain is a circle on the sphere with diameter 1.25×10^6 m, and the depth of the is given by the flat bathymetry b and the height of atmosphere model is given by the sum of the fluid thickness in each atmosphere layer. The ocean and atmosphere domains are partitioned in a sense that the bottom topography is flat, and not the sea surface height. This was done to avoid coupling the pressure terms in the ocean and atmosphere is this arrangement is typical in coupled climate models. We discretize each model in space using the so called *TRiSK scheme* as described in [4]. This is a finite volume scheme that is formally second order but depends on the mesh quality. It uses centroidal voronoi tessellations as the mesh on the sphere. The cells are on average 32 km in diameter on the quasi-uniform mesh. The TRiSK scheme is a staggered scheme and is mimetic so it maintains a discrete Hamiltonian framework leading to favorable properties. The TRiSK scheme also reduces the number of velocity variables by only requiring the velocities which are normal with respect to each grid cell. This is opposed to using both the normal and tangential velocity components and is achieve through the Flux reconstruction operator defined

Parameter	Value (Air)	Value (Ocean)	Unit	Description
b	0	-2500	m	Bottom topography (flat)
$\boldsymbol{u_j}$	0	0	m/s	Initial velocities for $j = 1, 2, 3$
ρ_j	[0.6599,0.9803,1.225]	[1025;1027;1028]	kg/m^3	Initial densities
T_j	[283.1,293.1,303.1]	From ocean temp eq	K	Initial temperatures
h_j	From T_j and σ_j	[250,450,1800]	m	Initial heights
σ_j	$g\rho_j h_j$	$g\rho_j h_j$	kg/(ms^2)	For $j = 1, 2, 3$

Table 3.1: 3-layer Air and 3-layer Ocean Models

in [3], which reconstructs the tangential flux and velocities. The grid used for the numerical results for each layer of either ocean or atmosphere has 8521 cells, 25898 edges, and 17378 vertices. The boundary conditions in the atmosphere are currently no penetration for the velocity, which creates an artificial boundary but is sufficient for these first tests. The height h_j and thickness weighted temperature $T_j h_j$ are defined on the cells, the normal component of the velocities with respect the cell edges $\boldsymbol{u}_j$ are defined on the edges, and the potential vorticities $q(h_j, \boldsymbol{u}_j)$ are defined on the vertices.

For test cases used in this paper, the model given by Equations 2.25 is implemented by discretizing spatially as in [4] and by using the time stepping method Runge-Kutta 4 (RK4). We used the numerical data parameters from the SOMA test case [3, 8] in our models. We consider a 3-layer air model and a 3-layer ocean model. For both models, using that $\sigma_j = \rho_j g h_j$, and that the densities are always chosen initially by the user, the user can either choose heights initially and the algorithm will find the corresponding σ_j variables and temperatures, or the user can choose the σ_j variables or temperatures for and the algorithm will find the corresponding heights. We add a drag term in the ocean

$$F_d = \frac{c_{\text{drag}}}{h_j}|\boldsymbol{u_j}|\boldsymbol{u_j}, \quad c_{\text{drag}} = 10^{-3} \tag{3.1}$$

as well whenever we have ocean layers as seen in [3]. This does not get applied to air layers. These additional components are defined in Equations 3.2 and Table 4.1 and applied to 2.25 as seen in Equations 4.19 along with the terms from the bulk condition later. We also add the horizontal and vertical smoothing to the velocity and tracer equations, $D_h\boldsymbol{u_j}$ and $D_v\boldsymbol{u_j}$, $D_h T_j h_j$ and $D_v T_j h_j$ [3], defined as

$$D_h\boldsymbol{u_j} = -\nu_{\boldsymbol{u}}\Delta_h^2\boldsymbol{u_j}\ , \tag{3.2}$$
$$D_v\boldsymbol{u_j} = K_j^m\Delta_v\boldsymbol{u_j} \tag{3.3}$$
$$D_h T_j h_j = \nu_T\Delta_h T_j h_j\ , \tag{3.4}$$
$$D_v T_j h_j = K_j^t\Delta_v T_j h_j\ , \tag{3.5}$$

for $j = \{a, o\}$, where the various viscosity and mixing values are obtained from [8]. The Δ_h^2 term is the thickness-weight h vector-biharmonic operator in the horizontal direction used in [6], Δ_h is the thickness-weighted scalar Laplacian operators in the horizontal direction from [6], and Δ_v is the vertical Laplacian used for vertical mixing in [6].

A description of these two models can be found in Table 3.1. We chose densities similar to well-known densities across literature. For all of the test cases in this paper, assume we are always starting at this base model and making any modifications thereafter.

3.2. Verification/Validation of Implementation. We want to verify that the rate of convergence for our RK4 implementation is correct, which we expect to be approaching

Parameter	Units	Description
N	N/A	Number of Layers (3)
i	N/A	Index of cells
e	N/A	Index of edges
j	N/A	Index of layers
h	m	Heights on cells
$\boldsymbol{u}$	m/s	Velocities on edges
σ	kg/(ms^2)	$\rho g h$
A_i	m^2	Areas of cells
A_e	m^2	Areas of edges
Tr	N/A	Corresponds to "true" solution

Table 3.2: Description of Error Calculation

k	R_k
1	3.9070
2	3.9995
3	4.0014
4	3.9990

Table 3.3: Convergence Results

4 as the time step size approaches 0. Verifying this correct rate on one test case should be sufficient to show this. The initial conditions were chosen for the 3-layer model such that bottom layer had a rotating Gaussian which was in geostrophic balance [3]. The initial condition for the fluid thickness was chosen to be a Gaussian (G), centered at the midpoint of the domain, a maximum height of 2 meters and standard deviation 8×10^{10}. The velocity on the bottom layer was chosen so that the test case was in approximate geostrophic balance:

$$v_3 = -\frac{g}{w} \operatorname{grad} G \,, \tag{3.6}$$

where $w = 0.00008$ [3]. The simulations were run for 600000 seconds, approximately a week, and the time step Δt_m for $\Delta t_1 = 192$, $\Delta t_2 = 96$, $\Delta t_3 = 48$, $\Delta t_4 = 24$, and $\Delta t_5 = 12$ all in seconds. In order to verify our code, we looked at the variables h_j, v_j, and σ_j for $j = 1, 2, 3$ at the final time of 600000 seconds. We used a time step of 1 second as an over-refined solution, acting in place of a true solution, which is difficult to define on this particular domain on the sphere. For the other five runs, we calculate the error E_m and rate of convergence R_k as follows:

$$E_m(U_m) = \sum_{j=1}^{N} \left(\sum_i (h_{i,j,m} - h_{i,j,Tr})^2 A_i + \sum_e (\boldsymbol{u}_{e,j,m} - \boldsymbol{u}_{e,j,Tr})^2 A_e + \sum_i (\sigma_{i,j,m} - \sigma_{i,j,Tr})^2 A_i \right)^{\frac{1}{2}} , \tag{3.7}$$

$$R_k = \log(E_k(\Delta t_k)/E_{k+1}(\Delta t_{k+1}))/\log(\Delta t_k/\Delta t_{k+1}) \quad : \quad \Delta t_k > \Delta t_{k+1} \tag{3.8}$$

where the remaining parameters are defined in Table 3.2. Observe $\Delta t_k/\Delta t_{k+1} = 2$ for all $k \in \{1, 2, 3, 4\}$. Since RK4 is a fourth order method, we expect that $R_k \to 4$ as k moves from 1 to 4. The results from this are in Table 3.3. We can see that the convergence asymptotically approaches fourth order in Δt_m, showing we have indeed implemented RK4 correctly.

4. Coupling Ocean and Air at Interface. We now present the numerical results from the coupled ocean-atmosphere model.

4.1. Bulk Condition. The so called *bulk condition* is a set of Robin boundary conditions [2,9] that describe the bulk effect of the shear stress of the 2-layer interface dragging on one another. It is important to note that the current configuration of the ocean-atmosphere model does not couple the pressures between the two models, which would lead to a very

Parameter(s)	Value	Units	Description
z	N/A	m	Positive Cartesian "upwards" direction
$\hat{z}$	1	m/s	Unit vector in the z direction
Γ	N/A	N/A	Air-sea interface
$\hat{n}_a$, $\hat{n}_o$	1, -1	m/s	Unit normal vectors w.r.t. Γ such that $\hat{n}_a = -\hat{n}_o$
τ	N/A	kg/(m^{2}s)	Surface wind stress
$\mathcal{T}$	N/A	s	Final time
Q	N/A	J/m^2	Heat flux
ρ_a, ρ_o	N/A	kg/m^3	Densities
$\boldsymbol{u}_a$, $\boldsymbol{u}_o$	N/A	m/s	Horizontal velocities
T_a, T_o	N/A	K	Temperatures
h_a, h_o	N/A	m	Layer heights
K_a^m, K_o^m	10^{-5}, 10^{-5}	m^2/s	Eddy viscosities
K_a^t, K_o^t	10^{-4}, 10^{-4}	m^2/s	Eddy diffusivities
$\nu_{\boldsymbol{u}}$	5×10^{13}	m^4/s	Scaling constant
ν_T	10^5	m^2/s	Scaling constant
c_a^p, c_o^p	1000, 4190	J/(kg K)	Specific heats
C_D, C_H	10^{-3}, 10^{-3}	N/A	Friction parameters
$\|\|\Delta U\|\|$	$((\boldsymbol{u}_o - \boldsymbol{u}_a)^2)^{1/2}$	m/s	Exponents are entry-wise operators

Table 4.1: Description of Bulk Condition

restrictive CFL condition from the accompanying gravity wave at the interface. This is typically done in partitioned ocean-atmosphere models.

$$\rho_a K_a^m \partial_z \boldsymbol{u}_a (\hat{z} \cdot \hat{n}_a) = \rho_o K_o^m \partial_z \boldsymbol{u}_o (\hat{z} \cdot \hat{n}_o) = \boldsymbol{\tau} \quad \text{on } \Gamma \times [0, \mathcal{T}] \tag{4.1}$$

$$\rho_a c_a^p K_a^t \partial_z (T_a h_a)(\hat{z} \cdot \hat{n}_a) = \rho_o c_o^p K_o^t \partial_z (T_o h_o)(\hat{z} \cdot \hat{n}_o) = Q \quad \text{on } \Gamma \times [0, \mathcal{T}] \tag{4.2}$$

$$\boldsymbol{\tau} = \rho_a C_D ||\Delta U|| (\boldsymbol{u}_a - \boldsymbol{u}_o) \tag{4.3}$$

$$Q = \rho_a c_a^p C_H ||\Delta U|| (T_a h_a - T_o h_o) \tag{4.4}$$

where the subscripts a and o refer to the air and ocean, respectively, and the other parameters are described in Table 4.1:

4.2. How to Insert into Model. The bulk condition gives the four following conditions:

$$\text{Wind forcing for air: } K_a^m \partial_z \boldsymbol{u}_a (\hat{z} \cdot \hat{n}_a) = \boldsymbol{\tau}/\rho_a \tag{4.5}$$

$$\text{Wind forcing for ocean: } K_o^m \partial_z \boldsymbol{u}_o (\hat{z} \cdot \hat{n}_o) = \boldsymbol{\tau}/\rho_o \tag{4.6}$$

$$\text{Thermal forcing for air: } K_a^t \partial_z (T_a h_a)(\hat{z} \cdot \hat{n}_a) = Q/(\rho_a c_a^p) \tag{4.7}$$

$$\text{Thermal forcing for ocean: } K_o^t \partial_z (T_o h_o)(\hat{z} \cdot \hat{n}_o) = Q/(\rho_o c_o^p) \tag{4.8}$$

Now we choose a normal convention for interface by letting $n_\Gamma = \hat{n}_a = -\hat{n}_o$. This just puts negative signs on one side so that the flux is conserved.

$$K_a^m \partial_z \boldsymbol{u}_a = -\boldsymbol{\tau}/\rho_a \tag{4.9}$$

$$K_o^m \partial_z \boldsymbol{u}_o = \boldsymbol{\tau}/\rho_o \tag{4.10}$$

$$K_a^t \partial_z (T_a h_a) = -Q/(\rho_a c_a^p) \tag{4.11}$$

$$K_o^t \partial_z (T_o h_o) = Q/(\rho_o c_o^p) \tag{4.12}$$

The bulk condition, a term representing the vertical stress at the interface is introduced as a boundary condition of the vertical mixing operator D_v for each of the respective equation.

To demonstrate this, we want to consider taking the discrete Laplacian over N layers of the same type of fluid. This means we have N variables, one for each layer, and $N+1$ interfaces, where the top and bottom maybe contain boundary conditions. Observe that for a given function F, (which will be $\boldsymbol{u}_j$ or $T_j h_j$ for $j = a, o$) then $K_j^m \Delta_v F = K_j^m \partial_z \partial_z F$ are the vertical mixing terms in F. Now consider F_i and h_i for $i = 1, 2, ..., N$ are the functions and heights at the N layers, and consider the discrete partial derivatives

$$(\partial_z F)_{i+1/2} = \frac{F_i - F_{i+1}}{\frac{1}{2}(h_i + h_{i+1})} \tag{4.13}$$

where $i = 1/2, 3/2, ..., N + 1/2$ are the layer interfaces. Now the discrete Laplacians are

$$(\partial_z \partial_z F)_i = \frac{\partial_z F_{i-1/2} - \partial_z F_{i+1/2}}{h_i}. \tag{4.14}$$

For the two values that do not exist: $\partial_z F_{1/2}$ and $\partial_z F_{N+1/2}$, we replace them with the bulk condition. Note that at the bottom of the ocean the boundary condition is represented by the drag term. Thus, for the coupled model, the bottom layer of air or the top layer of ocean have boundary conditions on the vertical mixing (vertical stress) that are represented by the bulk condition terms, which act like forcing terms in these layers. These four terms now have the correct units and are simplified as follows:

$$K_a^m \frac{\partial_z \boldsymbol{u}_a}{h_a} = -\frac{C_D}{h_a} ||\Delta U|| (\boldsymbol{u}_a - \boldsymbol{u}_o) \,, \tag{4.15}$$

$$K_o^m \frac{\partial_z \boldsymbol{u}_o}{h_o} = \frac{C_D \rho_a}{h_o \rho_o} ||\Delta U|| (\boldsymbol{u}_a - \boldsymbol{u}_o) \,, \tag{4.16}$$

$$K_a^t \frac{\partial_z (T_a h_a)}{h_a} = -\frac{C_H}{h_a} ||\Delta U|| (T_a h_a - T_o h_o) \,, \tag{4.17}$$

$$K_o^t \frac{\partial_z (T_o h_o)}{h_o} = \frac{\rho_a c_a^p C_H}{\rho_o h_o c_o^p} ||\Delta U|| (T_a h_a - T_o h_o) \,, \tag{4.18}$$

where the introduction of the factors $1/h_a$ and $1/h_o$ come from the vertical viscosity. We now add all of the forcing terms and bulk conditions into Equations 2.25.

$$\begin{aligned}
\frac{\partial h_j}{\partial t} &= -\operatorname{grad} \cdot h_j \boldsymbol{u}_j \\
\frac{\partial \boldsymbol{u}_j}{\partial t} &= -q(h_j, \boldsymbol{u}_j) \boldsymbol{k} \times h_j \boldsymbol{u_j} - \operatorname{grad} K_j - \frac{1}{\rho_j} \operatorname{grad} \left(\frac{\sigma_j}{2} \right) - \frac{1}{\rho_j} \operatorname{grad} \left(\sum_{i=1}^{j-1} \sigma_j \right) \\
&\quad - g \operatorname{grad} \left(b + \sum_{i=j+1}^{N} h_i + \frac{h_j}{2} \right) D_h \boldsymbol{u_j} + D_v \boldsymbol{u_j} - F_d + \delta_{a,j} \frac{\partial_z \boldsymbol{u}_a}{h_a} \bigg|_{\text{top}} + \delta_{o,j} \frac{\partial_z \boldsymbol{u}_o}{h_o} \bigg|_{\text{bottom}} \\
\frac{\partial (T_j h_j)}{\partial t} &= -\operatorname{grad} \cdot T_j h_j \boldsymbol{u}_j + D_h T_j h_j + D_v T_j h_j + \delta_{a,j} \frac{\partial_z (T_a h_a)}{h_a} \bigg|_{\text{top}} + \delta_{o,j} \frac{\partial_z (T_o h_o)}{h_o} \bigg|_{\text{bottom}}
\end{aligned} \tag{4.19}$$

where $\delta_{i,j}$ is the Kronecker delta.

4.3. Tests and Results. We have created an air-sea light test case that takes a single layer of air and a single layer of ocean and couples them together at the interface. For proof of concept, we would like to provide a thermal forcing in the ocean, which then spins up

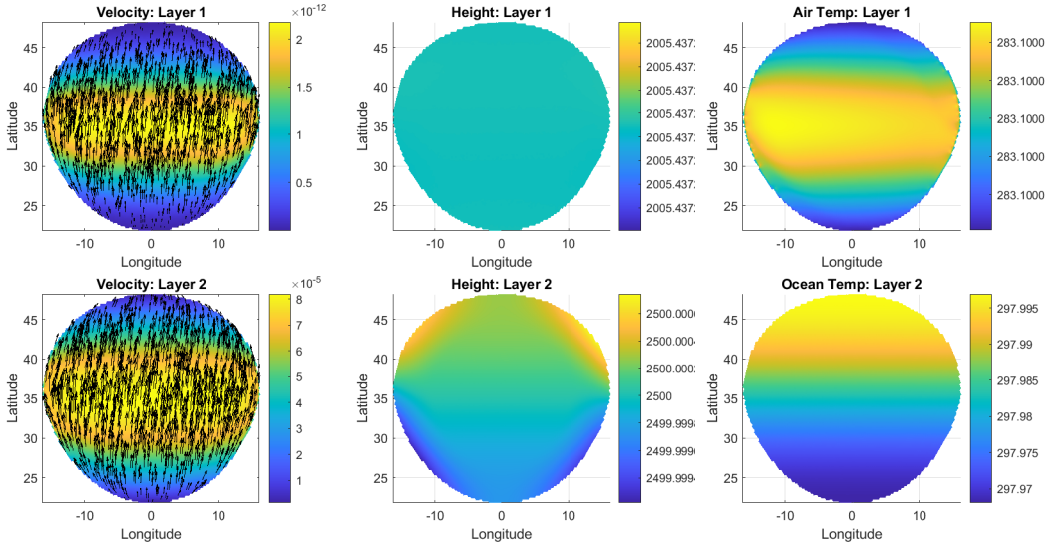


Fig. 4.1: Test Case 1: 1 Hour

the velocity in the atmosphere and ocean models through the coupling terms. We choose a temperature forcing that gives a larger temperature in the top of the domain so a double-gyre velocity can be achieved. The difference in temperatures creates a higher pressure in the top of the domain, and then through geostrophic balance, gives rise to a double-gyre.

To create this test-case, we took the parameters from the bottom layer of the 3-layer air model and the top layer of the 3-layer ocean model as described in Table 3.1 and coupled them using the bulk condition. We, however, made the height of the ocean layer 2500 meters instead of 250 meters. We used a time step of 180 seconds on the air, the ocean, and the coupling explicitly using the previous time step values t_n which are inserted in the coupling terms in the RK4 method, essentially lagging the bulk terms. The first test was to apply solar forcing to the ocean layer as follows:

$$S_1 = C_1(\sin(\pi dy) + 1) - C_2|T_2h_2|T_2h_2 \tag{4.20}$$

where these parameters are defined in Table 4.2. The second term acts an artificial heat dissipation terms to keep the temperature at a reasonable value. This solar forcing should create a double gyre in the ocean, as in the SOMA test case, and through coupling, a double gyre should be created in the air as well [3,8]. Furthermore, we expect to see the air and the ocean to follow similar patterns to each other over time in velocity, height, and temperature. We see in Figure 4.1 what the velocities, heights, and temperatures looked like after 1 hour to show the early effect of the solar forcing on each layer. We see that since the model is coupled, the solar forcing in the ocean indeed has an effect on the atmosphere. Figure 4.2 shows the solution after 3 days to show the air and ocean each having a double gyre formed in their velocities. This indeed shows that the coupling is working as the ocean successfully created a double gyre in an atmosphere that started at rest. This test case serves as a proof-of-concept to show off the effects of coupling; however, it still needs some fine tuning in order to achieve more realistic velocities and temperatures in the long run. We present a second test case to show a coupled model in the long run. We now have applied a thermal restoring

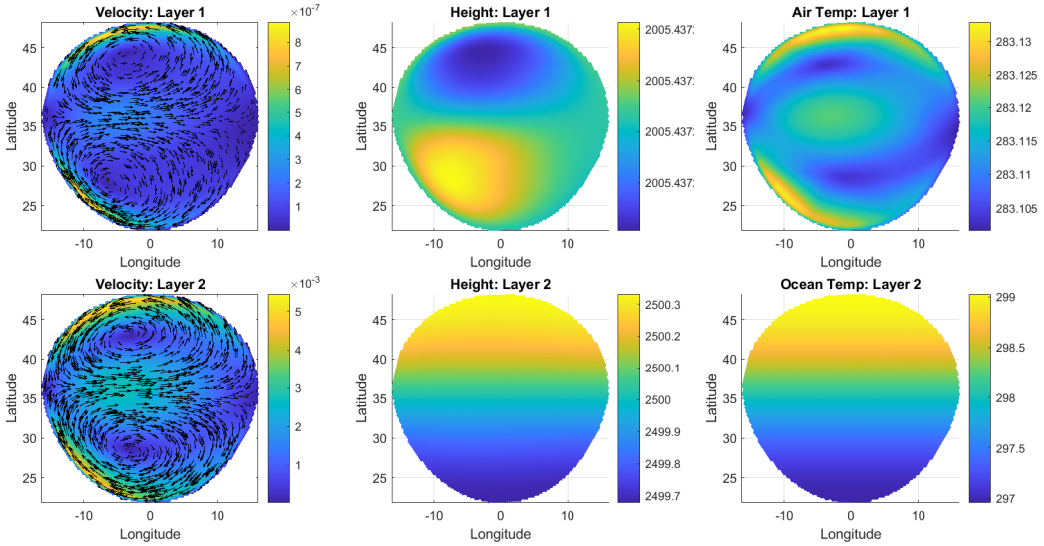


Fig. 4.2: Test Case 1: 3 Days

Parameter(s)	Value	Units	Description
C_1	$\frac{1}{100}$	N/A	Scaling constant
C_2	$\frac{1}{75^2} \times 10^{-10}$	N/A	Scaling constant
r	6371220	m	Radius of Earth
l	N/A	rad	Latitude coordinates at cell centers
dy	$Mr(l - 70\pi/360)$	rad	For all cells
dY_0	1.4684×10^6	rad	Maximum dy across all cells
dy_0	-1.4637×10^6	rad	Minimum dy across all cells
M	$1/(dY_0 - dy_0)$	m^{-1}	Scaling constant

Table 4.2: Description of Additional Forcing

force in the ocean and atmosphere. The thermal restoring forces for the atmosphere is

$$S_2 = -10^{-3}(T_1 - 283)\,, \tag{4.21}$$

and for the ocean is

$$S_3 = -10^{-3}(T_2 - (295 + 10\sin(\pi dy)))\,. \tag{4.22}$$

This choice of forcing maintains a desired temperature profile in each domain for long periods of time, avoiding the fine tuning required for the previous forcing. Once again the ocean has a sinusoidal based forcing to induce a double-gyre while the atmosphere has constant forcing to keep the temperature from getting too large or too small. These parameters are also explained more in Table 4.2. Figure 4.3 shows what this new model looks like after 1 year. We see in particular that the velocities between the ocean and atmosphere look similar to each other. Moreover, the model remains stable for the entire year and the restoring forces keep the temperatures from becoming unrealistic.

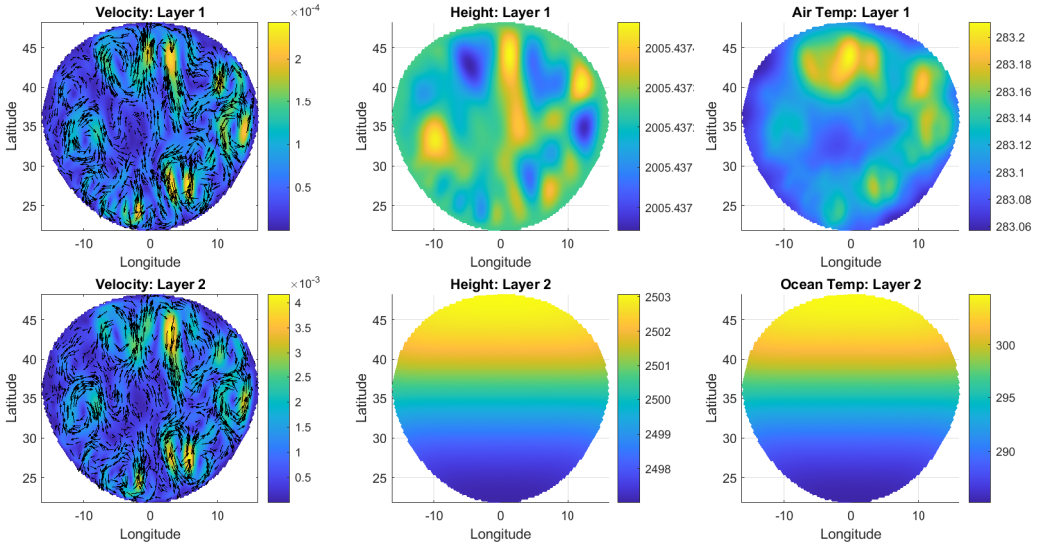


Fig. 4.3: Test Case 2: 1 Year

5. Conclusions. In this proceeding, we derived the Multilayer Thermal Rotating SWE using the Hamiltonian framework. We have implemented these equations in MATLAB using the RK4 time-stepping method [1, 3, 4, 7]. Using the data from the SOMA test case, we constructed a 3-layer air, and a 3-layer ocean model, and verified using the ocean model the correct RK4 convergence rate to show this implementation works correctly [3, 8]. We then applied the bulk condition to the air-sea light model and coupled a single layer of air and a single layer of ocean together at the interface [2, 9]. Future work would entail coupling together a multilayer model, such as 3 layers of air and 3 layers of ocean.

References.

[1] C. Eldred, T. Dubos, and E. Kritsikis, *A quasi-hamiltonian discretization of the thermal shallow water equations*, Journal of Computational Physics, 379 (2019), pp. 1–31.
[2] F. Lemarié, E. Blayo, and L. Debreu, *Analysis of ocean-atmosphere coupling algorithms: consistency and stability*, Procedia Computer Science, 51 (2015), pp. 2066–2075.
[3] K. Pieper, K. C. Sockwell, and M. Gunzburger, *Exponential time differencing for mimetic multilayer ocean models*, Los Alamos National Laboratory, (2019).
[4] T. D. Ringler, J. Thuburn, J. Klemp, and W. Skamarock, *A unified approach to energy conservation and potential vorticity dynamics for arbitrarily structured c-grids*, Journal of Computational Physics, 229 (2010), pp. 3065–3090.
[5] A. L. Stewart and P. J. Dellar, *Multilayer shallow water equations with complete coriolis force. part 1. derivation on a non-traditional beta-plane*, Journal of Fluid Mechanics, 651 (2010), pp. 387–413.
[6] L. A. N. L. C. O. S.-I. M. Team, *MPAS-Ocean Model User's Guide: Version: 2.0*, 2013.
[7] G. K. Vallis, *Atmospheric and oceanic fluid dynamics*, Cambridge University Press, 2017.
[8] P. J. Wolfram, T. D. Ringler, M. E. Maltrud, D. W. Jacobsen, and M. R. Petersen, *Diagnosing isopycnal diffusivity in an eddying, idealized midlatitude ocean basin via lagrangian, in situ, global, high-performance particle tracking (light)*, Journal of Physical Oceanography, 45 (2015), pp. 2114–2133.
[9] H. Zhang, Z. Liu, E. Constantinescu, and R. Jacob, *Stability analysis of interface conditions for ocean–atmosphere coupling*, Journal of Scientific Computing, 84 (2020), pp. 1–25.

PRECIPITATION MODEL AGGREGATION USING OPTIMAL TRANSPORT

KATHERINE E. GEROT* AND KELSEY L. DIPIETRO†

Abstract. In the management of infrastructure, the need for accurate precipitation models is crucial. For example, precipitation models inform the release of water to maintain safe water levels in reservoirs during rainfall. Some of the current forecast systems include the North American Mesoscale Forecast System (NAM), the European Centre for Medium-Range Weather Forecasts (ECMWF), and The Weather Channel Forecast System (TWC). These forecast systems provide model run predictions on a grid which are used in aggregation models to predict precipitation events at given locations. Current aggregate models use linear techniques, either nearest-neighbor or weighted nearest-neighbor, to verify the model predictions against observed data. This does not provide the most accurate model because precipitation is a more spatially varied phenomenon than can be measured in a linear system. Our model proposes using optimal transport and Wasserstein distance to spatially inform the validation between the observed values from Quantitative Precipitation Estimate (QPE) data and model runs and thus provide more accurate aggregate precipitation models. The model weights all nearby grid points by calculating the optimal transport cost for each forecast model using the optimal transport distance that contribute to a weighted regression model which creates a more informed aggregation model than a standard linear weighting could provide. In this article, we detail the necessary preprocessing steps required to use the precipitation data in optimal transport models and provides initial results for the optimal transport based model.

1. Introduction. The National Centers for Environmental Prediction's (NCEP) National Blend of Models (NBM) contains assorted forecast models which use various weather forecasting techniques. Examples include the Global Ensemble Forecasting System which generates 21 models to account for uncertainty in weather patterns, the flagship model for NOAA called the Global Forecast System, and the High-Resolution Rapid Response Ensemble. While ensemble members of the NBM provide a well-rounded aggregated forecast, using dynamic weighting has been shown to improve forecasting ability [1].

Through our collaborations with the Developmental and Operations Hydrologist and the Senior Hydrometeorological Analysis and Support Forecaster of the National Oceanic and Atmospheric Administration's West Gulf River Forecast Center (NOAA WGRFC), we have determined there is a need for an aggregation model that can more accurately predict the magnitude of large precipitation events. According to their internal studies, their models perform well for precipitation events of less than half an inch, but struggle to obtain high accuracy forecasts for larger rain events. This is particularly relevant for quickly assessing and quantifying the impact of precipitation events on hydrologic systems, such as reservoirs.

We propose an optimal transport based machine learning model that can provide additional accuracy to existing models. The proposed method uses ensemble forecast data paired with observational data to improve predictive capabilities that can be applied to hydrologic systems. In particular, the model uses aggregation approaches that combine ensemble forecasts according to a set of learned weights that are determined by minimizing the misfit between the model forecast and observational data in an appropriate norm. Unlike most aggregate models that use a L_2 norm to measure misfit, we propose using the Wasserstein distance since it integrates both magnitude and spatial shifts in the misfit function and has show success in wind forecasting [15] and seismic imaging [21].

The structure of the article is as follows. We begin with a brief discussion of optimal transport theory and its relevance to data driven modeling. In particular, we focus on the necessary transformations of the source and target data into probability distributions with equal measure. Then we shift gears to discuss a bulk of the work that has been done for the

*University of Nebraska–Lincoln, katie.gerot@huskers.unl.edu
†Sandia National Laboratories, kdpiet@sandia.gov

model, which entails collecting and processing the model and observational data so that it can be used by optimal transport algorithms. We finish with a brief discussion on the model construction and preliminary results of the model. Detailed construction and analysis of the model is left to future work. We finish with a discussion of the robust data analysis done in this project and the potential of extending the data analysis to additional model forecasts, such as the European Centre of Medium-Range Weather Forecasts (ECMWF).

2. Optimal Transport. The theory of optimal transport was originally posed by Gaspard Monge as a way to find the optimal way to transport a pile of sand from one location to another [13]. Mathematically, this is formulated as finding the transportation map between two probability measures μ,ν in the spaces X, Y, given by the following definition.

