Computational Linear Algebra 2021/22

Mastery Coursework

(01495449)

Imperial College London

January 13, 2023

1 Introduction

I will be investigating the effectiveness of the (Alternating-Direction implicit) ADI method using various numerical techniques. The ADI algorithm preconditions the problem $Ax = b$ to an easier problem $Mu = Ax = b$. The k^{th} iteration of the ADI algorithm gives an approximation to u, denoted as $u_k = M_k^{-1}Ax = M_k^{-1}b$. Note ADI algorithm works without directly computing the precondition M_k^{-1}.

In the algorithm later presented several steps of the ADI iteration are computed in parallel. One step in the ADI iteration involves two sweeps of mesh in the coordinate directions. When using finite differences, each sweep solves a tridiagonal system. Classical algorithms for solving tridiagonal systems are sequential, however, the ADI method can be made highly efficient on parallel architectures by using some parallel algorithms for solving tridiagonal systems. In this report, we will work with an ADI algorithm, where k-step consecutive iterations (k tridiagonal systems) are computed simultaneously.

2 ADI setup

We are trying to work out the solution to the partial differential equations

$$\begin{aligned} &-\tfrac{\partial}{\partial x}\left(a(x,y)\tfrac{\partial u}{\partial x}\right) - \tfrac{\partial}{\partial y}\left(b(x,y)\tfrac{\partial u}{\partial y}\right) = f(x,y), && (x,y) \in \Omega \\ &u = 0, && (x,y) \in \partial\Omega \end{aligned} \tag{1}$$

where $a > 0, b > 0$ and $\Omega = [0,1] * [0,1]$. The solution of the partial differential equation can be written as the solution to the following system of linear equations:

$$Au = (H + V)u = f \tag{2}$$

Where u and f are column vectors of size n^2. A is a positive definite matrix.
If (2) is separable then we can write H and V in tensor product form.

$$H = A_1 \otimes B_2 \tag{3}$$

$$V = B_1 \otimes A_2 \tag{4}$$

where A_1, A_2, B_1 and B_2 are n by n. Furthermore A_1^{-1}, A_2^{-1} exist and $A_1^{-1}B_1$ and $A_2^{-1}B_2$ are also positive definite.
We can solve the linear system (2) iteratively by applying a k-term ADI preconditioner M_k^{-1}

$$M^{-1}Au = M_k^{-1}f \tag{5}$$

In other words if $u^k = M_k^{-1}f$ is the k^{th} approximation of the solution of the linear system (2).

$$(H + \rho_{j+1}I)\, u^{j+1/2} = f - (V - \rho_{j+1}I)\, u^j \tag{6}$$

$$(V + \rho_{j+1} I)\, u^{j+1} = f - (H - \rho_{j+1} I)\, u^{j+1/2} \tag{7}$$

Here u^0 is an arbitrary initial vector and ρ are positive acceleration parameters.
Assuming H and V matrices are commutative. We can express the error of the k^{th} iteration as $e^k = u^k - u^*$, where u^* is the true solution to (2). The error satisfies

$$\left\| e^k \right\| \leqslant \max_{\substack{\lambda_i \in \sigma(H) \\ \mu_i \in \sigma(V)}} \left| \prod_{j=1}^{k} \frac{(\lambda_i - \rho_j)\,(\mu_i - \rho_j)}{(\lambda_i + \rho_j)\,(\mu_i + \rho_j)} \right| \left\| e^0 \right\| \tag{8}$$

where $\sigma(H)$ and $\sigma(V)$ are spectra of H and V respectively.
(6) and (7) aren't very computationally efficient and hence we can rewrite them together and solve for u^{j+1} gives:

$$u^{j+1} = (V + \rho_{j+1} I)^{-1} \left\{ f - (H - \rho_{j+1} I) \left[(H + \rho_{j+1} I)^{-1} \left[f - (V - \rho_{j+1} I)\, u^j \right] \right] \right\} \tag{9}$$

Using the proof in [1], we can simplify (9) into:

$$u^{j+1} = (V + \rho_{j+1} I)^{-1} \left\{ \tilde{u}^j + 2\rho_{j+1} (H + \rho_{j+1} I)^{-1} \left(f - \tilde{u}^j \right) \right\} \tag{10}$$

