

# Learning iterative methods through gradient-based optimization

18th Copper Mountain Conference On Iterative Methods

---

Nicolas Nytko<sup>1</sup>   Ali Taghibakhshi<sup>1</sup>   Tareq Uz Zaman<sup>2</sup>   Scott  
MacLachlan<sup>2</sup>   Luke Olson<sup>1</sup>   Matthew West<sup>1</sup>

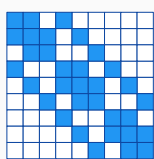
April 17, 2024

<sup>1</sup>University of Illinois at Urbana-Champaign

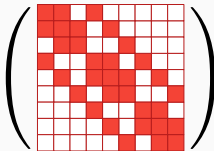
<sup>2</sup>Memorial University of Newfoundland

# Optimization Problems in Numerics

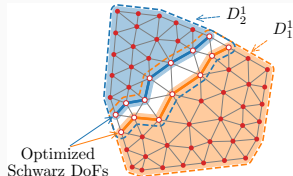
- Sparse Approximate Inverses
- Optimal Boundary Conditions for Schwarz Preconditioners<sup>1</sup>
- Coefficients for Jacobi Method



$\approx$



$-1$



<sup>1</sup>Taghibakhshi et al., "Learning Interface Conditions in Domain Decomposition Solvers". NeurIPS 2022. (Figure is from paper)

## Notation, Stating the optimization problem

- Define loss/objective function,

$$\arg \min_{\theta} f(\theta). \quad (1)$$

- How to compute minimum?
- Take gradient,  $\nabla_{\theta} f$ , run gradient descent steps

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} f. \quad (2)$$

- Simple in theory...

## Taking the gradient

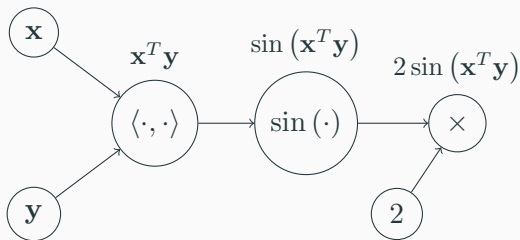
- How do we take derivatives of complex function?
- Assume they are compositions of simpler functions and apply chain-rule

$$f(\mathbf{x}, \mathbf{y}) = 2 \sin(\mathbf{x}^T \mathbf{y})$$

$$\frac{df}{d\mathbf{x}} = 2 \frac{d \sin(z)}{dz} \frac{d(\mathbf{x}^T \mathbf{y})}{d\mathbf{x}} = 2 \cos(\mathbf{x}^T \mathbf{y}) \mathbf{y}$$

## But I don't want to take a derivative by hand

- Automatic differentiation has been around since early 50's <sup>2</sup>
- Perform original calculation as normal, record intermediate operations (forward pass)

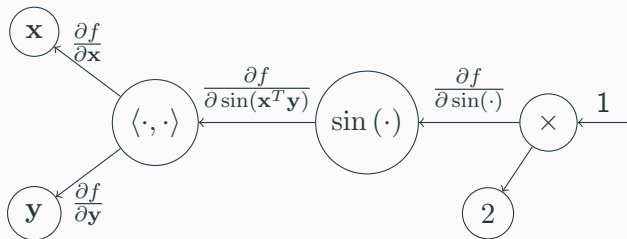


---

<sup>2</sup>John F. Nolan. "Analytical differentiation on a digital computer". Massachusetts Institute of Technology, 1953.

# Automatic Differentiation

- Flow gradient *backwards* by multiplying by transpose Jacobian through each node
- This is called *reverse-mode*, or *adjoint mode* differentiation
- Implemented in big frameworks: PyTorch, Tensorflow, etc.



# Detour: Sparse Matrix Operations

- Breadth of literature on the *forward mode* for sparse matrix ops<sup>345</sup>
  - Matrix matrix products, direct solves, etc.
- What about differentiation through sparse matrix expressions?
- What should that entail?