DEFINITION 2.1. ***Transport map:*** *For densities μ, ν in X, Y, respectively, the transport map $T : X \to Y$ is given as $\nu(T(A)) = \mu(A)$ for all μ measurable sets in $A = \{x \in X : T(x) \in B\}$.*

A schematic of one dimensional transport is given in Figure 2.1

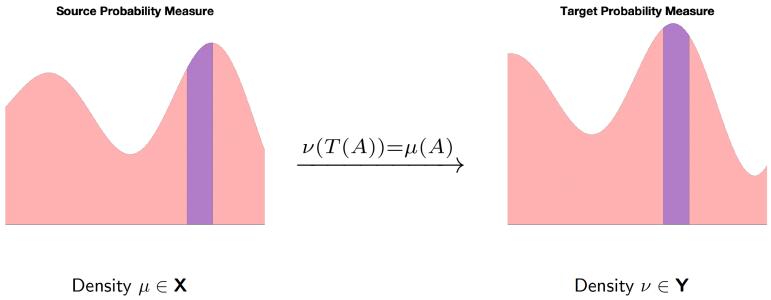


Fig. 2.1: Example of a transport map between two one-dimensional densities μ, ν in X and Y.

After defining the transport map, the optimal transport problem can be defined as follows.

DEFINITION 2.2. ***Optimal transport problem:*** *For $\mu \in X$ and $\nu \in Y$,*

$$\min_{T(x)} \int_X c(x, T(x)) d\mu(x)$$

over all μ measurable maps $T : X \to Y$ such that $\nu(T(A)) = \mu(A)$.

The cost function $c(x)$ can be a variety of functions depending on the application, with the original formulation considering Euclidean cost $c(x, y) = |x - y|$. There is a significant amount of theory surrounding solving the optimal transport with quadratic cost $c(x, y) = |x-y|^2$ [3,4,7,19]. Due to more robust theory and more straight forward numerical approximation, this work focuses on utilizing optimal transport with quadratic costs to improve discrete precipitation aggregation models.

2.1. Discrete optimal transport. Discrete optimal transport is extensively covered in [18] and this section gives a brief summary to better contextualize its use for meteorological applications. The discrete optimal transport problem is defined as,

DEFINITION 2.3. ***Discrete optimal transport:*** *For discrete distributions $\mu = \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$ and $\nu = \frac{1}{n}\sum_{j=1}^{n} \delta_{y_j}$, the optimal transport matrix is the solution to the mini-*

mization problem

$$\gamma^\star = \sum_{arg\ min_{i,j}} \gamma_{i,j} M_{i,j} \tag{2.1}$$

$$such\ that \gamma 1 = \mu;\ \gamma^T 1 = \nu, \gamma \geq 0, \tag{2.2}$$

where $M_{i,j}$ is the associated cost matrix of moving mass from x_i to y_j.

The numerical solution to the optimal transport problem is quite complex and is a large area of computational research [17]. The pseudo-code provided in Figure 2.2, gives the basics for computing the 1D discrete Wasserstein distance, which is straightforward but has poor scaling as data sets get larger. Since we aim to use optimal transport in to the large scale setting of improving precipitation models over the state of Texas, we opt for solving for the Wasserstein distance through a regularization which equates to finding the Sinkhorn distance. The Sinkhorn distance is less computationally expensive than directly computing the solution to the optimal transport problem, which is critical for dealing with the large amounts of precipitation data. For the sake of brevity, we refer to the paper [6] and the references therein for more of the details of computing the solution to optimal transport, specifically through the Sinkhorn algorithm.

Algorithm 1:

Input: S: source density distributions dictionary; T: target (observation) density distributions dictionary; each distribution contains a matching key and an x linspace and a kernel density estimation (KDE) probability density function (PDF)

Output: W: Wasserstein metrics

1 **foreach** $k \leftarrow$ key in S and T **do**

2 $\quad x_S \leftarrow$ x linspace of S at k

3 $\quad x_T \leftarrow$ x linspace of T at k

4 $\quad p_S \leftarrow$ KDE PDF of S at k

5 $\quad p_T \leftarrow$ KDE PDF of T at k

6 $\quad w = \min_\gamma \left(\sum_i \sum_j \gamma_{ij} ||x_S[i] - x_T[j]||^2 \right)^{1/2}$ s.t. $\gamma 1 = p_S, \gamma^T 1 = p_T, \gamma \geq 0$

7 $\quad$ Append w to W

8 **return** W

Fig. 2.2: Algorithm 1 calculates Wasserstein metric between source distribution and observation. The minimization problem on line 6 can be computed using `ot.wasserstein_1d` from the Python Optimal Transport package [8]

In summary, optimal transport provides a spatially informed measure for data sets. It changes the question from how similar are the two data sets in their magnitude to how similar are the data sets in both magnitude and location. Oftentimes, precipitation predictions may be off in either of these two metrics, magnitude or location, which motivates using optimal transport based metrics in an attempt to boost the predictive power of existing models.

3. Precipitation Data for Texas. Before the model can be built, we have to extract and process the model and validation data. Extracting and processing the model data is straightforward, since they are stored in simple NetCDF files in a grid format. The

validation data is more complicated, with multiple types of potential sources. The first set of validation data (ASOS) is point-wise data measurements taken from sites unequally scattered around Texas (usually at airports). This discrete set of points was not robust enough to build a model, since they are unequally distributed across the area of interest. These ASOS data points may be later used as additional validation data for the model. The second, more robust set of validation data (QPF) uses subject matter expertise to build qualitative forecast models on a grid similar to the model data. While this gives us a gridded set of equally distributed point wise measurements to pull from, it comes at the cost of a more complex data set that requires more processing before being used in any models. This section details the processing of these data sets using Python.

3.1. Automated Surface Observation Stations. Automated Surface Observation Stations (ASOS) are automated systems that record hourly weather conditions, including precipitation accumulation. These stations provide a validation dataset to compare the current forecast models and our aggregation model. For example, Figure 3.1 shows stations within a ten kilometer radius of ASOS T02 near Houston, Texas.

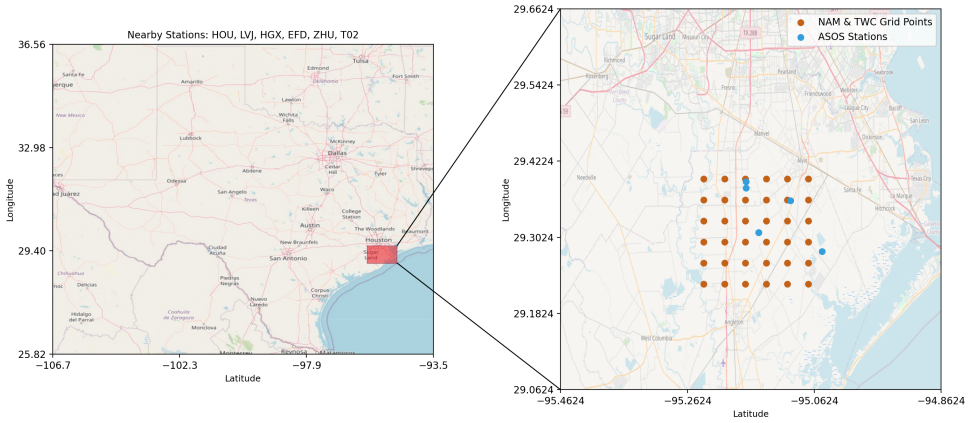


Fig. 3.1: ASOS Stations and TWC/NAM gridpoints surrounding Station T02 in Houston

Using ASOS data as the only observational data for the model poses several issues, particularly when comparing it to the relatively data rich model predictions. For example, if one wants to verify model prediction around Houston as seen in Figure 3.1, there are only 5 ASOS sites within a ten kilometer radius of the site. Since ASOS stations are generally associated with airport sites, this lack of validation data becomes even more prominent in more rural parts of Texas.

Recall that in order to use an optimal transport based metric 2, the data must be represented by discrete probability distributions. It is quite straightforward to transform the NAM and TWC model data into discrete measures using histograms of the data, since it includes several data points for a given area. The ASOS observations are much more sparse and creating an associated discrete distribution is significantly more complicated. In practice, one can build a normal distribution with mean as the amount of precipitation observed at the ASOS station and standard deviation of the error of the tool that collected the data. Figure 3.2, shows the associated one dimensional probability distributions of the section from Figure 3.1 for a selected model run. This distribution is regenerated for every model run based on the current precipitation observation and fixed tool error.

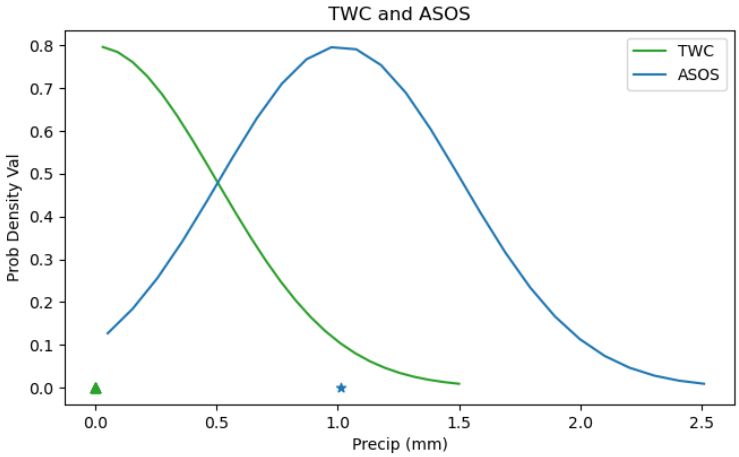


Fig. 3.2: Example of one dimensional TWC model and ASOS distributions for a single model run at Houston. Note that since the TWC model is centered around zero, the model predicted no rain and the ASOS station observed a little over one millimeter of rain.

It is clear from Figure 3.2 that by decomposing the data into one dimensional distributions based on the precipitation values of the model (NAM and TWC) data, we lose dependence on the location of the precipitation. This motivates using a different data set for the observation data set known as the quantitative precipitation estimate (QPE). Instead of posing the optimal transport problem between a set of model grid points (NAM and TWC) to a discrete set of observation points (ASOS), QPE data allows us to reformulate the optimal transport problem as a map between two dimensional images, thus preserving the spatial dependence of the model.

3.2. Quantitative Precipitation Estimate Data. As an alternate validation set, we use the quantitative precipitation estimate (QPE) grid data [11]. This data is generated through combination of several data sources and subject matter expert knowledge. The QPE data is public and available for download from the National Center for Atmospheric Research/University Corporation for Atmospheric Research (NCAR/UCAR) Earth Observing Laboratory (EOL) website.

The data for QPE is developed from both surface measurements and reflectivity measured by nearly 100 WSR-88D radars [2, 11]. Reflectivity is "[the] measure of the efficiency of a target in intercepting and returning radio energy. With hydrometeors, it is a function of the drop size distribution, number of particles per unit volume, physical state (ice or water), shape, and aspect" according to the Federal Meteorlogical Handbook Number 11, Part A [5]. The observations and reflectivity are then included in the National Precipitation Analysis (NPA) which merges them into the Stage IV data mosaic.

The `README` file that accompanies the QPE data download details that RFCs can manually inject quality control to the precipitation data before its inclusion in the QPE mosaic [11]. For example, frozen precipitation is inaccurately measured as less than what has fallen during snow events, since RFCs do not have the equipment to melt frozen precipitation. This data is not intentionally altered, and the lowered readings are sent to the QPE data as-is. Because our current data is restricted to Texas during the summer months, these inaccuracies will not affect its validity. For stations or RFCs that continuously give bad

readings, they are excluded from the analysis as a further quality control measure [2].

3.3. Data Processing. The forecast data for NAM and TWC are downloaded as NetCDF files. These are then handled with pandas `DataFrames` [12] as shown in Figure 3.3. Each column is labeled with a coordinate in degrees and each row is labeled with a Model Run (`ModRun`) which consists of the date in 'yyyymmddhh' format. For each Model Run and coordinate pair, the precipitation accumulation is recording in millimeters. `DataFrames` allow us to process large pieces of data in an organized format.

ModRun	(27.572964, -98.60017)	(27.572964, -98.56719)	...
2019010108	0.137620	0.064159	...
2019010109	0.141013	0.201357	...
2019010110	0.189398	0.360000	...
2019010111	0.341271	0.315510	...
...	...	...	

(a) NAM

ModRun	(27.572964, -98.60017)	(27.572964, -98.56719)	...
2019010207	1.3	1.3	...
2019010208	2.4	2.4	...
2019010209	2.1	2.1	...
2019010210	1.5	1.5	...
...	...	...	

(b) TWC

Fig. 3.3: Subsections of pandas DataFrames for NAM and TWC NetCDF data

The QPE data is downloaded as a GRIB (General Regularly-distributed Information in Binary form) file for each hour in the requested time period. Using the xarray Python package [9], we are able to access the data for each gridpoint as illustrated in Figure 3.4 for each file. The original files contain $987,601$ grid cells (881×1121) each.

GRIB standard requires latitude (variable name `latitude`) to range from -90 degrees south to 90 degrees north which matches our other data. For longitudinal coordinates (variable name `longitude`), GRIB standard requires values to be limited to 0 to 360 degrees east. The grid is entirely in the western hemisphere, so the longitudinal data is between 180 and 360 degrees. More information on GRIB regulations and formatting can be found in the documentation [20].

Precipitation data (variable `tp`) is described in units $kg \cdot m^{-2}$. This is equivalent to the forecast models' precipitation accumulation units of mm as one kilogram of water spread over a square meter is one millimeter deep. The precipitation grid uses the standard polar stereographic projection over the Continental United States (CONUS).

Figure 3.5 shows code for converting the gridded QPE data into the same format as the the forecast models. By performing the conversion and associated preprocessing, the number of files was compressed from 322 15 MB files to one 16 MB file.

prec[0], lat[0], lon[0]	prec[1], lat[1], lon[1]	... ...	prec[n-1], lat[n-1], lon[n-1]
prec[n], lat[n], lon[n]	prec[n+1], lat[n+1], lon[n+1]	... ...	prec[2n-1], lat[2n-1], lon[2n-1]
... ...	... ...	... ...	... ...
prec[(m-1)n], lat[(m-1)n], lon[(m-1)n]	prec[(m-1)n+1], lat[(m-1)n+1], lon[(m-1)n+1]	... ...	prec[mn-1], lat[mn-1], lon[mn-1]

Fig. 3.4: Visualization of $n \times m$ QPE data in GRIB format

```
def gridded_to_df(filename):
    dataset=nc.Dataset(''' Path to folder containing files ''')
    modelrun=filename.split('.')[0]

    # Enumerate to preserve index
    lat = enumerate(list(dataset.variables['latitude']).flatten())
    lon = enumerate(list(dataset.variables['longitude']).flatten())
    tp = enumerate(list(np.ma.getdata(dataset.variables['tp'])).flatten())

    # Filter data for values we want, preserving index
    tp_f = [x for x in list(tp_en) if not math.isnan(x[1]) and x[1] != 9999]
    lat_f = [x for x in list(lat_en) if 27.67 < x[1] < 30.43]
    lon_f = [(x[0], x[1] - 360) for x in list(lon_en) if 261.4 < x[1] < 265.16]

    # intersection of indicies
    lat_i = set([x[0] for x in lat_f])
    lon_i = set([x[0] for x in lon_f])
    tp_i = set([x[0] for x in tp_f])
    intersection = lat_i & lon_i  & tp_i

    # filter list by intersection ([x in set] is faster than [x in list])
    lat_fil = [x[1] for x in lat_fil if x[0] in intersection]
    lon_fil = [x[1] for x in lon_fil if x[0] in intersection]
    tp_fil = [x[1] for x in tp_fil if x[0] in intersection]

    df = pd.DataFrame([tp_fil], columns = list(zip(lat_fil,lon_fil)))
    df.insert(0, "ModRun", modelrun, True)
    return df

```

Fig. 3.5: Python code that converts grid data (in the form of a GRIB file) of the QPE data into the same format as the models (NAM and TWC).

4. Transport Regions. Beyond breaking up massive datasets, precipitation accumulation over large swaths of land in climates with occasional precipitation events will tend to skew towards zero. Figure 4.1 shows a major rain event along with the histogram of the precipitation values for the entire data set. One observes that even with a major storm, the histogram still has a large zero skew, which will greatly impact the related distributions needed for calculating the optimal transport distance. Figure 4.2 shows how the histograms are substantially more meaningful after dividing the larger domain into an arbitrary smaller domain.

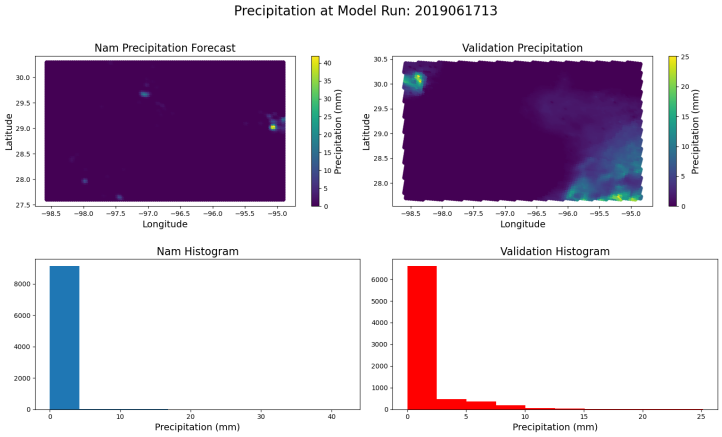


Fig. 4.1: Precipitation for the NAM and QPE sets with their corresponding histograms for a model run on June 17, 2019 taken at 13:00.

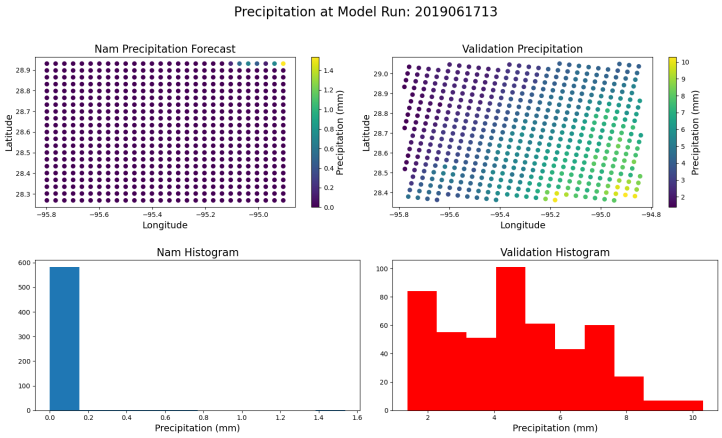
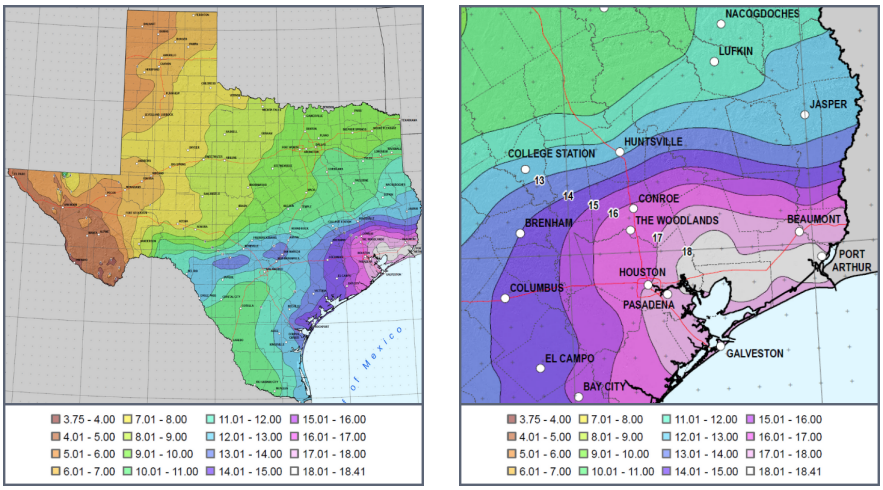


Fig. 4.2: Precipitation for the NAM and QPE sets with their corresponding histograms for a model run on June 17, 2019 taken at 13:00 after the domain has been partitioned into 16 arbitrary patches.

Thus, it is important to split larger regions into smaller, more manageable, subsections. These subsections will not be as granular as the forecast grid, as that would defeat the purpose of aggregation. Subsectioning land could be performed uniformly using a grid, but this does not take into account hydrologic or topological features, nor does it rely on expected precipitation distribution. Using meaningful subsections is important to maintain the model's usefulness for hydrologists and infrastructure planners.

The first type of division uses historical precipitation accumulation. Figure 4.3(a) shows the 24-hour isopluvials (or equivalent rainfall regions) for the state of Texas over the past 100 years. These isopluvials illustrate a radiating pattern of precipitation from the subtropical coastal region in the east to the arid plains region of west Texas. Figure 4.3(b) shows the same isopluvials centered on our current region of interest. By centering our model on

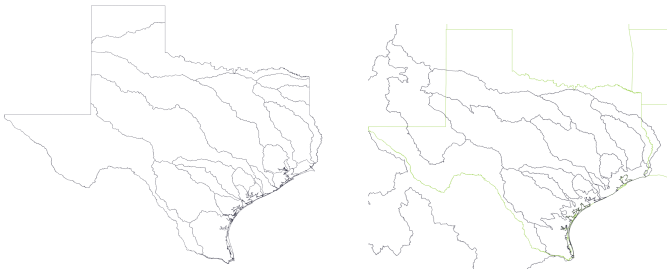
regions of similar historical precipitation, we can hope to avoid an unwanted skew.



(a) state of Texas (b) select region of Texas

Fig. 4.3: Isopluvials of 100-year 24-hour precipitation in inches for Texas. Adapted from map generated by the Precipitation Frequency Data Server (PFDS) using data from NOAA Atlas 14 Volume 11 Version 2, Precipitation-Frequency Atlas of the United States, Texas [16]

The second group of regions are basins which center on rivers in Texas. Figure 4.4(a) shows the twenty-three United States Geological Survey (USGS) defined river basins while Figure 4.4(b) shows the NOAA Forecast Groups over Texas. Both region groups were recommended by WGRFC as potential subsections. River basins provide a valuable form of subsectioning as they are already used by hydrologists. The Forecast Groups follow similar boundaries. While our target region within Texas does not contain as many basins as would be preferred, using a combination of basins or Forecast Groups in addition to historical precipitation may provide well-informed region bounds for building the model.



(a) 23 USGS-defined Texas River Basins (b) Texas Forecast Groups

Fig. 4.4: (a) Generated from Shapefile provided by the Texas Water Development Board; (b) Generated from Shapefile provided by WGRFC

5. Model Outline. Once the data has been converted into a NETCDF, grid format, we can begin to build the precipitation aggregation model based on differences using either the L_2 or Wasserstein W_2/Earth Mover's distance metric. Since the data is formatted as a grid for both the models and the QPE validation data, the model can be formulated similarly to comparing pixels between images. The main goal of the model is to create an aggregate model that is more accurate when compared to the validation data than a single model on its own. The precipitation aggregation model is still a work in progress, but we offer a high level outline to illustrate the future directions of this work.

5.1. Aggregate Model. To improve precipitation models for large rainfall events,we create an aggregate model based on the ridge regression formulation for ocean wave forecasting [10, 14]. In a ridge regression type model, the misfit of each model is measured against the validation data in an appropriate norm, then the various models are weighted such that they minimize the misfit to the validation data. This can be formulated using a time series approach and then used for future predictions.

Typically, a standard L_2 norm is used as the misfit function between the model and observation/validation data. In the context of the precipitation data, this entails finding the nearest validation point v_i to the model grid point m_i and calculating the L_2 norm between their precipitation values. This way of weighting the data will ignore any spatial variability in the data, giving a poor score for the model prediction even if it was off by only one grid point. For our model, we will use the L_2 norm as the benchmark to measure the performance of an optimal transport based model.

Since the data is still going through prepossessing steps, the optimal transport based model is a work in progress. Ideally, since optimal transport takes a more global approach to data than the nearest neighbor or L_2 approach, we expect it to have slightly better performance, particularly for major rain events. As a first pass, we have successfully found the optimal transport distance using using a Sinkhorn approximation from the Python optimal transport library [8]. This method essentially dampens any locations where there is precipitation in the source but not the target, but cannot create additional mass in the target if there was none in the source (see Figure 5.1). This is consistent with optimal transport theory and we are working on creating a more specific metric based on optimal transport that can allow for unbalanced distributions between the source and target measures.

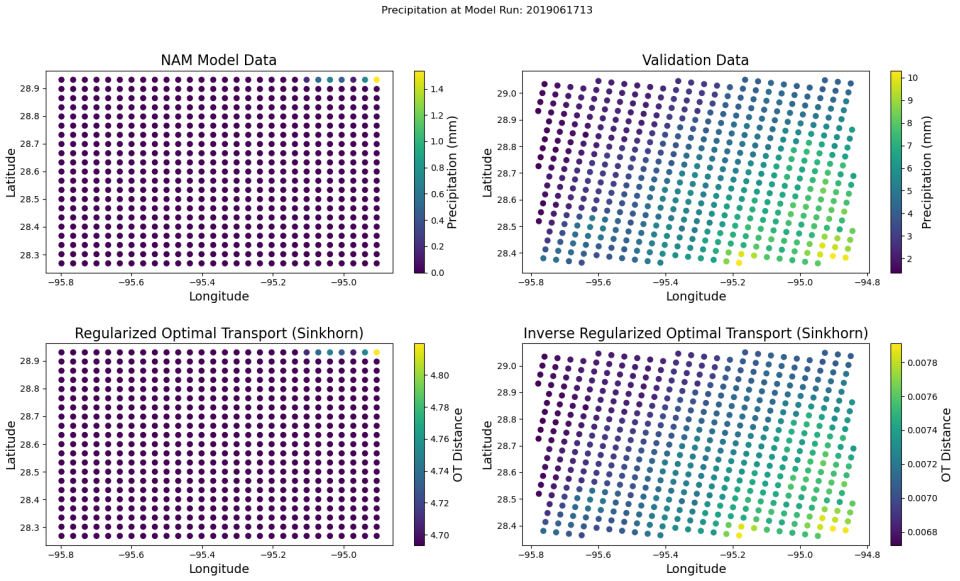


Fig. 5.1: Precipitation for NAM and QPE for a specific model run (top). The computed regularized discrete optimal transport from NAM to QPE (bottom left) and from QPE to NAM (bottom right).

6. Discussion. In this article we have summarized the necessary prepossessing steps needed to build a precipitation aggregation model utilizing optimal transport theory. We have included a high level outline of the desired model and its construction and detailed analysis is left to future work. Beyond model implementation, there are several improvements that could be made to the data collection process and could be beneficial to subsequent research, especially as the forecast area grows.

The more forecast models that are included in the aggregation models, the more informed it will be. Currently, the model consists of only NAM and TWC data. In the future it would be beneficial to also include the Global Forecast System (GFS), the European Centre for Medium-Range Weather Forecasts (ECMWF), and the High-resolution Rapid Response (HRRR) model. These are all members of the National Blend of Models and could provide a larger wealth of information. With such additions, the current calculations we are performing will have to be optimized within the model and integrated into the optimal transport model.

When computing the L_2 error norm for the forecast models' data against the QPE validation data, processing is performed with an extremely large number of operations (over 350 million) per model. For a validation dataset that is $m \times n$ and a model dataset that is $p \times q$, the time complexity is $O(mp + mn)$, because each of m coordinates must be checked against all p model coordinates to find the nearest neighbor. Once the nearest neighbor is found, all the gridpoints g_{mn} must be compared against the respective precipitation value in the model data.

To process 10 days of worth of data, it takes over two hours using parallel computing on eight cores. The time needed would increase for any calculations more complicated than Euclidean distance. Possible methods to try and improve this complexity is to compute the nearest neighbor coordinates once. Because the coordinates in our range do not change, this

would allow us to more quickly ($O(mn)$) compare data. Some optimization will need to be performed in order to allow for the addition of new models without exponentially increasing the time it takes to compute even basic functions and make the model computationally tractable for near real time use in the WGRFC.

References.

[1] J.-H. G. M. Alves, P. Wittmann, M. Sestak, J. Schauer, S. Stripling, N. B. Bernier, J. McLean, Y. Chao, A. Chawla, H. Tolman, G. Nelson, and S. Klotz", *"the ncep?fnmoc combined wave ensemble product: Expanding benefits of interagency probabilistic forecasts to the oceanic environment"*, "Bulletin of the American Meteorological Society", "94" ("2013"), pp. "1893 – 1905".

[2] M. Baldwin, *"national precipitation analysis faq"*, Environmental Modeling Center.

[3] Y. Brenier, *Polar factorization and monotone rearrangement of vector-valued functions*, Communications on Pure and Applied Mathematics, 54 (1991), pp. 375 – 417.

[4] L. Caffarelli, L. Ambrosio, Y. Brenier, G. Buttazzo, C. Villani, and S. Salsa, *The Monge-Ampère Equation and Optimal Transportation, an elementary review*, 01 2003, pp. 1–10.

[5] Federal Coordinator for Meteorological Services and Supporting Research, *WSR-88D meteorological observations fcm-h11a-2016*.

[6] S. Ferradans, N. Papadakis, G. Peyré, and J.-F. Aujol, *Regularized discrete optimal transport*, SIAM Journal on Imaging Sciences, 7 (2014), pp. 1853–1882.

[7] A. Figalli and G. Loeper, *C1 regularity of solutions of the monge–ampère equation for optimal transport in dimension two*, Calculus of Variations and Partial Differential Equations, 35 (2009), pp. 537–550.

[8] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer, *POT: Python Optimal Transport*, Journal of Machine Learning Research, 22 (2021), pp. 1–8.

[9] S. Hoyer and J. Hamman, *xarray: N-d labeled arrays and datasets in python*, Journal of Open Research Software, 5 (2017).

[10] S. C. James, Y. Zhang, and F. O'Donncha", *"a machine learning framework to forecast wave conditions"*, "Coastal Engineering", "137" ("2018"), pp. "1 – 10".

[11] Y. Lin, *Gcip/eop surface: Precipitation ncep/emc 4km gridded data (grib) stage iv data. version 1.0*, 2011.

[12] W. McKinney et al., *Data structures for statistical computing in python*, in Proceedings of the 9th Python in Science Conference, vol. 445, Austin, TX, 2010, pp. 51–56.

[13] G. "Monge, *"Mémoire sur la théorie des déblais et des remblais"*, "De l'Imprimerie Royale", 1781.

[14] F. O'Donncha, Y. Zhang, B. Chen, and S. C. James", *"ensemble model aggregation using a computationally lightweight machine-learning model to forecast ocean waves"*, "Journal of Marine Systems", "199" ("2019"), p. "103206".

[15] G. I. Papayiannis, G. N. Galanis, and A. N. Yannacopoulos, *Model aggregation using optimal transport and applications in wind speed forecasting*, Environmetrics, 29 (2018), p. e2531. e2531 env.2531.

[16] S. Perica, S. Pavlovic, M. S. Laurent, C. Trypaluk, D. Unruh, and O. Wilhite, *Precipitation-frequency atlas of the united states, texas*, 2018.

[17] G. Peyré and M. Cuturi, *Computational optimal transport*, 2020.

[18] C. "Villani, *"Topics in optimal transportation"*, "American Mathematical Society", 2003.

[19] C. Villani, *Optimal transport – Old and new*, vol. 338, 01 2008, pp. xxii+973.

[20] World Meteorological Association, *FM 92-XII Ext. GRIB - General Regularly-distributed Information in Binary form, Version 2*, 2 ed., 05 2003.

[21] Y. Yang, *Analysis and application of optimal transport for challenging seismic inverse problems*, 2019.

THE SCHWARZ ALTERNATING METHOD FOR MULTISCALE CONTACT MECHANICS

JONATHAN HOY*, IRINA TEZAUR†, AND ALEJANDRO MOTA‡

Abstract. Contact is an important research topic in the study of mechanical systems. State-of-the-art computational methods for simulating mechanical contact are prone to numerical difficulties, leading to poor performance (in terms of simulation time and accuracy) and a lack of robustness. Here, we describe and evaluate a novel approach for simulating contact based on the Schwarz alternating method. With this method, contact constraints are replaced with boundary conditions that are applied iteratively on the contact boundaries. Results from a canonical impact problem with an exact analytical solution suggest that the new Schwarz methodology has the potential to offer a significant improvement to established approaches.

