Gradient-based optimization of sparse numerical methods with automatic differentiation

21st Copper Mountain Conference On Multigrid Methods

Nicolas Nytko[1] Ali Taghibakhshi[1] Tareq Uz Zaman[2] Scott MacLachlan[2] Luke Olson[1] Matthew West[1]

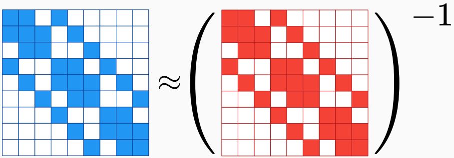
April 17, 2023

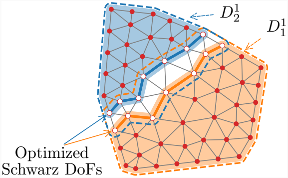
[1]University of Illinois at Urbana-Champaign

[2]Memorial University of Newfoundland

Optimization Problems in Numerics

- Sparse Approximate Inverses
- Optimal Boundary Conditions for Schwarz Preconditioners[1]
- Coefficients for Jacobi Method





[1] 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,

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

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

$$\theta \leftarrow \theta - \alpha \nabla_\theta f. \tag{2}$$

- Simple in theory...

Taking the gradient

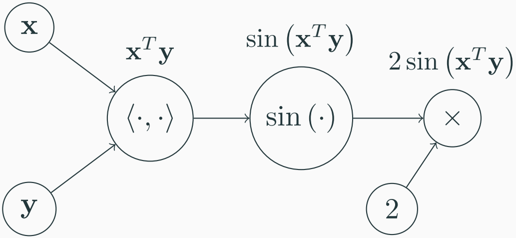
- How do we take derivatives of complex function?
- Assume they are compositions of simpler functions and apply chain-rule

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

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

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

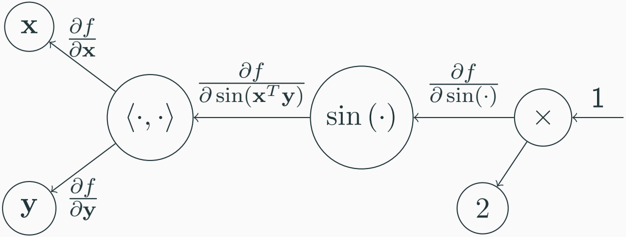
- Automatic differentiation has been around since early 50's [2]
- Perform original calculation as normal, record intermediate operations (forward pass)



[2] 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[3][4][5]
  - Matrix matrix products, direct solves, etc.
- What about differentiation through sparse matrix expressions?
- What should that entail?

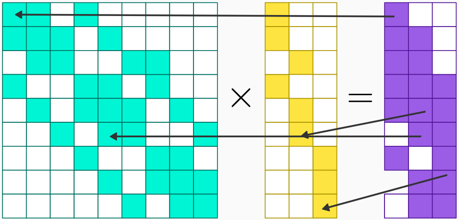
[3] Bank et al., "**Sparse matrix multiplication package (SMMP)**". Advances in Computational Mathematics, 1993

[4] Demmel et al., "**A Supernodal Approach to Sparse Pivoting**". SIAM Journal on Matrix Analysis and Applications, 1999

[5] 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
- Autogradient 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`

Mini-example of Sparse Optimization

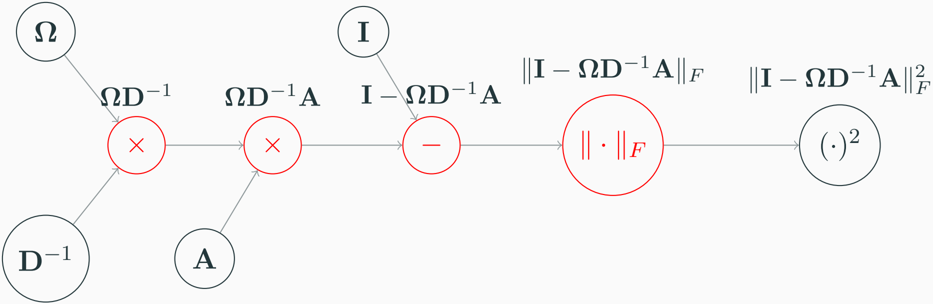
Consider Jacobi relaxation, we'll use a per-entry weight $\mathbf{\Omega}$,

