Non-Intrusive Parallel-in-Time Solvers for Challenging Problems

Research in Industry Projects for Student

Tran Duy Anh Le[1], Bryan Li[2], Margaret Luo[3], Daniel Agraz Vallejo[4]

Academic Mentor: Jean-Michel Maldague[5]
Industry Mentors: Robert Falgout[6], Rui Peng Li[6], Wayne Mitchell[6], Daniel Osei-Kuffuor[6]

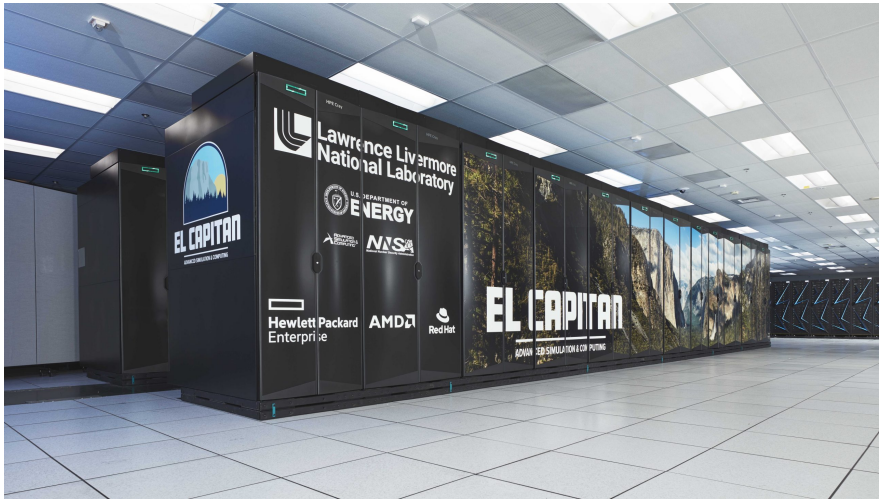
[1]University of Rochester, [2]University of California, Berkeley, [3]University of California, San Diego, [4]CETYS University

[5]University of California, Los Angeles, [6]Lawrence Livermore National Laboratory

August 20, 2024

Lawrence Livermore National Laboratory (LLNL)

Lawrence Livermore National Laboratory (LLNL)



- LLNL is a federally funded research institution located in Livermore, California established in 1952.
- Home of the World's fastest supercomputers including Sierra and El Capitan.
- Over 8,000 people works at LLNL.

LLNL's Center for Applied Scientific Computing (CASC)

- LLNL's branch that hosts 11 research groups that gathers various multidisciplinary teams to conduct scientific research.

Computational Physics Group

Data Science & Analytics Group

High Performance Computing Group

Informatics Group

Machine Intelligence Group

Mathematical Algorithms & Computing Group

Numerical Analysis & Simulations Group

Parallel Systems Group

Research Software Engineering Group

Scientific Computing Group

UQ Optimization Group

The need for parallelism

- Serial algorithms are bottlenecked on single-thread performance.
- Parallel algorithms take advantage of core count, which is growing exponentially.
  - Modern supercomputers (such as LLNL's Sierra) have core counts $\sim 10^6$–10^7.

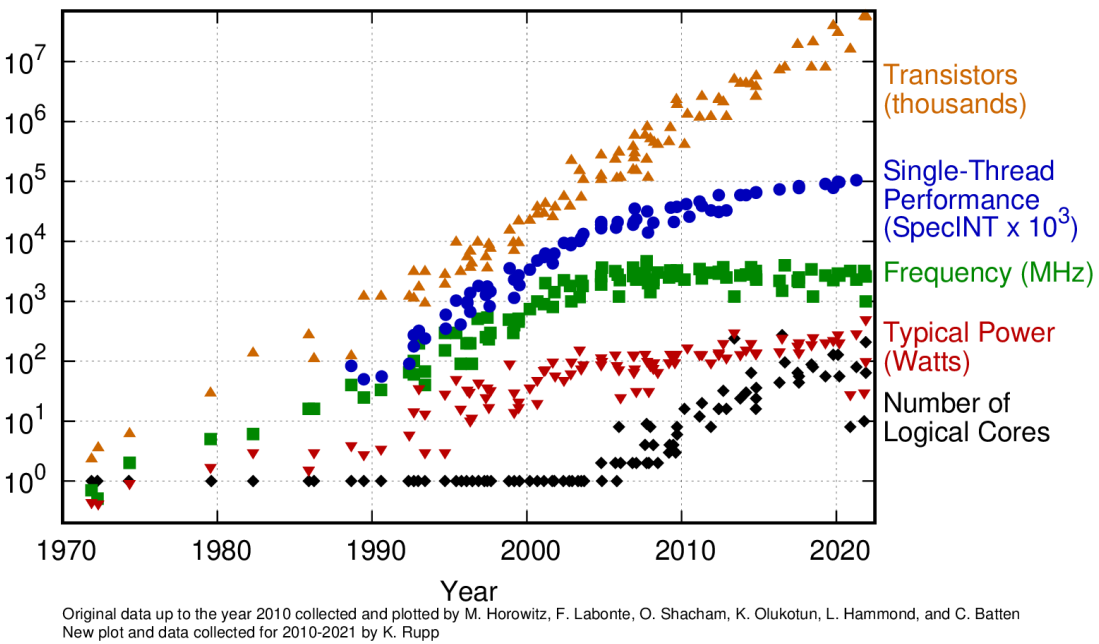


Figure: Stagnation of processor speed

Parallel-in-time algorithms

- Our target is parallel algorithms to simulate **evolutionary partial differential equations.**
- These model many physical phenomena that LLNL is interested in:
  - Heat transfer
  - Wave propagation
  - Quantum evolution
- Serial-in-time algorithms require many timesteps, which cannot be evaluated in parallel.
- Parallel-in-time algorithms evaluate timesteps in parallel, taking advantage of more cores to provide speedup.
  - This is the focus of our project!

Intrusive vs. Non-Intrusive Algorithms

Intrusive:

- Significant effort and additional code required to apply to new problems.
- Approach usually very heavily **problem-dependent.**

Intrusive vs. Non-Intrusive Algorithms

Intrusive:

- Significant effort and additional code required to apply to new problems.
- Approach usually very heavily **problem-dependent.**

Non-Intrusive:

- Little effort and additional code required to apply to new problems.
- Approach/algorithm **generalizes** to new problems.

Discretization using finite difference

- To deal with PDEs using a computer, we must first discretize them.
- For example, consider the 1D *steady-state diffusion* (Poisson) equation

$$-u_{xx} = f\,.$$

on the domain $[0, 1]$.