Where,

$$\tilde{u}^j = (V - \rho_{j+1} I)\, u^j \tag{11}$$

Furthermore,

$$\tilde{u}^j = \left[I - (\rho_{j+1} + \rho_j) (V + \rho_j I)^{-1} \right] \left\{ \tilde{u}^{j-1} + 2\rho_j (H + \rho_j I)^{-1} \left(f - \tilde{u}^{j-1} \right) \right\} \tag{12}$$

Most often H and V don't commute hence we manipulate (2) into a system with commutativity. Assuming A_1 and A_2 are invertible again. Multiplying (2) by $A_1^{-1} \otimes A_2^{-1}$ from the left, the result is

$$(\tilde{H} + \tilde{V}) u = \tilde{f}, \tag{13}$$

where

$$\tilde{H} = I \otimes A_2^{-1} B_2, \quad \tilde{V} = A_1^{-1} B_1 \otimes I, \tag{14}$$

and $\tilde{f} = (A_1^{-1} \otimes A_2^{-1}) f$. Now $\tilde{H}$ and $\tilde{V}$ clearly commute.
Using all this machinery I now present a parallel ADI preconditioner algorithm

2.1 ADI Algorithm 2.2.

In my work, I will use the five-point scheme setting $A_1 = A_2 = I$ and $B_1 = B_2 =$ tridiagonal matrix $(\frac{-1}{2}, 1, \frac{-1}{2})$

We start at an initial state u^0. Giving us $\tilde{u}^0 = \left(A_1^{-1} B_1 \otimes I - \rho_1 I \right) u^0$.

For $j = 1, ..., k-1$, we can calculate:

$$\tilde{u}^j = \left[I - (\rho_{j+1} + \rho_j)\left(A_1^{-1}B_1 \otimes I + \rho_j I\right)^{-1}\right] \times \left[\tilde{u}^{j-1} + 2\rho_j \left(I \otimes A_2^{-1}B_2 + \rho_j I\right)^{-1}\left(\tilde{f} - \tilde{u}^{j-1}\right)\right] \tag{15}$$

This gives us

$$u^k = \left(A_1^{-1}B_1 \otimes I + \rho_k I\right)^{-1}\left[\tilde{u}^{k-1} + 2\rho_k \left(I \otimes A_2^{-1}B_2 + \rho_k I\right)^{-1}\left(\tilde{f} - \tilde{u}^{k-1}\right)\right] \tag{16}$$

We can divide each iteration into the following steps:
(1) First calculate $r = \tilde{f} - \tilde{u}^{j-1}$
(2) Then compute $d = \left(I \otimes A_2^{-1}B_2 + \rho_j I\right)^{-1} r$
(3) Which allows us to calculate $r = \tilde{u}^{j-1} + 2\rho_j d$
(4) Now determine $d = \left(A_1^{-1}B_1 \otimes I + \rho_j I\right)^{-1} r$
(5) Finally we workout $\tilde{u}^j = r - (\rho_{j+1} + \rho_j)\, d$.

Algorithm 2.2 is equivalent to the implicit process defined as:

$$\left[A_1 \otimes (B_2 + \rho_{j+1}A_2)\right] u^{j+1/2} = f - \left[(B_1 - \rho_{j+1}A_1) \otimes A_2\right] u^j \tag{17}$$

$$\left[(B_1 + \rho_{j+1}A_1) \otimes A_2\right] u^{j+1} = f - \left[A_1 \otimes (B_2 - \rho_{j+1}A_2)\right] u^{j+1/2} \tag{18}$$

Algorithm 2.2 is more efficient than (17) and (18) as it requires solving fewer tri-diagonal systems and uses vector updates rather than matrix by vector multiplication. Hence

3 Acceleration parameters

The parameters required by Algorithm 2.2, namely $\{\rho_j\}$, for $j = 1, ..., k$ are calculated by minimising the upper bound on e^k in (8). This is done by finding the parameters such that the following is minimised:

$$\max_{\substack{\lambda \in [a,b] \\ \mu \in [a,b]}} \left| \prod_{j=1}^{k} \frac{(\lambda - \rho_j)(\mu - \rho_j)}{(\lambda + \rho_j)(\mu + \rho_j)} \right| \tag{19}$$

Where a and b are the lower and upper bounds of both $\sigma\left(A_1^{-1}B_1\right)$ and $\sigma\left(A_2^{-1}B_2\right)$.