1. Introduction. An important aspect of simulating mechanical systems, whe- ther engineered or natural, is understanding a given system's behavior when subjected to contact under normal or abnormal environments (e.g., touching surfaces, sliding, tightened bolts, impact, etc.). Whereas the methods and tools for simulating the bulk behavior of mechanical systems are well-developed and mature, the same cannot be said for contact mechanics. This situation is due to the complexity of the contact phenomenon itself and associated numerical difficulties. Traditionally, for computational simulation, the contact problem is divided into two steps: a proximity search, and the enforcement of contact constraints (introduced to prevent interpenetration of objects coming into contact). The proximity search is primarily a computer science problem, and has received much attention due to its paramount importance in other fields such as video game development. In relative terms, less attention has been devoted to the enforcement step, a multiscale physics phenomenon due primarily to the microscopic and macroscopic features of contact surfaces. Existing computational methods available for enforcement suffer from poor performance, both in terms of simulation timeand solution accuracy, and can lead both to long wait times and to physically incorrect predictions. Although traditionally, most of the computational expense in contact simulations is associated with the proximity search, there is also room for improvement when it comes to the efficiency of enforcement algorithms.

This paper introduces and evaluates numerically a fundamentally new approach to simulating mechanical contact based on the domain decomposition-based Schwarz alternating method [18]. The new approach leverages our previous work in Schwarz multiscale coupling [15, 16] and addresses two well-known problems in computational simulation of contact: (1) the accuracy of the contact constraint enforcement, and (2) the multiple scales involved. Rather than introducing contact constraints into the variational form of the problem, as done in conventional contact techniques, e.g., the penalty method [6, 11], the Lagrange multiplier method [1, 3] and the augmented Lagrangian method [1, 20, 26], the Schwarz alternating method decomposes the problem domain into two or more subdomains and prevents interpenetration by applying transmission (boundary) conditions in an iterative and alternating fashion on the subdomain boundaries. As shown in our earlier work [15, 16], the Schwarz alternating method has a number of desirable qualities, including its ability to use different element topologies and time integrators in different subdomains. We demonstrate herein that these advantageous properties carry over to the contact variant of the method.

Toward this effect, the remainder of this paper is organized as follows. Section 2 describes the variational formulation of the generic solid mechanics problem considered herein,

*University of Southern California, hoyj@usc.edu
†Sandia National Laboratories, ikalash@sandia.gov
‡Sandia National Laboratories, amota@sandia.gov

and details its spatio-temporal discretization. Section 3 presents our new Schwarz alternating contact formulation, which relies on non-overlapping subdomains and alternating Dirichlet-Neumann boundary conditions. Numerical results on a one-dimensional (1D) impact problem with an exact analytical solution are given in Section 4. The use of the 1D problem allows the investigation of the Schwarz algorithm as a viable contact method, circumventing other issues that are common to most contact algorithms, such as projection of fields from one contact surface to another. In addition to the Schwarz alternating method, we evaluate the performance of three conventional contact algorithms: the implicit and explicit penalty methods [6, 11], and the explicit Lagrange multiplier method (also known as the forward increment Lagrange multiplier method) [3]. Although the augmented Lagrangian method [1, 20, 26] is a popular approach for simulating mechanical contact, we do not consider this method in the present work. Our results demonstrate that the Schwarz alternating method predicts various quantities of interest (e.g., the contact point displacement, the impact and release time, the system energies) and conserves total energy better than the conventional contact methods, but introduces some oscillations in the contact point velocity and contact point forces. Ideas for minimizing these oscillations and general avenues for future work are discussed in Section 5.

2. Solid mechanics problem formulation. Consider the Euler-Lagrange equations for a generic dynamic solid mechanics problem in its strong form:

$$\text{Div}\ P + \rho_0\ B = \rho_0\ \ddot{\varphi} \quad \text{in} \quad \Omega \times I. \tag{2.1}$$

In (2.1), $\Omega \in \mathbb{R}^d$ for $d \in \{1, 2, 3\}$ is an open bounded domain, $I := \{t \in [t_0, t_1]\}$ is a closed time interval with $t_0 < t_1$, and $x = \varphi(X, t) : \Omega \times I \to \mathbb{R}^d$ is a mapping, with $X \in \Omega$ and $t \in I$, P denotes the first Piola-Kirchhoff stress, and $\rho_0\ B : \Omega \to \mathbb{R}^3$ is the body force, with ρ_0 denoting the mass density in the reference configuration. The over-dot notation denotes differentiation in time, so that $\dot{\varphi} := \frac{\partial\ \varphi}{\partial t}$ and $\ddot{\varphi} := \frac{\partial^2\ \varphi}{\partial t^2}$. Embedded within P is a constitutive model, which can range from a simple linear elastic model to a complex micro-structure model, e.g., that of crystal plasticity.

Suppose that we have the following initial and boundary conditions for the partial differential equations (PDEs) (2.1):

$$\begin{aligned} \varphi(X, t_0) &= X_0, \quad \dot{\varphi}(X, t_0) = v_0 \text{ in } \Omega, \\ \varphi(X, t) &= \chi \text{ on } \partial\Omega_{\varphi} \times I, \quad P\ N = T \text{ on } \partial\Omega_{T} \times I. \end{aligned} \tag{2.2}$$

In (2.2), it is assumed the outer boundary $\partial\Omega$ is decomposed into a Dirichlet and traction portion, $\partial\Omega_{\varphi}$ and $\partial\Omega_{T}$, respectively, with $\partial\Omega = \partial\Omega_{\varphi} \cup \partial\Omega_{T}$ and $\partial\Omega_{\varphi} \cap \partial\Omega_{T} = \emptyset$. The prescribed boundary positions or Dirichlet boundary conditions are $\chi : \partial\Omega_{\varphi} \times I \to \mathbb{R}^3$. The symbol N denotes the unit normal on $\partial\Omega_{T}$.

It is straightforward to show that the weak variational form of (2.1) with initial and boundary conditions (2.2) is

$$\int_I \left[\int_\Omega (\text{Div}\ P + \rho_0\ B - \rho_0\ \ddot{\varphi}) \cdot\ \xi\ \mathrm{d}V + \int_{\partial_T\Omega} T \cdot\ \xi\ \mathrm{d}S \right] \mathrm{d}t = 0, \tag{2.3}$$

where ξ is a test function in $\mathcal{V} := \{\ \xi \in W_2^1(\Omega \times I) :\ \xi =\ 0 \text{ on } \partial_{\varphi}\Omega \times I \cup \Omega \times t_0 \cup\ \Omega \times t_1\}$.

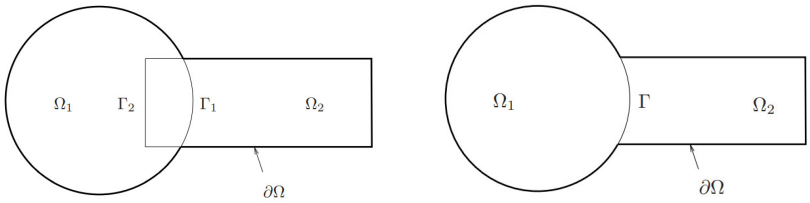
Discretizing the variational form (2.3) in space using the classical Galerkin finite element method (FEM) [10] yields the following semi-discrete matrix problem:

$$M\ \ddot{u} +\ f^{\text{int}} =\ f^{\text{ext}}. \tag{2.4}$$

In (2.4), M denotes the mass matrix, $u := \varphi(X,t) - X$ is the displacement, $\ddot{u}$ is the acceleration, f^{ext} is a vector of applied external forces, and f^{int} is the vector of internal forces due to mechanical and other effects inside the material. In the case of a problem with mechanical contact that is formulated with traditional contact methods, f^{ext} includes a contact contribution derived from a contact constraint, which must be enforced effectively.

A fully discrete problem is obtained by applying to (2.4) a time-integration scheme. A popular choice of time-integration scheme for solid mechanics problems such as those considered herein is the Newmark-beta method [17]. This scheme can be either first or second order accurate, depending on the values of its parameters β and γ. Additionally, it can be either implicit (and therefore unconditionally stable) or explicit, again depending on the values of β and γ.

3. The Schwarz alternating method. The purpose of the present work is to introduce and evaluate a *new* approach for simulating mechanical contact. This new approach is based on the Schwarz alternating method, an iterative domain decomposition-based approach that was first proposed in 1870 by Schwarz [18]. As mentioned earlier, in [15] and [16], the authors developed the Schwarz alternating method as a means for enabling continuum-to-continuum coupling in quasistatic and dynamic solid mechanics, respectively. In these works, the physical domain Ω is decomposed into two or more overlapping subdomains (Figure 3.1(a)), and the governing PDEs are solved within each subdomain in an iterative fashion, with information propagating through Dirichlet boundary conditions on the so-called Schwarz boundaries (Γ_1 and Γ_2 in Figure 3.1(a)). The method was shown to have a number of advantages over classical monolithic discretizations, enabling the seamless coupling in a plug-and-play manner of different mesh resolutions, different element types, and even different time integration schemes without introducing spurious errors or artifacts. Additionally, the method was shown to have a provable convergence guarantee [15, 16].



(a) Overlapping (b) Non-overlapping

Fig. 3.1: Illustration showing overlapping and non-overlapping domain decomposition.

3.1. Formulation. Motivated by the earlier work [15, 16] and the observation that a contact problem can be viewed as a coupled problem while two or more bodies are in contact, we propose herein to transform the Schwarz alternating method into a fundamentally new approach for simulating mechanical contact. A typical contact configuration is shown in Figure 3.1(b). The reader can observe that the configuration of interest involves two non-overlapping domains, connected by the contact boundary, denoted by Γ in this figure. Extending the Schwarz alternating method to contact problems hence requires extending the method's formulation to the case of non-overlapping subdomains. As first demonstrated in the late 1980s by Lions [13] and Zanolli *et al.* [25], obtaining a provably-converging Schwarz method for the case of non-overlapping domains requires specialized transmission

conditions. Specifically, whereas Dirichlet-Dirichlet transmission conditions ensure convergence in the case of overlapping domains, a convergent formulation in the non-overlapping domain case can be obtained by prescribing either Robin-Robin [5,8,9,13,14] or alternating Dirichlet-Neumann [4,7,12,25] boundary conditions on Γ (Figure 3.1(b)). In the present formulation, we consider the latter approach, which translates to alternating displacement-traction boundary conditions for the mechanical problem (2.3).

Consider, without loss of generality, a problem involving two subdomains, denoted by Ω_1 and Ω_2, that have come into contact, as depicted in Figure 3.1(b). Once contact has been detected using a global search algorithm based on specified contact conditions (described in Section 3.2), we begin the Schwarz iteration process. Following the Schwarz alternating method for transient solid dynamics [16], the present Schwarz alternating formulation includes the notion of a so-called "controller time stepper", which defines a set of global time-steps, denoted by $\triangle T$, at which the subdomains are synchronized (for more details, see Section 2.2 of [16]). For the specific case of (2.3), two subdomains and a controller time interval I_N, the Schwarz iteration takes the form:

$$\left\{\begin{array}{rcl} M_1\,\ddot{u}_1^{n+1} + f_1^{\text{int;n+1}} &=& f_1^{\text{ext;n+1}} \\ \varphi_1^{n+1} &=& \chi,\ \text{on}\ \partial_{\,\varphi}\Omega_1\backslash\Gamma, \\ \varphi_1^{n+1} &=& \varphi_2^{n},\ \text{on}\ \Gamma, \end{array}\right. \quad \left\{\begin{array}{rcl} M_2\,\ddot{u}_2^{n+1} + f_2^{\text{int;n+1}} &=& f_2^{\text{ext;n+1}} \\ \varphi_2^{n+1} &=& \chi,\ \text{on}\ \partial_{\,\varphi}\Omega_2\backslash\Gamma, \\ T_2^{n+1} &=& T_1^{n+1},\ \text{on}\ \Gamma. \end{array}\right. \tag{3.1}$$

In (3.1), $n = 0, 1, 2, \ldots$ denotes the Schwarz iteration number and N denotes the controller time-step number. We select to develop the alternating Dirichlet-Neumann formulation of the non-overlapping Schwarz method over the Robin-Robin formulation, as Dirichlet and Neumann (traction) boundary conditions are readily available in most solid mechanics codes, e.g., Sandia's Sierra/Solid Mechanics (Sierra/SM) code [19]. Alternate formulations, such as the formulation with Robin-Robin transmission conditions and various formulations involving numerical relaxation (see [22] and the references therein), may be considered in future work. The iteration (3.1) continues until convergence is reached. It is emphasized that the formulation (3.1) does not require that a conformal discretization be used within the various subdomains; as for the multiscale Schwarz alternating method [15,16], different discretizations, element types and even time-integration schemes can be used in different subdomains.

3.2. Contact criteria. An important part of any contact algorithm is defining a set of criteria to determine when contact has occurred. Herein, we consider the following set of contact criteria.

1. *Overlap condition*: triggered when two or more objects/domains have begun to overlap/penetrate each other.
2. *Push condition*: triggered when both of the following properties hold
   (a) *Compression*: the tractions at the interface are compressive.
   (b) *Sustainability*: there was contact in the previous time step.

Specifically, two or more bodies are determined to be in contact if either the overlap condition or the push condition hold.

The contact conditions enumerated above are roughly equivalent to the well-known Karush-Kuhn-Tucker (KKT) conditions [24,26] appearing in traditional mechanical contact formulations. It is noted that, unlike traditional contact formulations (penalty [6,11], Lagrange multiplier [1,3], augmented Lagrangian methods [1,20]), our Schwarz-based does not require the definition of contact constraints into the problem formulation.

4. Numerical results. The main contribution of this paper is the numerical evaluation of the Schwarz alternating method described in Section 3, as compared to several

state-of-the-art contact approaches. Section 4.1 succinctly summarizes the methods evaluated herein. Following this discussion, we describe the benchmark problem on which these methods are studied (Section 4.2) and present numerical results for several variants of this problem (Sections 4.3–4.4).

4.1. Summary of contact methods evaluated. We restrict our attention herein to three classes of methods: (1) the penalty method [6,11], (2) the Lagrange multiplier method [1,3], and (3) the Schwarz alternating method (Section 3).

The penalty method [6, 11] is one of the simplest approaches for mechanical contact, and applies a contact force that is linearly proportional to the amount of interpenetration by means of a penalty parameter τ. The penalty method is popular since it is very easy to implement into existing mechanics frameworks, but has the downside of having its accuracy and stability properties affected greatly by the choice of the penalty parameter, for which there is no exact science. If the penalty parameter τ is too low, the amount of interpenetration allowed can be too high, yielding inaccurate results; in contrast, selecting a τ that is too high can affect adversely the overall numerical stability of the method and can lead to inaccuracies/oscillations in the contact forces. The penalty method can be run with either an implicit or an explicit time-stepping scheme, both of which are considered in the present study.

In the Lagrange multiplier method [1, 3], contact constraints are imposed weakly using Lagrange multipliers; hence, unlike in the penalty method, the contact conditions are satisfied more precisely and there is no empirical parameter to tune. Lagrange multiplier methods present their own challenges, however. Care must be taken to design the Lagrange multiplier finite element space such that the *inf-sup* condition [2] is upheld, and implementing this mixed method in existing high-performance computing (HPC) codes such as Sierra/SM [19] can be cumbersome. Additionally, the Lagrange multiplier formulation gives rise to an indefinite discrete saddle point problem, which can be difficult to solve numerically and may require specialized preconditioning schemes. In [3], Carpenter *et al.* developed a specific variant of the Lagrange multiplier method with explicit time-stepping known as the "forward increment Lagrange multiplier method", which has been shown to deliver superior results over implicit Lagrange multiplier formulations for impact problems. For this reason, we restrict attention herein to the explicit (forward increment) Lagrange multiplier method, and do not consider the implicit variant of this method.

An important aspect of conventional contact methods such as the penalty and Lagrange multiplier methods is that they require the specification of contact constraints imposed within their respective formulations. Herein, we impose the so-called zero gap constraint, which ensures that the gap between a given pair of objects is never negative and hence the objects do not interpenetrate. We note here that it is not uncommon to impose in place of or in conjunction with the zero gap constraint a second constraint, namely that of a zero gap rate [21, 23]. The zero gap rate constraint was not considered in the numerical study performed herein, but would be an interesting future endeavor.

The Schwarz alternating method for simulating mechanical contact was described in Section 3. We evaluate herein three variants of the Schwarz alternating method: one in which an implicit Newmark-beta time-integration scheme is used in all subdomains, one in which an explicit Newmark-beta time-integration scheme is used in all subdomains, and one in which explicit and implicit Newmark-beta coupling is performed between the domains.

Table 4.1 summarizes the six methods evaluated in this paper.

4.2. One-dimensional impact problem. The Schwarz alternating method described in Section 3 and the three existing state-of-the-art contact methods described in Section 4.1 are evaluated on a simple 1D problem involving the impact of two identical linear elastic

Table 4.1: Summary of contact methods evaluated.

Method	Time-stepping scheme
Penalty	Implicit Newmark-beta
	Explicit Newmark-beta
Forward increment Lagrange multiplier [3]	Explicit Newmark-beta
Schwarz	Implicit-Implicit Newmark-beta
	Explicit-Implicit Newmark-beta
	Explicit-Explicit Newmark-beta

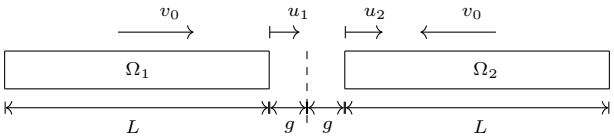


Fig. 4.1: Illustration of 1D impact problem

prismatic rods having density ρ, elastic modulus E and cross-sectional area A moving with equal speed in opposite directions. This test case is a variant of the problem considered in Section 5 of [3], and has an exact analytical solution. The configuration is depicted in Figure 4.1. The rods are initially undeformed and the configuration is symmetric about the plane at which the two rod faces impact. Let v_0 denote the initial speed of each rod. Let L denote the length of each rod, and assume the rods are initially separated by a distance $2g$. Per the derivation in [3], it is straightforward to shows that the position and velocity of the right end of the left rod are given by

$$x(t) = \begin{cases} -g + v_0(t - t_0), & t < t_{\text{imp}}, \\ 0, & t_{\text{imp}} \le t \le t_{\text{rel}}, \\ -v_0(t - t_{\text{rel}}), & t > t_{\text{rel}}, \end{cases} \quad v(t) = \begin{cases} v_0, & t < t_{\text{imp}}, \\ 0, & t_{\text{imp}} \le t \le t_{\text{rel}}, \\ -v_0, & t > t_{\text{rel}}, \end{cases} \tag{4.1}$$

respectively, where t_{imp} and t_{rel} are the impact and release times, respectively. The analytical values for these times are

$$t_{\text{imp}} = t_0 + \frac{g}{v_0}, \quad t_{\text{rel}} = t_{\text{imp}} + 2L\sqrt{\frac{\rho}{E}}, \tag{4.2}$$

where t_0 is the starting time of the simulation. Additionally, it can be shown that the contact force during impact is given by

$$f_{\text{contact}} = v_0\sqrt{E\rho}A, \tag{4.3}$$

and that the mass-averaged velocity, the kinetic energy (KE) and the potential energy (PE) take the form

$$\bar{v}(t) = \begin{cases} v_0, & t < t_{\text{imp}}, \\ v_0 - \frac{v_0\sqrt{E}}{L\sqrt{\rho}}(t - t_{\text{imp}}), & t_{\text{imp}} \le t \le t_{\text{rel}}, \\ -v_0, & t > t_{\text{rel}}, \end{cases} \tag{4.4}$$

$$KE = \begin{cases} \frac{1}{2}\rho A L v_0^2 & t < t_{\text{imp}}, \\ \frac{1}{2}\rho A L v_0^2 - \frac{1}{2}\sqrt{\rho E} A v_0^2 (t - t_{\text{imp}}) & t_{\text{imp}} \le t \le t_1, \\ \frac{1}{2}\sqrt{\rho E} A v_0^2 (t - t_1) & t_1 \le t \le t_{\text{rel}}, \\ \frac{1}{2}\rho A L v_0^2 & t > t_{\text{rel}}, \end{cases} \tag{4.5}$$

$$PE = \begin{cases} 0 & t < t_{\text{imp}} \\ \frac{1}{2}\sqrt{\rho E} A v_0^2 (t - t_{\text{imp}}) & t_{\text{imp}} \le t \le t_1, \\ \frac{1}{2}\rho A L v_0^2 - \frac{1}{2}\sqrt{\rho E} A v_0^2 (t - t_1) & t_1 \le t \le t_{\text{rel}}, \\ 0 & t > t_{\text{rel}}, \end{cases} \tag{4.6}$$

where

$$t_1 = t_{\text{imp}} + L\sqrt{\frac{\rho}{E}}, \tag{4.7}$$

is the time at which maximum PE and minimum KE are reached.

For the purpose of evaluating and comparing the various contact methods described herein, we have written a MATLAB code that discretizes the 1D impact problem using the FEM in space and the Newmark-beta time-integration scheme in time. This code is stored in an internal Sandia `git` repository. In the remainder of this document, N_x will denote the number of elements in the spatial discretization of each rod, and $\triangle t$ will denote the time-step used in the Newmark-beta time-stepper.

First, in Section 4.3, we verify the implementation of the conventional contact approaches considered herein (the penalty method and the forward increment explicit Lagrange multiplier method) on a low speed variant of the 1D impact problem, given by the parameters provided in the third column of Table 4.2. With this choice of parameters, the problem is identical to the test case considered in Section 5 of [3] and direct comparisons can be made for the purpose of verification.

Next, in Section 4.4, we evaluate the six contact methods considered herein, including our three Schwarz variants, on a high speed variant of the 1D impact problem, with parameters given in the fourth column of Table 4.2. This second high-speed variant of the 1D impact problem is qualitatively similar to the first low-speed variant, but more representative of typical Sandia applications.

4.3. Low speed impact variant: method verification. As mentioned earlier, our first task is to verify our implementation of the explicit penalty and explicit (forward increment) Lagrange multiplier (LM) methods on the low speed variant of the 1D impact problem. Toward this effect, we select parameters that match those used in the numerical study of Carpenter *et al.* [3], but converted to SI units (third column of Table 4.2). To match the setup of [3] we employed a spatial discretization having $N_x = 20$ elements, and a time-step of 2.226×10^{-6} seconds. For the explicit penalty method, we used the same value of the penalty parameter τ as the one used in [3], namely the SI equivalent of 7.5×10^6 lb/in. We did not consider the implicit penalty method, as it was not one of the methods considered in [3].

Figure 4.2 summarizes our main results for the low speed variant of the 1D impact problem (left column of the figure) compared to the results of Carpenter *et al* [3] (right column of the figure). The reader can observe that the solutions computed in our MATLAB code are qualitatively similar to those of Carpenter *et al.*, which provides verification of our implementation of these methods.

Table 4.2: 1D impact problem parameters for the two variants considered (low speed and high speed impact).

Parameter	Units	Low speed variant	High speed variant
ρ	kg/m^3	7844	1000
E	Pa	206.8×10^9	1.0×10^9
A	m^2	6.45×10^{-4}	1.0×10^{-6}
L	m	0.254	0.25
g	m	0.254×10^{-3}	0.02
v_0	m/s	5.136	100
t_s	s	0.0	-0.2×10^{-3}
t_f	s	20.0×10^{-5}	0.8×10^{-3}

4.4. High speed impact variant: method comparison. We now evaluate the contact methods summarized in Table 4.1 on the high speed impact problem described in Section 4.2. Our main results are summarized in Figures 4.3–4.8 and Tables 4.3–4.4. In the case of the Schwarz alternating method, each rod represents its own subdomain (Ω_1 and Ω_2), as shown in Figure 4.1. Unless otherwise noted, the bars are discretized using $N_x = 200$ linear elements. Also unless otherwise noted, a time-step of $\triangle t = 1.0 \times 10^{-7}$ is employed in all methods with the exception of Implicit-Explicit Schwarz. For this Schwarz variant, we utilize a time step of $\triangle t_i$ in subdomain Ω_i with $\triangle t_1 = 1.0 \times 10^{-7}$ and $\triangle t_2 = 1.0 \times 10^{-8}$, so as to illustrate the Schwarz alternating method's ability to couple not only different time-integrators but also different time-steps in different subdomains. It was verified that the time-steps employed were small enough to ensure satisfaction of the Courant-Friedrichs-Levy (CFL) condition for the explicit methods. For all the Schwarz methods considered, a controller time-step of $\triangle T = 1.0 \times 10^{-7}$ is employed. We select very tight relative and absolute Schwarz tolerances of 1.0×10^{-15} and 1.0×10^{-12}, respectively. These tolerances are applied to the Schwarz convergence criterion, which dictates that the change of position for all the subdomains at a given Schwarz iteration be less than these relative or absolute tolerances. For the two penalty methods evaluated, we chose a penalty parameter of $\tau = 7.5 \times 10^4$, as this value yielded the most accurate results.

4.4.1. Comparison of the Schwarz alternating method to conventional contact approaches. Figure 4.3 plots the contact point location of the right-most node of the left bar (Ω_1) as a function of time. The reader can observe that both penalty methods evaluated overpredict the contact point location between the impact and release times, similar to what was seen for the low speed impact variant of this problem (Figure 4.2). This behavior is not manifested by any of the Schwarz solutions. Although small oscillations can be observed in the Schwarz solutions while the bars are in contact, these are a tiny fraction of the exact contact point location. Additionally, while all three conventional approaches underpredict the release time, the Schwarz methods capture this quantity of interest to an accuracy of $\approx 0.01\%$. Similar conclusions can be drawn from Figure 4.4, which plots the mass-averaged velocity for the left bar as a function of time for the various methods: all three Schwarz variants calculate the mass-averaged velocity to a sufficiently greater accuracy than any of the conventional methods, especially near the time of release.

Figures 4.5–4.7 examine the kinetic, potential and total energies for the left bar as a function of time. It can be seen from Figure 4.5 that all three conventional methods exhibit noticeable errors in the kinetic energy after contact occurs. Halfway through the simulation, oscillations are observed in the solutions calculated using these methods. The explicit

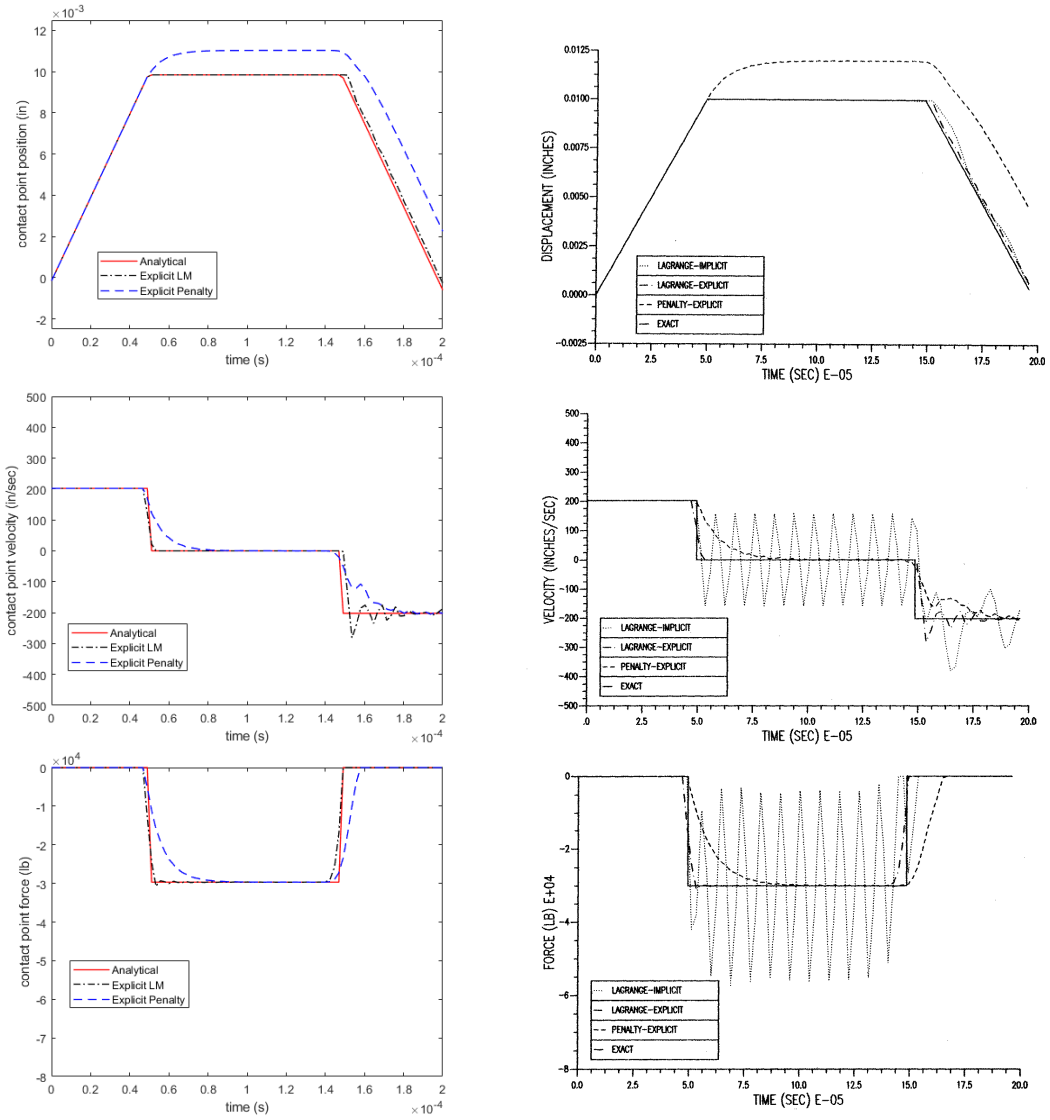


Fig. 4.2: Contact point position, contact point velocity and contact point force solutions computed using the explicit Lagrange multiplier and explicit penalty methods in our MATLAB code (left column) compared to the published solution in [3] (right column) for the low speed impact problem.

Lagrange multiplier and implicit penalty methods exhibit the largest errors in the kinetic energy following release, whereas the three Schwarz variants and the explicit penalty method exhibits the smallest error in this quantity. Remarkably, unlike any of the conventional methods, the Schwarz method is able to track the kinetic energy with great accuracy while the bars are in contact. The Implicit-Implicit Schwarz variant delivers the most accurate and least oscillatory kinetic energy solution. Similar conclusions can be drawn by inspecting Figure 4.6, which plots the potential energy of the left bar as a function of time. What is

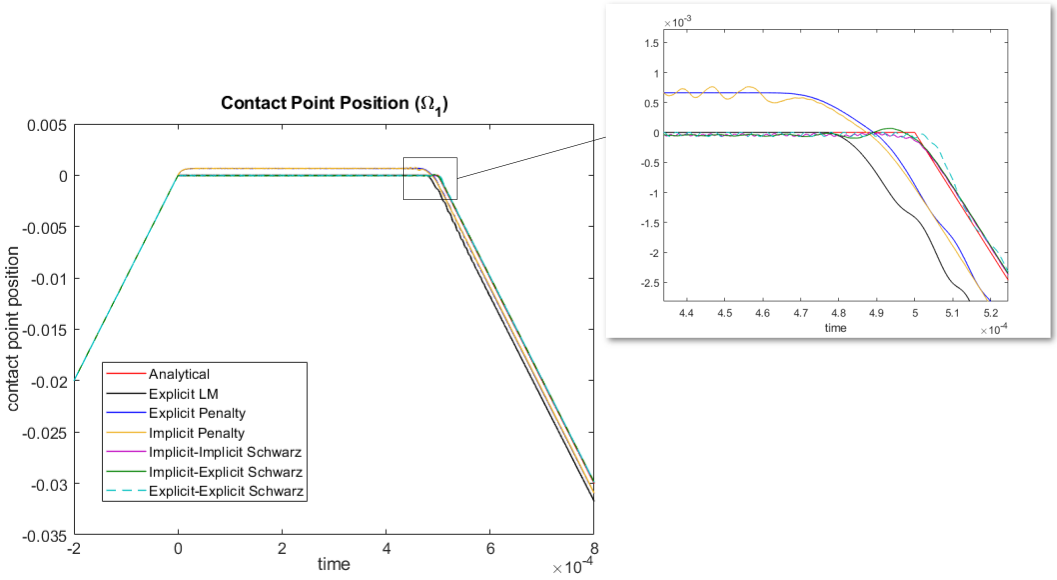


Fig. 4.3: Contact point position for the left bar (Ω_1) as a function of time (high speed impact)

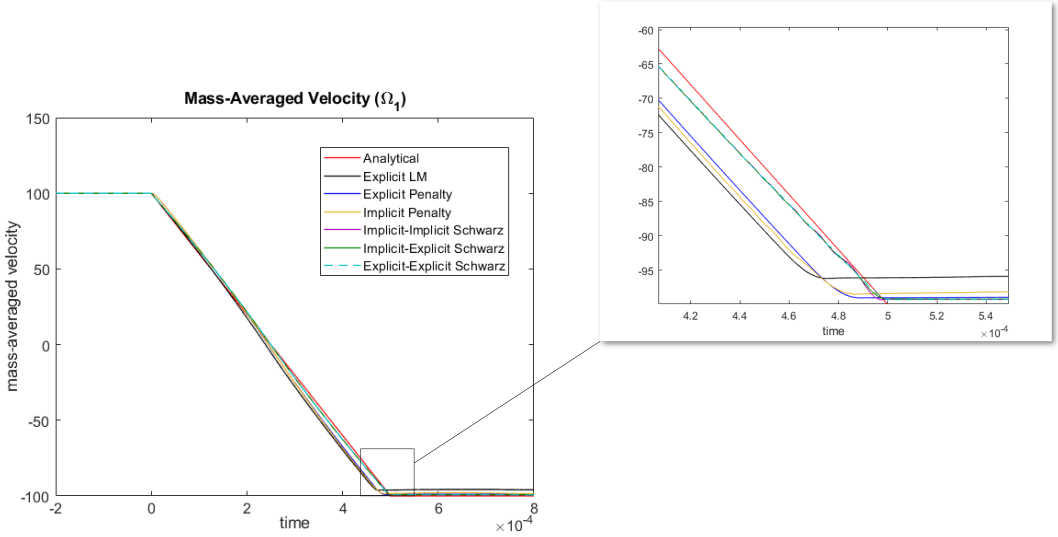


Fig. 4.4: Mass-averaged velocity for the left bar (Ω_1) as a function of time (high speed impact)

striking about this figure is the fact that all three conventional methods underpredict the peak potential energy by approximately 10%. This behavior is not seen in the Schwarz solutions, which capture the peak potential energy with a relative error of less than 0.1%.