Discretization using finite difference

- To deal with PDEs using a computer, we must first discretize them.
- For example, consider the 1D *steady-state diffusion* (Poisson) equation

$$-u_{xx} = f\,.$$

  on the domain $[0, 1]$.
- Divide domain up into N equal pieces of length $\delta x = 1/N$.
- Set nodes $x_i = i\delta x$ for $i = 0, \ldots, N$, and let $u_i \approx u(x_i)$, $f_i = f(x_i)$.

Discretization using finite difference

- To deal with PDEs using a computer, we must first discretize them.
- For example, consider the 1D *steady-state diffusion* (Poisson) equation

$$-u_{xx} = f.$$

  on the domain $[0, 1]$.
- Divide domain up into N equal pieces of length $\delta x = 1/N$.
- Set nodes $x_i = i\delta x$ for $i = 0, \ldots, N$, and let $u_i \approx u(x_i)$, $f_i = f(x_i)$.
- We can use the *finite difference approximation*

$$-u_{xx}(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{\delta x^2}.$$

Discretization using finite difference

- Using this approximation, for $i = 1, \ldots, N-1$, we obtain

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{\delta x^2} = f_i.$$

- Assuming Dirichlet boundary conditions $u_0 = u_N = 0$, we can write this in a linear system $A\mathbf{u} = \mathbf{f}$:

$$\frac{1}{\delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & -1 & \ddots & & 0 \\ \vdots & & & & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix}.$$

Relaxation methods

- Finite difference discretization produces a linear system we need to solve.
- Linear systems formed from discretization can be very large.
- Directly solving (e.g. Gaussian elimination) results in $O(n^3)$ runtime: too slow!
- **Relaxation methods**, which are iterative methods for solving linear systems, typically with $O(n^2)$ or better runtime per iteration, are used instead.

Relaxation methods and Multigrid

Relaxation methods are better at correcting oscillatory errors, but are worse at correcting low-frequency errors.

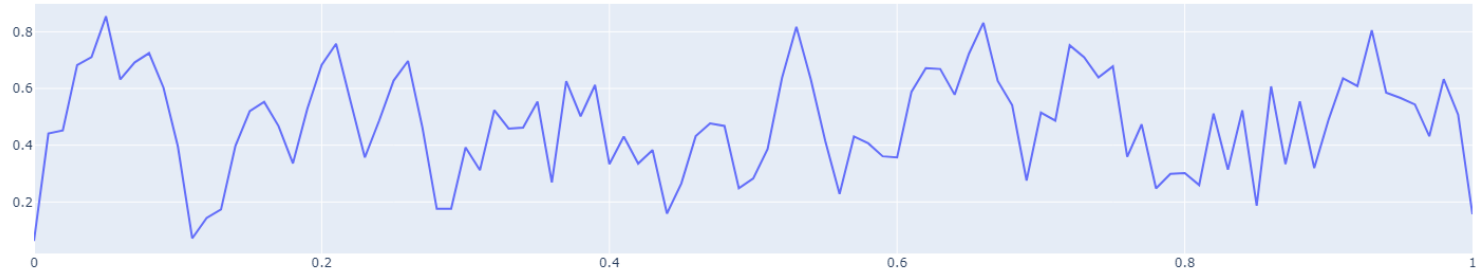


Figure: Fine Grid Iteration 0

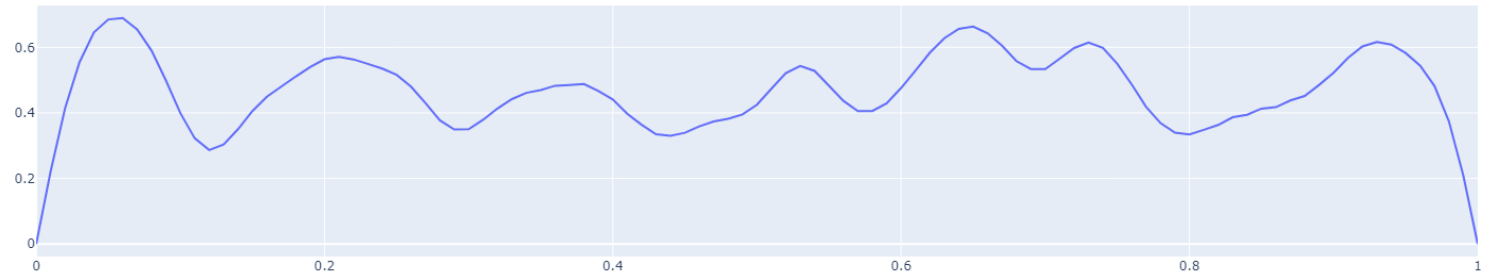


Figure: Fine Grid Iteration 5

Multigrid: the V-cycle

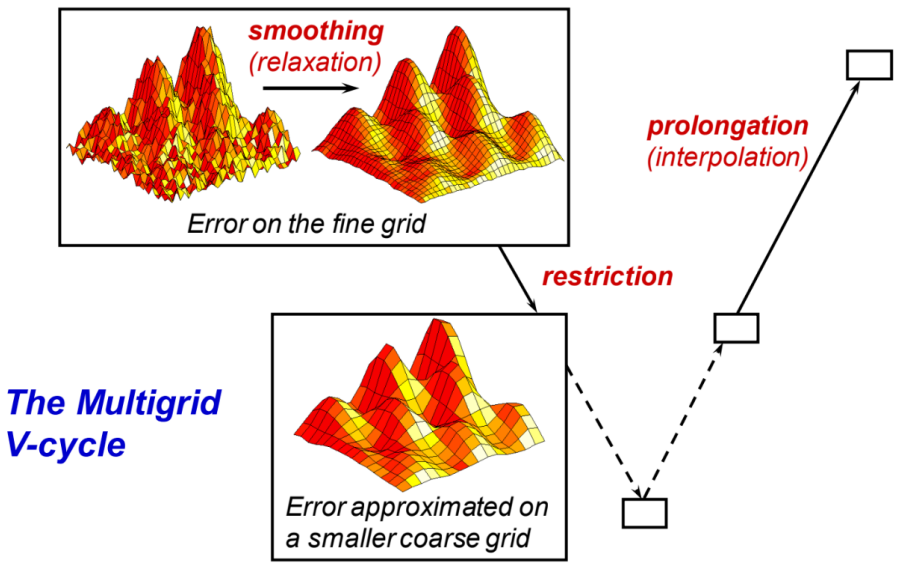


Figure: Multigrid V-cycle

- Multigrid attacks the weakness of relaxation methods by going through a hierarchy of grid sizes.