---

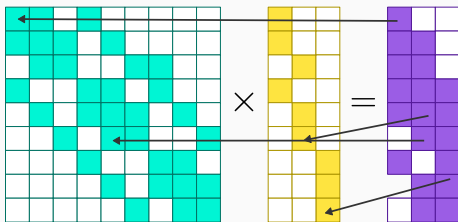
<sup>3</sup>Bank et al., "Sparse matrix multiplication package (SMMP)". Advances in Computational Mathematics, 1993

<sup>4</sup>Demmel et al., "A Supernodal Approach to Sparse Pivoting". SIAM Journal on Matrix Analysis and Applications, 1999

<sup>5</sup>Dalton et al., "Optimizing Sparse Matrix–Matrix Multiplication for the GPU". ACM Transactions on Mathematical Software, 2015

# Differentiation through Sparse Matrices?

- Large frameworks don't fully support sparse inputs
- Autograd on dense matrices gives dense gradient
- Not scalable, want to keep things sparse
- Ideally, differentiate wrt nozeros of sparse input



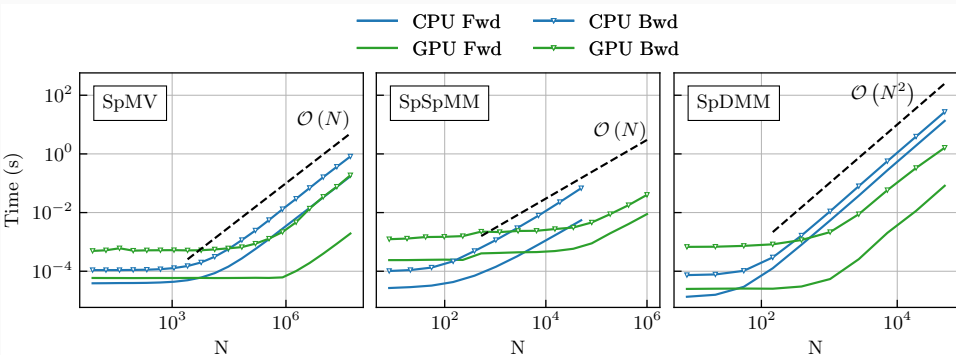


# Differentiable Sparse Kernels

- No general support exists (special cases for Graph-Nets, etc.), so we rolled our own
- Framework for PyTorch for general CSR support with differentiation
- Supports expressions that consist of primitives like:
  - SpMV, SpSpMM, SpDMM
  - Triangular solve, direct solve
  - Analogous to SciPy's sparse or CuPy with support for autodiff
- CUDA support for parallelization
- <https://github.com/nicknytko/numml>

# Timings, Scalings

Have scalable forward and backward pass on both CPU and GPU



## Mini-example of Sparse Optimization

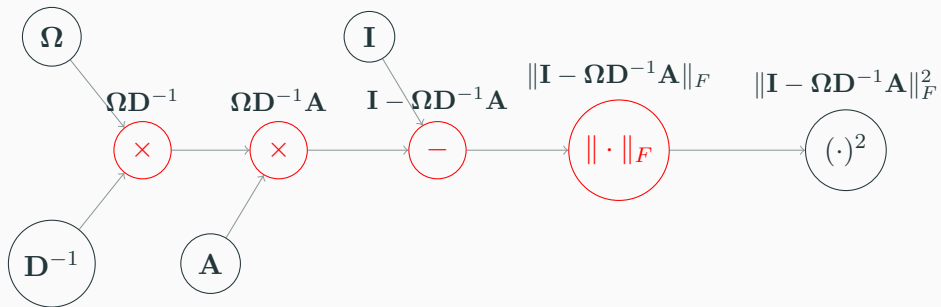
Consider Jacobi relaxation, we'll use a per-entry weight  $\Omega$ ,

$$\mathbf{x} \leftarrow \mathbf{x} + \Omega \mathbf{D}^{-1} (\mathbf{b} - \mathbf{A}\mathbf{x}). \quad (3)$$

How do we get  $\Omega$ ? Optimize over error propagator,

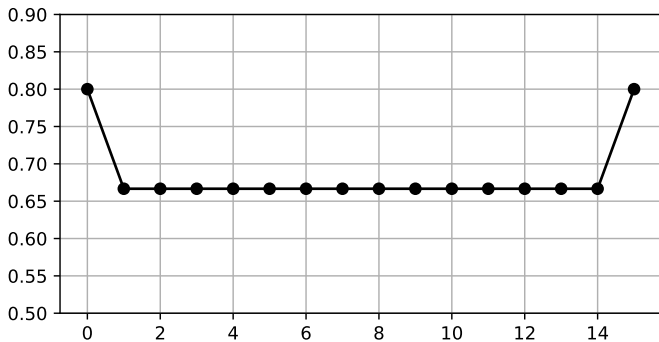
$$\arg \min_{\Omega} \|\mathbf{I} - \Omega \mathbf{D}^{-1} \mathbf{A}\|_F^2. \quad (4)$$