Next, we discuss the ability of the Schwarz alternating method to conserve the total energy, defined as the sum of the kinetic and potential energies. It is straightforward to

show that the total energy should be conserved for this problem [3]. Figure 4.7 plots the total energy relative error in the left bar as a function of time for the methods evaluated. It is clear from this figure that the total energy error is negative for all six methods. This indicates that none of the methods are gaining energy, which could lead to numerical instabilities. As expected from the potential energy results (Figure 4.6), the three conventional methods exhibit a total energy loss of up to 9% following the instantiation of contact. The explicit penalty method loses the most energy, followed by the implicit penalty method and the explicit Lagrange multiplier method. Unlike the conventional contact approaches, the Schwarz method achieves an error of at most 0.25% in the total energy. It can be observed that the Explicit-Explicit Schwarz variant is the most accurate, followed by the Implicit-Implicit Schwarz and the Implicit-Explicit Schwarz. Interestingly, the more accurate Schwarz methods exhibit larger amplitude oscillations in the total energy after contact occurs.

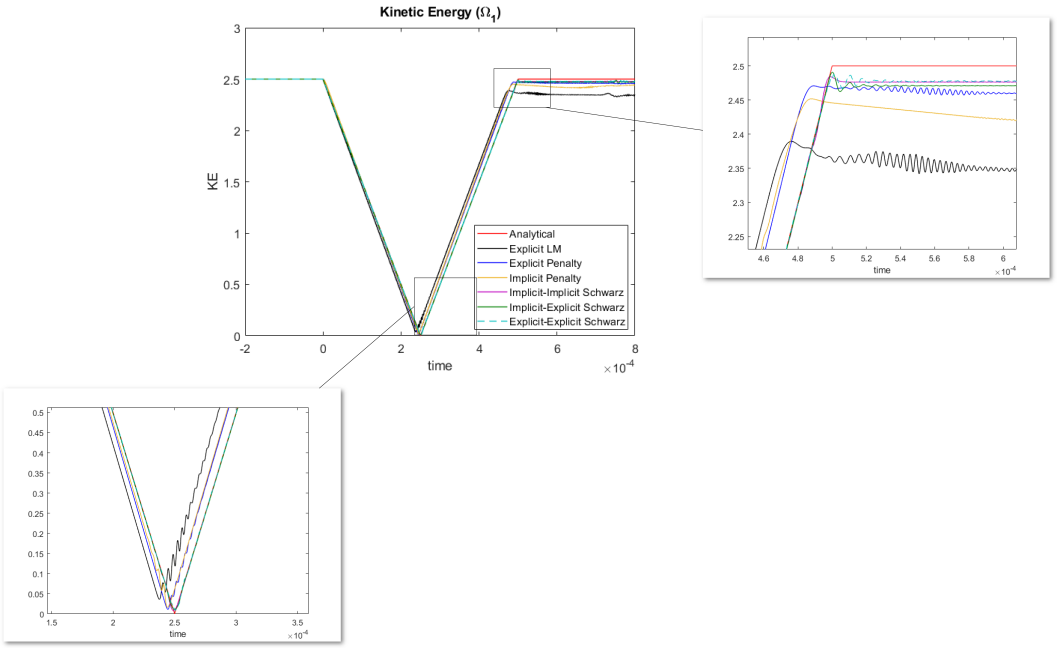


Fig. 4.5: Kinetic energy for the left bar (Ω_1) as a function of time (high speed impact)

In the results presented thus far, the Schwarz alternating method is better than the three conventional methods. The situation changes slightly when it comes to two other quantities of interest: the contact point force and the contact point velocity, plotted in Figures 4.8 and 4.9, respectively. While the three conventional methods exhibit several undesirable artifacts in the contact point force (e.g., an overshoot in the contact point force at the impact time for the explicit Lagrange multiplier method, oscillations in the contact point force around the time of release for the implicit penalty method, an under-prediction of the release time; see Figure 4.8(a)), these methods deliver in general a smooth contact force solution while the bars are in contact and after the bars separate. The same cannot be said of the Schwarz solutions, which exhibit in some cases significant oscillations following the instantiation of contact (Figure 4.8(b)). Similar results are seen in the contact point velocity (Figure 4.9), and suggest that the Schwarz methods may be suffering from a well-

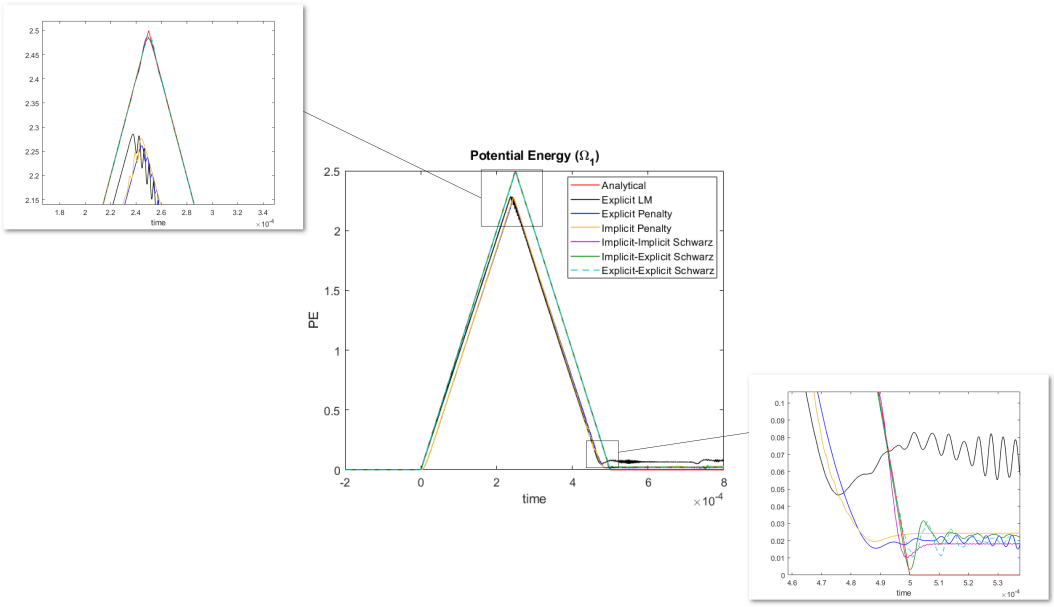


Fig. 4.6: Potential energy for the left bar (Ω_1) as a function of time (high speed impact

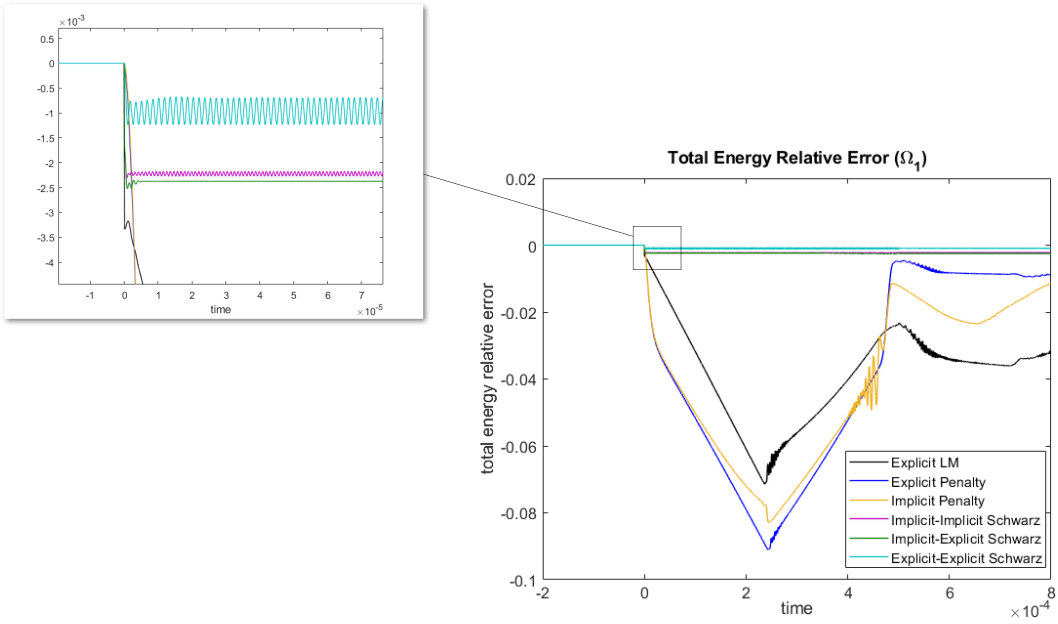
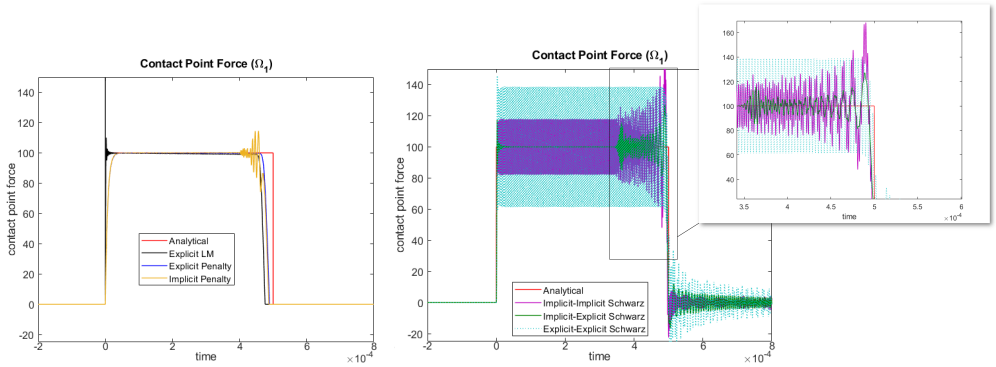


Fig. 4.7: Relative error in the total energy for the left bar (Ω_1) as a function of time (high speed impact)

known problem in the simulation of mechanical contact known as chatter, in which contact is lost and reestablished, sometimes numerous times (as seen here). Numerical experiments

reveal that the oscillations do not appear to be sensitive to the Schwarz convergence tolerances. It is interesting to observe that the chatter problem is significantly ameliorated by performing Implicit-Explicit Schwarz coupling. This observation suggests that the amount of chatter may be related to the predictor employed within the Schwarz coupling time integration scheme. Future work will focus on understanding the role of this predictor when it comes to the chatter problem, and potentially designing alternate predictors to reduce the amount of chatter present in the Schwarz solutions. Equally intriguing is the connection between the amount of chatter and the total energy loss (Figure 4.7). Comparing Figure 4.7 with Figures 4.8 and 4.9, it can be seen that the method with the largest total energy loss exhibits the least amount of chatter. This result is consistent with published results demonstrating that energy dissipation is necessary for the establishment of persistent contact [21], and suggests that it may be possible to reduce the amount of chatter in the Schwarz solutions by introducing numerical dissipation either through the Newmark-beta time-integrator (by selecting different parameters within this time-integration scheme) or directly into the Schwarz formulation.



(a) Conventional methods (b) Schwarz

Fig. 4.8: Contact point force for the left bar (Ω_1) as a function of time (high speed impact)

4.4.2. Convergence studies of the Schwarz alternating method. Having compared the Schwarz alternating method to our three conventional methods, we now turn our attention to evaluating the former method's convergence. We consider a single quantity of interest (QOI) in this study, namely the kinetic energy in the left bar. Figures 4.10, 4.11 and 4.12 depict the convergence of the Implicit-Implicit, Implicit-Explicit and Explicit-Explicit Schwarz methods as the mesh is refined from $N_x = 50$ to $N_x = 400$ elements. For the purpose of studying convergence in space, the time-step in this study was fixed to $\triangle t = 1.0 \times 10^{-8}$ in both subdomains in these calculations. The reader can observe convergence of the computed solution to the exact analytical solution with mesh refinement for all three couplings. It is curious to remark that oscillations can be seen in the Schwarz solution calculated using the finest mesh resolution ($N_x = 400$) when performing Implicit-Explicit Schwarz coupling (Figure 4.11). These oscillations begin shortly after the onset of contact and appear to grow in time, until the bars separate. The nature of these oscillations is currently unknown and will be studied in future work. Figure 4.13(a) depicts the mesh convergence of the three Schwarz variants considered with respect to the kinetic energy QOI. All three approaches converge at a rate of ≈ 0.82. This convergence rate is comparable to the convergence rate observed for a low-speed variant of this problem simulated using Sandia's ALEGRA code

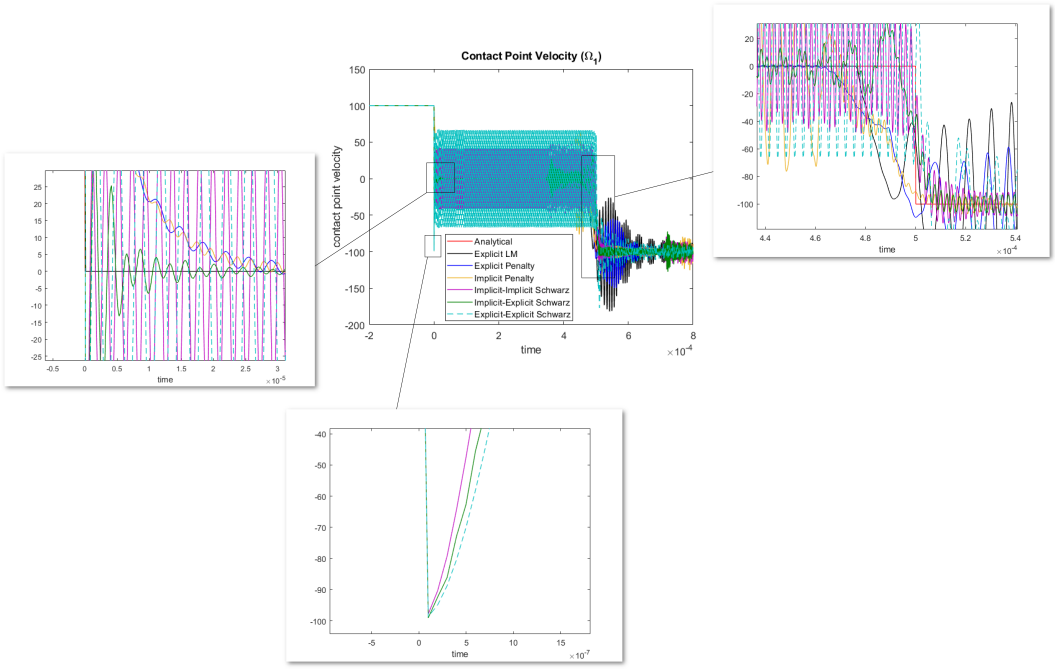


Fig. 4.9: Contact point velocity for the left bar (Ω_1) as a function of time (high speed impact)

base, which implements the forward increment explicit Lagrange multiplier contact method [23]. The Explicit-Explicit Schwarz method is seen to be the most accurate, but only by a very small margin.

We lastly provide some data on the number of Schwarz iterations required for convergence, a measure of computational efficiency. Figure 4.13(b) plots the number of Schwarz iterations required for convergence for the three Schwarz variants when a spatial resolution of $N_x = 200$ and a time-step of $\triangle t = 1.0 \times 10^{-7}$ is employed. The reader can observe that the method converges in between two and five Schwarz iterations, depending on the type of coupling despite our selection of very tight Schwarz convergence tolerances (a relative tolerance of 1.0×10^{-15} and an absolute tolerance of 1.0×10^{-12}). Curiously, Explicit-Explicit Schwarz requires the fewest number of iterations to achieve convergence at this resolution (between two and three). As expected, no Schwarz iterations are required before the bars come into contact and after the bars separate. Tables 4.3 and 4.4 summarize the maximum and the average number of Schwarz iterations as a function of N_x (with a fixed time-step of $\triangle t = 1.0 \times 10^{-8}$) and as a function of $\triangle t$ (with a fixed spatial resolution of $N_x = 200$), respectively. The reader can observe that the number of Schwarz iterations increases in general as the mesh is refined. Additionally, Implicit-Implicit coupling requires the most Schwarz iterations in general. The number of Schwarz iterations required for convergence does not change significantly as the time-step is reduced, with the exception of the case in which the time-step is reduced from 1.0×10^{-7} to 1.0×10^{-8} and Explicit-Explicit Schwarz coupling is employed (Table 4.4).

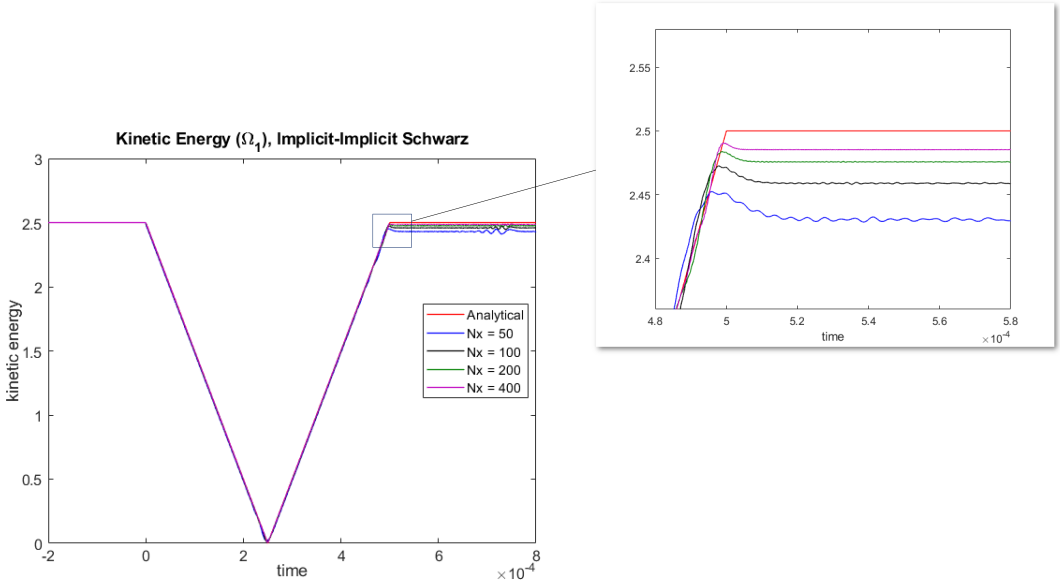


Fig. 4.10: Mesh convergence of the kinetic energy for the left bar (Ω_1) as a function of time computing using Implicit-Implicit Schwarz coupling with a time-step of $\triangle t = 1.0 \times 10^{-8}$ in both subdomains (high speed impact).

Table 4.3: Maximum/average number of Schwarz iterations as a function of N_x with the time-step fixed to $\triangle t = 1.0 \times 10^{-8}$ for various Schwarz couplings (high speed impact).

N_x	Implicit-Implicit	Implicit-Explicit	Explicit-Explicit
50	4/1.7354	4/1.7483	4/1.7562
100	5/1.9105	4/1.7506	4/1.7569
200	5/2.2184	5/2.0225	5/2.1953
400	6/2.5882	5/2.3142	5/2.2505

Table 4.4: Maximum/average number of Schwarz iterations as a function of $\triangle t$ with the spatial resolution fixed to $N_x = 200$ for various Schwarz couplings (high speed impact).

$\triangle t$	Implicit-Implicit	Implicit-Explicit	Explicit-Explicit
1.0×10^{-7}	5/2.447	5/1.8768	3/1.2532
1.0×10^{-8}	5/2.2184	5/2.0225	5/2.1953
1.0×10^{-9}	5/2.2195	5/2.0607	5/2.1964

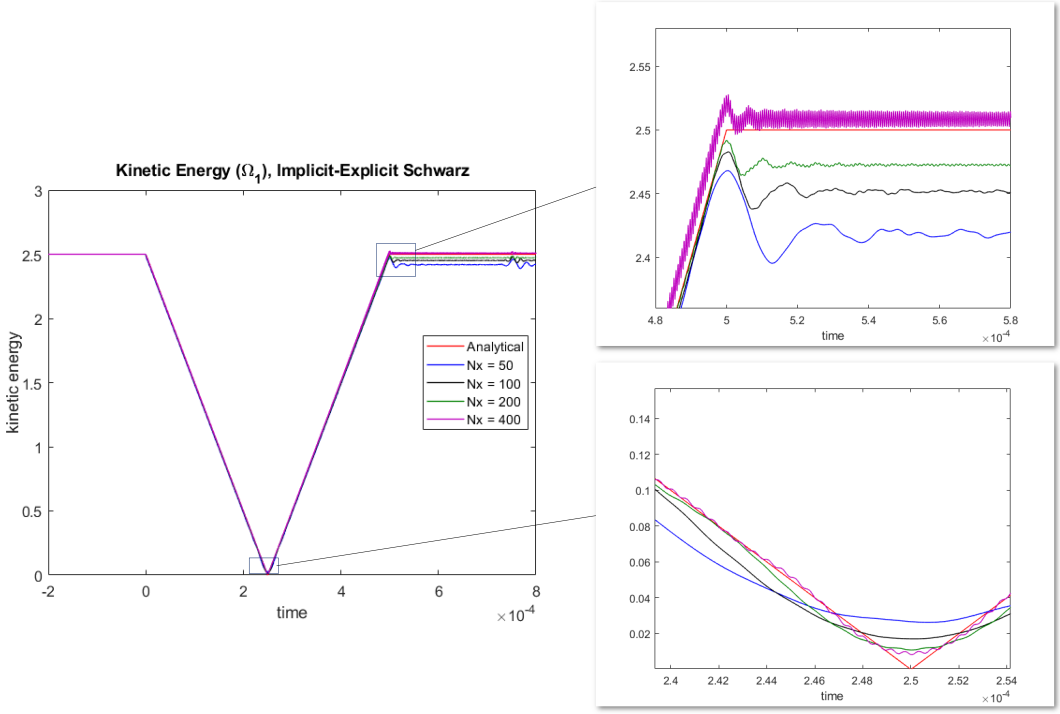


Fig. 4.11: Mesh convergence of the kinetic energy for the left bar (Ω_1) as a function of time computing using Implicit-Explicit Schwarz coupling with a time-step of $\triangle t = 1.0 \times 10^{-8}$ in both subdomains (high speed impact).

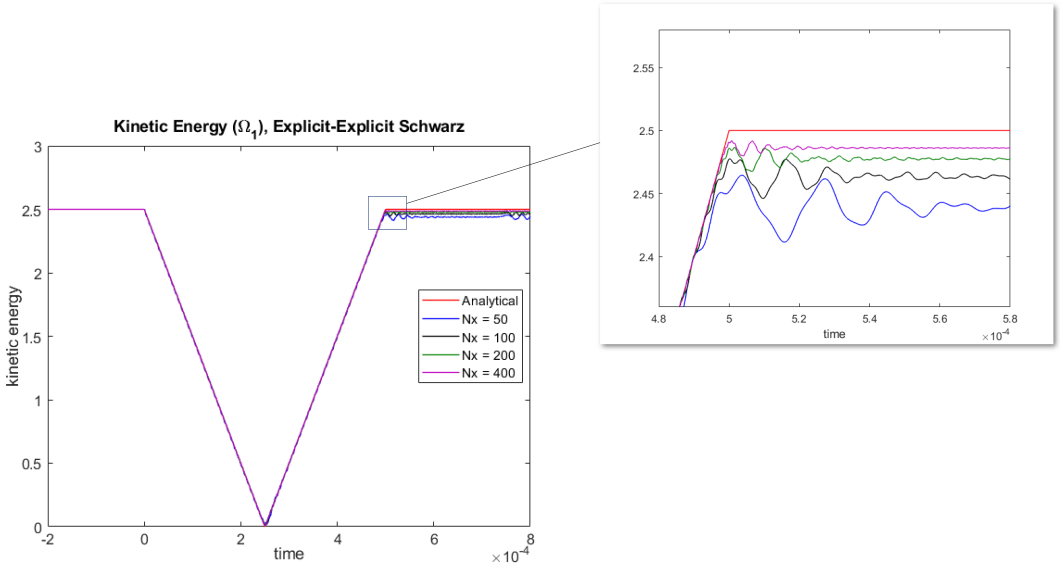
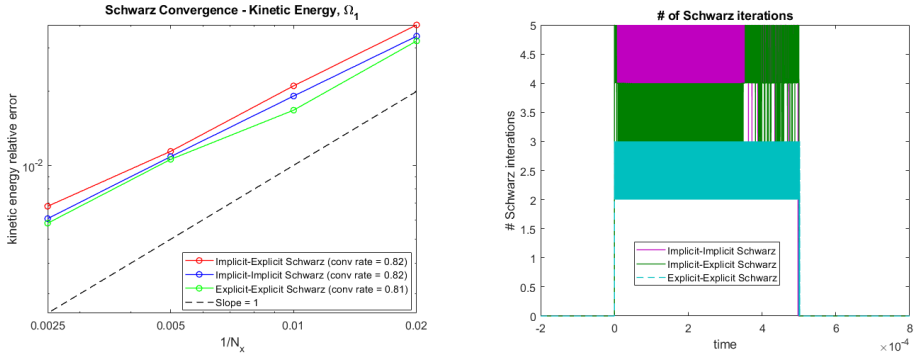


Fig. 4.12: Mesh convergence of the kinetic energy for the left bar (Ω_1) as a function of time computing using Explicit-Explicit Schwarz coupling with a time-step of $\triangle t = 1.0 \times 10^{-8}$ in both subdomains (high speed impact).



(a) Mesh convergence of the kinetic energy for the left bar (Ω_1) when $\triangle t = 1.0 \times 10^{-8}$

(b) Number of Schwarz iterations required for convergence ($N_x = 200$, $\triangle t = 1.0 \times 10^{-7}$)

Fig. 4.13: Convergence metrics for various Schwarz couplings (high speed impact).

5. Summary. This paper presents a new computational framework for simulating mechanical contact based on the Schwarz alternating method. In this approach, contact constraints are replaced with alternating Dirichlet-Neumann boundary conditions that are applied iteratively on the contact boundaries following a non-overlapping domain decomposition of the problem geometry. After describing the Schwarz methodology, we evaluate the method on a 1D impact problem with an exact analytical solution and compare the method's accuracy with that of conventional contact algorithms, namely the penalty method and the Lagrange multiplier method. We consider three variants of the Schwarz method in which different Newmark-beta time-integrators are used in different subdomains, so as to demonstrate the method's flexibility in coupling different time integrators with possibly disparate time-steps. Our results demonstrate the the Schwarz alternating method delivers a solution with substantially better accuracy than the conventional approaches for QOIs such as the contact point displacement, the mass-averaged velocity, the impact time, the release time, and the kinetic and potential energies. Additionally, the new method conserves energy significantly better than the conventional approaches. An unfortunate consequence of the method's ability to conserve energy so well appears to be the introduction of oscillations in the contact point velocity and contact point force. Future work will focus on better understanding the cause of these oscillations and devising approaches to mitigate them. Preliminary results suggest that the introduction of some slight dissipation [21] and/or numerical relaxation [22] can ameliorate the problem.

Future work will also include the following additional studies and extensions: (1) the introduction of additional or alternate contact constraints to those discussed in Section 3.2 to the Schwarz formulation, (2) a comparison of the Schwarz alternating method to conventional contact formulations in which a zero gap rate constraint is used in place of or in conjunction with a zero gap constraint, (3) an investigation of why the use of Implicit-Explicit Schwarz coupling introduces oscillations in QOIs such as the kinetic energy when sufficiently fine meshes are employed, and (4) an implementation and evaluation of the Schwarz alternating method in multiple spatial dimensions. The third of these tasks will require the development of operators for consistent transfer of contact traction boundary conditions using the concept of prolongation and restriction, common in multigrid methods.

Acknowledgment. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the U.S. Government.

References.

[1] T. Belytschko and S. P. Xiao, *Coupling Methods for Continuum Model with Molecular Model*, International Journal for Multiscale Computational Engineering, 1 (2003), pp. 115–126.
[2] D. Boffi, F. Brezzi, and M. Fortin, *Mixed finite element methods and applications*, Springer, 2013.
[3] N. J. Carpenter, R. L. Taylor, and M. G. Katona, *Lagrange constraints for transient finite element surface contact*, International journal for numerical methods in engineering, 32 (1991), pp. 103–128.
[4] J. Côté, M. J. Gander, L. Laayouni, and S. Loisel, *Comparison of the dirichlet-neumann and optimal schwarz method on the sphere*, in Domain Decomposition Methods in Science and Engineering, T. J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, eds., Berlin, Heidelberg, 2005, Springer Berlin Heidelberg, pp. 235–242.
[5] Q. Deng, *A nonoverlapping domain decomposition method for nonconforming finite element problems*, Communications on Pure and Applied Analysis, 2 (2003), pp. 297–310.
[6] O. Faltus, *Object-oriented design and implemention of the contact mechanics into finite element code "OOFEM"*, PhD thesis, Czech Technical University, Prague, Czech Republic, 2020.

[7] D. Funaro, A. Quarteroni, and P. Zanolli, *An iterative procedure with interface relaxation for domain decomposition methods*, SIAM J. Numer. Anal., 25 (1988), pp. 1213–1236.
[8] M. Gander, *Schwarz methods over the course of time*, Electronic Transactions on Numerical Analysis, 31 (2008), pp. 228–255.
[9] L. Gerardo-Giorda and M. Perego, *Optimized schwarz methods for the bidomain system in electrocardiology*, ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique, 47 (2013), pp. 583–608.
[10] T. Hughes, *Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Dover Publications, Inc., Mineola, New York, 2000.
[11] I. Hunek, *On a penalty formulation for contact-impact problems*, Computers and Structures, 48 (1993), pp. 193–203.
[12] F. Kwok, *Neumann–neumann waveform relaxation for the time-dependent heat equation*, in Domain Decomposition Methods in Science and Engineering XXI, J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. Widlund, eds., Cham, 2014, Springer International Publishing, pp. 189–198.
[13] P. L. Lions, *On the Schwarz alternating method III: A variant for nonoverlapping subdomains*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, 1990, pp. 202–223.
[14] S. Lui, *On accelerated convergence of nonoverlapping schwarz methods*, Journal of Computational and Applied Mathematics, 130 (2001), pp. 309–321.
[15] A. Mota, I. Tezaur, and C. Alleman, *The Schwarz alternating method*, Comput. Meth. Appl. Mech. Engng., 319 (2017), pp. 19–51.
[16] A. Mota, I. Tezaur, and G. Phlipot, *The Schwarz alternating method for transient solid dynamics*, Int. J. Numer. Meth. Engng. (under review), (2021).
[17] S. S. Rao, *The finite element method in engineering*, Butterworth-heinemann, 2017.
[18] H. Schwarz, *Über einen Grenzübergang durch alternierendes verfahren*, Vierteljahrsschriftder Naturforschenden Gesellschaft in Zurich, 15 (1870), pp. 272–286.
[19] SIERRA Solid Mechanics Team, *Sierra/solidmechanics 4.22 user's guide*, Tech. Rep. SAND2011-7597, Sandia National Laboratories Report, 10 2011.
[20] J. Simo and T. Laursen, *An augmented lagrangian treatment of contact problems involving friction*, Computers and Structures, 42 (1992), pp. 97–116.
[21] J. M. Solberg and P. Papadopoulos, *A finite element method for contact/impact*, Finite Elements in Analysis and Design, 30 (1998), pp. 297–311.
[22] I. Tezaur, *Schwarz variants*, tech. rep., Unpublished manuscript, 2021.
[23] I. Tezaur, T. Voth, J. Niederhaus, J. Robbins, and J. Sanchez, *An eXtended Finite Element Method (XFEM) Formulation for Multi-Material Eulerian Solid Mechanics in the ALEGRA Code*.
[24] P. Wriggers and G. Zavarise, *Encyclopedia of Computational Mechanics, Volume 2: Solids and Structures*, John Wiley & Sons, Ltd. Edited by E. Stein, R. de Borst and T.J.R. Hughes, 2004.
[25] P. Zanolli, *Domain decomposition algorithms for spectral methods*, CALCOLO, 24 (1987), pp. 201–240.
[26] O. C. Zienkiewicz and R. L. Taylor, *The finite element method for solid and structural mechanics*, Elsevier, 2005.