Multigrid: the V-cycle

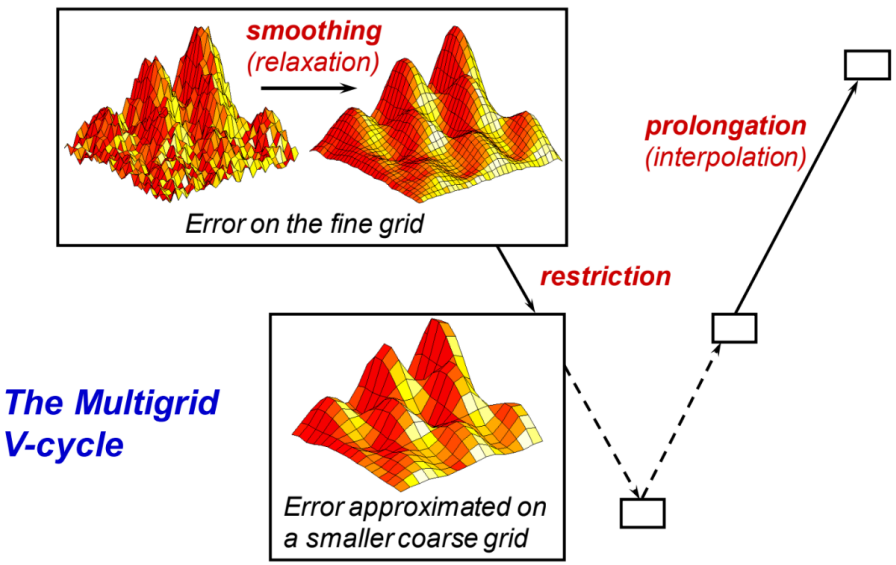


Figure: Multigrid V-cycle

- Multigrid attacks the weakness of relaxation methods by going through a hierarchy of grid sizes.
- Low-frequency (smooth) errors on the fine grid turn into high-frequency (oscillatory) error on the coarser grid and are corrected more effectively through relaxation.

Relaxation methods and Multigrid

Relaxation methods are better at correcting oscillatory errors.

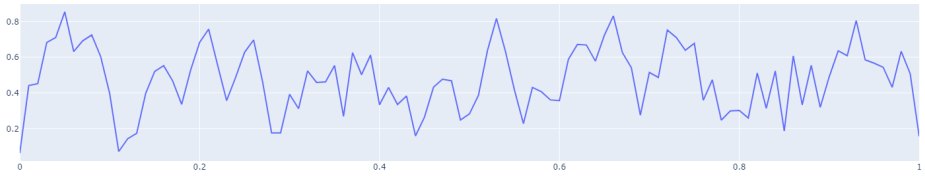


Figure: Fine Grid Iteration 0

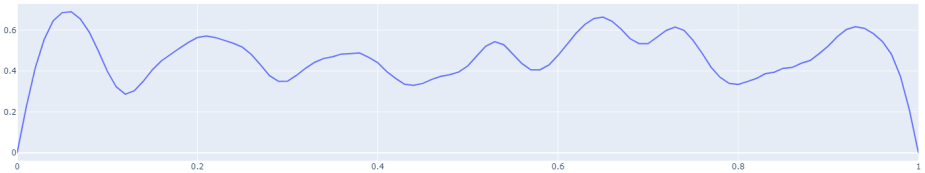


Figure: Fine Grid Iteration 5

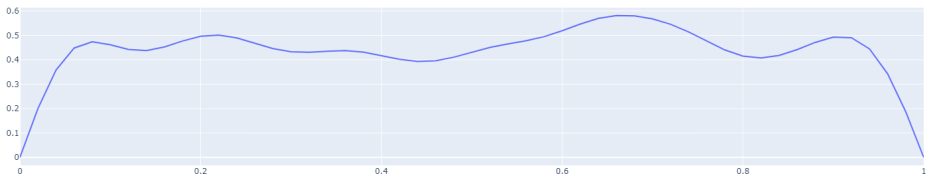


Figure: Coarse Grid Iteration 0

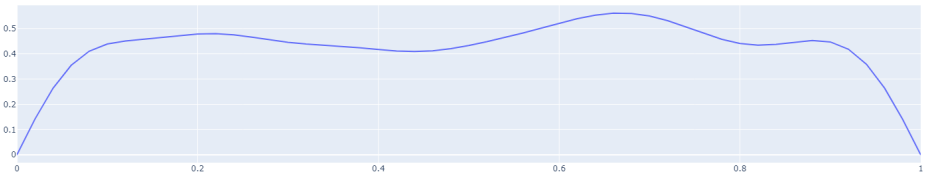


Figure: Coarse Grid Iteration 5

Multigrid reduction in time (MGRIT)

MGRIT algorithm is a parallel-in-time approach for solving time dependent problems that is designed to be non-intrusive.

MGRIT

Consider the diffusion equation $u_t - u_{xx} = f$ with the following discretization

$$u_t \approx \frac{u_i^{j+1} - u_i^j}{\delta t}, \quad -u_{xx} \approx \frac{-u_{i-1}^j + 2u_i^j - u_{i+1}^j}{\delta x^2}$$

where $u_i^j \approx u(x_i, t_j)$.

Using Forward Euler, we get the update rule

$$\mathbf{u}^j = \Phi \mathbf{u}^{j-1} + \delta t \mathbf{f}^{j-1} \text{ for } j = 1, 2 \ldots, n \text{ and } \mathbf{u}^0 = \mathbf{g}^0,$$

where $\mathbf{u}^j \approx \begin{bmatrix} u_0^j & u_1^j & \cdots & u_N^j \end{bmatrix}^T$ and Φ has the stencil $[\beta \;\; 1-2\beta \;\; \beta]$, where $\beta = \delta t / \delta x^2 \leq 1/2$ by stability condition.

MGRIT: the coarse-grid operator

Using the partition below, we let $t_j = j\delta t$ for the fine time grid and $T_j = mj\delta t$ for the coarse time grid with coarsening factor m.

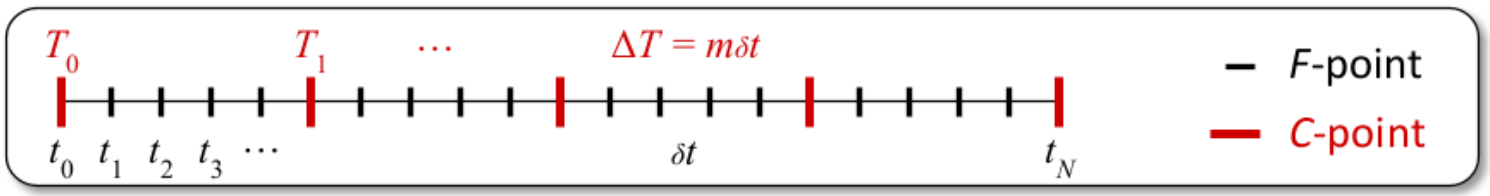


Figure: Coarse and fine time grids

Then to solve each coarse time step is to solve the following system

$$T(m)\mathbf{u} = \begin{pmatrix} I & & & & \\ -\Phi^m & I & & & \\ & -\Phi^m & I & & \\ & & \ddots & \ddots & \\ & & & -\Phi^m & I \end{pmatrix} \begin{pmatrix} u_0 \\ u_m \\ u_{2m} \\ \vdots \\ u_N \end{pmatrix} = \mathbf{f}.$$