Run through optimizer to get entries of  $\Omega$



```
1 import numl.sparse as sp
2 import torch
3 |
4 N = 16
5 A = sp.eye(N) * 2 - sp.eye(N,k=-1) - sp.eye(N,k=1)
6 Dinv = sp.diag(1./A.diagonal())
7 I = sp.eye(N)
8
9 Omega = sp.eye(N)
10 Omega.requires_grad = True
11 optimizer = torch.optim.Adam([Omega.data], lr=0.01)
12
13 for i in range(50):
14     optimizer.zero_grad()
15     loss = ((I - Omega @ Dinv @ A) ** 2.).sum()
16     loss.backward()
17     optimizer.step()
```

For 1-D Poisson with Dirichlet conditions, obtain  $\omega_i \approx \frac{2}{3}$  on interior, higher on boundary



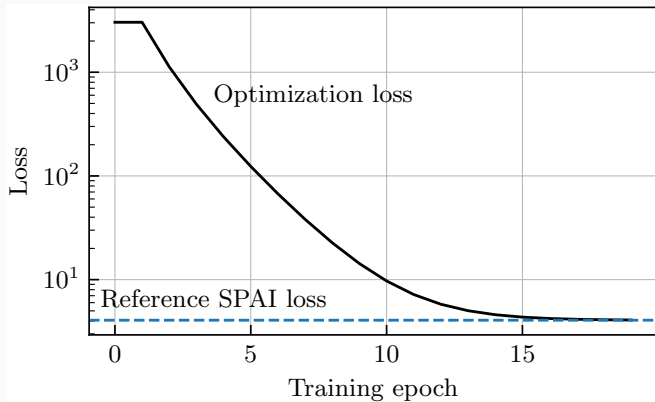
## (Easy!) Sparse approximate inverses

Given  $\mathbf{M}$  with certain sparsity pattern, want entries so that  $\mathbf{M} \approx \mathbf{A}^{-1}$

$$\arg \min_{\mathbf{M}} \|\mathbf{I} - \mathbf{MA}\|_F^2$$

Easy to implement in code:

```
1 opt.zero_grad()  
2 ell = ((sp.eye(N) - M @ A)**2.).sum()  
3 ell.backward()  
4 opt.step()
```





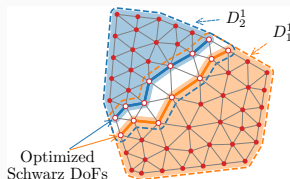
# Boundary Conditions for Schwarz Preconditioners

Based on work by Taghibakhshi et al.

Want to find an additive overlapping Schwarz preconditioner,

$$\mathbf{A}^{-1} \approx \mathbf{M}^{(\theta)} = \sum_{i=1}^S \tilde{\mathbf{R}}_i^T \left( \mathbf{R}_i \mathbf{A} \mathbf{R}_i^T + \mathbf{L}_i^{(\theta)} \right)^{-1} \mathbf{R}_i, \quad (5)$$

where  $\mathbf{L}_i^{(\theta)}$  is the output of a *graph neural network*

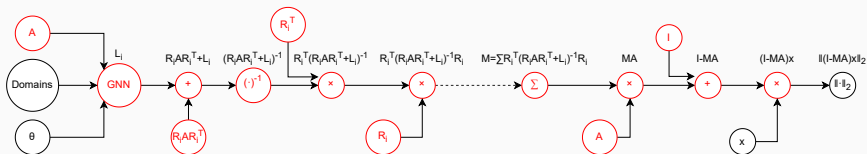


$$\mathbf{A}^{-1} \approx \mathbf{M}^{(\theta)} = \sum_{i=1}^S \tilde{\mathbf{R}}_i^T \left( \mathbf{R}_i \mathbf{A} \mathbf{R}_i^T + \mathbf{L}_i^{(\theta)} \right)^{-1} \mathbf{R}_i, \quad (5)$$

Optimize stochastic approximation to error propagator,

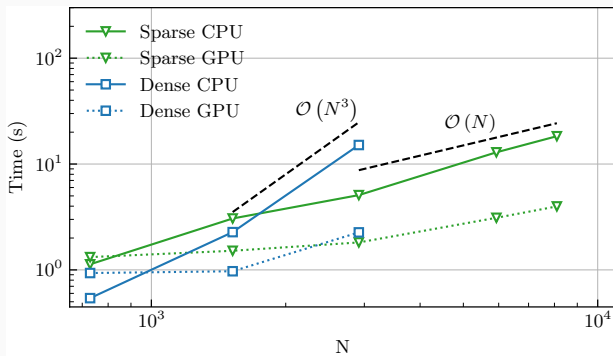
$$\ell = \max_{\mathbf{x} \in \mathcal{X}} \left\| \left( \mathbf{I} - \mathbf{M}^{(\theta)} \mathbf{A} \right) \mathbf{x} \right\|_2, \quad (6)$$

for random unit vectors  $\mathcal{X}$



		$N = 1,521$	$N = 2,916$	$N = 5,929$	$N = 8,100$
Sparse	CPU	3.060	5.088	12.906	18.321
	GPU	1.516	1.817	3.106	3.979
Dense	CPU	2.274	15.126	–	–
	GPU	0.971	2.262	–	–