LEARNING TRANSFERABLE NEURAL NETWORK SURROGATES FOR KOHN-SHAM DENSITY FUNCTIONAL THEORY

KYLE R. LENNON* AND SIVASANKARAN RAJAMANICKAM†

Abstract. Density functional theory (DFT) is widely used to compute properties of many-body systems from first principles, and often serves as an intermediate step to computing forces on atomic nuclei in molecular dynamics (MD) simulations. However, DFT calculations are computationally intensive, and their cost becomes prohibitive for large systems or long MD simulations. Recently, efforts to circumvent expensive DFT calculations by learning system-specific neural network surrogates have been met with some success. Still, these surrogates require the generation of *ab initio* training data and long training times for every individual system and thermodynamic state. Here, we investigate whether this approach can be made more widely applicable by leveraging modern machine learning techniques, such as transfer learning and Bayesian approaches, to produce surrogate models that transfer between systems and states, or adapt quickly without the need for extensive training.

1. Introduction. In problems of materials engineering, where a principal focus is to find a specific material whose properties are most desirable for a certain engineering application, the design space – namely, the space of all candidate materials – is oftentimes extraordinarily vast. Due to the numerous candidate materials in such problems, launching experimental campaigns to directly characterize the properties of interest in each material is typically infeasible due to the cost and time associated with physical experiments. Thus, experimental studies are usually much more limited in scope, and many candidate materials are left unexplored.

Recently, however, the inability to investigate the numerous materials applicable to certain science and engineering problems has diminished, as computational methods take the place of experimental studies. This effort is highlighted in the Integrated Computational Materials Engineering initiatives, which support simulations across a diverse range of length scales for materials design and discovery [8, 14]. In computational materials science, and in particular for problems where the microstructural and electronic properties of materials are of interest, density functional theory (DFT) is a computational method that has been revolutionary [17, 20]. DFT drastically reduces the computational effort needed to solve the many-body Schrödinger equation, making solutions to problems with tens or hundreds of atoms feasible, when they previously would have been prohibitively expensive [6]. These solutions may then be directly used to estimate properties of a material, or as an intermediate step in molecular dynamics (MD) simulations, which evolve the atomic configuration in a material according to Newton's laws [2, 22]. In fact, from MD simulations that use DFT to compute the forces on atomic nuclei (often referred to as *ab initio* MD), one may compute any material property of interest, provided that the simulation converges in a feasible amount of time [4, 19].

Although DFT unlocks the potential to simulate a number of atoms far larger than by direct solution of the Schrödinger equation, its computational cost does grow quickly with the size of the simulation domain, and again becomes prohibitively large for systems bigger than a few hundred atoms [27, 28]. This is particularly true in *ab initio* MD simulations, where DFT calculations are performed at every time step. Therefore, MD simulations of larger materials, which are often needed to observe material properties that are relevant at macroscopic length scales, typically replace DFT with empirical interatomic potentials (IAPs) [10]. Although these IAPs are much less expensive to evaluate, they are inexact, and their degree of applicability for unstudied materials is often not known. Thus, it is of

*Department of Chemical Engineering, Massachusetts Institute of Technology, krlennon@mit.edu
†Sandia National Laboratories, srajama@sandia.gov

particular interest to substantially speed up DFT calculations, to retain the accuracy of *ab initio* MD while maintaining the scalability of empirical methods.

Recent efforts to accelerate DFT calculations have focused in particular on developing surrogates to the governing equations and their solutions using machine learning [7,9]. This machine learning DFT (ML-DFT) framework trains an artificial neural network to map from local descriptors of the atomic environment to the output of a DFT calculation. Networks are trained on the output of DFT calculations from snapshots of *ab initio* MD simulations for a particular material at a set temperature. The forward passes through the trained ML-DFT neural network are much faster than full DFT calculations, and scale linearly in the number of simulation grid points. This speed and scalability allows ML-DFT approaches to tackle simulations of larger scale and duration than those involving direct DFT calculations, and may help to extend *ab initio* calculations to scales on which bulk properties of materials emerge.

However, state-of-the-art ML-DFT models require the generation of new data and conducting long training runs for every material and temperature of interest. Temperature, for instance, is an important state parameter in molecular simulations, and may substantially affect the configurational state-space explored by a collection of atoms due to enhanced thermal fluctuations. Thus, a model trained at one temperature may not be applicable to another, due to the difference in the underlying domain of explored configurations in the training data set. The high computational cost of the simulations needed to generate training data and of the training campaigns partially offsets the performance gains realized by the surrogate models. Models that transfer to different conditions, or more quickly adapt in light of limited training data, could dramatically reduce this overhead cost, and lead to more widespread and rapid adaptation of ML-DFT in practice.

In this work, we explore the development of these general, or adaptable, models by frontier methods in machine learning, in particular using Bayesian methods for transfer learning. After developing the foundations of DFT in Section 2, we introduce these machine-learning methods in 3. Section 4 presents preliminary results from applying these machine learning methods to the ML-DFT framework. Finally, Section 5 presents a discussion of the preliminary results, and provides guidance for the direction of future explorations to refine and expand these new ML-DFT models.

2. Density Functional Theory.

2.1. The Many-Body Schrödinger Equation. The foundation of quantum mechanics, and the starting point for any *ab initio* calculation or simulation of a material, is the many-body Schrödinger equation. One well-founded and widely applied approximation in nonrelativistic quantum mechanics is to assume that, in a system composed of N_i (ionic) nuclei and N_e electrons, the light electrons equilibrate on a time scale much faster than movements of the much heavier nuclei, therefore the nuclei may be considered fixed when solving for the electronic structure, which is called the Born-Oppenheimer approximation [1]. In this approximation, for a system with electronic positions $\underline{\boldsymbol{r}} = \{\boldsymbol{r}_1, \dots \boldsymbol{r}_{N_e}\}$ and nuclear positions $\underline{\boldsymbol{R}} = \{\boldsymbol{R}_1, \dots \boldsymbol{R}_{N_i}\}$, the many-body Schrödinger equation governing the many-electron wavefunction $\Psi(\underline{\boldsymbol{r}})$ is:

$$\left[\sum_{i=1}^{N_e}\left(-\frac{1}{2}\nabla^2\right) + \sum_{i=1}^{N_e} V(\boldsymbol{r}_i; \underline{\boldsymbol{R}}) + \sum_{i=1}^{N_e}\sum_{j=1}^{N_e}\frac{1}{2}U(\boldsymbol{r}_i, \boldsymbol{r}_j)\right]\Psi(\underline{\boldsymbol{r}}) = E(\underline{\boldsymbol{R}})\Psi(\underline{\boldsymbol{r}}), \tag{2.1}$$

where $V(\boldsymbol{r}_i; \underline{\boldsymbol{R}})$ represents the collective Coulombic potential felt by electron i due to the nuclei, $U(\boldsymbol{r}_i, \boldsymbol{r}_j)$ represents the Coulombic potential between electrons i and j, and $E(\underline{\boldsymbol{R}})$ represents the total energy. Atomic units have been used here such that $\hbar = m_e = e^2 = 1$.

2.2. Kohn-Sham Density Functional Theory. The solution of the many-body Schrödinger equation is complicated by the pairwise interaction term $U(\boldsymbol{r}_i, \boldsymbol{r}_j)$, which necessitates working with the very high dimensional wavefunction $\Psi(\underline{\boldsymbol{r}})$. For the purpose of *ab initio* MD simulations, the total energy is desired to compute forces on the atomic nuclei, where the force on the nucleus α is: $F_\alpha = -\partial E(\underline{\boldsymbol{R}})/\partial \boldsymbol{R}_\alpha$. However, the first Hohenberg-Kohn theorem [13] states that the total energy is a unique functional $E[n]$ of the electronic density $n(\boldsymbol{r}; \underline{\boldsymbol{R}})$. The electronic density is a much lower dimensional function, as it varies only with spatial position $\boldsymbol{r}$ and not with the position of all electrons $\underline{\boldsymbol{r}}$ (note the distinction in notation here). By making use of this theorem, the Schrödinger equation may be reformulated as:

$$\left[-\frac{1}{2}\nabla^2 + V_s(\boldsymbol{r}; \underline{\boldsymbol{R}})\right] \phi_i(\boldsymbol{r}; \underline{\boldsymbol{R}}) = \epsilon_i \phi_i(\boldsymbol{r}; \underline{\boldsymbol{R}}). \tag{2.2}$$

This is called the Kohn-Sham equation, which is the basis for Kohn-Sham DFT (often simply called DFT, as herein). The Kohn-Sham equation resembles the single-electron Schrödinger equation governing the Kohn-Sham orbitals ϕ_i (with corresponding energy eigenvalue ϵ_i) in an effective potential:

$$V_s(\boldsymbol{r}; \underline{\boldsymbol{R}}) = \frac{\partial U[n]}{\partial n} + \frac{\partial E_{XC}[n]}{\partial n} + V(\boldsymbol{r}; \underline{\boldsymbol{R}}). \tag{2.3}$$

All derivatives in this equation represent functional derivatives, with the functional $U[n]$ representing the known electron interaction functional, and $E_{XC}[n]$ representing the exchange-correlation functional. The latter functional is now known precisely, but is approximated in practice [5, 12, 15, 23]. The Kohn-Sham DFT formalism is finalized by defining the electronic density in terms of the Kohn-Sham orbitals and energies:

$$n(\boldsymbol{r}; \underline{\boldsymbol{R}}) = \sum_j f^\beta(\epsilon_j) |\psi_i(\boldsymbol{r}; \underline{\boldsymbol{R}})|^2, \tag{2.4}$$

with $f^\beta(\epsilon)$ representing the Fermi-Dirac distribution at temperature $\beta = (k_B T)^{-1}$.

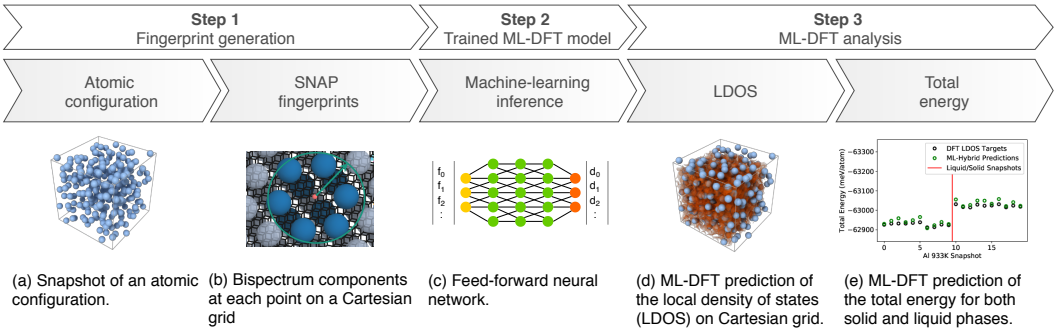
Now, the Kohn-Sham equation may be solved independently for each orbital ϕ_i, representing a much more tractable problem for a many-body system. However, the eigenvalue problem still scales with the cube of the system size N_e. Further, we notice that the effective potential $V_s(\boldsymbol{r}; \underline{\boldsymbol{R}})$ actually depends on the electronic density n via functional derivatives. Therefore, the Kohn-Sham equations must be solved by an iterative, variational approach to converge to a self-consistent solution. Thus, while DFT makes many-body problems computationally feasible for moderate system sizes, its cost still becomes prohibitive for larger systems.

Once the electronic density is known, it may be used to compute the total energy and subsequently the forces on nuclei for an MD time step. Here, we instead use a related quantity that may be computed from the electronic density, called the local density of states (LDOS):

$$D(\epsilon, \boldsymbol{r}; \underline{\boldsymbol{R}}) = \sum_i |\phi_i(\boldsymbol{r}; \underline{\boldsymbol{R}})|^2 \delta(\epsilon - \epsilon_i), \tag{2.5}$$

where $\delta(.)$ represents the Dirac delta function. Using the LDOS, the total energy functional $E[D](\underline{\boldsymbol{R}})$ takes a compact form:

$$E[D] = E_b[D] - S_s[D]/\beta - U[D] + E_{XC}[D] - V_{XC}[D] + V^{ii}(\underline{\boldsymbol{R}}). \tag{2.6}$$



(a) Snapshot of an atomic configuration. (b) Bispectrum components at each point on a Cartesian grid (c) Feed-forward neural network. (d) ML-DFT prediction of the local density of states (LDOS) on Cartesian grid. (e) ML-DFT prediction of the total energy for both solid and liquid phases.

Fig. 2.1: Summary of the ML-DFT pipeline. (a) Snapshots of *ab initio* MD simulations are taken to obtain feasible atomic configurations. (b) Local atomic descriptors of the snapshot at points on a Cartesian grid are computed. (c) and (d) These fingerprints are fed through an artificial neural network to compute the LDOS on the Cartesian grid. (e) The LDOS may be used to compute a number of material properties, including the total energy of the material. Taken with permission from [9].

The exchange-correlation potential $V_{XC}[D]$ requires a suitable approximation consistent with $E_{XC}[D]$. The remaining functionals, including the band energy $E_b[D]$ and Kohn-Sham entropy $S_s[D]$, have known forms (which are not repeated here for brevity), and $V^ii(\underline{\boldsymbol{R}})$ represents the pairwise Coulombic potential between the nuclei.

2.3. Machine Learning Density Functional Theory. Having developed the framework for traditional Kohn-Sham DFT in the previous section, we now turn to the development of ML-DFT, where an artificial neural network replaces the solution of the Kohn-Sham equations. Using the LDOS as the output quantity from which the total energy and other properties are computed, the solution to the Kohn-Sham equation represents a map from the space of atomic configurations, which are defined by the atomic number Z_i and position $\boldsymbol{R}_i$ of the ith atomic nuclei, to the LDOS: $\{Z_i, \boldsymbol{R}_i\} \to D(\epsilon, \boldsymbol{r}; \underline{\boldsymbol{R}})$. In principle, an artificial neural network may be trained to approximate this map directly, with the LDOS as its output and the set of $\{Z_i, \boldsymbol{R}_i\}$ as its input. However, in this context the space of inputs is very broad and high-dimensional, and it would be nearly impossible to populate it with enough training data to properly condition the neural network. Moreover, the input dimensionality would change with the system size. Similarly, the output represents the distribution of the LDOS over all space, which suffers from the same high-dimensionality and variability as the input domain.

Instead, previously developed ML-DFT approaches separate this map into two distinct components [9]. First, the local environment at a coordinate $\boldsymbol{r}$, defined by $\{Z_i, \boldsymbol{R}_i\}$, is converted to a local atomic descriptor, or *fingerprint*, denoted $\boldsymbol{B}(\boldsymbol{r}; \underline{\boldsymbol{R}})$. In this work and in others, the atomic descriptor is selected to be the SNAP bispectrum components [24]. The artificial neural network is then tasked from mapping the local descriptors at $\boldsymbol{r}$ to the LDOS at $\boldsymbol{r}$. Therefore, the conversion of the atomic configuration to the LDOS proceeds as follows: $\{Z_i, \boldsymbol{R}_i\} \to \boldsymbol{B}(\boldsymbol{r}; \underline{\boldsymbol{R}}) \to D(\epsilon, \boldsymbol{r}; \underline{\boldsymbol{R}})$. This ML-DFT pipeline is depicted in Figure 2.1. Because the neural network takes as input a compact descriptor of fixed length located at the single spatial coordinate $\boldsymbol{r}$ and outputs a suitable discretization in ϵ-space of the LDOS at the same coordinate $\boldsymbol{r}$, it needs only be trained on a set of $(\boldsymbol{B}, D)$ data, each of which may be taken at different spatial coordinates. The dimensionality of the input and

output spaces are therefore reduced to 91 bispectrum components and 250 energy levels of the LDOS, respectively, at each of the grid points taken from a snapshot of a molecular simulation. The distribution of LDOS throughout space may be obtained by applying the same trained neural network at every grid point in a simulation domain. For all snapshots in this study, the simulation box is discretized into a $200 \times 200 \times 200$ grid, with eight million total grid points.

3. Machine Learning Methods.

3.1. Fully-Connected Feedforward Neural Networks. In some previous work investigating ML-DFT, the machine learning surrogate has been composed of a single, fully-connected feedforward neural network. This simple architecture may be defined by a sequence of linear transformations and nonlinear *activations*, the pair of which comprises a *layer*. A single layer is described by the following equation:

$$\boldsymbol{v}^{l+1} = a(\boldsymbol{W}^l \boldsymbol{v}^l + \boldsymbol{b}^l), \tag{3.1}$$

where $\boldsymbol{v}^l$ is a $(n_l \times 1)$ vector representation of the data at layer l, $\boldsymbol{W}^l$ is an $(n_{l+1} \times n_l)$ matrix representing the *weights* of the neural network at layer l, and $\boldsymbol{b}^l$ is an $(n_{l+1} \times 1)$ vector representing the *biases* of the neural network at layer l. The function $a(.)$ is the activation, which (typically) acts element-wise on its vector argument. For the ML-DFT pipeline, at layer 0, $\boldsymbol{v}^0 = \boldsymbol{B}(\boldsymbol{r}; \underline{\boldsymbol{R}})$, and at layer L, $\boldsymbol{v}^L = \hat{D}(\boldsymbol{\epsilon}, \boldsymbol{r}; \underline{\boldsymbol{R}})$ (the latter representing a vector of predicted LDOS values at particular energy levels $\boldsymbol{\epsilon}$ at the coordinate $\boldsymbol{r}$). In other works, a bi-directional recurrent neural network is appended to the feedforward network to smooth the predicted LDOS; however, in this work we investigate feedforward networks only.

This feedforward neural network, like all networks in this work, is trained iteratively using a stochastic optimizer. The goal of such an optimizer is to minimize the mean squared loss between the true and predicted LDOS values:

$$L(\hat{D}, D; \boldsymbol{W}, \boldsymbol{b}) = \sum_{i=1}^{N_{train}} \sum_{j=1}^{N_{levels}} [\hat{D}_i(\epsilon_j) - D_i(\epsilon_j)]^2, \tag{3.2}$$

where D_i represents the LDOS sampled from a particular coordinate of with a particular global atomic configuration, of which there are N_{train} examples. Note that the LDOS here has been discretized to N_{levels} energy levels. Optimization proceeds by computing the gradient of this loss function with respect to the network weights and biases, $\boldsymbol{W}$ and $\boldsymbol{b}$, with the goal of finding the weights and biases that minimize the loss:

$$\hat{\boldsymbol{W}}, \hat{\boldsymbol{b}} = \underset{\boldsymbol{W}, \boldsymbol{b}}{\operatorname{argmin}} L(\hat{D}, D; \boldsymbol{W}, \boldsymbol{b}). \tag{3.3}$$

However, N_{train} is typically quite large, and therefore the gradient computations may be expensive. Therefore, the gradient is approximate by Monte Carlo sampling, with random batches of N_{batch} data points taken at a time to compute the gradient. In this work, we employ the ADAM optimizer with $N_{batch} = 1000$ to train these neural networks.

3.2. Transfer Learning with Variational Information Bottlenecks. One limitation of the previous ML-DFT investigations is that each model has been trained and applied on systems of a single atomic species at a single temperature. In practice, however, there may be many atomic species and temperatures of interest. Retraining new networks for each species or temperature requires a substantial amount of work, including the time

needed to generate large amounts of training data, train neural networks, and run independent hyperparameter optimization studies (to determine, for example, the network size best suited for different systems). One method to reduce or eliminate this limitation is called *transfer learning*, which aims to train models that perform well in circumstances where the model application is slightly different than the training application [21]. In the case of ML-DFT, this difference may come in the form of slightly different input domains at different temperatures, or more vastly different domains in the case of new atomic species.

Transfer learning does not always require special model architectures. In fact, previous ML-DFT models trained on data at a single temperature may be applied to new temperature data, although with limited success. However, recent approaches have found that statistically motivated network architectures and training routines may enhance transfer performance of models. One approach that we will apply here is called the *Deep Variational Information Bottleneck* (Deep VIB), which seeks to learn a model that makes predictions based off of features of the input data that are transferable across different domains, while ignoring features of the input data that are specific to certain domains, so that the model may more robustly generalize to new data domains [3].

The Deep VIB is achieved by passing the input data X through a "bottleneck", or a latent representation Z, where the domain-specific details of X are discarded [26]. Then, the outputs Y are predicted directly from Z. The dimensionality of Z is typically chosen to be smaller than that of X to accomplish a certain minimum compression; here, we choose Z to be a 32-dimensional vector. The objective of the Deep VIB is to maximize the shared information $I(Z, Y; \boldsymbol{\theta})$ between the representation Z and the output Y, while minimizing the information $I(Z, X; \boldsymbol{\theta})$ between the representation Z and the input X, given model parameters $\boldsymbol{\theta}$:

$$R_{IB}(\boldsymbol{\theta}) = I(Z, Y; \boldsymbol{\theta}) - \beta I(Z, X; \boldsymbol{\theta}) \tag{3.4}$$

which may be achieved if predictions are made by using only the minimum amount of necessary information (that which transfers between domains). Using Bayesian statistics, we may write the optimization in terms of likelihoods and prior beliefs. In particular, $I(Z, Y, \boldsymbol{\theta})$ may be related to the negative log-likelihood of observing true output given the prediction, which in the case of regression is proportional to the mean-squared error under the assumption of uniform variance in the outputs. The penalty term $I(Z, X; \boldsymbol{\theta})$ is related to the Kullback-Leibler between the observed distribution of latents given a data point x_n, $p(Z|x_n)$, and a prior expectation of this distribution $r(Z)$. Thus, the Deep VIB objective for ML-DFT is to minimize:

$$J(\hat{D}, D; \boldsymbol{W}, \boldsymbol{b}) = L(\hat{D}, D; \boldsymbol{W}, \boldsymbol{b}) + \sum_{i=1}^{N_{train}} \beta KL[p(Z|\boldsymbol{B}_i), r(Z)]. \tag{3.5}$$

Because the Deep VIB is statistical in nature, it must be treated with a slightly different architecture than the standard fully-connected feedforward network. In particular, we apply the "reparameterization trick" to train the Deep VIB [16]. In this architecture, an "encoder" portion of the network predicts the mean vector $\mu(Z|x_i)$ and variance vector $\sigma^2(Z|x_i)$ of a multi-dimensional normal distribution defining $p(Z|x_i) \sim N(\mu, \sigma^2)$. Then, a vector is drawn at random from this distribution, which is fed to a standard feedforward neural network to predict the output y_i. Through this random number generation, we perform Monte Carlo estimation to obtain an estimate of the gradient, and may train the network in the same way as for other feedforward networks.

3.3. Training Data Generation. The training and testing data used in this work was generated for solid aluminum (Al) at temperatures of 298 K and 933 K, with ambient

density (2.699 g/cm^3). The simulation box consists of 256 atoms of Al. To obtain viable atomic configurations, *ab initio* MD simulations were performed using the Vienna Ab initio Simulation Package (VASP) [18]. After the atoms equilibrate, their positions at later simulation time steps are recorded as snapshots for training data generation. The simulation box for a single snapshot is divided into $200 \times 200 \times 200$ grid points (8,000,000 total points). Local SNAP descriptors at each grid point are computed using the LAMMPS software [22], and the LDOS at each grid point is computed using Quantum Espresso with an $8 \times 8 \times 8$ grid spacing in k-space [11]. For SNAP descriptors, the bispectrum components were cut off such that there were 91 scalar components per descriptor (i.e. at each grid point). For the LDOS, the energy line was discretized with a spacing of 0.1 eV between -10 eV and 15 eV, for a total of 250 scalar value per grid point. For a single fingerprint, the 8,000,000 data points were divided into a training set of 5,000,000 data points, a validation set of 1,000,000 data points, and a test set of 2,000,000 data points. Each of the 91 SNAP components in the training data was standardized by the mean and variance of the component in the training data set, and the entire LDOS training data set was normalized by the maximum observed value over all energy levels. Data used in this report is taken with permissions from [9].

3.4. Architectures. For the baseline feedforward, fully-connected neural network (FCNN), we use a total of $L = 5$ layers, with [4000, 4000, 4000, 250, 250] channels. The first four of these layers represent *hidden* layers, and the last represents the output layer, or the predicted LDOS. LeakyReLU activations were applied after each layer, with a linear activation on the final layer.

For the Deep VIB network, we divide the architecture into three *blocks*: an encoder, a decoder, and a dense block. The encoder is responsible for converting the input fingerprints to their latent representation z, by virtue of predicting the mean μ and variance σ^2 of $p(Z, x_i)$. This block consists of four fully-connected layers, with [128, 64, 64, 32] channels (the last representing the dimensionality of z). The decoder simply decompressed this representation, with three layers of size [64, 128, 4000]. Finally the dense block acts similarly to the FCNN, to predict the LDOS from the decompressed representation, with four layers of size [4000, 4000, 250, 250]. To train the network variationally, in a training forward pass, a random vector z is drawn from $N(\mu, \sigma^2)$ (with μ and σ^2 output from the encoder), and z is fed to the decoder and dense blocks to predict the LDOS. LeakyReLU activations are applied after each layer, with the exception of the network output and the encoder output, which are purely linear.

4. Results.

4.1. Fully-Connected Neural Network. We first test the baseline performance of the FCNN trained separately on the 5,000,000 training grid points from a single snapshot at either 298 K or 933 K. This provides some indication of how well previous ML-DFT approaches would transfer to unseen conditions. For both temperatures, the FCNN adequately fits the training data with low relative error, as depicted in the parity plots in Figure 4.1 and presented in Table 4.1. The low error is mirrored in the 2,000,000 test grid points from the same snapshot as the training data. This realization may already represent an advance over current ML-DFT approaches, which use distinct network architectures (i.e. different number of hidden channels) to fit data at these two temperatures. Having a single network architecture capable of sufficiently fitting data at different temperatures may substantially reduce the time needed to train an ML-DFT model, as it eliminates the need for expensive calculations such as architecture searches and hyperparameter optimization over the network size.

To assess the baseline generalization and transfer performance of the FCNN model

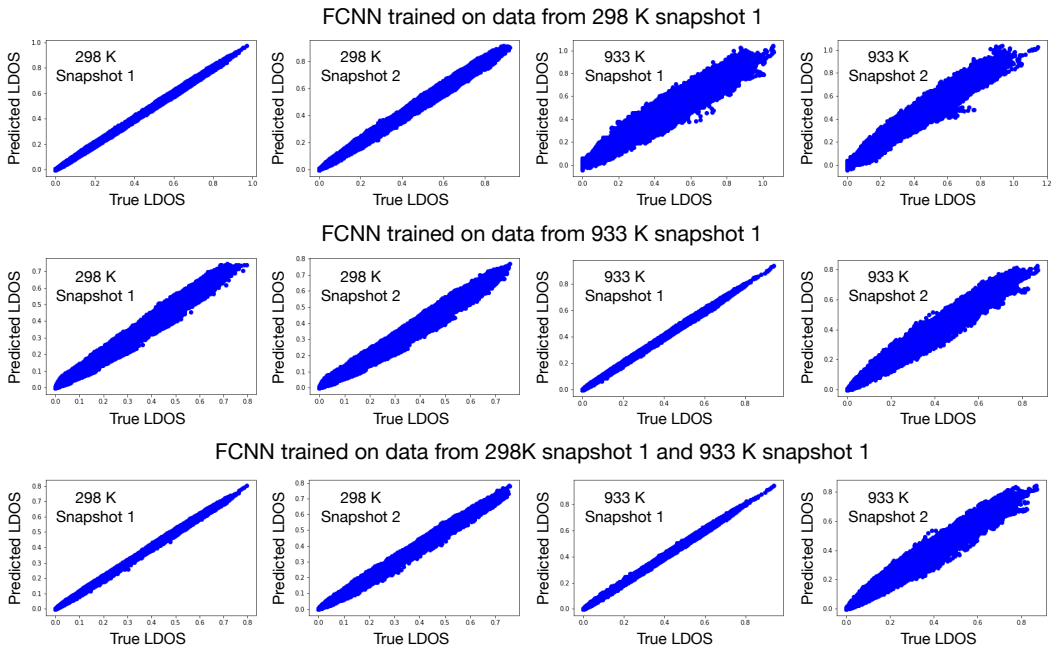


Fig. 4.1: Parity plots of the test performance of the FCNN model in predicting the LDOS. From top to bottom: the model trained only on data from snapshot 1 at 298 K, the model trained only on data from snapshot 1 at 933 K, and the model trained on snapshot 1 at 298 K and snapshot 1 at 933 K. From left to right: model performance on the test set from snapshot 1 at 298 K, performance on the test set from snapshot 2 at 298 K, performance on the test set from snapshot 1 at 933 K, and performance on the test set from snapshot 2 at 933 K. The LDOS has been normalized to the maximum value in the training set in each case.

Architecture	Standard FCNN			Deep VIB		
Training Data	298 K	933 K	Both	298 K	933 K	Both
298 K Snapshot 1	1.47	11.7	1.29	5.65	10.5	1.12
298 K Snapshot 2	6.34	11.8	4.39	5.53	10.4	5.10
933 K Snapshot 1	33.8	1.17	1.52	30.2	8.77	1.32
933 K Snapshot 2	42.5	18.7	19.2	37.7	19.2	20.0

Table 4.1: The mean squared error test performance of different ML-DFT models, over test data sets from different snapshots and different temperatures. All values are the true mean squared error of the max-normalized LDOS multiplied by a factor of 10^5.

trained separately at each temperature, we perform two tests. In the first, we deploy the model on a test set of grid points taken from a different snapshot, but one still at the same temperature as the training data. Different snapshots at the same temperature represent a slight change in the input domain, but one much smaller than the change in moving to a different temperature. The second test, therefore, is to deploy the model on a test set of points from a snapshot taken at a different temperature (at 298 K in the case of the model

trained on 933 K data, and vice versa). This represents a more difficult transfer task, as the domains for different temperatures may be quite different.

The model trained on 298 K data appears to transfer with only slight additional error to the different snapshot at 298 K. The transfer performance of the model trained on 933 K to the different 933 K snapshot is slightly worse, as shown in Figure 4.1 and Table 4.1. This observation is consistent with those in previous ML-DFT works, which demonstrated the need for additional 933 K training data to obtain a model that produces high-fidelity predictions for new snapshots, compared to the ability to obtain such a model at 298 K from a more limited data set. The difference in transfer performance here is most likely due to the increased number of states accessed at higher temperature due to increased thermal fluctuations, which results in a more varied domain of local configurations and LDOS.

When testing the transfer performance of the models on snapshots from the other temperature, however, the opposite trend is observed. Specifically, the 298 K trained model performs worse on the 933 K data than on the 298 K data from unseen snapshots, while the 933 K trained model performs similarly on the 298 K data and the 933 K data from unseen snapshots. This suggests that the states explored by the 298 K simulation are a subset of the states explored by the 933 K simulation. In this case, the 933 K training data would span the domain of the 298 K training data, and therefore predictions at 298 K would require interpolation in the training data space, leading to predictions similar to those for unseen 933 K data. Conversely, for the model trained at 298 K, predictions at 933 K require extrapolation from the training domain, producing lower quality predictions.