Approximating the coarse-grid operator

However, Φ^m is expensive to compute with, thus we ultimately want to **approximate Φ^m with a sparser operator Ψ.**

Approximating the coarse-grid operator

However, Φ^m is expensive to compute with, thus we ultimately want to **approximate Φ^m with a sparser operator Ψ.**

So we have the "target matrix" $T(m)$ which we want to approximate with some matrix A

$$T(m) = \begin{pmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{pmatrix}, \quad A = \begin{pmatrix} I & & & \\ -\Psi & I & & \\ & \ddots & \ddots & \\ & & -\Psi & I \end{pmatrix}.$$

Approximating the coarse-grid operator (cont'd)

$$T(m) = \begin{pmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{pmatrix}, \quad A = \begin{pmatrix} I & & & \\ -\Psi & I & & \\ & \ddots & \ddots & \\ & & -\Psi & I \end{pmatrix}$$

For a good approximation, we want A to be *spectrally equivalent* to $T(m)$, i.e. the following is bounded independent of problem size

$$\|I - AT^{-1}\|.$$

$$\begin{aligned} \|I - AT^{-1}\| &= \sup_x \frac{\|(I - AT^{-1})x\|}{\|x\|} \\ &= \sup_y \frac{\|(T - A)y\|}{\|Ty\|} \end{aligned}$$

Why Machine Learning?

We will utilize Machine Learning techniques for this project.

Why Machine Learning?

We will utilize Machine Learning techniques for this project.

- Current approaches of approximating Φ^m applicable to only certain types of PDEs.

Why Machine Learning?

We will utilize Machine Learning techniques for this project.

- Current approaches of approximating Φ^m applicable to only certain types of PDEs.
- We want a "non-intrusive" way to estimate coarse-grid operators.

Why Machine Learning?

We will utilize Machine Learning techniques for this project.

- Current approaches of approximating Φ^m applicable to only certain types of PDEs.
- We want a "non-intrusive" way to estimate coarse-grid operators.
- Machine learning is good at coming up at more general solutions.

Neural Network and PyMGRIT

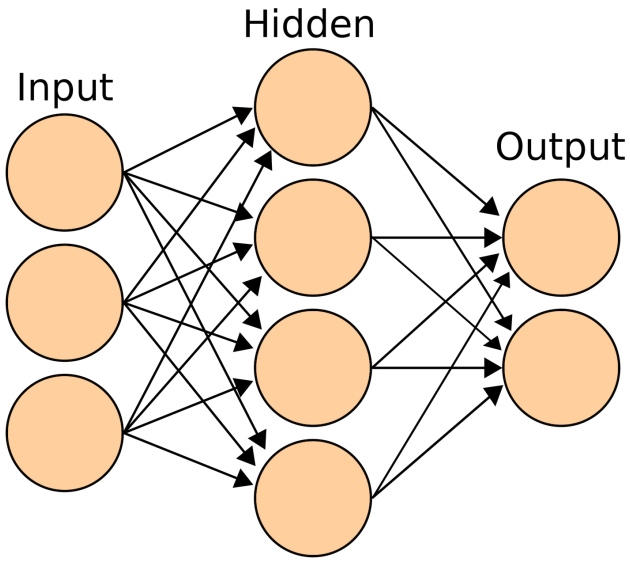
- Construct a Neural Network whose output is spectrally equivalent to the target matrix

Neural Network and PyMGRIT

- Construct a Neural Network whose output is spectrally equivalent to the target matrix
- Evaluate the performance of the Neural Network model integrated into PyMGRIT

Neural Network Architecture

For 1D diffusion problem:
Input: $(\phi, m) \in \mathbb{R}^3 \times \mathbb{N}$.
Output: $\psi \in \mathbb{R}^3$.



ϕ is the "stencil" of Φ. For example, in our periodic problem,

$$L = \frac{1}{\delta x^2} \begin{bmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ -1 & & & -1 & 2 \end{bmatrix},$$

and $\Phi = I - \delta t L$ has stencil
$\phi = \begin{bmatrix} \beta & 1 - 2\beta & \beta \end{bmatrix}$ where $\beta = \delta t / \delta x^2$

Loss Function

- Goal: Find an A that's *spectrally equivalent* to T. In other words, minimize $\|I - AT^{-1}\|$.

Loss Function

- Goal: Find an A that's *spectrally equivalent* to T. In other words, minimize $\|I - AT^{-1}\|$.
- Recall, the ℓ^2 norm of a vector $v \in \mathbb{R}^d$ is $\|v\|_{\ell^2}^2 = v_1^2 + \cdots + v_d^2$.

Loss Function

- Goal: Find an A that's *spectrally equivalent* to T. In other words, minimize $\|I - AT^{-1}\|$.
- Recall, the ℓ^2 norm of a vector $v \in \mathbb{R}^d$ is $\|v\|_{\ell^2}^2 = v_1^2 + \cdots + v_d^2$.
- The norm of a matrix is defined as the induced 2-norm (or spectral norm)

$$\|A\| = \sup_{0 \neq v \in \mathbb{R}^d} \frac{\|Av\|_{\ell^2}}{\|v\|_{\ell^2}}.$$

Loss Function

- Goal: Find an A that's *spectrally equivalent* to T. In other words, minimize $\|I - AT^{-1}\|$.
- Recall, the ℓ^2 norm of a vector $v \in \mathbb{R}^d$ is $\|v\|_{\ell^2}^2 = v_1^2 + \cdots + v_d^2$.
- The norm of a matrix is defined as the induced 2-norm (or spectral norm)

$$\|A\| = \sup_{0 \neq v \in \mathbb{R}^d} \frac{\|Av\|_{\ell^2}}{\|v\|_{\ell^2}}.$$

- Equivalently, this is the largest magnitude of eigenvalue of A.

Loss Function 1

By choosing some vectors $\{y_k\}_{k=1}^{K}$, we can get the following loss function

Loss Function 1

By choosing some vectors $\{y_k\}_{k=1}^K$, we can get the following loss function

1st Loss function

$$\mathcal{L}_1 = \frac{\|Ty_k - Ay_k\|^2}{\|Ty_k\|^2} = \frac{\sum_r (T_r y_k - A_r y_k)^2}{\sum_r (T_r y_k)^2} \leq \sum_r \frac{(T_r y_k - A_r y_k)^2}{(T_r y_k)^2}$$

where T_r, A_r denote row r of T and A respectively.