There is a method described in [2] which represents how to find the optimum set of $\{\rho_j\}$ parameters in terms of elliptic functions.

Since I implemented the 5-point scheme, I used the closed-form equations below to calculate the acceleration parameters. These were given in [1] as they approximate the solutions to the optimal parameters condition (19).

Let

$$a = \sin^2(\frac{\pi}{2(n+1)}), \quad b = \sin^2(\frac{n\pi}{2(n+1)}) \tag{20}$$

and set

$$\rho_j = b(\frac{a}{b})^{\frac{2j-1}{2k}} \tag{21}$$

for j=1,2,...,k

4 GMRES

GMRES stands for Generalised Minimum Residual method. In this algorithm, we compute the orthogonal basis for the Krylov subspaces one by one and at each iteration, we solve the projection of Ax=b into the Krylov basis until the residual reaches below a tolerance. GMRES utilises Arnoldi iteration in this.

Arnoldi iteration uses modified gram Schmidt to construct orthogonal vectors $q_1, q_2, ...$ such that $q_1, ..., q_n$ span the Krylov subspace K_n.

Krylov subspace K_r is defined as:
$K_r(A, b) = span(b, Ab, A^2b, ..., A^{r-1}b)$

5 Analysis

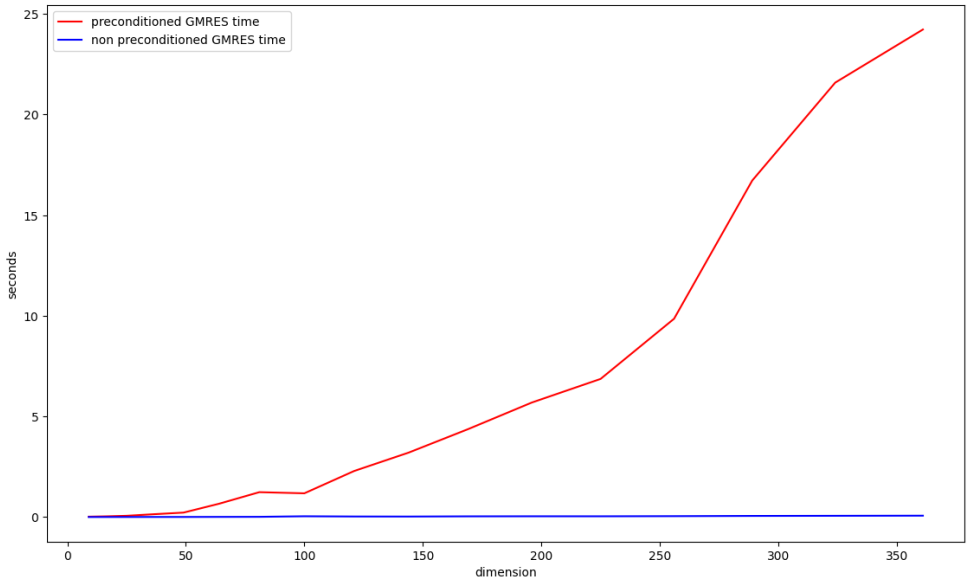

Figure 1: This shows that the time needed for convergence of the preconditioned GMRES is longer than that of non-preconditioned GMRES. This separation grows rapidly for larger dimensions. The reason for this is that the preconditioned GMRES calls on the Minv function. This function is inefficient as it utilises np.linalg.inv to calculate the inverse of large matrices. We could speed up Algorithm 2.2 by implementing parallel computing. The steps (1) to (5) on page 3, can be done efficiently on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems

