An Examination of Algebraic Multigrid as a Preconditioner for Krylov Subspace Iterative Methods

Perry Hooker

May 14, 2010
1 Introduction: Systems of Linear Equations

Matrix equations are interesting because they can represent very large (or very detailed) systems. These equations take the form

\[ Ax = b \]  

(1)

where \( A \) is a matrix and \( x \) and \( b \) are Euclidean vectors. Such a system of linear equations is inherently discrete: each equation in such a system “stands alone” in the sense that it defines a true statement about some set of variables. However, most useful systems are comprised of numerous equations that share many of the same variables, and only one set of values for these variables satisfies all of the equations at once. In these non-trivial cases the system is fundamentally interconnected - a change to one part of the system (one variable) is “felt” by every other part of the system.\(^1\) Quantifying the degree to which such a system “feels” a change to one of its parts is the crux of computational science and engineering, which tasks itself with answering the following question: Given a large, interconnected system, how much does a change to one element of the system affect the system as a whole?

2 Iterative Methods: Algorithms for Solving Large Systems of Equations

Iterative methods provide one way to answer this question. Iterative methods have been around since the 1700s, and they work by finding successively better approximations to a system’s solution: the output of one iteration (of the method) becomes the starting point for the next go ’round. In this way, an arbitrary starting guess at the solution is systematically refined toward the system’s true solution. Assurance that each iteration brings the approximate solution closer to the true solution is obtained by examining the residual \( r \) at each step. If \( \hat{x} \) is an approximate solution\(^2\) to a system of linear equations, the residual \( r \) is given by

\[ r = b - A\hat{x}. \]  

(2)

If a system’s true solution \( x \) is known,\(^3\) another measure of the approximate solution’s accuracy is the algebraic error, given by

\[ e = x - \hat{x}. \]  

(3)

\(^1\)This notion of “discrete-ness” is important. By analogy, a digital clock gives a discrete representation of time - a series of individual “ticks” that are not subdivided. Analog, by contrast, is the opposite of discrete. An analog clock is constantly moving, and thus does not break time into “chunks.”

\(^2\)The vector \( \hat{x} \) can also be some initial guess at a system’s solution. Refined approximations are typically obtained as output from an iterative solver, which is an implementation of an iterative method algorithm.

\(^3\)For most useful systems, this is VERY unlikely and often impossible!
Equations (2) and (3) can be manipulated to relate the error and residual:
\[ Ae = r. \] (4)

This relationship is called the “residual equation,” and it is fundamental to many techniques for solving linear systems. Note that \( r \) and \( e \) (and \( x \) and \( \hat{x} \)) are all vectors. Unlike a scalar quantity, a vector has no inherent “size” - instead, its magnitude is measured in a relative sense by examining the vector’s components. The magnitude of a vector is known as its norm. There are a number of ways to define a vector’s norm, but the two most common are the maximum norm and the Euclidean (or \( l^2 \)) norm, defined respectively as
\[ \| x \|_\infty = \max_{1 \leq j \leq n} |e_j| \] (5)
\[ \| x \|_2 = \left( \sum_{j=1}^{n} e_j^2 \right)^{1/2} \] (6)

(In Euclidean space, the \( l^2 \) norm corresponds exactly to the formula for determining the distance between two points.) Thus, the norm of the residual is used to quantify “how close” an approximate solution \( \hat{x} \) is to a system’s true solution \( x \) - without knowing \( x \) itself. Often, iterative methods involve a series of matrix multiplication operations, which are quite computationally expensive. Thus, many modern iterative methods avoid matrix-matrix operations altogether. Instead, these techniques multiply vectors by the matrix and work with the resulting vectors. Starting with a vector \( b \), one computes \( Ab \), then multiplies the resulting vector by \( A \) to find \( A^2 b \) and so on. Such methods are known as Krylov subspace methods, after the Russian naval engineer Alexei Krylov who published a paper on the topic in 1931 \[7\]. This paper examines the following Krylov methods: GMRES (Generalized Minimum RESidual) \[12\], LGMRES \[1\], MINRES \[9\], conjugate gradient \[5\], and conjugate gradient squared \[13\]. Iterative methods come in many flavors in addition to the Krylov subspace methods, and they are generally very effective - but they’re not perfect.