Loss Function 1

By choosing some vectors $\{y_k\}_{k=1}^K$, we can get the following loss function

1st Loss function

$$\mathcal{L}_1 = \frac{\|Ty_k - Ay_k\|^2}{\|Ty_k\|^2} = \frac{\sum_r (T_r y_k - A_r y_k)^2}{\sum_r (T_r y_k)^2} \leq \sum_r \frac{(T_r y_k - A_r y_k)^2}{(T_r y_k)^2}$$

where T_r, A_r denote row r of T and A respectively.

By focusing on one row, and letting $y_k(i)$ be the restriction of y_k to the coarse time point i, we obtain a simplified expression

$$\mathcal{L}_1' = \sum_{k=1}^K \frac{((\Phi^m)_r y_k - \Psi_r y_k(i))^2}{(1 - (\Phi^m)_r y_k(i))^2}.$$

Loss Function 2

Our second idea for a loss function is inspired by Ru Huang et al (2023).

Loss Function 2

Our second idea for a loss function is inspired by Ru Huang et al (2023). Let (λ_k, y_k) be the orthogonal eigenpairs of Φ^m, sorted such that $|\lambda_i| \geq |\lambda_{i+1}|$. Let $K \leq d$ be the number of testing vectors.

Loss Function 2

Our second idea for a loss function is inspired by Ru Huang et al (2023). Let (λ_k, y_k) be the orthogonal eigenpairs of Φ^m, sorted such that $|\lambda_i| \geq |\lambda_{i+1}|$. Let $K \leq d$ be the number of testing vectors.
Then we get the following loss function

2nd Loss Function

$$\mathcal{L}_2 = \sum_{k=1}^{K} \|\Phi^m y_k - \Psi y_k\|_{\ell^2}^2.$$

Loss Function 3

Another loss function is developed similarly to the second one.

Loss Function 3

Another loss function is developed similarly to the second one. Consider (λ_k, y_k) be orthogonal eigenpairs of Φ, sorted such that $|\lambda_i| \geq |\lambda_{i+1}|$. Let $K \leq d$ be the number of testing vectors.

Loss Function 3

Another loss function is developed similarly to the second one.
Consider (λ_k, y_k) be orthogonal eigenpairs of Φ, sorted such that $|\lambda_i| \geq |\lambda_{i+1}|$. Let $K \leq d$ be the number of testing vectors.
By expanding out the ℓ_2 norm of the vectors, if M is the number of block rows of $T(V)$, we have a new loss function.

3rd Loss Function

$$\mathcal{L}_{3,M} = \sum_{k=1}^{K} \frac{(M-1)\|(\Psi - \Phi^m)v_k\|_{\ell_2}^2}{1 + (M-1)|1 - \lambda_k^m|^2}$$

Currently, M is chosen to be a sufficiently large number or $M = \frac{N}{m} + 1$ where N is a large number and m is the coarsening factor.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.
- For example, for the equation $u_t = u_{xx}$, if we consider N gridpoints, then the eigenvalues are given by

$$\lambda_k = 1 - 2\beta\left(1 - \cos\left(\frac{k\pi}{N+1}\right)\right), 1 \leq k \leq N$$

where $\beta = \delta t/\delta x^2 \leq 1/2$.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.
- For example, for the equation $u_t = u_{xx}$, if we consider N gridpoints, then the eigenvalues are given by

$$\lambda_k = 1 - 2\beta\left(1 - \cos\left(\frac{k\pi}{N+1}\right)\right), 1 \leq k \leq N$$

  where $\beta = \delta t/\delta x^2 \leq 1/2$.
- This makes approximation for Ψ easier as smaller eigenvalues' decay faster.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.
- For example, for the equation $u_t = u_{xx}$, if we consider N gridpoints, then the eigenvalues are given by

$$\lambda_k = 1 - 2\beta\left(1 - \cos\left(\frac{k\pi}{N+1}\right)\right), 1 \leq k \leq N$$

  where $\beta = \delta t/\delta x^2 \leq 1/2$.
- This makes approximation for Ψ easier as smaller eigenvalues' decay faster.
- However, the distribution of $|\lambda_k|$ varies between different types of PDEs (hyperbolic, elliptic, etc).

Loss Functions we are comparing

- $\mathcal{L}_1 = \sum_{k=1}^{K} \frac{((\Phi^m)_r y_k - \Psi_r y_k(i))^2}{(1-(\Phi^m)_r y_k(i))^2 + \epsilon}$ where y_k are random vectors with an additional constant vector of all 1's, and ϵ is added to avoid dividing by zero.
- $\mathcal{L}_1 = \sum_{k=1}^{K} \frac{((\Phi^m)_r y_k - \Psi_r y_k(i))^2}{(1-(\Phi^m)_r y_k(i))^2 + \epsilon}$ where y_k are largest eigenvectors of Φ.
- $\mathcal{L}_2 = \sum_{k=1}^{K} \|\Phi^m y_k - \Psi y_k\|_{\ell^2}^2$ where y_k are largest eigenvectors of Φ.
- $\mathcal{L}_{3,M} = \sum_{k=1}^{K} \frac{(M-1)\|\Phi^m y_k - \Psi y_k\|_{\ell^2}^2.}{(1+(M-1)|\lambda_k^m - 1|)^2}$ where (λ_k, y_k) are largest pairs of eigenvalues and eigenvectors of Φ
- $\mathcal{L}_{3,\text{Hybrid}}$, which has a similar structure to $\mathcal{L}_{3,M}$ but we update M during training.

Loss Functions we are comparing (cont'd)

We also consider modifications of the preceding loss functions, including:

- Using eigenvectors instead of random vectors for $\mathcal{L}_1$.
- Adding the constant vector of all 1s to the test vector list for $\mathcal{L}_2, \mathcal{L}_3$ and $\mathcal{L}_1$ with eigenvectors.

Loss Function Landscapes

Consider the problem set up with

$$\delta x = \frac{1}{16}, \quad \delta t = \frac{1}{4096}, \quad \beta = \frac{1}{16} \quad m = 4$$

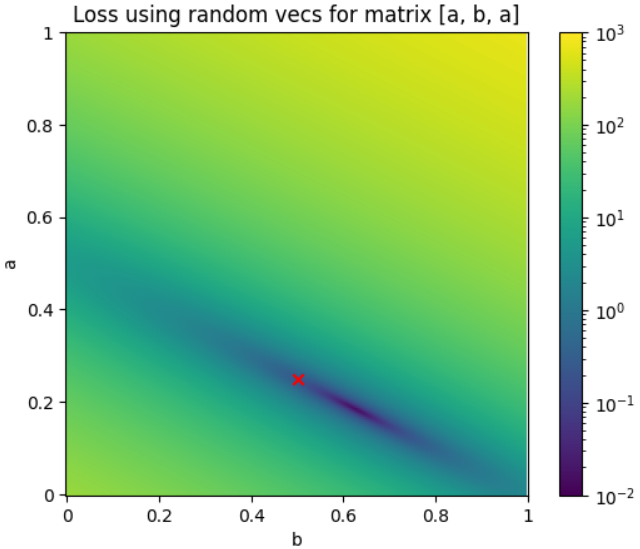
The stencil for $\Phi = \begin{bmatrix} 0.0625 & 0.8750 & 0.0625 \end{bmatrix}$, and our input to the loss function is $[0.0625, 0.8750, 0.0625, 4.0000]$.
The re-discretization stencil is $\begin{bmatrix} 0.2500 & 0.5000 & 0.2500 \end{bmatrix}$

