MIE 1210: Computational Fluid Dynamics: Assignment 1

Declan Bracken

October 2, 2023

1 Introduction

The purpose of this assignment is the development of a linear solver capable of solving large ($N = 10^6$) sparse square matrices. The form of the sparse matrix to be solved is not specified exactly, but the minimum requirement is to solve a banded matrix. Due to the size of the sparse matrix, iterative solving methods are preferred, specifically high efficiency solvers which implement krylov subspace methods to converge to a solution within a given tolerance. The primary iterative solving methods tested are General Minimal Residual (gmres) and Conjugate Gradient (CG), as well as some of their derivative methods such as limited-memory gmres (lgmres), conjugate gradient squared (cgs), and bi-conjugate gradient stabilized (bicg-stab). Each of these solvers possess their own advantages and disadvantages including limits on what kind of matrices they can be applied to and convergence speed. In addition to testing solving methods, the use of a preconditioner, specifically incomplete LU factorization, is shown to drastically improve convergence times and is used for essentially all solvers applied to large sparse systems.

Python is the coding language of choice because it's Scipy library has pre-built functions which can easily implement any of the aforementioned solving methods, including preconditioning matrices. Python is also open-source, free, and has a large user base which helps with using external libraries and troubleshooting.

2 Iterative Solving Methods

2.1 Conjugate Gradient

The CG solving method is the fastest iterative solver tested, but it's performance is also limited by the type of matrix it can be applied to. A CG solver in Scipy will only converge on a hermitian positive-definite matrix. CG functions by taking an initial guess x_0 at the solution x for the linear system;

$$\{A\}\{x\} = \{b\}, \tag{1}$$

where it may then construct an approximate solution using the shifted krylov subspaces;

$$x_0 + K^k(A; r_0) = x_0 + \{r_0, Ar_0, ..., A^{k-1}r_0\}, \tag{2}$$

where $r_0 = b - Ax_0$ is the residual of the initial guess. this leads to a Richardson iteration;

$$x_k = (I - A)x_{k-1} + b. \tag{3}$$

This is the fundamental math behind all iterative solving methods, but conjugate gradients specifically functions by constructing residuals which are orthogonal to the current subspace [1]. This makes CG converge incredibly quickly, aiming to converge in n steps for an n dimensional matrix, but also imposes the aforementioned limitations. This makes CG the ideal choice for handling SPD matrices which are well conditioned, but not much else.

A derivative of the CG method is the CGS method or conjugate gradient squared. The CGS method is designed to function on non-symmetric matrices, making it's application more general than CG, but often at the cost of convergence speed.

One other derivative tested for this assignment was the bicg-stab, or biconjugate gradient stabilized solver. bicg-stab functions as another generalization to the conjugate gradient method which allows it to be applied to asymmetric matrices. bicg, as the name would suggest, has the solver store 2 sets of search directions, one for the left and one for the right preconditioning of the matrix. Since these 2 directions need not be orthogonal to each other, this creates flexibility in the solver, but also results in some unstable convergence patterns. bicg can then be combined however with some minimal residual steps to create a more stable convergence method, which is bicg-stab.

2.2 General Minimal Residual

The gmres solver functions with a similar idea to conjugate gradient in that it uses shifted krylov subspaces, but instead of constructing subspaces which are purely orthogonal, gmres aims to minimize the euclidean norm $||b - Ax_k||_2$ [2]. Gmres must store a full orthogonal basis for each iteration of the solver, which increases memory demands when compared to the cg method. For a high number of iterations, this can make gmres an obsolete solver without preconditioning. A common approach is to perform a restart after a certain number of iterations to keep memory requirements low.

A derivative of gmres is lgmres or loose general minimal residual is a method which addresses how the residuals of the gmres solver behaves after restarts. Often consecutive restarts of gmres cause the residuals to alternate directions cyclically, the lgmres algorithm aims to disrupt this alternation and converge in fewer iterations.

2.3 Preconditioning

A preconditioner is a matrix which can be used to improve the convergence of iterative solvers by improving the conditioning of the matrix A. Often large sparse matrices can be categorized by a condition number which is proportional to the difference in their maximum and minimum eigenvalues. This condition number directly informs the convergence speed of many iterative linear solvers. By preconditioning a matrix, we may drastically improve convergence speed by reducing the condition number. There are several preconditioners commonly applied in linear solvers, but for this assignment ILU or incomplete LU factorization will be applied. Now each solver is swept through N for $N = [10^3, 10^4, 2 * 10^4, 10^5, 2 * 10^5, 10^6]$

3 Results

Using a custom built function in python and the Scipy module, precondition with ILU, solver tolerance, and solver method can all be manually selected before solving a linear system of equations given the matrix A and vector b. The 5 solver methods (gmres, lgmres, cg, cgs, and bicg-stab) are then used to solve an arbitrary banded matrix for different dimensions of N upto the assignment requirement of $N = 10^6$. To test their performance on a banded matrix, a tridiagonal matrix is constructed for A where the diagonal is all 2 (arbitrarily) and the off diagonal elements are all -1;

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix},$$

Then the solution vector b is generated randomly. This matrix is symmetric and functions with cg, but other matrices may not, which is why the built function can easily swap solving methods on a whim based off user input. Without preconditioning, the gmres solver won't even converge to a solution on this matrix past $N = 1500$. For this reason, it was left out of the non-preconditioned analysis and the 4 other solvers were swept for $N = [100, 200, 500, 1000, 2000, 5000, 10000]$.

in Figure 1, for all matrix sizes the cg solver is the fastest, followed by cgs, then bicg-stab, then lgmres. Of note is that after $N = 1000$ bicg-stab solving time begins to diverge from cgs. If gmres were plotted on in Figure 1, it would be significantly higher (greater than 10^0 at $N = 1000$) than the other solvers.

