Adaptive Algebraic Multigrig Preconditionering
with Graph Modularity Matching

Austen Nelson [1] and Panayot S. Vassilevski[1]

[1]Portland State University, ajn6@pdx.edu, panayot@pdx.edu

MTH 510 — Multilevel Methods

June 15, 2023

Portland State
UNIVERSITY

Overview

1. Background
   - Problem Statement
   - Algebraic vs Geometric Multigrid
   - Anisotropy
   - Heterogeneous Coefficients
2. Algorithm
3. Results

Problem Statement

Want to solve the linear matrix system:

$$A\boldsymbol{x} = b$$

Where A is symmetric positive definite.
A common source of problems like this is from numerical discretizations of elliptic partial differential equations:

$$\begin{cases} -\Delta u = f & , \text{ in } \Omega \\ u = 0 & , \text{ on } \partial\Omega \end{cases}$$

Algebraic vs Geometric Multigrid

Geometric Multigrid

- Requires a hierarchy of refinements (h or p)
- Interpolation and restriction operators come from this hierarchy

Algebraic Multigrid

- No knowledge of problem structure/nature required
- 'Black Box' for the end user
- Takes a matrix and algebraically finds the interpolation and restriction operators

Anisotropy

Let $\Omega \subset \mathbb{R}^3$.
We can modify our original PDE to be of the form:

$$\begin{cases} -\nabla \cdot (\beta \nabla u) = f & , \text{ on } \Omega \\ u = 0 & , \text{ on } \partial\Omega \end{cases}$$

Where

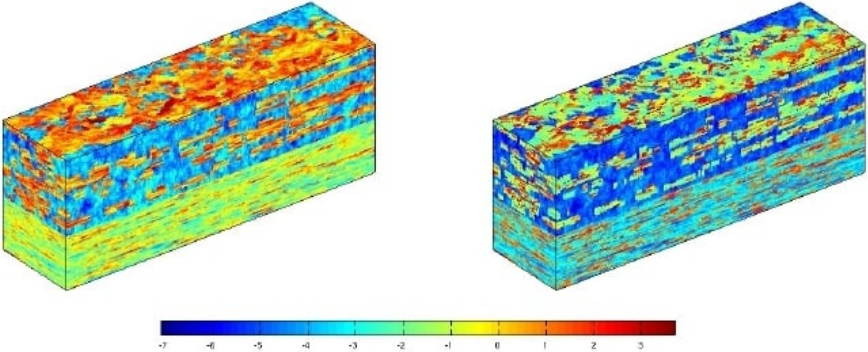
$$\beta = \varepsilon I + \boldsymbol{b}\boldsymbol{b}^T$$

for small $\varepsilon > 0$ and

$$\boldsymbol{b} = \begin{bmatrix} \cos\theta\cos\phi \\ \sin\theta\cos\phi \\ \sin\phi \end{bmatrix}$$

Heterogeneous Coefficients

$$\begin{cases} -\nabla \cdot (\beta \nabla u) = f & , \text{ on } \Omega \\ u = 0 & , \text{ on } \partial\Omega \end{cases}$$



Adaptivity Algorithm

Start with the 'identity' preconditioner, $B = I$.

- Finding 'Near Null' Vectors
  - Search for an 'algebraically smooth' error vector by iterating on the system $A\boldsymbol{x} = 0$ with a random starting guess with the preconditioner B.
  - When the convergence stalls, $A\boldsymbol{x} \approx 0$ and $\boldsymbol{x}$ is a linear combination of eigenvectors of A associated with small eigenvalues.
  - If the test converged, i.e
  $$\frac{\|x\|_A}{\|x_{init}\|_A} < \varepsilon$$
  then return the current preconditioner B.
  - Normalize this iterate to create the weight vector:
  $$\boldsymbol{w} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_A}$$
- Construct the weighted matrix $\overline{A} = (\overline{a}_{ij})_{i,j=1}^{n}$ where $\overline{a}_{ij} = -w_i a_{ij} w_j$. Note that if $\boldsymbol{w}$ is 'good enough' then $\overline{A}$ has positive row sums:
$$r_i \equiv \sum_j \overline{a}_{ij} \approx a_{ii} w_i^2 \geq 0 \text{ since } a_{ii} > 0$$

Adaptivity Algorithm

- With positive row sums we can perform clustering of A through pairwise aggregation iterations on the modularity matrix, $B = (b_{ij})_{i,j=1}^{n}$, associated with $\overline{A}$:
  Let $\boldsymbol{r} = (r_i)_{i=1}^{n}$ and $T = \sum_{i=1}^{n} r_i$.

$$B = \overline{A} - \frac{1}{T}\boldsymbol{r}\boldsymbol{r}^T$$

- Performing these actions recursively gives a hierarchy aggregations, $\{P_k\}_{k=1}^{\ell}$, which in turn define a hierarchy of matrices, $\{A_k\}_{k=1}^{\ell+1}$:

$$A_1 = A, \quad A_{k+1} = P_k^T A_k P_k$$

Adaptivity Algorithm

- Now we put $\boldsymbol{w}$ into the range of P_1, specifically such that $\boldsymbol{w} = P_1 \mathbf{1}_2$. We do this by scaling row i of P_1 by w_i.
- Using this modified hierarchy of aggregation operators and the resulting hierarchy of matrices we define an AMG V-cycle, denoted B_{new}.
- Create the composite preconditioner by composing B with B_{new} with the following algorithm:
  Initialize: $\boldsymbol{x} = 0$.
  - $\boldsymbol{x} = \boldsymbol{x} + B\boldsymbol{r}$
  - $\boldsymbol{r} = \boldsymbol{r} - A\boldsymbol{x}$
  - $\boldsymbol{x} = \boldsymbol{x} + B_{new}\boldsymbol{r}$
  - $\boldsymbol{r} = \boldsymbol{r} - A\boldsymbol{x}$
  - $\boldsymbol{x} = \boldsymbol{x} + B\boldsymbol{r}$

  Output $\boldsymbol{x}$.
- Define the new preconditioner, B as the steps above.
- Return to the first step to search for new near null.

Results