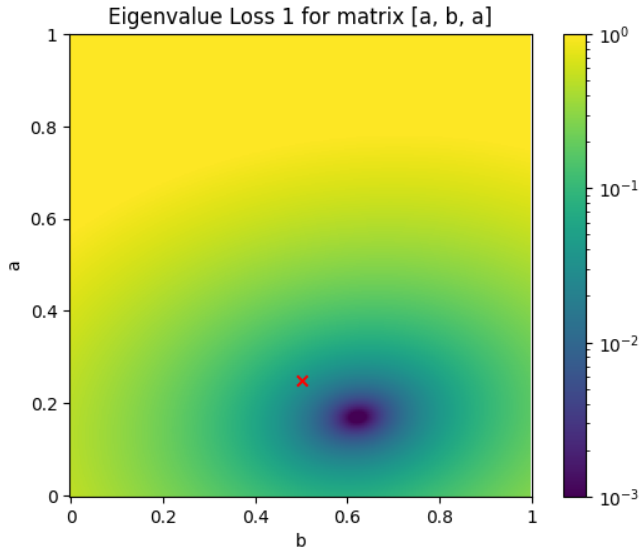
If we restrict the approximation Ψ to have stencil of the form $\begin{bmatrix} a & b & a \end{bmatrix}$, we can plot the loss at (a, b) for each loss function.

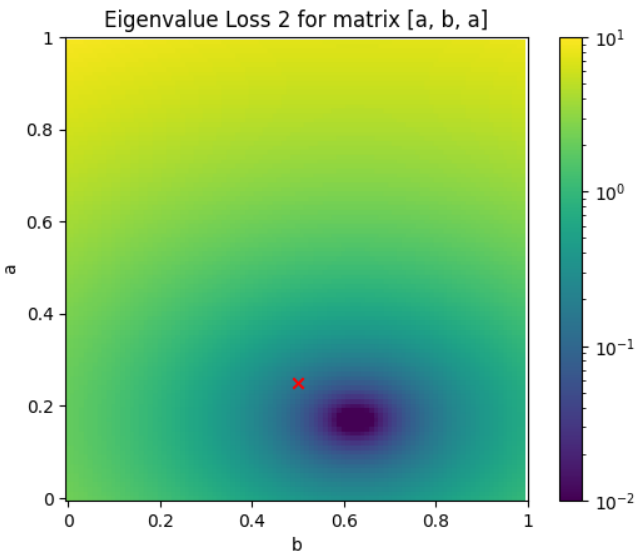
This allows us to observe the loss landscape for each loss function, as well as where the re-discretization stencil, which we know converges well in PyMGRIT, lies.

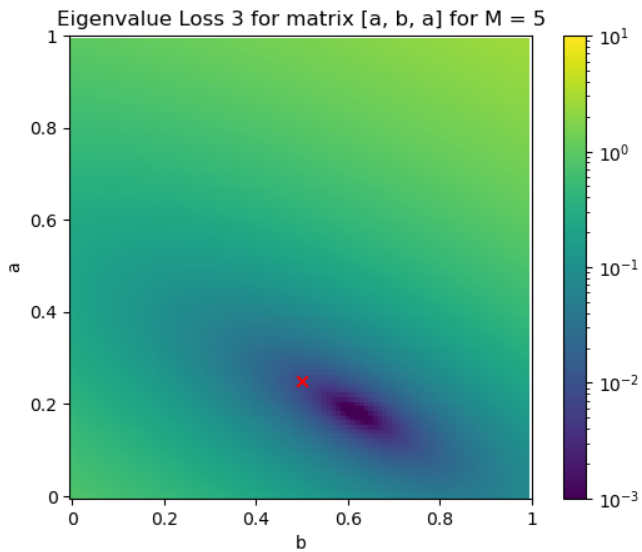
Loss Function Landscapes (cont'd)

x is the re-discretization stencil.









Loss Function Landscapes (cont'd)

$\mathcal{L}_1$ with eigenvectors vs eigenvectors and constant vector.