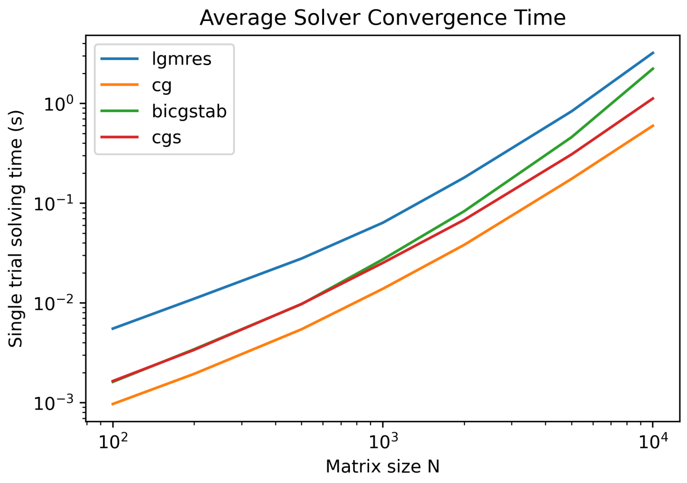

Figure 1: Average solver convergence time (not including gmres) to a tolerance of 10^{-5} without preconditioning.

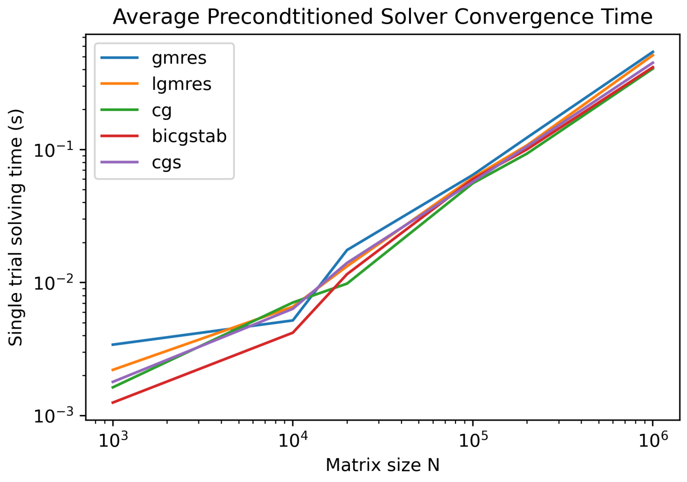

Figure 2: Average solver convergence time to a tolerance of 10^{-5} while using ILU preconditioning on matrix A.

As seen in Figure 2, cg, bicg-stab, and cgs are the 3 lowest times, followed by lgmres and gmres. In general however, all the values are close to each other for this matrix which makes selection between them generally unimportant. At $N = 10^6$, the residual norm of each solver is $0.00701, 0.00702, 0.00701, 0.00458, 0.00458$ for cg, cgs, bicg-stab, lgmres, and gmres respectively. Use of a particular solver will depend on the system being solved. In this case cg converges the fastest, however, future matrices will not likely be symmetric and positive-definite, and the other solving methods will be required.

References

[1] H. A. van der Vorst, "Efficient and reliable iterative methods for linear systems," *Journal of Computational and Applied Mathematics*, vol. 149, no. 1, pp. 251–265, 2002.

[2] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Other Titles in Applied Mathematics, SIAM, second ed., 2003.

MIE 1210 Assignment 1

Friday, September 22, 2023 9:59 AM

From Proof 2:

$$f''(x_{i-1}) = \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2} + O(h^2)$$

Now:

$$f(x_i) = f(x_{i-1}) + h f'(x_{i-1}) + \frac{h^2}{2} f''(x_{i-1}) + O(h^3)$$

$$\Rightarrow f'(x_{i-1}) = \frac{f(x_i) - f(x_{i-1})}{h} + \frac{h}{2} f''(x_{i-1}) + O(h^3)$$

Substitute:

$$f'(x_{i-1}) = \frac{f(x_i) - f(x_{i-1})}{h} + \frac{1}{2}\left(\frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h}\right) + O(h^2) \cancel{+ O(h^3)}$$

$$\boxed{\therefore f'(x_{i-1}) = \frac{(3f(x_i) - 4f(x_{i-1}) + f(x_{i-2}))}{2h} + O(h^2)} \quad (1)$$

Leading Truncation Error is $\underline{O(h^2)}$

$$f(x_{i+1}) = f(x_i) + h f'(x_i) + \frac{h^2 f''(x_i)}{2}$$

$$f(x_{i-1}) = f(x_i) - h f'(x_i) + \frac{h^2 f''(x_i)}{2}$$

$$\Rightarrow f(x_{i+1}) + f(x_{i-1}) = 2f(x_i) + h^2 f''(x_i)$$

$$\boxed{\therefore f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}} \quad (2)$$

with Truncation error $\underline{O(h^2)}$ (Lecture Derivation)