3 Preconditioning: Improving the Iterative Methods

The performance of iterative solvers can often be improved with a technique known as preconditioning. Simplistically, a preconditioner is a matrix based on the original operator \( A \) that improves the “guess” used to initiate each cycle of the iterative solver algorithm. \[11\] Necessarily, it must do this in such a way that the formulation of this preconditioner is computationally cheaper than running the iterative solver alone! (More formally, a preconditioner \( P \) of a linear operator \( A \) is a matrix such that \( P^{-1} A \) has a smaller condition number than \( A \).) This paper examines the effectiveness of one technique, known as algebraic multigrid, for preconditioning the above-mentioned Krylov subspace methods.
4 Algebraic Multigrid: A Multilevel Method

Algebraic multigrid is a technique which can be used as either a standalone solver or as a preconditioner to a Krylov subspace iterative method \[2\]. Algebraic multigrid is a generalization of geometric multigrid, which relies on the underlying geometry of the problem to be solved. By contrast, algebraic multigrid needs only the definition of the system \(Ax = b\) to function, and can be implemented in cases where the problem’s geometry is prohibitively complex, unknown, or nonexistent.

In general, multigrid methods work by taking advantage of certain properties of discrete systems and iterative solvers. Many interesting systems of equations arise as discretizations of continuous systems; a discretization can be thought of as a sampling of some continuous system at regular intervals.\(^4\) Iterative solvers work with discrete systems, and (naïvely) these solvers “smooth” the error associated with some approximation \(\hat{x}\) of the system’s solution. However, “smoothing” \(\hat{x}\) does not necessarily reduce its magnitude: it only ensures that adjacent\(^5\) components of \(\hat{x}\) have comparable values. Figure 1 represents this notion graphically; in these images, a error of 0 would be shown as a flat plane.

5 Experimental Methods

Here, a Python-based implementation of algebraic multigrid called pyAMG was used to examine the effectiveness of algebraic multigrid as a preconditioner to a selection of Krylov subspace methods. Iterative solvers were obtained from the SciPy code library of scientific tools. Some trials made use of the NetworkX extension to generate random graphs, which were then modified and used as linear operators. Since the original theory of algebraic multigrid was developed for symmetric M-matrices \[3\], a short Python function was also created to generate random M-matrices.\(^6\)

To test the effectiveness of pyAMG, a selection of sparse linear systems \(Ax = b\) were constructed with random right-hand-sides \(b\) and various linear operators \(A\). pyAMG’s “smoothed aggregation solver” was used in all cases.\(^7\) Experiments were performed with a number of linear operators \(A\), but only a few of these operators returned interpretable results. Ultimately, data were generated for random systems based on the stiffness ma-

\(^{4}\)To continue the time analogy: the minute is a continuous unit of time which, for convenience, is discretized into 60 seconds. There are still units of time “in between” the seconds, but for common everyday usage these units are too small to matter. Thus, a discretization is, in some sense, a simplification of a system which might otherwise be prohibitively complex.

\(^{5}\)[3], [10] provide rigorous definitions of adjacency.

\(^{6}\)M-matrices are non-singular square matrices with non-positive off-diagonal entries, positive diagonal entries, non-negative row sums, and at least one positive row sum. [11]

\(^{7}\)pyAMG provides two algebraic multigrid solvers: ‘smoothed aggregation’ and ‘Ruge-Stüben.’ In early trials, the Ruge-Stüben solver was tested on the Poisson problem with Dirichlet boundary conditions, and it returned a small number of values that were either < 0 or > 1. No hypothesis is forthcoming regarding what the heck is going on there! The smoothed aggregation solver returned accurate results.
Figure 1: (a) shows the error associated with some poor approximation $\hat{x}$. It contains high-frequency components (small jagged spikes) and low-frequency components (large peaks and valleys). Part (b) shows the error after being processed by an iterative solver: the error is smoother, and error values at adjacent points are relatively closer than they were before. Lower-frequency errors are mostly unchanged. The iterative solver has smoothed the error, but not reduced it enough. In part (c), multigrid techniques change the size of the grid so that the grid spacing is more comparable to the frequency of the remaining error. This new grid will be used to effectively reduce low-frequency error before returning to the original grid size.
matrix $K$, the “fixed-free” matrix $T$, the discrete Laplacian operator with Dirichlet boundary conditions