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

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

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

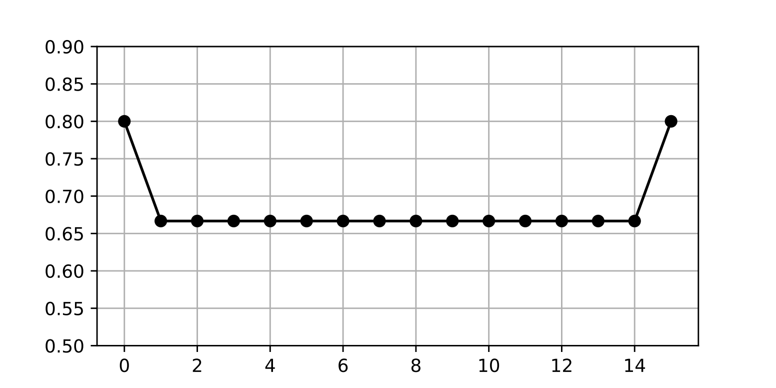
Run through optimizer to get entries of $\mathbf{\Omega}$


$\boldsymbol{\Omega}$
$\mathbf{D}^{-1}$
$\mathbf{A}$
$\mathbf{I}$
$\boldsymbol{\Omega}\mathbf{D}^{-1}$
$\boldsymbol{\Omega}\mathbf{D}^{-1}\mathbf{A}$
$\mathbf{I} - \boldsymbol{\Omega}\mathbf{D}^{-1}\mathbf{A}$
$\|\mathbf{I} - \boldsymbol{\Omega}\mathbf{D}^{-1}\mathbf{A}\|_F$
$\|\mathbf{I} - \boldsymbol{\Omega}\mathbf{D}^{-1}\mathbf{A}\|_F^2$
$\times$
$\times$
$-$
$\|\cdot\|_F$
$(\cdot)^2$

```python
import numml.sparse as sp
import torch

N = 16
A = sp.eye(N) * 2 - sp.eye(N,k=-1) - sp.eye(N,k=1)
Dinv = sp.diag(1./A.diagonal())
I = sp.eye(N)

Omega = sp.eye(N)
Omega.requires_grad = True
optimizer = torch.optim.Adam([Omega.data], lr=0.01)

for i in range(50):
    optimizer.zero_grad()
    loss = ((I - Omega @ Dinv @ A) ** 2.).sum()
    loss.backward()
    optimizer.step()
```

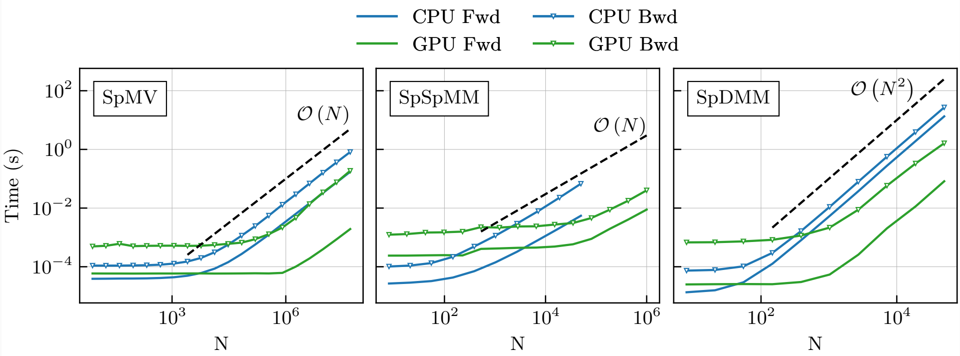
0 Ln 3, Col 1 Spaces: 4 UTF-8 LF Python

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



Timings, Scalings

Have scalable forward and backward pass on both CPU and GPU



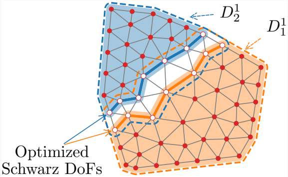
Boundary Conditions for Schwarz Preconditioners

Based on work by Taghibakhshi et al.[6]

Want to find an additive overlapping Schwarz preconditioner,

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

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



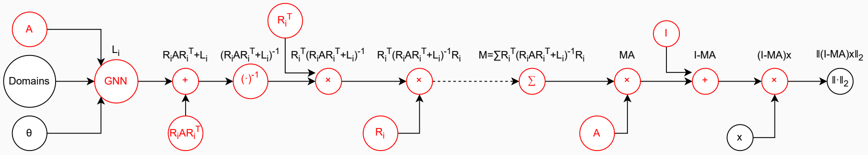
[6] Presenting on Tuesday's session, go check it out.