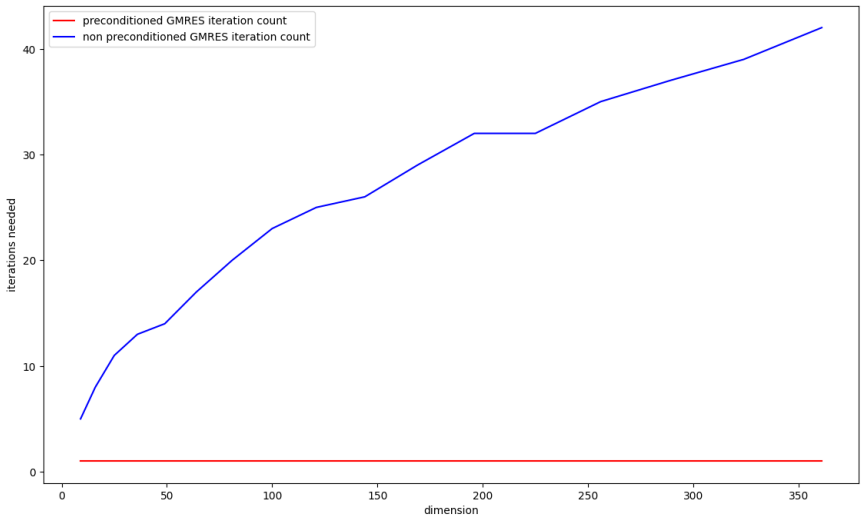

Figure 2: This confirms that the non-preconditioned GMRES algorithm takes more iterations to converge and this grows in a quadratic manner as the dimension increases. However, the preconditioned GMRES takes only one iteration to converge confirming it takes fewer computational operations than non-preconditioned GMRES.

5.1 Investigating Acceleration parameters

I am now going to show my results that support that the optimum acceleration parameters used from (21) speed up the convergence of GMRES by comparing it to other acceleration parameters.

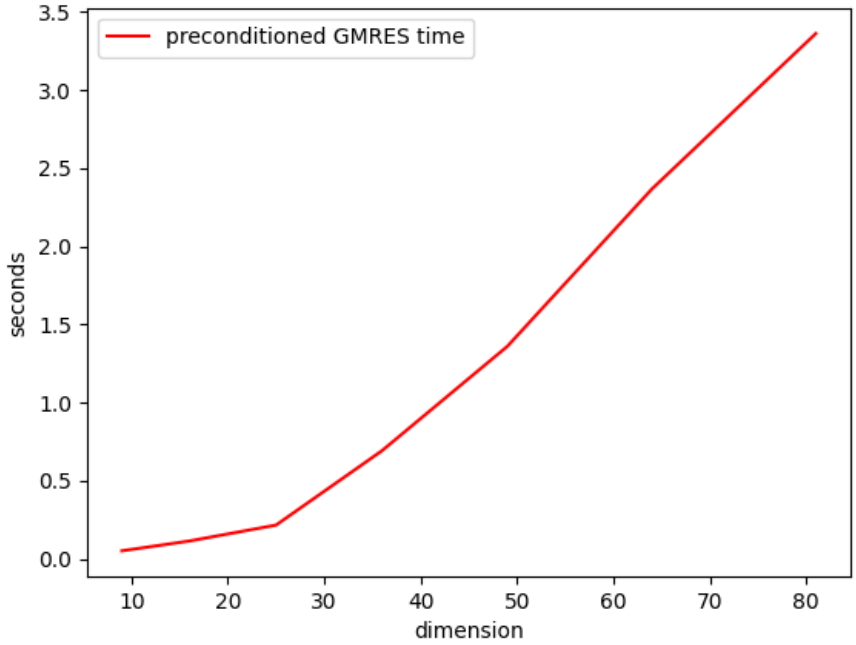

Figure 3: This shows how quickly preconditioned GMRES converged for different dimensions of matrices using the optimal acceleration parameters derived in (21)

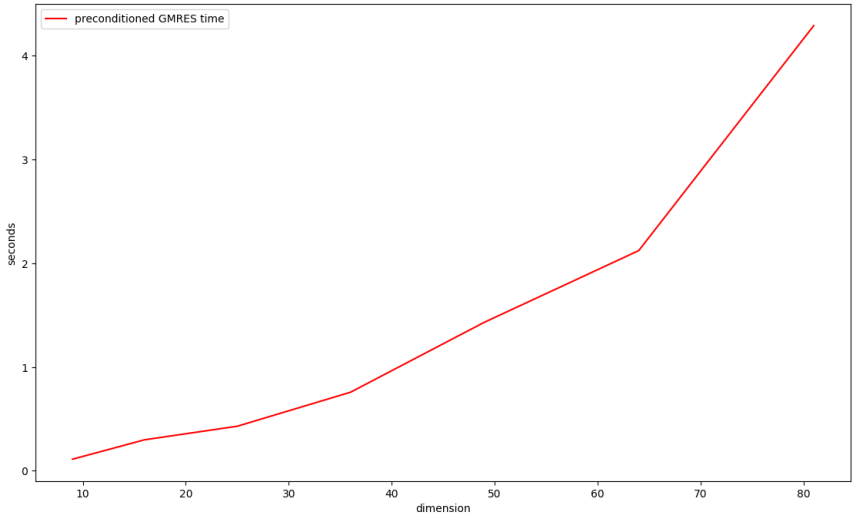

Figure 4: This shows how quickly preconditioned GMRES converged for different dimensions of matrices where each acceleration parameter is set to 4.