$$\begin{align*}
\rho(x = 0) &= \rho(x = 1) = 1 \\
\rho(y = 0) &= \rho(y = 1) = 0
\end{align*}$$

, the discrete Laplacian operator with Dirichlet boundary conditions

$$\begin{align*}
\rho(x = 0) &= \rho(x = 1) = 1 \\
\rho(y = 0) &= \rho(y = 1) = 0
\end{align*}$$

, a discretization of the linear elasticity problem called the finite element Q1 stiffness matrix, a matrix based on a random regular graph with nodes of degree 3 featuring off-diagonal elements equal to -1 and diagonal elements equal to 3, an operator based on the Laplacian matrix of an Erdős-Renyi random graph with 2048 nodes and an edge-creation probability of 0.001, and a random M-matrix generated by the author.\(^8\) In each of these trials, systems preconditioned with algebraic multigrid were solved with different iterative solvers. Performance was measured by examining the number of iterations required to reduce the $l^2$ norm of the residual below a threshold of $10^{-8}$. Results appear below.

Conclusions

It was quickly determined that pyAMG’s implementation of the smoothed aggregation algebraic multigrid method vastly outperformed the unaccelerated Krylov methods in virtually all circumstances: both algebraic multigrid solvers and algebraic multigrid preconditioners yielded orders-of-magnitude improvements in performance when compared to unaccelerated Krylov methods. HOWEVER, in many cases the accelerated iterative solvers (and in a number of unreported trials, the standalone pyAMG solver itself!) failed utterly to converge on a solution, and instead demonstrated behavior wherein the residual increased with each iteration of the method. The author speculates (without any rigor whatsoever!) that this is due to properties of BOTH the matrices and the iterative algorithms: something weird happens when certain matrices are fed to certain solvers. GMRES was the outstanding algorithm in these experiments: in many cases, it solved systems that were unapproachable by other iterative methods or algebraic multigrid solvers. Finally, the LGMRES method showed intriguing results in some cases, converging quite quickly to a low residual norm, though once it reached this low norm (typically around $10^{-8}$) further iterations yielded no further improvements.

\(^8\)Something strange happened with these systems. The “operator complexity” (as calculated by the pyAMG software) was higher (as much as a factor of 10, though those systems never got solved) for these matrices than for any of the others. What’s more, up to a point the complexity increased when the matrices were initialized with fewer entries, then decreased as the number of non-zeros approached zero.
Figure 2: Convergence history for the K-matrix. All results are for pyAMG-accelerated methods except for ‘Pure GMRES.’ ‘Standalone’ refers to the pyAMG smoothed aggregation solver. Operator complexity: 1.498
Figure 3: Convergence history for the T-matrix. All results are for pyAMG-based methods except for ‘Pure GMRES.’ ‘Standalone’ refers to the pyAMG smoothed aggregation solver. Operator complexity: 1.498
Figure 4: Convergence history for the Poisson problem. E-W boundaries = 1, N-S boundaries = 0. Operator complexity: 1.262
Figure 5: Convergence history for the Poisson problem. E-W boundaries = N-S boundaries = 0. Operator complexity: 1.262
Figure 6: Convergence history for the linear elasticity problem. Operator complexity: 1.280
Figure 7: Convergence history for a matrix based on a random regular graph of degree 3. Operator complexity: 1.892
Figure 8: Convergence history for a matrix based on an Erdős-Renyi random graph with 2048 nodes and edge creation probability 0.001. Operator complexity: 2.289
Figure 9: Convergence history for a random M-matrix. Operator complexity: 4.909
Bibliography