Next, we explore whether the model performance is substantially affected by training on 298 K and 933 K data, simultaneously. We observe that this model maintains the best-case performance compared to either of the models trained separately on the two temperatures, as evident in Figure 4.1 and Table 4.1. In particular, the model accurately predicts the LDOS for the test set held out from the training snapshots at both temperatures. The model transfers well to a new snapshot at 298 K, and transfers with some added error to a new snapshot at 933 K. However, in both cases, the model performs no worse than either model trained specifically on a single temperature. This suggests firstly that the information capacity of the network architecture is enough to accommodate two slightly different data domains without sacrificing performance, and secondly that it is possible to form a single map from local descriptors to LDOS that holds across different temperatures. Neither observation was obvious *a priori*, and this realization represents an advance in ML-DFT studies, at it is the first observation of a model that is, at least partially, portable between temperatures.

4.2. Deep Variational Information Bottleneck. We test the transferability of the Deep VIB architecture in the same way as the FCNN: first by training the network separately on each temperature, and next by training it simultaneously on both. Here, we demonstrate results for $\beta = 10^{-6}$, a value selected based on previous studies with Deep VIBs, which is found to balance the prediction error with the information penalty in the latent domain. In all cases, the performance and trends, both on tests sets from training snapshots, and transferred to other snapshots of the same or different temperatures, are very similar to those observed for the FCNN, as evident in Figure 4.2 and Table 4.1.

The similarity between the performance of the Deep VIB and the FCNN suggests a few different conclusions. Firstly, it is notable that the Deep VIB does not exhibit enhanced transfer performance to a new snapshot at 933 K. The principle of the Deep VIB is that it should reduce learning a predictor that is specific to a single domain by finding the features of the input data which generalize well. However, it is still possible that the training data set does not span this domain, meaning that the network may still have to extrapolate from

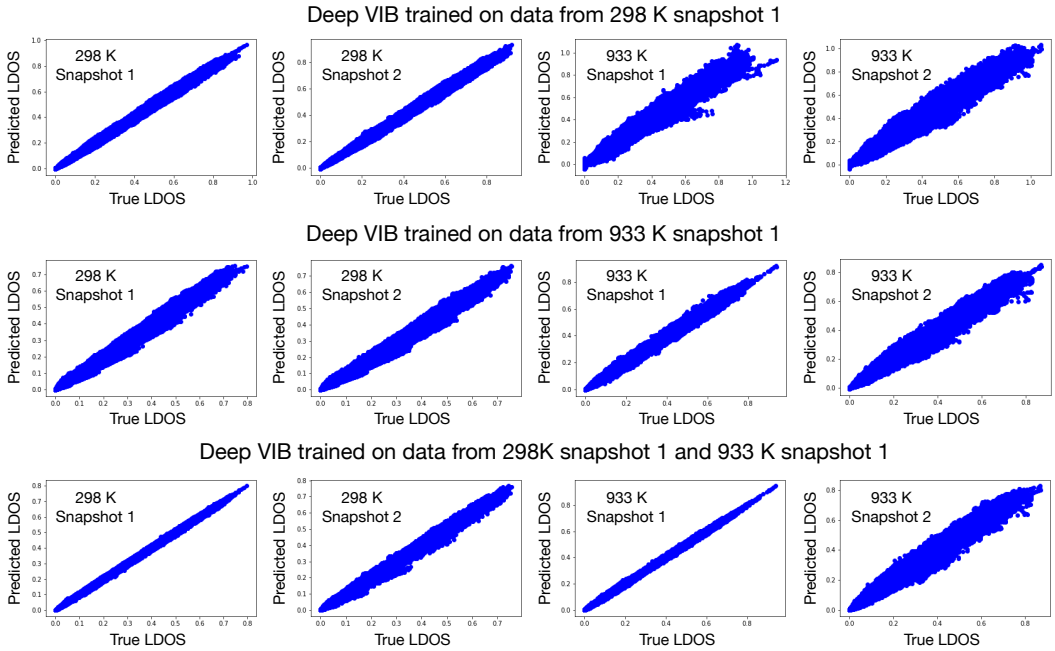


Fig. 4.2: Parity plots of the test performance of the Deep VIB model in predicting the LDOS. From top to bottom: the model trained only on data from snapshot 1 at 298 K, the model trained only on data from snapshot 1 at 933 K, and the model trained on snapshot 1 at 298 K and snapshot 1 at 933 K. From left to right: model performance on the test set from snapshot 1 at 298 K, performance on the test set from snapshot 2 at 298 K, performance on the test set from snapshot 1 at 933 K, and performance on the test set from snapshot 2 at 933 K. The LDOS has been normalized to the maximum value in the training set in each case.

its training domain to make predictions for new domains, such as another high-temperature snapshot, despite the fact that it is using feature representations that may generalize well. Therefore, it seems that there is indeed still a need for more data at higher temperatures to train a model whose applicable domain spans that which may be observed at these temperatures.

The second conclusion suggested by these results is that there are features of the input SNAP fingerprints that are not useful for computing the LDOS. The Deep VIB is able to compress the 91-dimension input fingerprint to a 32-dimension latent vector by architectural design, nearly a third of the size. Moreover, we observe that the Kullback-Leibler divergence for this latent space is, on average, ten times lower with $\beta = 10^{-6}$ than with $\beta = 0$, suggesting further compression of information has been achieved. Therefore, there is a representation of the input domain with substantially less information than the SNAP fingerprints that is still sufficient to compute LDOS predictions without substantially more error. Similar results may be observed by analyzing the variance in the LDOS explained by the principal components of the bispectrum fingerprints. Indeed, the substantial compression observed in the Deep VIB suggests that a small number of principal components may be sufficient to form a predictive map from fingerprints to LDOS. This observation also suggests that, when presented with more training data, the Deep VIB may indeed transfer

to new conditions better than the FCNN, as the latent space will be more densely populated with training data than the space of SNAP fingerprints. Moreover, it suggests that the information capacity of the FCNN architecture is not fully utilized, and indeed the network may be able to accommodate much more training data without observing diminished performance due to information overload.

5. Discussion. This report has detailed the first steps towards developing ML-DFT models that transfer between different data domains, in particular, between temperatures, with no or limited additional training. The results presented in the previous section indicate that it is possible to train the same network architectures previously used on single-temperature data to fit data at two different temperatures, without substantial loss of performance at either temperature. Moreover, the Deep VIB approach suggests that it is possible to compress the input domain into a more compact, lower-dimensional domain, such that it may be densely populated by less training data than the full SNAP fingerprint domain. Together, these results suggest a pathway towards training transferable neural network surrogates for DFT calculations.

However, substantially more work is needed to assess the viability of these approaches on more interesting cases, such as interpolation in temperature, where the test snapshot is taken at a temperature between those present in the training data. Moreover, additional training data is required to achieve transferable performance between snapshots at the training temperatures. These studies are primarily limited by the availability of training data. We plan to generate such data and explore these features of the models in upcoming work.

The focus of this report has been on transfer learning approaches for ML-DFT, which aim to apply a trained model to a slightly different problem domain. However, other approaches exist which may be able to achieve better performance at the cost of some slight additional training, but not nearly to the extent of retraining a model for each domain of interest. This is the domain of meta-learning, where the goal is to develop a model that is able to rapidly adapt to new problems, or tasks. In future work, we plan to explore a two-stage meta-learning approach, in which a low-fidelity but general Deep VIB model is trained on snapshots from a range of temperatures, and the learned embeddings are used to train a second, smaller model that predicts the LDOS, which is temperature specific. Similar approaches have demonstrated success in few-shot, fine-grained image classification, dramatically reducing the amount of training data and the training time to achieve a performant model on difficult classification tasks [25]. This highlights the main trade-off inherent in these transfer and meta-learning ML-DFT approaches: an increased flexibility in predicting data from different states with limited additional training, at the cost of a more complex model with larger upfront training cost.

6. Conclusions. We have discussed the recently-developed paradigm of ML-DFT, including its limitations on rapidly adapting to unseen input domains, such as data from different temperatures or chemical species. A transfer-learning approach to creating such a general model was examined, with preliminary results indicating that current ML-DFT model architectures are capable of describing more than one temperature. This analysis also revealed that transference from models trained at high temperature to lower-temperature test data is more accurate than that in the opposite direction, indicating the need specifically for a sufficient amount of high-temperature training data. An exploration of modern variational approaches to assist in transfer learning, specifically the Deep VIB, revealed that substantially less information than that stored in the full 91-dimensional SNAP fingerprint is required to achieve accurate predictions of the LDOS. However, more data is required to examine whether this approach outperforms the standard FCNN in transfer performance.

Finally, we briefly introduced the concept of meta-learning, which opens future studies into rapidly adaptable, but highly accurate, ML-DFT models. The development of such models will be critical in the rapid adaptation of scalable electronic structure calculations for a variety of applications, as the reduced need for training data and the low time to train substantially mitigates two of the largest barriers in the wide-spread accessibility of the ML-DFT approach.

References.

[1] A. Abedi, N. T. Maitra, and E. K. U. Gross, *Correlated electron-nuclear dynamics: Exact factorization of the molecular wavefunction*, The Journal of Chemical Physics, 137 (2012), p. 22A530.

[2] B. J. Alder and T. E. Wainwright, *Studies in molecular dynamics. i. general method*, The Journal of Chemical Physics, 31 (1959), pp. 459–466.

[3] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy, *Deep variational information bottleneck*, 2019.

[4] A. D. Becke, *Perspective: Fifty years of density-functional theory in chemical physics*, The Journal of Chemical Physics, 140 (2014), p. 18A301.

[5] E. W. Brown, J. L. DuBois, M. Holzmann, and D. M. Ceperley, *Exchange-correlation energy for the three-dimensional homogeneous electron gas at arbitrary temperature*, Phys. Rev. B, 88 (2013), p. 081102.

[6] K. Burke, *Perspective on density functional theory*, The Journal of Chemical Physics, 136 (2012), p. 150901.

[7] A. Chandrasekaran, D. Kamal, R. Batra, C. Kim, L. Chen, and R. Ramprasad, *Solving the electronic structure problem with machine learning*, npj Computational Materials, 5 (2019), p. 22.

[8] N. R. Council, *Integrated Computational Materials Engineering: A Transformational Discipline for Improved Competitiveness and National Security*, The National Academies Press, Washington, DC, 2008.

[9] J. A. Ellis, L. Fiedler, G. A. Popoola, N. A. Modine, J. A. Stephens, A. P. Thompson, A. Cangi, and S. Rajamanickam, *Accelerating finite-temperature kohn-sham density functional theory with deep neural networks*, Phys. Rev. B, 104 (2021), p. 035120.

[10] M. Finnis, *Interatomic forces in condensed matter*, vol. 1, OUP Oxford, 2003.

[11] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, *QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials*, Journal of Physics: Condensed Matter, 21 (2009), p. 395502.

[12] S. Groth, T. Dornheim, T. Sjostrom, F. D. Malone, W. M. C. Foulkes, and M. Bonitz, *Ab initio exchange-correlation free energy of the uniform electron gas at warm dense matter conditions*, Phys. Rev. Lett., 119 (2017), p. 135001.

[13] P. Hohenberg and W. Kohn, *Inhomogeneous electron gas*, Phys. Rev., 136 (1964), pp. B864–B871.

[14] M. F. Horstemeyer, *Multiscale Modeling: A Review*, Springer Netherlands, Dordrecht, 2010, pp. 87–135.

[15] V. V. Karasiev, T. Sjostrom, J. Dufty, and S. B. Trickey, *Accurate homogeneous electron gas exchange-correlation free energy for local spin-density calculations*, Phys. Rev. Lett., 112 (2014), p. 076403.

[16] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2014.

[17] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev., 140 (1965), pp. A1133–A1138.

[18] G. Kresse and J. Hafner, *Ab initio molecular dynamics for liquid metals*, Phys. Rev. B, 47 (1993), pp. 558–561.

[19] K. Lejaeghere, G. Bihlmayer, T. Björkman, P. Blaha, S. Blügel, V. Blum, D. Caliste, I. E. Castelli, S. J. Clark, A. Dal Corso, S. de Gironcoli, T. Deutsch, J. K. Dewhurst, I. Di Marco, C. Draxl, M. Dułak, O. Eriksson, J. A. Flores-Livas, K. F. Garrity, L. Genovese, P. Giannozzi, M. Giantomassi, S. Goedecker, X. Gonze, O. Grånäs, E. K. U. Gross, A. Gulans, F. Gygi, D. R. Hamann, P. J. Hasnip, N. A. W. Holzwarth, D. Iuşan, D. B. Jochym, F. Jollet, D. Jones, G. Kresse, K. Koepernik, E. Küçükbenli, Y. O. Kvashnin, I. L. M. Locht, S. Lubeck, M. Marsman, N. Marzari, U. Nitzsche, L. Nordström, T. Ozaki, L. Paulatto, C. J. Pickard, W. Poelmans, M. I. J. Probert, K. Refson, M. Richter, G.-M. Rignanese, S. Saha, M. Scheffler, M. Schlipf, K. Schwarz, S. Sharma, F. Tavazza, P. Thunström, A. Tkatchenko, M. Torrent, D. Vanderbilt, M. J. van Setten, V. Van Speybroeck, J. M. Wills, J. R. Yates,

G.-X. Zhang, and S. Cottenier, *Reproducibility in density functional theory calculations of solids*, Science, 351 (2016), p. aad3000.
[20] N. D. Mermin, *Thermal properties of the inhomogeneous electron gas*, Phys. Rev., 137 (1965), pp. A1441–A1443.
[21] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on Knowledge and Data Engineering, 22 (2010), pp. 1345–1359.
[22] S. Plimpton, *Fast Parallel Algorithms for Short-Range Molecular Dynamics*, Journal of Computational Physics, 117 (1995), pp. 1–19.
[23] A. Pribram-Jones, D. A. Gross, and K. Burke, *Dft: A theory full of holes?*, Annual Review of Physical Chemistry, 66 (2015), pp. 283–304. PMID: 25830374.
[24] A. P. Thompson, L. P. Swiler, C. R. Trott, S. M. Foiles, and G. J. Tucker, *Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials*, Journal of Computational Physics, 285 (2015), pp. 316–330.
[25] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola, *Rethinking few-shot image classification: a good embedding is all you need?*, 2020.
[26] N. Tishby, F. C. Pereira, and W. Bialek, *The information bottleneck method*, arXiv preprint physics/0004057, (2000).
[27] P. F. Weck, K. R. Cochrane, S. Root, J. M. D. Lane, L. Shulenburger, J. H. Carpenter, T. Sjostrom, T. R. Mattsson, and T. J. Vogler, *Shock compression of strongly correlated oxides: A liquid-regime equation of state for cerium(iv) oxide*, Phys. Rev. B, 97 (2018), p. 125106.
[28] M. A. Wood, M. A. Cusentino, B. D. Wirth, and A. P. Thompson, *Data-driven material models for atomistic simulation*, Phys. Rev. B, 99 (2019), p. 184305.

IMPROVED VERTICAL REMAPPING ACCURACY IN THE NH-HOMME ATMOSPHERE DYNAMICAL CORE

JASON L. TORCHINSKY* AND MARK A. TAYLOR†

Abstract. A vertical Lagrangian coordinate has been used in global atmosphere models for nearly two decades and has several advantages over other discretizations, including reducing the dimensionality of the physical problem. As the Lagrangian surfaces deform over time, it is necessary to accurately and conservatively remap them back to fixed Eulerian surfaces. A popular choice of remapping algorithm is the piecewise-parabolic method, a modified version of which is used in the dynamical core of the atmosphere component of the Energy Exascale Earth System Model. However, this version of the remapping algorithm creates unwanted noise at the model top and planetary surface for several standard test cases. We explore four alternative modifications to the algorithm, specifically to the treatment of the domain boundaries, and show that the most accurate of these eliminates this noise.

1. Introduction. The use of a vertical Lagrangian coordinate in global circulation models (GCMs) was first introduced almost eighty years ago by V. P. Starr in [9], and was re-introduced in its modern form less than twenty years ago by S.-J. Lin in [5]. Since then, a vertical Lagrangian coordinate has been a popular choice for the dynamical cores of present-day non-hydrostatic GCMs, including a version of ENDGame used in the Met Office atmospheric Unified Model and the Non-Hydrostatic High Order Method Modeling Environment (NH-HOMME) used in the Energy Exascale Earth System Model (E3SM) [3, 10].

The essence of the vertical Lagrangian coordinate is as follows: we start the time integration process with a terrain-following coordinate surface, and allow each Lagrangian surface to float, compress, or expand as the simulation progresses. After some number of time-steps, we remap each state variable from the deformed Lagrangian surfaces back to the terrain-following vertical Eulerian surfaces to avoid the Lagrangian surfaces from becoming too close or crossing. The key advantage of the vertical Lagrangian coordinate is that there is no flow through the Lagrangian surfaces, and so we may eliminate vertical advection terms from the equations of motion [10]. However, many remapping algorithms introduce excessive vertical dissipation and undesirable errors [2, 6–8, 10].

One common choice of remapping algorithm, and the one we will discuss throughout this work, is the piecewise-parabolic method (PPM) introduced by P. Colella and P. Woodward in [2]. Many have introduced modifications and improvements to the PPM, for example [1, 6–8, 13]. Originally, the PPM was introduced as a numerical method to solve advection problems in a periodic domain and did not propose an alternative algorithm for the case of non-periodic domains. One modification to the PPM to handle non-periodic domains is to instead use one-sided stencils at the domain boundary [6, 13]. Another is to instead extrapolate the known data beyond the domain, and apply the original PPM algorithm utilizing the ghost-cell data at the domain boundaries, such as in [6].

The dynamical core used in the E3SM utilizes this latter strategy of extrapolating ghost-cell data to remap a vertical Lagrangian pressure coordinate back to a terrain-following Eulerian pressure coordinate. Specifically, the original data is extrapolated linearly beyond the domain, the extrapolated data is then adjusted to be bounded by the global bounds of the original data, and finally the original PPM algorithm is applied. This has been shown to create unwanted noise at the domain boundary for several standard test cases introduced by the Dynamical Core Model Intercomparison Project (DCMIP).

*University of Wisconsin-Madison Department of Mathematics, jason.torchinsky@wisc.edu
†Sandia National Laboratories, mataylor@sandia.gov

In this work, we review the discrete remapping problem (Section 2), examine four alternative strategies for treating the extrapolated data (Section 3), analyze their accuracy in a suite of sandbox tests (Section 4), and compare the most accurate of these strategies against the remapping algorithm currently used in NH-HOMME (Section 5).

2. The Discrete Remapping Problem. In the most general sense, the discrete remapping problem may be stated as follows:

> Given information about a function in a discretization of a domain, how may we "nicely" calculate the same information about this function in a *different* discretization of the same domain?

We consider the discrete remapping problem in one spatial dimension. In particular, let the source grid be made up of n cells with centers x_i $(i = 1, \ldots, n)$ each with boundary points $x_{i\pm\frac{1}{2}}$, and the target grid made up of m cells with centers ξ_j $(j = 1, \ldots, m)$ each with boundary points $\xi_{j\pm\frac{1}{2}}$. The source grid and the target grid are discretizations of the same domain, with $x_{\frac{1}{2}} = \xi_{\frac{1}{2}}$ and $x_{n+\frac{1}{2}} = \xi_{m+\frac{1}{2}}$.

We are given the average value $\langle a \rangle_i$ of some function $a(x)$ in each cell of the source grid

$$\langle a \rangle_i = \frac{1}{\Delta x_i} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} a(x)\, dx, \qquad i = 1, \ldots, n, \tag{2.1}$$

where $\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$ is the width of the i^{th} cell on the source grid. We wish to remap the cell averages $\langle a \rangle_i$ of the source grid to the cell averages $\overline{a}_j$ of the target grid

$$\overline{a}_j = \frac{1}{\Delta \xi_j} \int_{\xi_{j-\frac{1}{2}}}^{\xi_{j+\frac{1}{2}}} a(\xi)\, d\xi, \qquad j = 1, \ldots, m, \tag{2.2}$$

where $\Delta \xi_j = \xi_{j+\frac{1}{2}} - \xi_{j-\frac{1}{2}}$ is the width of the j^{th} cell on the target grid. We require that this remapped solution has three properties:

(i) the integral of $a(x)$ over the domain does not change because of the remapping

$$\int_{x_{\frac{1}{2}}}^{x_{n+\frac{1}{2}}} a(x)\, dx = \sum_{i=1}^{n} \Delta x_i \, \langle a \rangle_i = \sum_{j=1}^{m} \Delta \xi_j \, \overline{a}_j = \int_{\xi_{\frac{1}{2}}}^{\xi_{m+\frac{1}{2}}} a(\xi)\, d\xi, \tag{2.3}$$

i.e., the remapping conserves "mass",

(ii) preserves local bounds on the interior of the domain, i.e., for all cells on the target grid away from the domain boundary, the cell average $\overline{a}_j$ lies within the cell averages $\langle a \rangle_i$ used to obtain $\overline{a}_j$, and

(iii) preserves global bounds on the boundary, i.e., for all cells on the target grid near the domain boundary, the cell average $\overline{a}_j$ lies within the cell averages $\langle a \rangle_i$ $(i = 1, \ldots, n)$.

The current algorithm used to accomplish this in NH-HOMME is a modified version of the piecewise-parabolic method (PPM) introduced in [2], which we describe in the following section.

3. A Modified Piecewise-Parabolic Method. The first step of the modified PPM used in NH-HOMME is to generate ghost-cell data on the original grid, in particular the

values $\langle a \rangle_{-1}$, $\langle a \rangle_0$, $\langle a \rangle_{n+1}$, and $\langle a \rangle_{n+2}$ using a simple linear extrapolation

$$\langle a \rangle_{-1} = \left(1 - \frac{\Delta x_{-1} + 2\,\Delta x_0 + 2\,\Delta x_1 + \Delta x_2}{\Delta x_1 + \Delta x_2}\right) \langle a \rangle_2 + \frac{\Delta x_{-1} + 2\,\Delta x_0 + 2\,\Delta x_1 + \Delta x_2}{\Delta x_1 + \Delta x_2} \langle a \rangle_1 , \tag{3.1a}$$

$$\langle a \rangle_0 = \left(1 - \frac{\Delta x_0 + 2\,\Delta x_1 + \Delta x_2}{\Delta x_1 + \Delta x_2}\right) \langle a \rangle_2 + \frac{\Delta x_0 + 2\,\Delta x_1 + \Delta x_2}{\Delta x_1 + \Delta x_2} \langle a \rangle_1 , \tag{3.1b}$$

$$\langle a \rangle_{n+1} = \left(1 - \frac{\Delta x_{n-1} + 2\,\Delta x_n + \Delta x_{n+1}}{\Delta x_{n-1} + \Delta x_n}\right) \langle a \rangle_{n-1} + \frac{\Delta x_{n-1} + 2\,\Delta x_n + \Delta x_{n+1}}{\Delta x_{n-1} + \Delta x_n} \langle a \rangle_n , \tag{3.1c}$$

$$\langle a \rangle_{n+2} = \left(1 - \frac{\Delta x_{n-1} + 2\,\Delta x_n + 2\,\Delta x_{n+1} + \Delta x_{n+2}}{\Delta x_{n-1} + \Delta x_n}\right) \langle a \rangle_{n-1} + \frac{\Delta x_{n-1} + 2\,\Delta x_n + 2\,\Delta x_{n+1} + \Delta x_{n+2}}{\Delta x_{n-1} + \Delta x_n} \langle a \rangle_n , \tag{3.1d}$$

where the cell-widths Δx_{-1}, Δx_0, Δx_{n+1}, and Δx_{n+2} are chosen to mirror those in the domain, e.g., $\Delta x_{-1} = \Delta x_2$, $\Delta x_0 = \Delta x_1$, etc. Making these substitutions, Eqs. 3.1 may be expressed in simpler terms

$$\langle a \rangle_{-1} = \left(1 - \frac{4\,\Delta x_1 + 2\,\Delta x_2}{\Delta x_1 + \Delta x_2}\right) \langle a \rangle_2 + \frac{4\,\Delta x_1 + 2\,\Delta x_2}{\Delta x_1 + \Delta x_2} \langle a \rangle_1 , \tag{3.2a}$$

$$\langle a \rangle_0 = \left(1 - \frac{3\,\Delta x_1 + \Delta x_2}{\Delta x_1 + \Delta x_2}\right) \langle a \rangle_2 + \frac{3\,\Delta x_1 + \Delta x_2}{\Delta x_1 + \Delta x_2} \langle a \rangle_1 , \tag{3.2b}$$

$$\langle a \rangle_{n+1} = \left(1 - \frac{\Delta x_{n-1} + 3\,\Delta x_n}{\Delta x_{n-1} + \Delta x_n}\right) \langle a \rangle_{n-1} + \frac{\Delta x_{n-1} + 3\,\Delta x_n}{\Delta x_{n-1} + \Delta x_n} \langle a \rangle_n , \tag{3.2c}$$

$$\langle a \rangle_{n+2} = \left(1 - \frac{2\,\Delta x_{n-1} + 4\,\Delta x_n}{\Delta x_{n-1} + \Delta x_n}\right) \langle a \rangle_{n-1} + \frac{2\,\Delta x_{n-1} + 4\,\Delta x_n}{\Delta x_{n-1} + \Delta x_n} \langle a \rangle_n . \tag{3.2d}$$

Currently, we then limit these ghost-cell cell averages to obey global bounds of the original data, e.g., if $\langle a \rangle_{n+1}$ is greater than the greatest $\langle a \rangle_i$ then we reduce $\langle a \rangle_{n+1}$ to the greatest $\langle a \rangle_i$.

With the ghost-cell cell averages, we may apply the original PPM of [2] to obtain a piecewise-parabolic reconstruction of $a(x)$ on the source grid that we then integrate to obtain the cell averages $\overline{a}_j$ ($j = 1, \ldots, m$) on the target grid. The original PPM algorithm guarantees that our remapped solution conserves "mass" and preserves local bounds on the interior of the domain. By limiting the ghost-cell cell averages to obey global bounds, we also guarantee that our remapped solution obeys global bounds on the boundary of the domain. We henceforth refer to this strategy as strategy (0).

As we will discuss in Section 5, the limiting of the ghost-cell cell averages introduces undesirable noise at the model top and surface to several DCMIP tests, which are stated in detail in [11,12]. We investigated several alternative strategies to eliminate this noise:

(1) Instead of limiting extrapolated ghost-cell cell averages for all variables, only limit ghost-cell cell averages for some state variables. This relaxes the condition of global bounds preservation on the domain boundary for these variables, leading to less dissipation than strategy (0).
(2) In addition to the changes for strategy (1), we utilize "mass-borrowing" on the remapped solution to guarantee that global bounds are preserved on the domain

boundary. Specifically, we add/subtract "mass" from the remapped cell averages $\overline{a}_j$ on the so that global bounds on the boundary are preserved, and subtract/add this "mass" to neighboring cells on the interior of the domain. This relaxes the condition of local bounds preservation on the interior for these variables.

(3) Use a linear combination of the remapped solution obtained from the original strategy and strategy (1) to guarantee that global bounds are preserved on the domain boundary. This guarantees that the remapped solution satisfies conditions (i)–(iii).

(4) In addition to the changes for strategy (1), we change the parabolic piece of the reconstructions to be piecewise-constant on the outermost cells of the source grid. This guarantees that the remapped solution satisfies conditions (i)–(iii).

4. Sandbox Test Results. To directly compare the five strategies, we remapped various test functions from a non-uniform source grid to a uniform target grid, each with the same number of cells. In particular, these source grids discretized the interval $[0,\ 1]$ and may be obtained from a uniform discretization $u_0,\ \ldots,\ u_n$ via the following mappings

$$g_1(u) = u^2, \tag{4.1a}$$

$$g_2(u) = u^3, \tag{4.1b}$$

$$g_3(u) = \frac{1}{2}\left(1 - \cos(\pi u)\right), \tag{4.1c}$$

$$g_4(u) = \left(1 + e^{-30(x-0.5)}\right)^{-1}, \tag{4.1d}$$

$$g_5(u) = u + \frac{1}{32}\mathcal{R}, \tag{4.1e}$$

$$g_6(u) = u + \frac{1}{4}\mathcal{R}, \tag{4.1f}$$

where $\mathcal{R}$ is a uniform random number from the interval $[-\Delta x_u,\ \Delta x_u]$ (Δx_u being the uniform grid spacing) and we guarantee that $g_4(0) = g_5(0) = g_6(0) = 0$ and $g_4(1) = g_5(1) = g_6(1) = 1$. We chose each of these source grids with the following intent:

- the resolution of the grid discretized via g_1 is biased towards one side of the domain,
- the resolution of the grid discretized via g_2 is even more biased towards one side of the domain than that discretized via g_1,
- the resolution of the grid discretized via g_3 is symmetrically biased toward the domain boundaries,
- the resolution of the grid discretized via g_4 is even more symmetrically biased toward the domain boundaries than that discretized via g_3,
- the grid discretized via g_5 is a small perturbation of a uniform grid which we expect in our use cases, and
- the grid discretized via g_6 is a larger perturbation of a uniform grid than that discretized via g_5.

The test functions we investigated were

$$a_1(x) = 4\left(x - \frac{1}{2}\right)^2, \tag{4.2a}$$

$$a_2(x) = e^{-\frac{1}{2}\left(20\left(x-\frac{1}{2}\right)\right)^2}, \tag{4.2b}$$

$$a_3(x) = e^{x-1}, \tag{4.2c}$$

$$a_4(x) = e^{-x}, \tag{4.2d}$$

$$a_5(x) = \frac{16}{9}\left(x - \frac{1}{4}\right)^2, \tag{4.2e}$$

$$a_6(x) = \frac{1}{2}\left(1 + \sin\left(8\,\pi\,x\right)\right), \tag{4.2f}$$

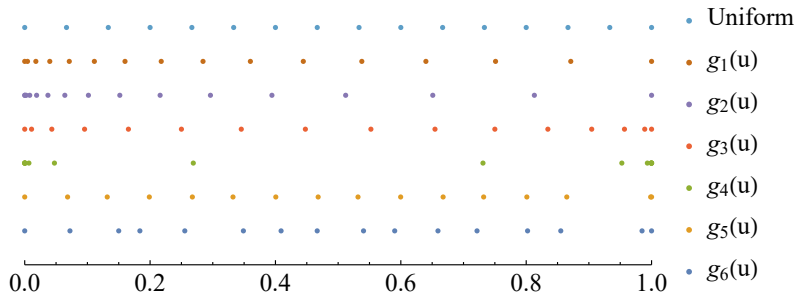
which were intended to examine the following situations:

- the test function a_1 has symmetric steep global maxima at the domain boundaries and a global minimum at the domain center,
- the test function a_2 has a global maximum at the domain center and gradual symmetric global minima at the domain boundaries,
- the test function a_3 has a steep global maximum on the right side of the domain and a gradual global minimum on the left side of the domain,
- the test function a_4 has a steep global maximum on the left side of the domain and a gradual global minimum on the right side of the domain,
- the test function a_5 has asymmetric steep local maxima at the domain boundaries with one (on the right) being the global maximum,
- the test function a_6 is highly oscillatory and has steep local extrema on the domain boundaries and achieves global extrema on the interior of the domain.

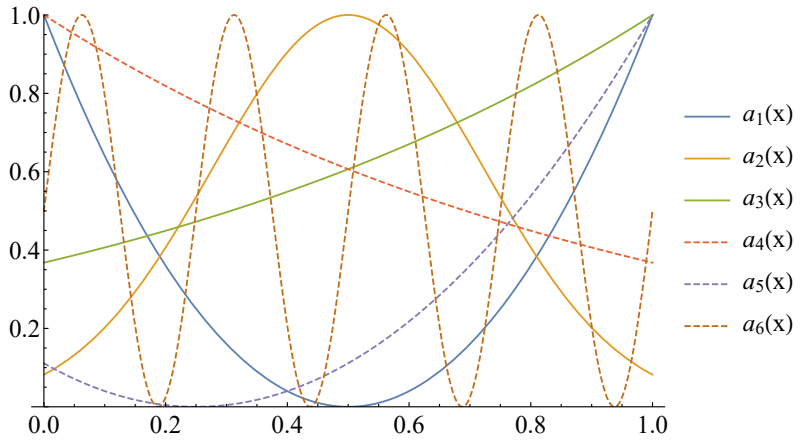
Each source grid and test function is shown in Figure 4.1.

