Project Report

GMRES Implementation with a Preconditioner

CS 575: Numerical Linear Algebra
Spring 2023
John Tran

May 2, 2023

Introduction

Note: link the `git` repo here

The main topic is to explore the code implementation of the Generalized Minimal RESidual method (GMRES) applied with a simple preconditioner (usually to reduce the condition number of the problem and speed up convergence – i.e., with fewer iterations).

Due to time constraints, [1] will be heavily referenced as the main linear system and most pertinent details of GMRES are contained there (especially in the Problem, Algorithm, and Theoretical Results sections).

Problem

The linear system investigated for this project will be the steady-state solution (time independent) advection-diffusion problem of the form

$$au'(x) = \kappa u''(x) + \psi(x)$$
$$u(0) = \alpha, u(1) = \beta$$

as a 2-point boundary value problem with Dirichlet boundary conditions α, β on the interval $[0, 1]$ for x (equation (2.87) in [2])

We can rewrite the above equation by dividing both sides by a and gathering the derivatives of u on the left-hand side

$$\epsilon u''(x) - u'(x) = f(x)$$

where we let the inverse of the Péclet number $\epsilon = \kappa/a$ and $f(x) = \psi(x)/a$ (equation (2.88) in [2])

However, since the main focus will be on implementing GMRES and exploring the relative performance with other methods (discussed more in the Code and Implementation Details section), we can use a simplified version of the advection-diffusion equation above mentioned in [1] (equation (2.20) where $L = 1$). The advection-diffusion equation is also referred to as the convection-diffusion equation in [1]

$$-au''(x) + bu'(x) = 0, \quad 0 < x < 1 \tag{1}$$
$$u(0) = 0, u(1) = 1 \tag{2}$$

where there are some constant a, b for both u'', u', respectively, this time along with substitutions $f(x) = 0$ and the boundary constants $\alpha = 0, \beta = 1$ from above.

Given this simplification, we are given the exact solution [1]

$$u(x) = \frac{1 - e^{Rx}}{1 - e^{R}} \tag{3}$$

where the Péclet number $R = b/a$

If we look at the linear differencing equation for solving the linear equation $Ax = y$ for $A \in \mathbb{R}^{m \times m}$ (b replaced with y to prevent confusion with the b constant above in front of u') where $y = f(x) = 0$ for our simplification, we can use the centered schemas for both the first and second order derivatives of u

$$u' = \frac{u_{i+1} - u_{i-1}}{2h} \qquad \text{centered difference} \tag{4}$$
$$u' = \frac{u_{i+1} - u_{i}}{h} \qquad \text{forward difference} \tag{5}$$
$$u' = \frac{u_{i} - u_{i-1}}{h} \qquad \text{backward difference} \tag{6}$$
$$u'' = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \tag{7}$$

where h is the mesh step size ($h = 1/(m+1)$) and plugging these derivatives above,

$$-au''(x) + bu'(x) = 0 \tag{8}$$
$$-a\Big(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\Big) + b\Big(\frac{u_{i+1} - u_{i-1}}{2h}\Big) = 0 \tag{9}$$
$$-a\Big(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\Big) + b\Big(\frac{u_{i+1} - u_{i}}{h}\Big) = 0 \tag{10}$$
$$-a\Big(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\Big) + b\Big(\frac{u_{i} - u_{i-1}}{h}\Big) = 0 \tag{11}$$

where the equations (9) - (11) are given in the same order as above for equations (4) - (6) for centered, forward, and backward differences, respectively. In addition, any future equations given in relation to these three differences will be shown in the same order.

So, in order to construct the matrix A, we can group terms by u_{i-1}, u_i, u_{i+1} for values $i \in \{1, 2, \ldots, m\}$, remembering we have the boundary values $u_0 = u_{m+1} = 0$ so no extra vector is needed to account for them.

$$-(1-c)u_{i+1} + 2u_i - (1+c)u_{i-1} = 0 \tag{12}$$
$$-(1-c)u_{i+1} + (2-c)u_i - u_{i-1} = 0 \tag{13}$$
$$-u_{i+1} + (2+c)u_i - (1+c)u_{i-1} = 0 \tag{14}$$

as shown in [1] (equation (2.21)) where $c = Rh/2 = bh/2a$ for equation (12) and $c = Rh = bh/a$ for equations (13) and (14). The shared terms u_{i+1}, u_i, u_{i-1} are collected from equations (9) - (11) and multiplied on both sides by h^2/a to get equations (12) - (14).