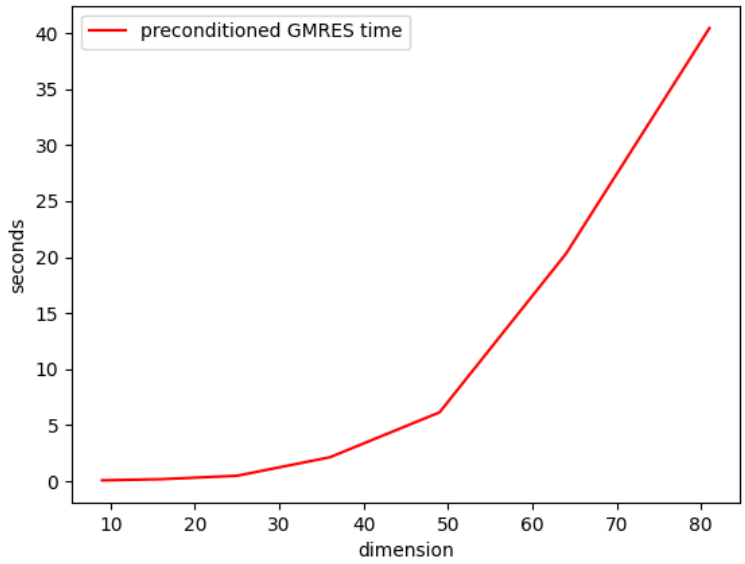

Figure 5: This shows how quickly preconditioned GMRES converged for different dimensions of matrices where $\rho_j = 5^{j-1}$.

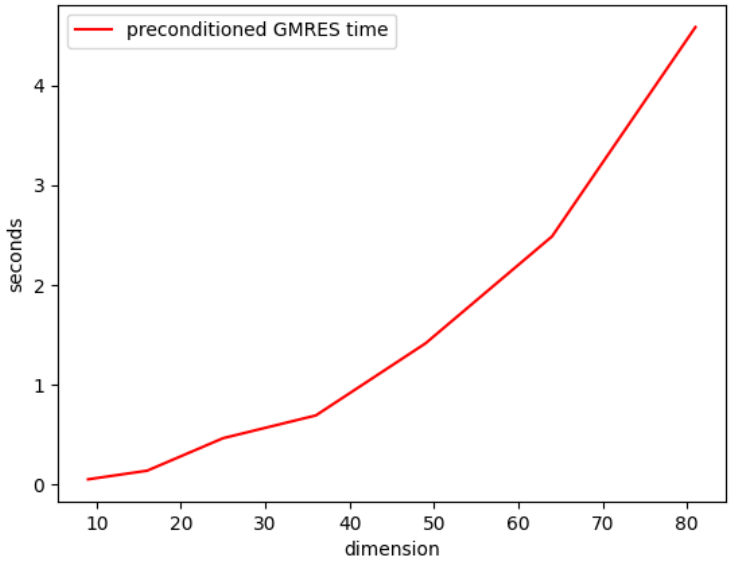

Figure 6: This shows how quickly preconditioned GMRES converged for different dimensions of matrices where $\rho_j = \frac{1}{2}^{j-1}$.

From the above figures, with the optimal acceleration parameters, GMRES takes less time to converge when compared to the 3 other lists of acceleration parameters. The lists vary by having constant values, quadratically increasing and quadratically decreasing values.

6 References

[1] A parallel alternating direction implicit preconditioning method *, by Jiang and Wong

[2] G.A. Watson, Approximation Theory and Numerical Methods (Wiley, New York, 1980).

[3] The Numerical Solution of Parabolic and Elliptic Differential Equations, by Peaceman and Rachford, Page 38

[4] Computational Linear Algebra lecture notes, Prof. Colin Cotter, Section 6,