The cell averages are initialized on the source grid by evaluating the true function at the mid-point of each source cell. The 2–norm and max–norm errors are calculated as described in Appendix A.5 of [4], taking the difference between the remapped solution and the true function evaluated at the mid-point of each target cell for domains decomposed into 16, 32, 64, 128, 256, 512, and 1024 cells. Figure 4.2 shows the order of accuracy for each remapping strategy for each combination of grid function and test function.

As may be observed from Figure 4.2, strategy (0) is significantly less accurate than strategy (1) in most test cases, is notably less accurate then strategies (2) and (3) in test cases involving source grid g_5 which we expect in our use cases, and is more accurate than strategy (4) in several test cases.

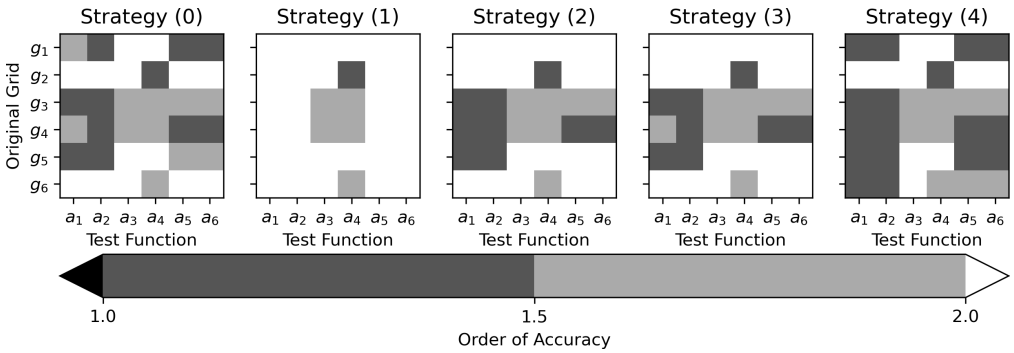


(a) Source Grids

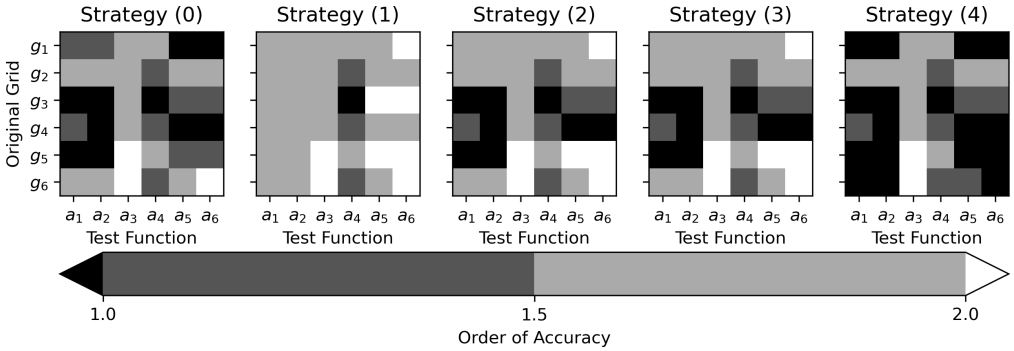


(b) Test Functions

Fig. 4.1: The source grid cell interfaces (Subfigure 4.1(a)) and test functions (Subfigure 4.1(b)), used in the sandbox test cases.



(a) 2-Norm Accuracy



(b) Max-Norm Accuracy

Fig. 4.2: The order of accuracy of each remapping strategy for every combination of grid function and test function, in 2–norm (Subfigure 4.2(a)) and max–norm (Subfigure 4.2(b)), obtained by calculating the error for domains decomposed into 16, 32, 64, 128, 256, 512, and 1024 cells and remapping from the source grid to the target grid a single time.

5. DCMIP Test Results. The Dynamical Core Model Intercomparison Project (DCMIP) introduced a suite of test cases in 2012 and 2016 with the goal to understand the treatment of the equations of motion within several atmospheric general circulation models [11,12]. These test cases cover a range of complexities, from pure advection tests to non-orographic gravity waves on a small planet, and beyond.

Here, we compare the result of using strategy (0), which is used in the current version of NH-HOMME, and strategy (1), where we use unbounded ghost cell data for only the thermodynamic variable. We present results for the following DCMIP test cases in the subsequent subsections:

- 2012-2.0.0 - steady-state atmosphere at rest in the presence of orography with moderately-steep slopes,
- 2012-2.1 - non-hydrostatic mountain waves over a Schär-type mountain without vertical wind shear,
- 2012-2.2 - non-hydrostatic mountain waves over a Schär-type mountain with vertical wind shear,
- 2012-3.1 - non-hydrostatic gravity waves with an overlaid potential temperature perturbation at the equator,
- 2016-1.1 - moist baroclinic wave with terminator chemistry, and
- 2016-1.2 - idealized tropical cyclone.

Although strategies (2) and (3) are more accurate than strategy (0) for several sandbox test cases, they were found to perform nearly identically to strategy (0) for these DCMIP test cases.

5.1. DCMIP Test 2012-2.0.0 Results. The DCMIP test 2012-2.0.0 is designed to give insight to the pressure gradient calculation in the presence of orography. In particular, the model is initialized with a hydrostatically-balanced atmosphere at rest on a non-rotating Earth-size planet. The surface topography, illustrated in Figure 5.1 by the black shape on the model surface, consists of a highly oscillatory mountain range and flat terrain away from the mountain range. Further details for DCMIP test 2012-2.0.0 are given in Subsection 2.0 of [11].

From a physical standpoint, in this test there is no driving force to cause the atmosphere to flow. The flow we observe in the simulation is entirely driven by numerical errors in the calculation of the pressure gradient in the discretized model, which gives insight into the discrete pressure gradient terms. Figure 5.1 shows a snapshot of the simulated flow on the sixth day of simulation at the equator.

There are two main points of comparison between strategy (0) and strategy (1) for DMIP test 2012-2.0.0: wind velocity extrema and noise. As mentioned at the end of Section 3, strategy (0) is more dissipative than strategy (1), as strategy (1) allows for the augmentation of extrema at the model surface and top. Thus, strategy (1) leads to a larger wind velocity than strategy (0) for this test case. Although larger wind velocity extrema is undesirable for this test case, the errors in the discrete pressure gradient terms may be eliminated entirely by choosing special forms of the equations of motion [11]. As we may "solve" the issue of the numerical errors of the discrete pressure gradient terms independently of our choice of remapping strategy, we are more concerned with using the remapping strategy which minimizes the noise in the simulation.

Strategy (1) does indeed reduce the noise when compared to strategy (0). Most notably, the noise present at the model surface toward the edges of the mountain range (approximately 15E and 160E) is almost entirely eliminated by using strategy (1).

5.2. DCMIP Tests 2012-2.1 and 2012-2.2 Results. Similarly to DCMIP test 2012-2.0.0, DCMIP tests 2012-2.1 and 2012-2.2 are designed to illustrate the impact of

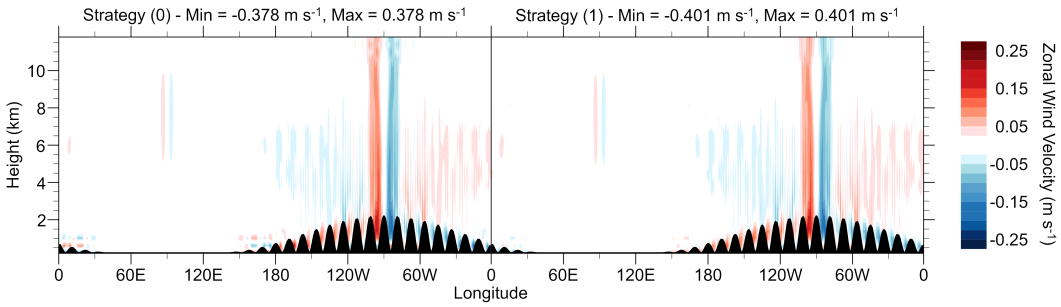


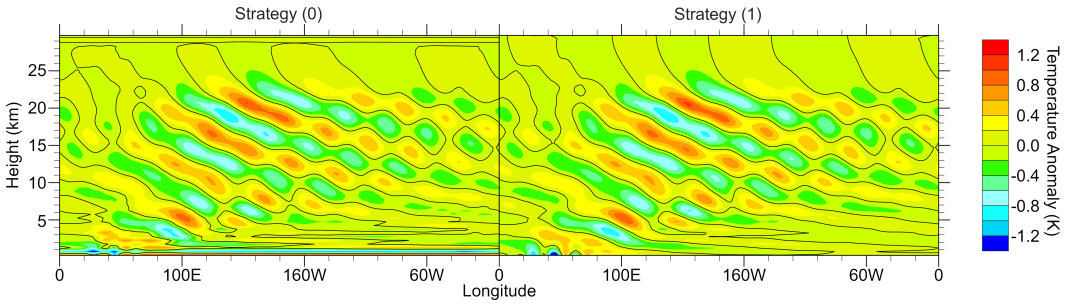
Fig. 5.1: A comparison of strategy (0) (left) and strategy (1) in which only the thermodynamic variable is remapped using unbounded ghost cell data (right) for DCMIP test 2012-2.0.0 on the sixth day of the simulation at the equator. The surface topography is show by the black shape on the model surface. Although strategy (1) results in a larger zonal wind velocity perturbation, there is less noise toward the model surface, e.g., above the mountains at approximately 160E.

mountain profiles on simulated atmospheric flow. These two tests have almost identical setups; both are initialized with an atmosphere in purely zonal flow on a non-rotating reduced-size Earth with a reduction factor of 500, including mountainous topography similar to that of DCMIP test 2012-2.0.0. The reduced-radius of the planet ensures that the simulated response to the mountains contain hydrostatic and non-hydrostatic features. The difference between DCMIP test 2012-2.1 and 2012-2.2 is that the initial wind profile of the latter has zonal velocity increasing vertically, whereas the initial wind profile is the former remains contrast with respect to altitude. Further details of these tests are given in Sections 2.X, 2.1, and 2.2 or [11]. This test utilizes a vertically-constant zonal wind velocity on a reduced-size planet with a reduction factor of 500, which is intended to trigger a non-hydrostatic response [11]. A snapshot after 7200 s of simulation time for both tests is shown in Figure 5.2.

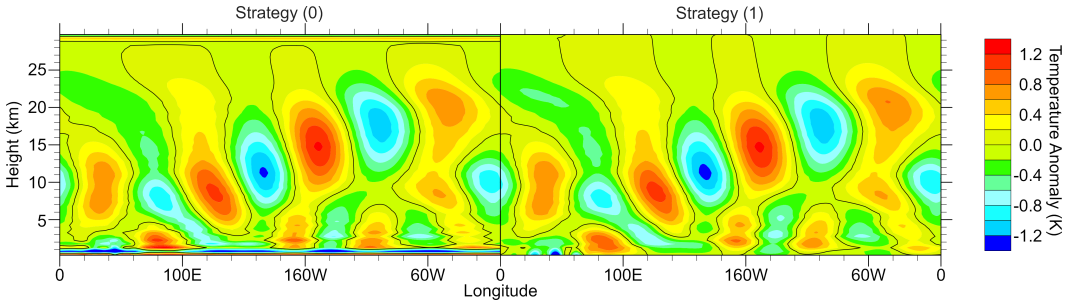
When using strategy (0) for both of these tests, the temperature anomaly from a constant background state at the model surface and top is highly oscillatory. In particular, at the model surface there is a steep decrease in temperature at approximately 1 km altitude followed by a sharp increase below that altitude. At the model top, there is a slightly shallower increase in temperature at approximately 29 km followed by a decrease above that altitude. These noisy artifacts are entirely eliminated when using strategy (1).

5.3. DCMIP Test 2012-3.1 Results. The DCMIP test 2012-3.1 illustrates the response of models to short time-scale wave-motion triggered by a localized perturbation. The initial state of this test is in both hydrostatic balanced and gradient-wind balance, and the overlay of a potential temperature perturbation at the equator triggers the evolution of gravity waves. Similarly to DCMIP tests 2012-2.1 and 2012-2.2, this test takes place on a non-rotating reduced-radius Earth, instead with a reduction factor of 125, ensuring that both hydrostatic and non-hydrostatic responses can be observed. Unlike DCMIP tests 2012-2.x, the topography of this planet is entirely flat. Further details of this test are given in Section 3 of [11]. A snapshot after 3600 s of simulation time is shown in Figure 5.3.

Similarly to DCMIP tests 2012-2.1 and 2012-2.2, when using strategy (0) there are unnatural potential temperature anomalies which oscillate vertically near the model top and surface, especially toward the center of the gravity wave. These oscillations are entirely



(a) DCMIP Test 2012-2.1



(b) DCMIP Test 2012-2.2

Fig. 5.2: A comparison of strategy (0) (left) and strategy (1) in which only the thermodynamic variable is remapped using unbounded ghost cell data (right) for DCMIP test 2012-2.1 (Subfigure 5.2(a)) and test 2012-2.2 (Subfigure 5.2(b)) at 7200 s simulation time. The temperature anomaly reported is a perturbation from a constant 300 K. The noise present in the model surface and top using strategy (0) in both tests is eliminated by using strategy (1).

absent when using strategy (1).

5.4. DCMIP Test 2016-1.1 Results. The DCMIP test 2016-1.1 examines the emergence and evolution of wave modes associated with baroclinic instability. In particular, the initial condition is a small perturbation from a geostrophic, hydrostatic reference state that satisfies the condition for baroclinic instability. Similarly to DCMIP test 2012-2.0.0, a perfect model would maintain this reference state indefinitely, but errors introduced by discrete operators trigger the wave modes to form. The small perturbation added to this reference state is notably larger than this numerical error, which allows for greater control in the evolution of the baroclinic wave. In addition, the model surface topography is flat and the velocity field goes to zero at the model surface. Further details are given in Section 1 of [12]. Figure 5.4 shows the temperature at 850 hPa after 15 days of simulation time.

Unlike the DCMIP 2012 tests reported in the previous subsection, there is no obvious noise present in the simulation for strategy (0) and no noise is introduced by strategy (1).

5.5. DCMIP Test 2016-1.2 Results. The DCMIP test 2016-1.2 examines the propagation of an analytic vortex on an Earth-size planet initialized in a background environment which is tractable to the augmentation of tropical cyclones [12]. Figure 5.5 shows a snapshot of the surface pressure on the 10^{th}.

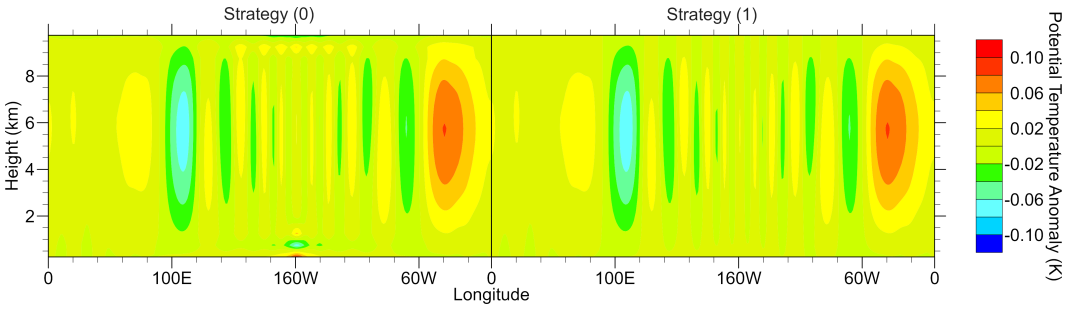


Fig. 5.3: A comparison of strategy (0) (left) and strategy (1) in which only the thermodynamic variable is remapped using unbounded ghost cell data (right) for DCMIP test 2012-3.1 at 3600 s simulation time. The potential temperature anomaly reported is a perturbation from the mean potential temperature. The noise present in the model surface and top using strategy (0) is eliminated by using strategy (1).

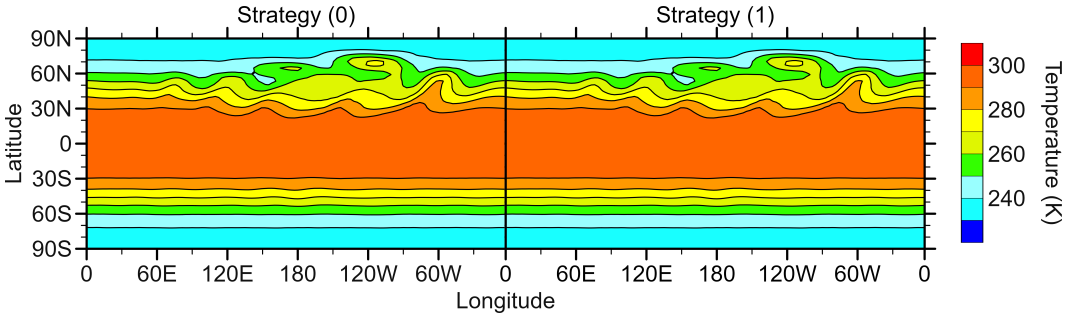
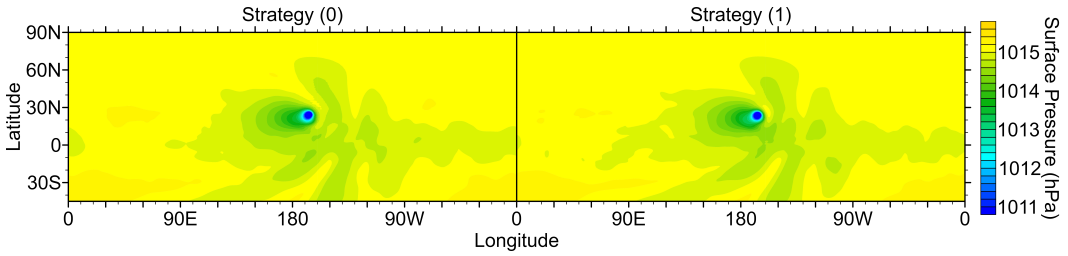


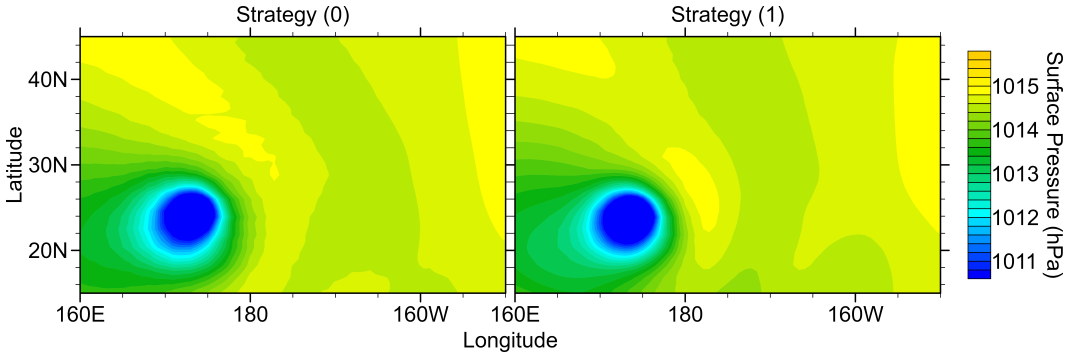
Fig. 5.4: A comparison of strategy (0) (left) and strategy (1) in which only the thermodynamic variable is remapped using unbounded ghost cell data (right) for DCMIP test 2016-1.1 on the 15$^{\text{th}}$ day of simulation. The temperature reported is at the 850 hPa pressure level. Unlike the DCMIP 2012 tests reported in previous subsections, there is no meaningful difference between the results obtained from strategy (0) and strategy (1).

Unlike all previously discussed DCMIP test cases, the noise present toward the center of the cyclone (Subfigure 5.5(b)) is present both when we use strategy (0) and when we use strategy (1) in which we remap the thermodynamic variable with unbounded ghost cell data. When we additional remap the water vapor mixing ratio with unbounded ghost cell data, this noise is eliminated.

Furthermore, using strategy (1) on both the thermodynamic variable and water vapor mixing ratio causes the idealized tropical cyclone to evolve differently, as may be noticed in, e.g., the 'branch' of lower surface pressure about the equator between 90W and 0 longitude in Subfigure 5.5(a). We attribute this difference in evolution to the sensitivity of the tropical cyclone to surrounding atmosphere conditions, i.e., the differences between strategy (0) and strategy (1) accumulate and build up over time.



(a) Full-view of cyclone



(b) Zoom-view of center of cyclone

Fig. 5.5: A comparison of strategy (0) (left) and strategy (1) in which the thermodynamic variable and water vapor mixing ratio are remapped using unbounded ghost cell data (right) for DCMIP test 2016-1.2 on the 10^{th} day of simulation. The noise at the center of the cyclone using strategy (0), as well as the propagation of noise from throughout the domain, is eliminated only when we use strategy (1) for the water vapor mixing ratio in addition to the thermodynamic variable.

6. Conclusions. In this work, we examined different treatments of ghost-cell data in a modified version of the PPM on a variety of sandbox test cases (Section 4), and compared the best of these strategies against the treatment of ghost-cell data currently used in NH-HOMME in several DCMIP tests (Section 5).

In the sandbox test cases, we found the the most accurate strategy for treating ghost-cell data of those compared was unbounded linear extrapolation (strategy (1)), which was at least as accurate as bounded ghost-cell data (strategy (0)) in every sandbox test case. With this result in mind, we chose to utilize strategy (1) for the thermodynamic variable for the DCMIP test case comparisons, as the relaxation of the requirement to preserve global bounds at the domain boundary is an acceptable trade-off for some state variables.

In all but one of the DCMIP test cases we examined, applying strategy (1) to the thermodynamic variable alone was sufficient to reduce or eliminate the noise found in several state variables at the domain boundary. However, in DCMIP test 2016-1.2 (idealized tropical cyclone), we found that it was necessary to use strategy (1) for the water vapor mixing ratio as well to eliminate this noise.

In future work, it would be interesting to investigate the following

- compare the use of strategy (0) to strategy (1) on a subset of the state variables for all DCMIP 2012 and 2016 test cases,

- compare the use of PPM and the piecewise-quartic method, which has been shown to be of higher order on a periodic domain [13], or other remapping algorithms, and
- compare the strategies here for the treatment of the ghost-cell data with more strategies, including using higher-order extrapolants and smaller ghost-cells,

among other possible future directions.

Acknowledgments. We would like to acknowledge Andrew Bradley and Oksana Guba (both of Sandia National Laboratories, Albuquerque, NM) for insightful conversations, ideas, and feedback. We also thank Andrew Salinger and Michael Wolf (both of Sandia National Laboratories, Albuquerque, NM) for assistance in organizing the execution of this project. J. L. Torchinsky was funded by the Department of Energy's Computational Science Graduate Fellowship, under grant number DE-SC0020347.

References.

[1] P. Colella and M. Sekora, *A limiter for PPM that preserves accuracy at smooth extrema*, Journal of Computational Physics, 227 (2008), pp. 7069–7076.
[2] P. Colella and P. Woodward, *The piecewise parabolic method (PPM) for gas-dynamical simulations*, Journal of Computational Physics, 54 (1984), pp. 174–201.
[3] I. Kavčič and J. Thuburn, *A Lagrangian vertical coordinate version of the ENDGame dynamical core. Part I: Formulation, remapping strategies, and robustness*, Quarterly Journal of the Royal Meteorological Society, 144 (2018), pp. 1649–1666.
[4] R. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, SIAM, Philadelphia, PA, Jun. 2007.
[5] S.-J. Lin, *A "vertically Lagrangian" finite-volume dynamical core for global models*, Monthly Weather Review, 132 (2004), pp. 2293–2307.
[6] P. McCorquodale and P. Colella, *A high-order finite-volume method for conservation laws on lacally refined grids*, Communications in Applied Mathematics and Computational Science, 6 (2011), pp. 1–25.
[7] W. Rider, *Reconsidering remap methods*, International Journal for Numerical Methods in Fluids, 76 (2014), pp. 587–610.
[8] M. Sekora and P. Colella, *Extremum-preserving limiters for MUSCL and PPM*, ArXiv: Computational Physics, (2009), pp. 1–26.
[9] V. Starr, *A quasi-Lagrangian system of hydrodynamical equations*, Journal of Meteorology, 2 (1945), pp. 227–237.
[10] M. A. Taylor, O. Guba, A. Steyer, P. Ullrich, D. Hall, and C. Eldred, *An energy consistent discretization of the nonhydrostatic equations in primitive variables*, Journal of Advances in Modeling Earth Systems, 12 (2020), pp. 1649–1666.
[11] P. Ullrich, C. Jablonowski, J. Kent, P. Lauritzen, R. Nair, and M. Taylor, *Dynamical core model intercomparison project (DCMIP) test case document*. Electronic, technical report, 2013.
[12] P. Ullrich, C. Jablonowski, K. Reed, C. Zarzycki, P. Lauritzen, R. Nair, J. Kent, and A. Verlet-Banide, *Dynamical core model intercomparison project (DCMIP2016) test case document*. Electronic, technical report, 2016.
[13] L. White and A. Adcroft, *A high-order finite volume remapping scheme for nonuniform grids: The piecewise quartic method (PQM)*, Journal of Computational Physics, 227 (2008), pp. 7394–7422.

MACHINE-LEARNING FOR SINGLE PARTICLE MOTION IN PLASMAS

SONATA M. VALAITIS* AND KATHRYN A. MAUPIN†

Abstract. The application of machine learning to particle advancement in kinematic plasma simulations is the motivation behind this work. Fully kinetic particle-in-cell simulations are necessary for resolution of high-frequency effects in magnetic confinement fusion plasmas near the wall, but the application of this simulation methodology to larger domains of the plasma is limited by computational cost and speed limitations. Both gyrokinetic and fully kinetic particle-in-cell methods often employ the Boris-Bunemann particle advancement algorithm due to its excellent long-simulation-time accuracy and efficiency. This work discusses the implementation of machine learning in plasma physics for the purpose of regression and classification tasks relating to experimental and computational data. The implementation of a neural network to the task of particle advancement is addressed, where training data is provided by the Boris-Bunemann algorithm with and without frequency correction. The algorithm is trained and tested for various cases including uniform E and B fields, and nonuniform fields where the E × B drift, grad-B drift and curvature drift are present. The aim of this work is to implement a neural network to accurately predict single charged particle motion in an electromagnetic field, capturing relevant physical effects and potentially providing a computationally efficient alternative to the Boris-Bunemann particle advancement algorithm for use in kinematic plasma simulations.

1. Introduction and Motivation. Simulation of a magnetically confined plasma is a multi-scale physics problem which is essential to the development of a viable commercial fusion reactor. Particle-in-cell (PIC) codes are a simulation methodology used to determine the positions and velocities of individual plasma particles over time and to calculate the macroscopic properties of a plasma [29]. While highly accurate, PIC simulations are computationally intensive, making analyses such as uncertainty quantification or optimal experimental design intractable.

Machine-learning (ML) algorithms have the potential to reduce the computational cost of high-fidelity simulation [6,11]. In the physical sciences, ML has been used in a wide range of applications, including data analysis, prediction and inference, and comparison between experimental and computational data. Applications include fitting scattered data, vectors, images, and time-series [3]; analyzing signals [16]; constructing smooth functions [15]; and approximating partial differential equations [26]. ML has also been successfully applied to both experimental data and numerical simulations in plasma physics.

The goal of this work is to look at one method of increasing the computational efficiency of fully kinetic PIC codes using a machine learning approach. The paper is outlined as follows: An overview of nuclear fusion is given in Section 2, with simulation details given in Section 3. Section 4 discusses relevant machine learning techniques, the training and testing of which is described in Section 5. Concluding remarks are then given in Section 6.

2. Overview of Nuclear Fusion. In nuclear fusion, multiple atomic nuclei combine to form a different chemical element. Isotopes of hydrogen such as deuterium and tritium are commonly used as fuel for nuclear fusion reactions, which produce helium. Large amounts of energy are released during this process via the emission of subatomic particles and various forms of radiation. Nuclear fusion releases nearly four million times as much energy as chemical reactions such as burning coal or gas and four times as much energy as nuclear fission reactions at equal mass [1]. Fusion occurs within "plasma": an ionized, electrically conductive state of matter that is created when gases are heated and subjected to strong electromagnetic fields.

Fusion is the process that powers active stars such as our sun, and it is being investigated

*University of Illinois Urbana-Champaign, sonatav2@illinois.edu
†Sandia National Laboratories, kmaupin@sandia.gov

as a potential source of sustainable, carbon-free power production here on Earth. There are several factors that make nuclear fusion an attractive power source. Unlike nuclear fission, fusion does not produce long-lived radioactive waste that must be monitored and stored. Because it does not require fissile materials such as uranium and plutonium, it also poses a low risk of nuclear weapons proliferation. Fusion is argued to be environmentally safer than nuclear fission because it does not create the chain reactions that could lead to meltdown of a fission plant reactor core. Fuel availability is also a beneficial factor. Deuterium is abundantly available and tritium can be produced via reactions with a lithium blanket within the reactor. Lastly, fusion power produces no greenhouse gases and therefore does not contribute to climate change [1].

Magnetic confinement fusion is the predominant approach to fusion energy research and consists of using electromagnetic fields to heat and confine the plasma. Tokamaks and stellarators are two of the leading magnetic confinement device configurations used to confine plasma into a toroidal shape using a magnetic field. Tokamaks use a toroidal plasma current to achieve a necessary rotational transform of the magnetic field, whereas stellarators employ external non-axisymmetric magnetic coils to achieve the same purpose [31]. Both experimental designs have been extensively simulated using the particle-in-cell method.

3. PIC Simulations. Though many PIC algorithms and approaches exist, they all have the same general time-step loop. The general procedure consists of integrating the equations of motion for each particle, tracking each particle within the field mesh grid, applying any boundary conditions or collision terms, interpolating the current and charge source terms to the mesh grid, integrating the field equations on the mesh grid, and interpolating the field values from the grid to particle locations. A schematic of this time-step loop is shown in Figure 3.1.

Efforts to produce PIC simulations of the entire domain of a tokamak while accurately including plasma-surface interactions and plasma sheath effects traditionally take three possible approaches: fully kinetic PIC, gyrokinetic PIC, and a coupling of the two. Fully kinetic PIC codes employ six-dimensional representations of plasma particles that fully describe their positions and velocities in three-dimensional space. This approach is widely used in plasma physics modeling because it reduces the number of assumptions made in the physical model to a minimum. Fully kinetic PIC codes enable complex, highly accurate simulations, which include plasma-surface interactions, at the cost of high computation time [29]. For this reason, they have been widely applied to small-domain simulations of the plasma sheath region near the wall of magnetic confinement fusion devices, such as tokamaks and stellarators. It is necessary to integrate full cyclotron orbits when studying charged particle dynamics in plasmas with waves at frequencies above cyclotron frequency. If the computational efficiency of fully kinetic simulations could be increased, it would allow long time scale simulations of tokamaks which resolve not only the lower frequency dynamics in the bulk plasma but also the higher frequency effects in the plasma sheath near the wall.

Gyrokinetic PIC codes employ an alternative simulation methodology in which fast scale gyromotion is removed from plasma dynamics. This is a five-dimensional approach that describes the three-dimensional location of the "guiding center" of the particle along with its parallel and perpendicular velocity components relative to the magnetic field line. Gyrokinetic PICs significantly reduce the computation time required to simulate strongly magnetized plasmas and expand the achievable domain size of a given simulation. However, these codes are only applicable for describing plasma oscillations in slowly varying electromagnetic fields [29]. They are therefore limited in applicability to the bulk plasma and scrape-off layer regions of a magnetic confinement fusion device. Gyrokinetic PICs must therefore make a

choice of boundary condition approximations to represent the plasma sheath region and plasma-surface interactions near the wall. This technique has been extensively implemented on high-performance computing resources over the past several decades [14]. Gyrokinetic reduction procedures are widely employed by many modern particle-in-cell codes and have played a fundamental role in advancing our ability to simulate strongly magnetized plasmas over long time periods [19, 27]. However, the importance of plasma-surface interactions on bulk plasma dynamics is just beginning to be understood. As a result, there is increasing interest in optimizing the computational efficiency of fully kinetic PIC methods in order to examine the impact of surface interactions and dust-particle transport on the bulk plasma [10, 20].