Here, it would be important to discuss which differencing scheme should be used for u'. As shown in [1], we would need to look at the solution found by difference equations given above in equations (13) and (14). For brevity, the solutions will be given and in the form (same general form for all schemas)

$$u_i = \alpha + \beta\sigma^i \tag{15}$$

where $\alpha = -\beta$ because of the boundary condition $u_0 = 0$ and the boundary condition $u_{m+1} = 1$ produces

$$\alpha = \frac{1}{1-\sigma^{m+1}} \tag{16}$$
$$\sigma_{center} = \frac{1+c}{1-c} \tag{17}$$
$$\sigma_{forward} = \frac{1}{1-c} \tag{18}$$
$$\sigma_{backward} = 1+c \tag{19}$$

where c is defined below equations (13) and (14) for the different schemas. Plugging in everything for equation (15), we get the final solution for u_i

$$u_i = \frac{1-\sigma^i}{1-\sigma^{m+1}} \tag{20}$$

Looking at the equations for σ in (17) - (19) and the final solution for u_i in (20), we can see that as long as c is chosen so that $\sigma > 0$, then we should not encounter schema-specific issues (i.e., oscillations in u_i that would result in inaccurate solutions relative to the true solution in equation (3)). So, we have the following restraints on c for the different schemas

$$\begin{aligned} -1 < c_{center} &< 1 \\ c_{forward} &< 1 \\ c_{backward} &> -1 \end{aligned}$$

We can see that even though the centered schema produces the most accurate approximation of the three schemas, there are stricter constraints on c for solving the linear system numerically. It is also important to note that none of the three schemas produce symmetric tridiagonal matrices for A, but if $c \ll 1$, then A could be approximately symmetric. For exploration into GMRES as a general iterative method on non-symmetric (sparse) matrices, we would want to choose a c so the symmetry is broken (and compare performance with Conjugate Gradient that prefers SPD matrices).

For a simple visualization of the tridiagonal matrix A derived from these discretizations, we can let $m = 5$ and remember to multiply A by a/h^2 (reciprocal of the factor we premultiplied to get equations (12) - (14)) and recover our original system (to account for boundary conditions, modifying b). The entries in A are given in equations (12) - (14), but reversed since A from left to right uses increasing index of u_i instead of decreasing index and places where there are zeros will be omitted.

$$A_{center} = \frac{a}{h^2}\begin{bmatrix} 2 & -(1-c) & & & \\ -(1+c) & 2 & -(1-c) & & \\ & -(1+c) & 2 & -(1-c) & \\ & & -(1+c) & 2 & -(1-c) \\ & & & -(1+c) & 2 \end{bmatrix}$$

$$A_{forward} = \frac{a}{h^2}\begin{bmatrix} 2-c & -(1-c) & & & \\ -1 & 2-c & -(1-c) & & \\ & -1 & 2-c & -(1-c) & \\ & & -1 & 2-c & -(1-c) \\ & & & -1 & 2-c \end{bmatrix}$$

$$A_{backward} = \frac{a}{h^2}\begin{bmatrix} 2+c & -1 & & & \\ -(1+c) & 2+c & -1 & & \\ & -(1+c) & 2+c & -1 & \\ & & -(1+c) & 2+c & -1 \\ & & & -(1+c) & 2+c \end{bmatrix}$$

So y in $Ax = y$ would be equal to the zero vector (from the right-hand side of equation (1)) subtracted with our boundary conditions in equation (2). Since $u(0) = 0$, the only nonzero element of y would be the last element of y to account for $u(1) = 1$. However, the ***exact*** value of this element would change depending on what kind of schema used for the first order derivative of u. So y should be in the form (again, using $m = 5$ as a visualization)

$$y = \frac{a}{h^2}\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -\gamma \end{bmatrix}$$

where $\gamma = -(1-c)$ for centered and forward schemas and $\gamma = -1$ for backward schema as the coefficient in front of u_{i+1} (remembering how c was defined below equations (12) - (14)).

Note: since the constant a/h^2 is present on both sides of the equation $Ax = y$ (since 0 is on the right-hand side of equation (1)), we can ignore it in the implementation.

Now, to finally choose which schema should be used, the centered schema will be ignored because of the stronger presence of symmetry (want more non-symmetric for GMRES) in favor of either forward or backward schemas because of a (weak) diagonal dominance. From here, either forward or backward could be chosen so backward will be arbitrarily used where we need $c > -1$, or an easier constraint $c > 0$, and $c = Rh = bh/a$ and a, b are both chosen to be > 0 to satisfy the constraint on c. To reduce approximate symmetry when $c \ll 1$, we need to choose a large enough b relative to a so even for small values of h (testing large array sizes), the lack of symmetry is maintained. However, we also need to maintain the (weak) diagonal dominance to allow for reasonable convergence and not make c too large.

From here, the linear system is established and it is a matter of coding up GMRES to solve this system (and experimenting with different ways to implement GMRES and choice of parameters, discussed in the next section).

Algorithm

All the potential sources that I have found of interest will be found at the end of the report in the References section. Due to time constraints, only brief details concerning the algorithms are discussed for GMRES and potential preconditioners.

Note: it is interesting to see that the author of [1] was one of the people that helped develop the GMRES algorithm, so that will certainly be a useful reference form GMRES in general.

Another note: many of the sources mentioned in the proposal were cut out due to time constraints and [1] providing the pseudocode for GMRES and the structure of the preconditioning.

GMRES with Plane Rotations

To match the notation of [1], let $A \in \mathbb{R}^{n \times n}$.