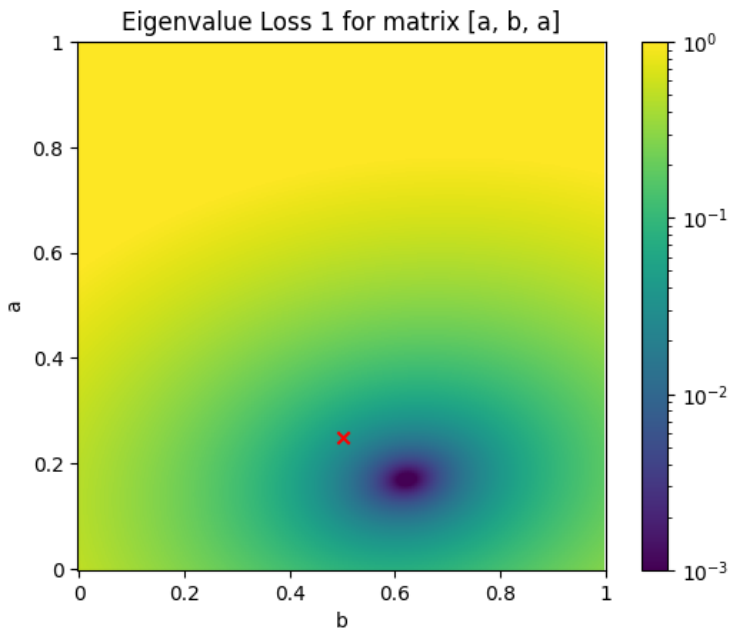


Figure: $\mathcal{L}_1$ with evecs

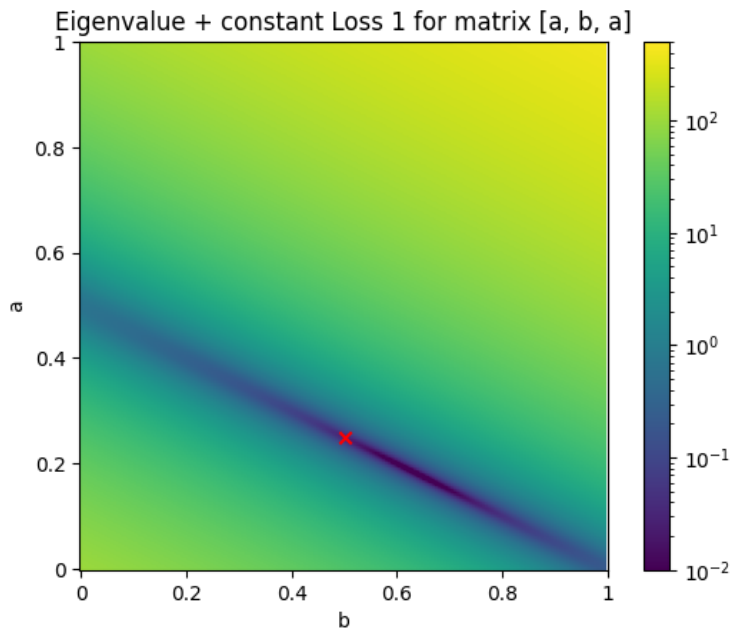


Figure: $\mathcal{L}_1$ with evecs and the constant vector

Loss Function Landscapes (cont'd)

$\mathcal{L}_2$ with eigenvectors vs eigenvectors and constant vector.

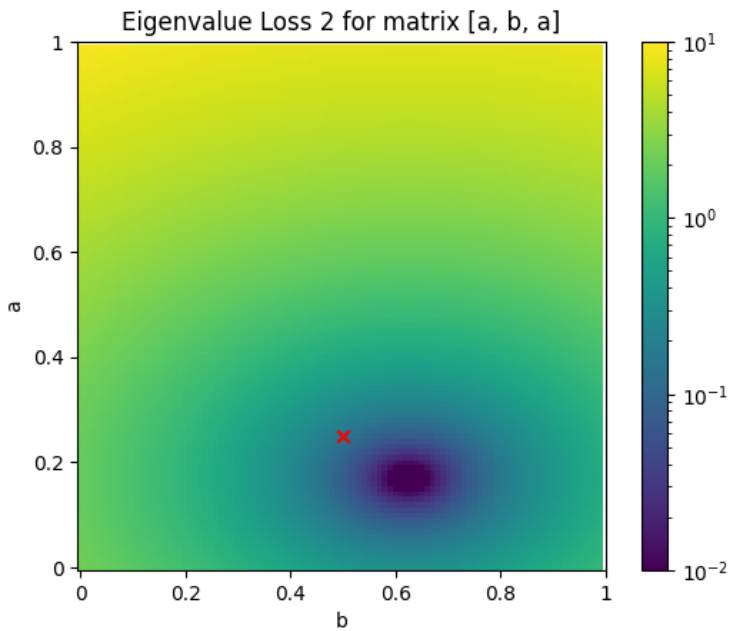


Figure: $\mathcal{L}_2$ with evecs

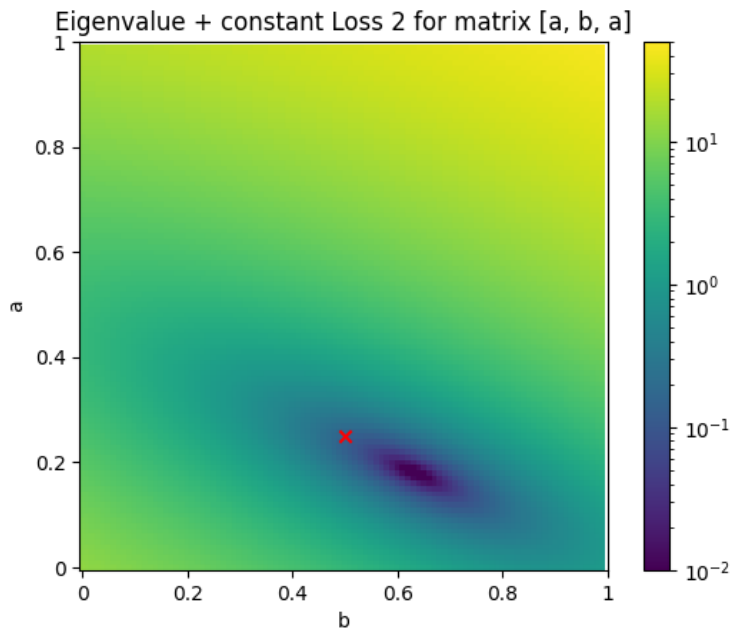


Figure: $\mathcal{L}_2$ with evecs and the constant vector

Loss Function Landscapes (cont'd)

$\mathcal{L}_{3,5}$ with eigenvectors vs eigenvectors and constant vector.