3.1. Boris-Bunemann Algorithm. The Boris-Bunemann algorithm is a widely used particle advancement algorithm in PIC codes [21, 23]. It is based on a discretization of the Newton-Lorentz equation of motion (or "Lorentz force law"), as shown in Equations 3.1-3.2 for α species with N_α particles each ($i = 1, \ldots, N_\alpha$)

$$\frac{d\mathbf{x}_{\alpha i}}{dt} = \mathbf{v}_{\alpha i} \tag{3.1}$$

$$\frac{d\mathbf{v}_{\alpha i}}{dt} = \frac{eZ_\alpha}{m_\alpha} \left[\mathbf{E}(\mathbf{x}, t) + \mathbf{v}_{\alpha i} \times \mathbf{B}(\mathbf{x}, t)\right] |_{x=x_{\alpha,i}} \tag{3.2}$$

where $\mathbf{x}$ is position, $\mathbf{v}$ is velocity, Z_α is the atomic number and m_α is the mass of species α, and $\mathbf{E}$ and $\mathbf{B}$ are the electric and magnetic fields, respectively.

The Boris-Bunemann algorithm is shown in Equations 3.3 - 3.10. It is a type of leapfrog method in that it requires a half-time-step offset between the velocity and position updates [23].

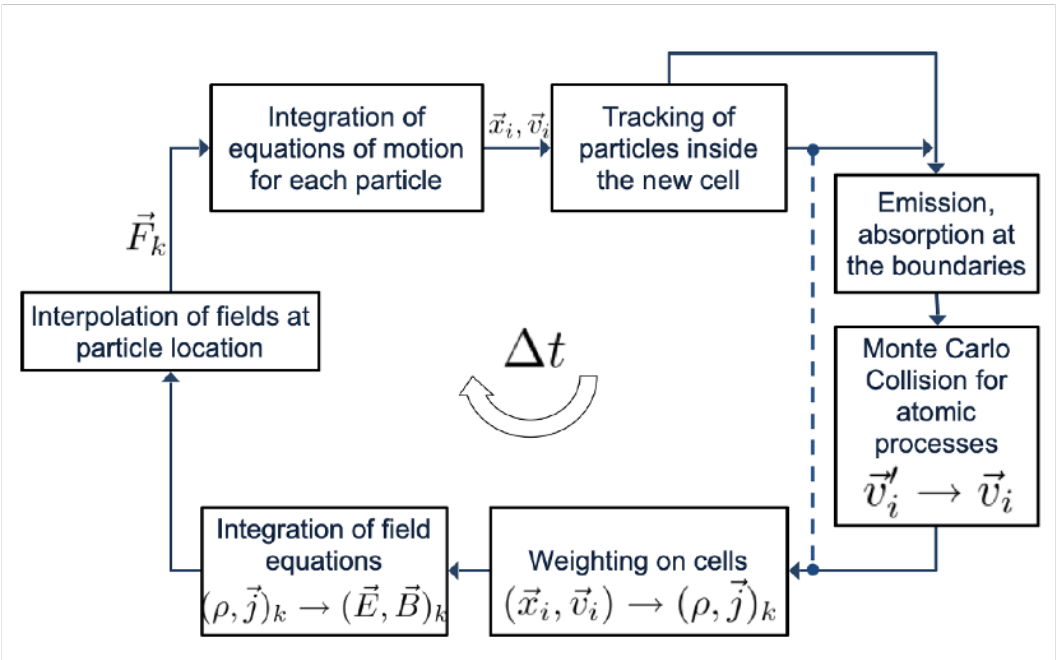


Fig. 3.1: Time-step loop of particle-in-cell method [7]

First, a half-acceleration due to the electric field is applied

$$\overrightarrow{v}^{-} = \overrightarrow{v}_{n-1/2} + \frac{q}{m}\frac{\Delta t}{2}\overrightarrow{E_n} \tag{3.3}$$

The rotation due to the magnetic field is then applied

$$\overrightarrow{v}' = \overrightarrow{v}^{-} + \overrightarrow{v}^{-} \times \overrightarrow{t} \tag{3.4}$$

$$\overrightarrow{v}^{+} = \overrightarrow{v}^{-} + \overrightarrow{v}' \times \overrightarrow{s} \tag{3.5}$$

There are "frequency-corrected" and "non-frequency-corrected" versions of the Boris-Bunemann algorithm, where the former places bounds on absolute error but also requires a computationally costly tangent function calculation. The "frequency-corrected" Boris-Bunemann algorithm employs

$$\overrightarrow{t} \equiv -\overrightarrow{b}\tan\frac{\theta}{2} \tag{3.6}$$

whereas the more computationally efficient "non-frequency-corrected" Boris-Bunemann uses the approximation

$$\overrightarrow{t} \approx \frac{q\overrightarrow{B}}{m}\frac{\Delta t}{2} \approx \frac{\overrightarrow{\Omega}\Delta t}{2} \tag{3.7}$$

where

$$\overrightarrow{s} = \frac{2\overrightarrow{t}}{1+|\overrightarrow{t}|^2} \tag{3.8}$$

Lastly, a second half-rotation from the electric field is then utilized

$$\overrightarrow{v}_{n+1/2} = \overrightarrow{v}^{+} + \frac{q}{m}\frac{\Delta t}{2}\overrightarrow{E_n} \tag{3.9}$$

The position of the particle at the new time step $n+1$ is updated

$$\overrightarrow{x}_{n+1} = \overrightarrow{x}_n + \overrightarrow{v}_{n+1/2}\Delta t \tag{3.10}$$

The Boris-Bunemann algorithm is considered the "de facto standard" for advancing charged particles in PIC simulations. Although it is not symplectic, it possesses excellent long-term accuracy and the same global bound on energy error typically observed in symplectic algorithms. This is has been shown to be a result of the fact that it preserves phase space volume [21].

4. Machine-Learning for the Physical Sciences. Machine-learning models typically represent a functional relationship between some set of independent inputs and a dependent output. Supervised learning techniques ranging from regression schemes to deep learning are often used to model these functions, while unsupervised learning techniques are often applied to reduce input space dimensionality [26]. ML models have advantages relating to both uncertainty quantification and computational efficiency.

4.1. Introduction to Machine Learning. ML models are frequently applied to statistical, data-based tasks such as classification, clustering, dimensional reduction, and regression. Regression involves using ML algorithms to approximate real-valued functions, while classification returns discrete or categorical values. In their most basic form, machine-learning models consist of algorithms that predict numerical outputs based on numerical inputs and that learn or improve their performance with experience, i.e. as data is added. If an ML algorithm was modeling a real-valued function such as $y = f(x)$, the "experience" would consist of the input and output values $(X = \{\mathbf{x}_i\}, Y* = \{\mathbf{y}_i^*\})$ for $i = 1, \ldots, N$ data points produced by either observation or simulation. "Learning" is the improvement of the algorithm's performance with increasing experience. A simple and frequently-used way of measuring performance is to calculate the squared error by comparing the predicted values y to the correct output values $y*$: $SE = \sum(y_i^* - y_i)^2$ [26]. Another common choice is to use the mean squared error $MSE = \sum_{i=1}^{n}(y_i^* - y_i)^2/n$, which can be consistently applied to data sets with varying numbers of data points n.

There are two major categories of ML algorithms used for regression: parametric and non-parametric methods. The simplest parametric method is the linear least squares method, and it is often used with explicitly parametrized model functions that have a linear dependence on their parameters. It can also be applied to nonlinear model functions, provided that a nonlinear solution technique is employed. Parametric methods typically require labeled or "supervised" data.

Non-parametric methods do not require explicit parametrization of the model function in advance. These include decision trees, support vector machine, k-nearest neighbor algorithms, and some neural networks. Without the constraint of parametrization based on prior knowledge of the data, non-parametric methods tend to require more data and time for training. They also offer increased flexibility because they function by setting a very large number of parameters and combining a large number of simple functions to approximate the model function [26].

4.2. Applications in Plasma Physics. In the area of inertial confinement fusion, deep neural networks and random forests have been applied to learn the response of a radiation hydrodynamics code over high-dimensional parameter spaces and to identify optimal simulation parameters [8,17]. In hydrodynamic models, neural networks have been applied to model the source terms in discretized partial differential equations and to provide less computationally costly alternatives to traditional algorithm implementations [28]. Random forests, neural networks, and support vector machines have been applied to the classification task of predicting disruptions in the tokamaks such as DIII-D and JET with a high degree of accuracy [22,26,30]. Deep learning techniques such as multilayer perceptrons and convolutional neural networks have been used to calculate the electric field from the electron phase space in particle-in-cell simulations. While this method was able to produce the correct growth rate in a two-stream instability test, the deep-learning based method did not conserve total energy and momentum [2].

4.3. Neural Networks. A neural network is a set of nested, nonlinear functions that can be altered to fit input and output data as shown in equation 4.1. They are also often represented as directed acyclic graphs

$$y = f^{(J)}\left(...f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right)\right) \tag{4.1}$$

Input values in neural networks undergo a nonlinear operation known as the "activation function" within each layer of the network. Multi-layer networks are referred to as "deep" neural networks and often exhibit better performance than neural networks with fewer layers.

Each neuron takes the output values, x, from the previous layer as its inputs and produces a combination $f(Wx+b)$ of them as the output, where f is the activation function mentioned earlier. For a linear combination $f(x) = x$, the output would be $z = Wx + b$. The weights W and biases b are among many free parameters used to fit the machine-learning model to the data [26]. The number of layers in a neural network is referred to as its "depth" and the number of nodes per layer is its "width."

While neural networks are generally regarded as parametric methods, fully-connected networks at the limit of infinite width are equivalent to non-parametric Gaussian Processes (GPs). In other words, an i.i.d. prior over the weights and biases of a fully-connected, infinitely wide neural network is equivalent to a corresponding Gaussian Process prior over functions, and the function produced by the neural network is a function drawn from the GP [12]. This equivalence allows exact Bayesian inference to be employed for regression using infinitely wide neural networks by evaluating the corresponding GPs. It has been shown that GP predictions usually outperform those made by finite-width neural networks, that the network prediction error decreases with increasing width, and that GP uncertainty is correlated with network prediction error [12]. For the case of a single-layer infinitely wide network, the form of the GP kernel is well understood. Recent work has been done to develop kernel-based methods of understanding infinite-width deep neural networks as well [4].

4.4. Uncertainty Quantification. Computational simulation has become the third pillar of science [18], complementing the two traditional pillars, theory and experimentation. These three pillars come together in predictive science, where high-fidelity models, such as PIC codes, are derived from first principles, calibrated to experimental data, and implemented computationally. Correspondingly, uncertainties in model predictions can come from any of these three sources of information.

Generally, computational simulation allows scientists to make predictions about physical systems that have not yet been observed or to make inferences about physical systems that are difficult or impossible to observe directly. Quantifying the extent to which these predictions can be trusted is a central tenet of the field of uncertainty quantification (UQ). In addition to understanding uncertainties in model predictions, UQ processes provide insights into sources of this uncertainty, such as calibration data, missing or unknown physics, or numerical errors [25].

UQ analyses require many (i.e. hundreds, thousands, or more) model runs. High-fidelity simulations, such as those provided by PIC codes, are typically expensive to run, making quality uncertainty analysis infeasible. By replacing high-fidelity codes with ML models at a reduced computational cost, UQ processes become tractable. Thus, the development of a ML model for PIC not only reduces the cost of making predictions, it provides a better understanding of how reliable these predictions are.

4.5. Computational Efficiency. Machine-learning algorithms are often implemented with the goal of increasing the computational efficiency of physics-based simulations. For example, deep learning applied to computational fluid dynamics for modeling two-dimensional turbulent flows has accomplished 40- to 80-fold gains in the computational speed of simulations while remaining stable during long-time simulations and generalizing to input parameters outside of the range of the training data [11].

One drawback is that the computational cost of training a neural network can sometimes be significant. There are many methods that have been developed to address this problem, including changing the activation function from a threshold to either sigmoidal or a rectified linear unit (ReLU) function, each of which offer advantages related to the ability to use gradient-based methods [13]. Over-specification, or training a network which is significantly

larger than needed, can also be an effective method of reducing the necessary training time and increasing accuracy [13]. Lastly, regularizing the weights of a neural network has the effect of accelerating convergence [13].

4.6. Application to Particle-in-Cell Simulations. Since the goal of the machine-learning algorithm is to serve as a computationally efficient alternative to the Boris-Bunemann particle pusher, the basic goal of the algorithm is to correctly predict the position and velocity of a single charged super-particle one time step ahead of the input values. The inputs are therefore $\{q, m, \overrightarrow{E_n}, \overrightarrow{B_n}, \Delta t, v_{n-1/2}, x_n\}$ and the outputs are $\{x_{n+1}, v_{n+1/2}\}$ where n refers to the time step index. It requires the ability to make a sequence of correct position and velocity predictions that produce a correct particle orbit for a long-time simulation of charged particle motion in an electromagnetic field.

Since this problem formulation resembles a time-series with an oscillating, nonlinear, and periodic quality, the neural network architecture should be able to accurately predict this behavior. The "Long Short-Term Memory" network is a type of recurrent neural network architecture that is able to classify and predict time series events given time lags of unknown duration, and has demonstrated success in predicting sine functions [9, 24].

There are several considerations regarding use cases for an ML model within the PIC simulation time loop, potential gains in computational efficiency, and accuracy and generalizability. It may be beneficial to include the gradients and time derivatives of the electromagnetic fields as inputs in order to improve the accuracy of the ML model in capturing more complex drifts. In addition, while the current model assumes infinite domain, an ML model could also be extended to make predictions on finite systems, including boundary conditions and particle collisions. Compared with solely modeling the particle advancement step, potentially increased gains in computational efficiency could be achieved by including splitting and merging steps in the ML model functionality, or by including a method of updating the fields and making longer-term predictions that over-step traditional time-scale restrictions on the numerical integrator.

5. Training and Testing Procedure for Single Particle Motion Simulations. In order to verify that the machine-learning algorithm can capture each of the drifts that typically occur in plasmas, the training procedure includes data from Boris-Bunemann simulations designed to produce each of the major effects outlined below. Note that a more detailed description can be found in Chapter 2 of [5].

5.1. Uniform E and B Fields.

5.1.1. E = 0. For the case of E = 0 and a uniform B field ($\mathbf{B} = B\hat{z}$), the charged particle motion is well-described as a simple harmonic oscillator at the cyclotron frequency ω_c,

$$\omega_c = \frac{|q|B_G}{mc} \tag{5.1}$$

where B_G indicates B in units of Gauss [5]. The Larmor radius of the particle's cyclotron gyration is

$$r_L \equiv \frac{v_\perp}{\omega_c} = \frac{mv_\perp c}{|q|B_G} \tag{5.2}$$

For this case, the charged particle motion is a circular orbit around the fixed guiding center (x_0, y_0),

$$x = x_0 + r_L \sin \omega_c t \tag{5.3}$$

$$y = y_0 \pm r_L \cos \omega_c t \tag{5.4}$$

where ± reflects the sign of the particle charge q.

A training orbit produced by the Boris-Bunemann algorithm (with and without frequency correction, as explained in Equations 3.6-3.7) for this case is shown in Figures 5.1-5.2. The non-frequency-corrected Boris algorithm has an absolute error that increases with time, but it is often employed in PIC simulations because it is significantly more computationally efficient than the frequency-corrected version of the algorithm, which requires a trigonometric operation. For orbits where an analytical solution exists, that solution can be sampled as another method of providing training data to the neural network. Performance can then be compared between the neural networks trained using Boris-Bunemann and those trained using analytical solutions, where available.

5.1.2. $\mathbf{E} \neq \mathbf{0}$. The charged particle trajectory for this case is described by

$$v_x = v_\perp \exp(i\omega_c t) \tag{5.5}$$

$$v_y = \pm i v_\perp \exp(i\omega_c t) - \frac{E_x}{B} \tag{5.6}$$

where $v_\perp$ reflects the speed of the particle in the plane perpendicular to $\mathbf{B}$. This trajectory can be visualized as a slanted helix with changing pitch [5]. The electric field drift of the guiding center is

$$v_E = \frac{\mathbf{E} \times \mathbf{B}}{B^2} = 10^8 \frac{E(V/cm)}{B(gauss)} \cdot \frac{cm}{sec} \tag{5.7}$$

5.2. Nonuniform B Field.

5.2.1. $\nabla B \perp B$: Grad-B Drift. For this case there is a gradient in $|B|$ that causes the Larmor radius to vary in size depending on the position of the particle in its gyrating orbit. Because ions and electrons gyrate in opposite directions, the effect of ∇B causes electrons and ions to drift in opposite directions that are perpendicular to both $\mathbf{B}$ and ∇B. The guiding center grad-B drift velocity is given by

$$v_{\nabla B} = \pm \frac{1}{2} v_\perp r_L \frac{\mathbf{B} \times \nabla B}{|B|^2} \tag{5.8}$$

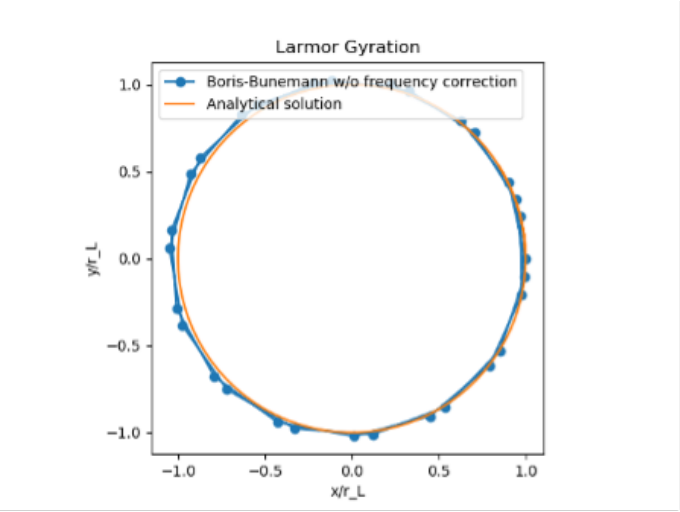
where ± reflects the sign of the charge [5].

5.2.2. Curved B : Curvature Drift. If $|B|$ is assumed constant and the lines of force possess a constant radius of curvature R_c, the centrifugal force exerted on the particles as they travel along the field lines will create the guiding center "curvature drift" as given by

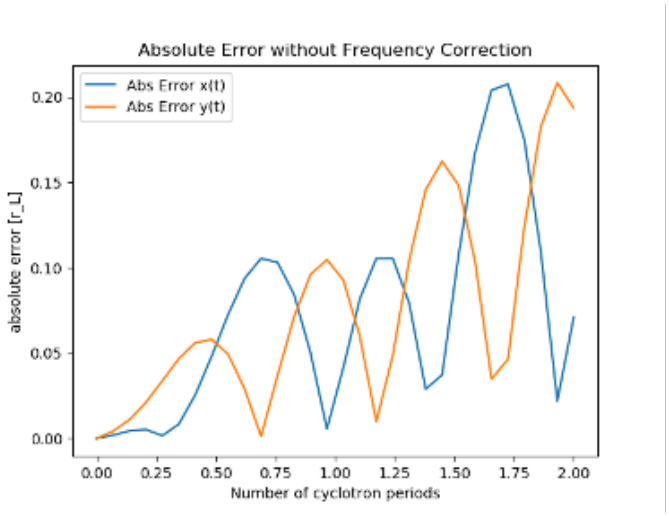
$$v_R = \frac{1}{q} \frac{\mathbf{F_{cf}} \times \mathbf{B}}{B^2} = \frac{m v_{||}^2}{qB^2} \frac{\mathbf{R_c} \times \mathbf{B}}{R_c^2} \tag{5.9}$$

When the decrease of $|B|$ with increasing radius is accounted for, the total guiding center drift in a curved vacuum field is the sum of the curvature drift and the grad-B drift resulting in

$$v_{\nabla B} + v_R = \frac{m}{q} \frac{\mathbf{R_c} \times \mathbf{B}}{R_c^2 B^2} \left(v_{||}^2 + \frac{1}{2} v_\perp^2 \right) \tag{5.10}$$



(a) Particle orbit for E = 0



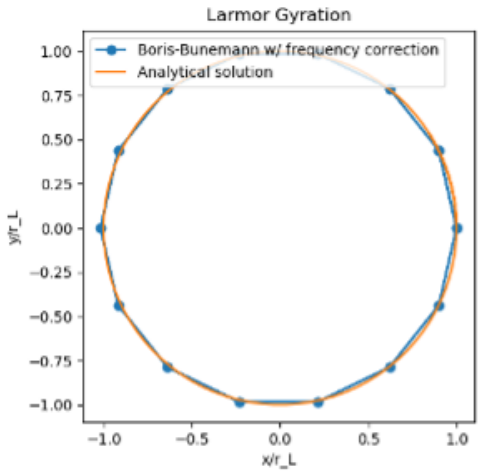
(b) Particle orbit error for E = 0

Fig. 5.1: Boris-Bunemann algorithm without frequency correction

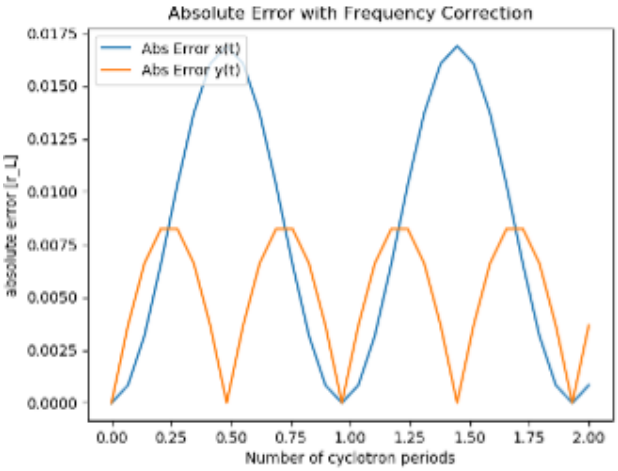
5.3. Nonuniform E Field. If the magnetic field is uniform and the electric field is nonuniform in space, the ordinary $\mathbf{E}\times\mathbf{B}$ drift is influenced by the inhomogeneity to produce the finite-Larmor-radius effect as shown in

$$v_E = \left(1 + \frac{1}{4} r_L^2 \nabla^2\right) \frac{\mathbf{E}\times\mathbf{B}}{B^2} \tag{5.11}$$

Unlike in uniform E fields, v_E is now dependent on particle species since r_L is larger for ions than for electrons [5]. The grad-B drift is another type of finite-Larmor-radius effect, but it does not become significant until the scale length of the field inhomogeneity is much larger compared with that of the nonuniform-E effect. If it is assumed that $\mathbf{E} \equiv E_0(\cos ky)\hat{\mathbf{x}}$,



(a) Particle orbit for E = 0



(b) Particle orbit error for E = 0

Fig. 5.2: Boris-Bunemann algorithm with frequency correction

then the drift can be represented as

$$v_E = \left(1 - \frac{1}{4}k^2 r_L^2\right) \frac{\mathbf{E} \times \mathbf{B}}{B^2} \tag{5.12}$$

5.4. Time-Varying E Field. If **E** and **B** are uniform in space, but **E** varies in time like $\mathbf{E} \equiv E_0 \exp i\omega t \hat{\mathbf{x}}$, we can define the quantities

$$\tilde{v_p} \equiv \pm \frac{i\omega}{\omega_c} \frac{\tilde{E_x}}{B} \tag{5.13}$$

$$\tilde{v_E} \equiv -\frac{\tilde{E_x}}{B} \tag{5.14}$$

Then the following equations of motion hold:

$$\ddot{v_x} = -\omega_c^2 \left(v_x - \tilde{v_p}\right) \tag{5.15}$$

$$\ddot{v_y} = -\omega_c^2 \left(v_y - \tilde{v_E}\right) \tag{5.16}$$

If **E** is assumed to vary slowly so that $\omega^2 << \omega_c^2$, these can be solved in a way that sums the velocities from drift and gyration separately:

$$v_x = v_\perp \exp i\omega_c t + \tilde{v_p} \tag{5.17}$$

$$v_y = \pm i v_\perp \exp i\omega_c t + \tilde{v_E} \tag{5.18}$$

The resulting guiding center drift has two parts. The x-component, called the polarization drift, is parallel to **E** and has opposite directions for ions and electrons. It is generally stated as

$$\mathbf{v_p} = \pm \frac{1}{\omega_c B} \frac{d\mathbf{E}}{dt} \tag{5.19}$$

The y-component is the ordinary $\mathbf{E} \times \mathbf{B}$ drift where v_E oscillates slowly with the electric field oscillation frequency ω [5].

6. Conclusions. The particle-in-cell simulation methodology was discussed, as well as the practical differences between the gyrokinetic and fully kinetic variations. The motivation of increasing the computational efficiency of PIC algorithms was established with the goal of extending the application of the fully kinetic methodology to whole-domain magnetic confinement fusion device simulations so that sheath and wall effects can be accurately included. The Boris-Bunemann algorithm was introduced and its computational advantages in accuracy and efficiency were discussed relative to other particle advancement methods. Machine learning was introduced in the context of the physical sciences and, more specifically, plasma physics. The advantages of machine learning in terms of potential gains in uncertainty quantification and computational efficiency were discussed. A neural network architecture was outlined for application to the Boris-Bunemann particle advancement problem. A training and evaluation methodology was outlined in which the Boris-Bunemann algorithm would provide training and reference solutions for the machine learning algorithm in order to ensure that effects such as the E × B drift, grad-B drift and curvature drift are accurately captured and accounted for by a machine learning model. The motivation of this work is to develop and test a machine learning model trained by the Boris-Bunemann particle advancement algorithm for use in kinematic plasma simulations and to compare its accuracy and computational cost with the original Boris-Bunemann algorithm.

References.

[1] *Advantages of fusion*. https://www.iter.org/sci/fusion. Accessed: 2010-08-27.
[2] X. Aguilar and S. Markidis, *A Deep Learning-Based Particle-in-Cell Method for Plasma Simulations*, arXiv:2107.02232 [physics], (2021). arXiv: 2107.02232.
[3] R. K. Archibald, M. Doucet, T. Johnston, S. R. Young, E. Yang, and W. T. Heller, *Classifying and analyzing small-angle scattering data using weighted k nearest neighbors machine learning techniques*, Journal of Applied Crystallography, 53 (2020), pp. 326–334.
[4] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang, *On Exact Computation with an Infinitely Wide Neural Net*, arXiv:1904.11955 [cs, stat], (2019). arXiv: 1904.11955.

[5] F. Chen, *Introduction to Plasma Physics and Controlled Fusion*, Plenum Press, 1984.
[6] N. F. Y. Chen, M. F. Kasim, L. Ceurvorst, N. Ratan, J. Sadler, M. C. Levy, R. Trines, R. Bingham, and P. Norreys, *Machine learning applied to proton radiography of high-energy-density plasmas*, Physical Review E, 95 (2017), p. 043305.
[7] D. Curreli, *npre598 lecture notes*, University of Illinois Urbana-Champaign, (2020).
[8] K. D. Humbird, J. L. Peterson, and R. G. McClarren, *Deep neural network initialization with decision trees*, arXiv:1707.00784 [cs], (2018). arXiv: 1707.00784.
[9] M. Jiménez-Guarneros, P. Gómez-Gil, R. Fonseca-Delgado, M. Ramírez-Cortés, and V. A. Aquino, *Long-Term Prediction of a Sine Function Using a LSTM Neural Network*, in Nature-Inspired Design of Hybrid Intelligent Systems, P. Melin, O. Castillo, and J. Kacprzyk, eds., vol. 667 of Studies in Computational Intelligence, Springer, 2017, pp. 159–173.
[10] R. Khaziev and D. Curreli, *hPIC: A scalable electrostatic Particle-in-Cell for Plasma–Material Interactions*, Computer Physics Communications, 229 (2018), pp. 87–98.
[11] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, *Machine learning–accelerated computational fluid dynamics*, Proceedings of the National Academy of Sciences, 118 (2021).
[12] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, *Deep Neural Networks as Gaussian Processes*, arXiv:1711.00165 [cs, stat], (2018). arXiv: 1711.00165.
[13] R. Livni, S. Shalev, Shwartz, and O. Shamir, *On the computational efficiency of training neural networks*, (2014).
[14] K. Madduri, E.-J. Im, K. Z. Ibrahim, S. Williams, S. Ethier, and L. Oliker, *Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms*, Parallel Computing, 37 (2011), pp. 501–520.
[15] M. Mahdavi, L. Zhang, and R. Jin, *Mixed Optimization for Smooth Functions*.
[16] R. J. Martis, C. Chakraborty, and A. K. Ray, *Wavelet-based Machine Learning Techniques for ECG Signal Analysis*, in Machine Learning in Healthcare Informatics, S. Dua, U. R. Acharya, and P. Dua, eds., Intelligent Systems Reference Library, Springer, Berlin, Heidelberg, 2014, pp. 25–45.
[17] R. Nora, J. L. Peterson, B. K. Spears, J. E. Field, and S. Brandon, *Ensemble simulations of inertial confinement fusion implosions*, Statistical Analysis and Data Mining: The ASA Data Science Journal, 10 (2017), pp. 230–237. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sam.11344.
[18] J. Oden, R. Moser, and O. Ghattas, *Computer predictions with quantified uncertainty, Part I*, SIAM News, 43 (2010).
[19] N. Ohana, A. Jocksch, E. Lanti, T. M. Tran, S. Brunner, C. Gheller, F. Hariri, and L. Villard, *Towards the optimization of a gyrokinetic Particle-In-Cell (PIC) code on large-scale hybrid architectures*, Journal of Physics: Conference Series, 775 (2016), p. 012010. Publisher: IOP Publishing.
[20] A. Y. Pigarov, S. Krasheninnikov, T. Soboleva, and T. Rognlien, *Dust-particle transport in tokamak edge plasmas: Physics of Plasmas: Vol 12, No 12*.
[21] H. Qin, S. Zhang, J. Xiao, J. Liu, Y. Sun, and W. Tang, *Why is Boris algorithm so good*, Physics of Plasmas, 20 (2013), p. 084503.
[22] C. Rea and R. S. Granetz, *Exploratory Machine Learning Studies for Disruption Prediction Using Large Databases on DIII-D*, Fusion Science and Technology, 74 (2018), pp. 89–100. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/15361055.2017.1407206.
[23] B. Ripperda, F. Bacchini, J. Teunissen, C. Xia, O. Porth, L. Sironi, G. Lapenta, and R. Keppens, *A Comprehensive Comparison of Relativistic Particle Integrators*, The Astrophysical Journal Supplement Series, 235 (2018), p. 21.
[24] A. Sherstinsky, *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network*, Physica D: Nonlinear Phenomena, 404 (2020), p. 132306.
[25] R. Smith, *Uncertainty Quantification: Theory, Implementation, and Applications*, Computational Science and Engineering, SIAM, 2013.
[26] B. K. Spears, J. Brase, P.-T. Bremer, B. Chen, J. Field, J. Gaffney, M. Kruse, S. Langer, K. Lewis, R. Nora, J. L. Peterson, J. Jayaraman Thiagarajan, B. Van Essen, and K. Humbird, *Deep learning: A guide for practitioners in the physical sciences*, Physics of Plasmas, 25 (2018), p. 080901.
[27] E. Sánchez, A. Mishchenko, J. M. García-Regaña, R. Kleiber, A. Bottino, L. Villard, and t. W.-X. team, *Nonlinear gyrokinetic PIC simulations in stellarators with the code EUTERPE*, Journal of Plasma Physics, 86 (2020), p. 855860501. arXiv: 2004.14605.
[28] B. D. Tracey, K. Duraisamy, and J. J. Alonso, *A Machine Learning Strategy to Assist Turbulence Model Development*, in 53rd AIAA Aerospace Sciences Meeting, AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, Jan. 2015.
[29] D. Tskhakaya, K. Matyash, R. Schneider, and F. Taccogna, *The Particle-In-Cell Method*, Contributions to Plasma Physics, 47 (2007), pp. 563–594. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ctpp.200710072.

[30] J. Vega, S. Dormido-Canto, J. M. López, A. Murari, J. M. Ramírez, R. Moreno, M. Ruiz, D. Alves, and R. Felton, *Results of the JET real-time disruption predictor in the ITER-like wall campaigns*, Fusion Engineering and Design, 88 (2013), pp. 1228–1231.
[31] Y. Xu, *A general comparison between tokamak and stellarator plasmas*, Matter and Radiation at Extremes, 1 (2016), pp. 192–200.