(– ran out of memory)



# Sparse Relaxation for Multigrid

For an algebraic multigrid solver with levels

$$\mathbf{A}^{(l)} \leq \mathbf{A}^{(l-1)} \leq \dots \leq \mathbf{A}^{(1)},$$

can we find an optimal relaxation scheme at each level?

Find sparse  $\mathbf{M}^{(i)}$  for each level with same sparsity as  $\mathbf{A}^{(i)}$ ,

$$\mathbf{x} \leftarrow \mathbf{M}^{(i)} (\mathbf{b} - \mathbf{A}^{(i)} \mathbf{x}), \quad (7)$$

that complements coarse-grid correction

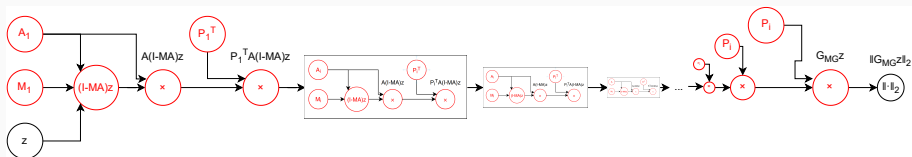
$$\arg \min_{\mathbf{M}^{(\dots)}} \|\mathbf{G}_{\text{MG}}\| \quad (8)$$

For full multigrid error propagator  $\mathbf{G}_{\text{MG}}$

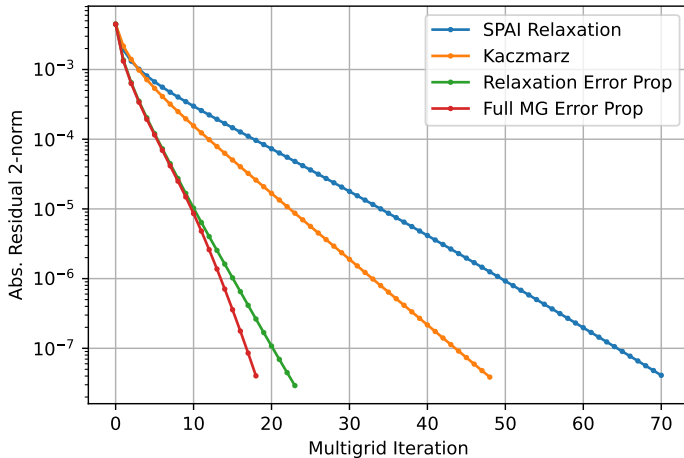
Claim: one iteration of  $\mathbf{Ax} = \mathbf{0}$  with AMG using guess  $\mathbf{z}$  is  $\mathbf{G}_{MG}\mathbf{z}$

$$\arg \min_{\mathbf{M}(\dots)} \|\mathbf{G}_{MG}\mathbf{z}\|, \quad (9)$$

for nonzero unit guess  $\mathbf{z}$



Residual History, Recirculating Flow



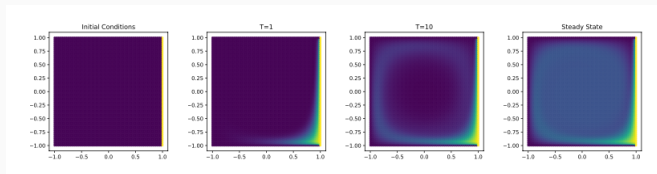
# Conclusions

- We have a general framework for sparse optimization with automatic differentiation
- Huge number of optimization problems to explore
  - Sparse relaxation
  - Learning Schwarz Preconditioners
  - Optimizing Heavyball to get CG?
  - Lets chat about ideas afterwards
- <https://github.com/nicknytko/numml>

Future:

- Scale up to multi-GPU and multi-node computation
- Investigate (neural) symbolic regression for interpretable learned methods

# Recirculating Flow Setup



$$\frac{\partial u}{\partial t} - \varepsilon \nabla^2 u = \nabla \cdot (\mathbf{w}u) = 0, \quad (10)$$

$$\mathbf{w}(x, y) = \left[ 2y(1 - x^2) \quad -2x(1 - y^2) \right]^T. \quad (11)$$

On unit square domain, Dirichlet condition: 1 on right boundary, 0 elsewhere.  $\varepsilon = 0.005$ .