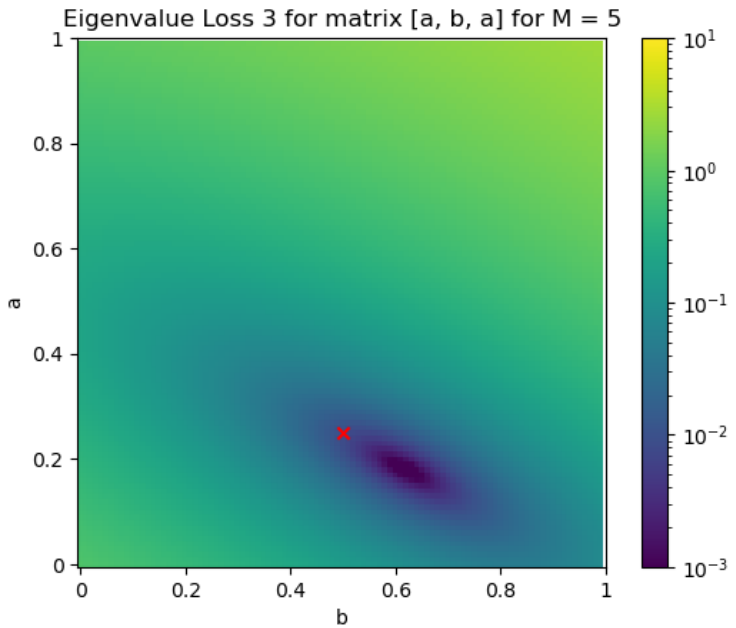


Figure: $\mathcal{L}_{3,5}$ with evecs

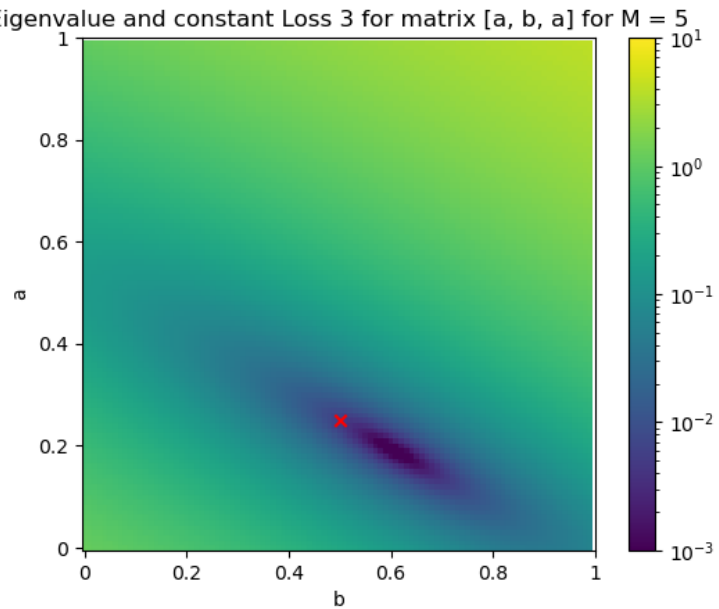


Figure: $\mathcal{L}_{3,5}$ with evecs and the constant vector

Performance of optimal stencil in PyMGRIT

$$\delta x = \frac{1}{16}, \quad \delta t = \frac{1}{4096}, \quad m = 4$$

Loss Function	Optimal Stencil			Residual	Conv. Factor
$\mathcal{L}_1$ Random	[0.1814	0.6324	0.1814]	6.60e-5	4.04e-1
$\mathcal{L}_1$ EVecs	[0.1700	0.6221	0.1700]	2.13e-2	9.40e-1
$\mathcal{L}_1$ EVecs, const.	[0.1824	0.6341	0.1824]	6.60e-6	2.82e-1
$\mathcal{L}_2$	[0.1705	0.6223	0.1705]	2.11e-2	9.59e-1
$\mathcal{L}_2$ const.	[0.1797	0.6346	0.1797]	1.55e-4	4.17e-1
$\mathcal{L}_{3,5}$	[0.1891	0.6071	0.1891]	6.82e-2	9.03e-1
$\mathcal{L}_{3,5}$, const.	[0.1977	0.5945	0.1977]	3.32e-2	7.62e-1
$\mathcal{L}_{3,20}$	[0.2028	0.5895	0.2028]	1.63e-2	4.48e-1
$\mathcal{L}_{3,20}$, const.	[0.2061	0.5840	0.2061]	2.87e-3	3.47e-1
Re-disc	[0.2500	0.5000	0.2500]	1.73e-8	9.93e-2

Neural Network Testing Setup

Training set: $\beta \in \{1/8, 1/12, 1/16, 1/24\}$, $m = \{1, 2, 3, 4\}$ (4 stencils).
Test problem: $\beta = 1/10$, $\delta x = 1/16, \delta t = 1/2560$. In PyMGRIT, $n_t = 2561$, $n_x = 17$.

We use a feedforward neural network with 3 hidden layers of 50 neurons each with the LeakyReLU activation function.

Neural Network Results ($m = 2$)

Loss Function	Output Stencil			Output Loss	Redisc. Loss
$\mathcal{L}_1$ Random	[0.131	0.743	0.131]	0.177	0.143
$\mathcal{L}_1$ Evecs	[0.127	0.726	0.121]	6.75e-3	6.15e-3
$\mathcal{L}_1$ Evecs, const.	[0.159	0.693	0.150]	1.21e-3	4.83e-3
$\mathcal{L}_2$	[0.128	0.733	0.120]	4.03e-2	3.48e-2
$\mathcal{L}_2$ const.	[0.130	0.738	0.125]	4.15e-2	3.48e-2
$\mathcal{L}_{3,5}$	[0.209	0.509	0.255]	9.64e-3	8.07e-4
$\mathcal{L}_{3,5}$ const.	[0.159	0.664	0.165]	1.11e-3	3.62e-3
$\mathcal{L}_{3,10}$	[0.223	0.505	0.256]	7.26e-3	5.67e-4
$\mathcal{L}_{3,10}$ const.	[0.168	0.666	0.166]	1.53e-3	2.76e-3
$\mathcal{L}_{3,20}$	[0.239	0.485	0.266]	5.97e-3	3.40e-4
$\mathcal{L}_{3,20}$ const.	[0.196	0.619	0.199]	5.97e-3	3.40e-4
$\mathcal{L}_{3,\text{Hybrid}}$	[0.246	0.473	0.275]	5.14e-3	2.48e-4
$\mathcal{L}_{3,\text{Hybrid}}$ const.	[0.243	0.420	0.334]	5.05e-2	1.47e-3
Re-disc.	[0.200	0.600	0.200]	—	—

PyMGRIT Results ($m = 2$)

Loss Function	Output Stencil			Residual	Conv. Factor
$\mathcal{L}_1$ Random	[0.131	0.743	0.131]	2.28	5.74e-1
$\mathcal{L}_1$ Evecs	[0.127	0.726	0.121]	1.85e-2	9.42e-1
$\mathcal{L}_1$ Evecs, const.	[0.159	0.693	0.150]	2.07e-4	4.03e-1
$\mathcal{L}_2$	[0.128	0.733	0.120]	1.13e-2	9.03e-1
$\mathcal{L}_2$ const.	[0.130	0.738	0.125]	1.69e-3	6.3e-1
$\mathcal{L}_{3,5}$	[0.209	0.509	0.255]	9.25e-2	9.86e-1
$\mathcal{L}_{3,5}$ const.	[0.159	0.664	0.165]	6.40e-2	9.13e-1
$\mathcal{L}_{3,10}$	[0.223	0.505	0.256]	7.84e-2	9.57e-1
$\mathcal{L}_{3,10}$ const.	[0.168	0.666	0.166]	2.19e-8	1.67e-1
$\mathcal{L}_{3,20}$	[0.239	0.485	0.266]	5.08e-2	8.64e-1
$\mathcal{L}_{3,20}$ const.	[0.196	0.619	0.199]	1e12	1.75
$\mathcal{L}_{3,\text{Hybrid}}$	[0.246	0.473	0.275]	1.43e-2	6.57e-1
$\mathcal{L}_{3,\text{Hybrid}}$ const.	[0.243	0.420	0.334]	1.54e-2	6.81e-1
Re-disc.	[0.200	0.600	0.200]	7.60e-8	6.77e-2

Further research recommendations

- 1D Diffusion only proof-of-concept – generalize to more types of PDEs, such as advection (which is harder for MGRIT).
- Currently, Ψ hard-coded to be tridiagonal. For larger coarsening factors, larger stencil may result in better convergence. Use second neural network to select nonzero entries of Ψ, following Ru Huang et als' paper.
- Use bootstrapping ideas:
  - Use the output to train the model.
  - Solve $A\mathbf{u} = \mathbf{0}$ to generate error vectors and update the training set with them.
- Work on periodic condition.

Acknowledgement

Figure: Lawrence Livermore National Laboratory Site Visit Photo

Q & A

- Thank you for listening!
- Questions?