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

Optimize stochastic approximation to error propagator,

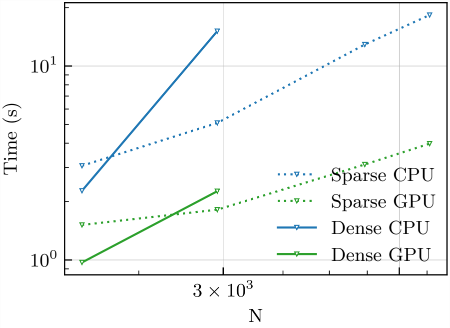
$$\ell = \max_{\mathbf{x} \in \mathcal{X}} \left\| \left(\mathbf{I} - \mathbf{M}^{(\theta)} \mathbf{A} \right) \mathbf{x} \right\|_2, \tag{6}$$

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

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

can we find an optimal relaxation scheme at each level?

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

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

that complements coarse-grid correction

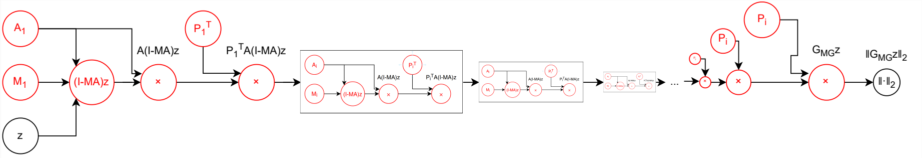
$$\arg\min_{\mathbf{M}^{(\dots)}} \|\mathbf{G}_{\mathsf{MG}}\| \tag{8}$$

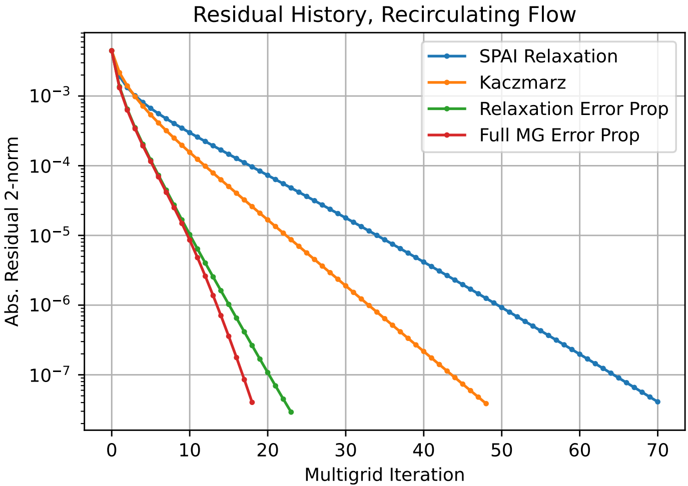
For full multigrid error propagator $\mathbf{G}_{\mathsf{MG}}$

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

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

for nonzero unit guess $\mathbf{z}$





Residual History, Recirculating Flow
SPAI Relaxation
Kaczmarz
Relaxation Error Prop
Full MG Error Prop
Abs. Residual 2-norm
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
10
20
30
40
50
60
70
Multigrid Iteration

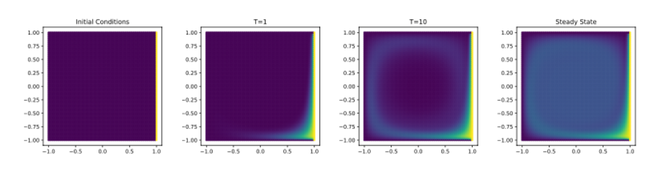
Conclusions

- We have a new, 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

Recirculating Flow Setup



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

$$\mathbf{w}(x, y) = \begin{bmatrix} 2y(1 - x^2) & -2x(1 - y^2) \end{bmatrix}^T. \tag{11}$$

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