First, GMRES is a Krylov subspace method in $\mathcal{K}_m$ of dimension m where any vector x in $x_0 + \mathcal{K}_m$ can be rewritten as (equation (6.25) in [1])

$$x = x_0 + V_m y \tag{21}$$

where V_m is the orthogonal matrix that acts as a basis of K_m (along the columns), writing x as a linear combination of the basis vectors in V_m.

For the approximation given by GMRES, it provides a unique vector in $x_0 + \mathcal{K}_m$ with the following solutions (equations (6.29) and (6.20) in [1]).

$$\begin{aligned} x_m &= x_0 + V_m y_m && (22) \\ y_m &= \underset{y}{\operatorname{argmin}} \|b - A(x_0 + V_m y)\| && (23) \\ &= \underset{y}{\operatorname{argmin}} \left\|\beta e_1 - \bar{H}_m y\right\| && (24) \end{aligned}$$

where $\bar{H}_m$ is the $m \times (m+1)$ Hessenberg matrix with the last row removed from H_m, the original square $(m+1) \times (m+1)$ Hessenberg matrix

From here, we have our basic algorithm for GMRES where V_m can be found using the Arnoldi iteration method (using Modified Gram-Schmidt (MGS) for simplicity and overall better performance for linear sparse systems, as mentioned in [1]). However, under Section 6.5.3 Practical Implementation Issues in [1], plane rotations are introduced as a way to measure the residual at each iteration, giving us a stopping condition. Without the rotations, the original GMRES algorithm has no way to access the residual in between iterations since the solution x_m is calculated at the very end. Towards the end of the section, details are discussed to implement the rotations in a progressive manner so we can calculate the norm of the residual at every iteration. Since the details are long and tedious, the use of rotations is mentioned and please look at the code for more information.

Adapting Algorithm 6.9 in [1] to include the plan rotations, we have the general algorithm for GMRES (some of the details are expanded further in the Theoretical Results section)

Algorithm 1 GMRES with plane rotations

```
 1: Choose an initial guess $x_0$
 2: Compute $r_0 = b - Ax_0, \beta = \|r_0\|_2, v_1 = r_0/\beta$
 3: for j = $1, 2, \ldots, m$ do
 4:     START Arnoldi iteration step
 5:     Compute $w_j = Av_j$
 6:     for $i = 1, 2, \ldots, j$ do
 7:         $h_{ij} = (w_j, v_i)$
 8:         $w_j = w_j - h_{ij} v_i$
 9:     end for
10:     $h_{j+1,j} = \|w_j\|_2$
11:     if $h_{j+1,j} == 0$ then                ▷ encountered with preconditioning
12:         Set $m = j$ and goto 22
13:     end if
14:     $v_{j+1} = w_j / h_{j+1,j}$
15:     END Arnoldi iteration step
16:     START plane rotation step
17:     Apply rotation matrix to rows $i, i+1$ of $h$ along columns $1, 2, \ldots j$
18:     Update $\bar{g}_m$ on indices $i, i+1$ of $\bar{g}_m$ according to equations given by
    [1]
19:     If the norm of the residual $|\gamma_{m+1}|$ is less than some threshold, goto 22
20:     END plane rotation step
21: end for
22: Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = \{h_{ij}\}$ for $1 \leq i \leq$
    $m+1, 1 \leq j \leq m$
23: With plane rotations, let $\bar{R}_m = \bar{H}_m$ where $\bar{R}_m$ is the upper triangular
    form of $\bar{H}_m$ and removing the zero'd row as an $(m+1) \times m$ matrix
24: Compute $y_m$ the minimizer of $\left\|\bar{g}_m - \bar{R}_m y\right\|_2$
25: Compute $x_m = x_0 + V_m y_m$
```

Preconditioning (Left)

The lack of robustness is a major weakness of iterative solvers when compared to direct solvers [1]. However, both the efficiency and robustness of iterative methods in general can be improved with preconditioning. The performance in general can also be influenced more by choice of preconditioner rather than the particular Krylov subspace method itself.

So, we would want the preconditioning matrix M to be a good approximation to A while also being inexpensive to solve the system $Mx = y$ (since the preconditioning step will be executed at every iteration). Due to time constraint and for simplicity, the left preconditioned GMRES algorithm will be used where

$$M^{-1}Ax = M^{-1}y$$

So, in Algorithm 1, there will be two lines that will be modified to apply the preconditioning step: line 2 where we compute $r_0 = M^{-1}(b - Ax_0)$ and line 5 at each iteration of the Arnoldi step where $w_j = M^{-1}Av_j$ is calculated. Usually, it would be hard to know what M^{-1} is in advance (but we do have access to M) so the modified systems $Mr_0 = b - Ax_0$ and $Mw_j = Av_j$ are solved instead.

For the choice of preconditioners, we have already discussed two simple ones in class from an iterative method perspective but can also be used as a preconditioner: Jacobi and Gauss Seidel [3] [4] [5] [6]. So, details on them will be skipped and the precondition matrix M will be given below

$$\begin{aligned} M_{JA} &= D \\ M_{GS} &= D + L \end{aligned}$$

where A can be decomposed into the parts D (main diagonal), L (strictly lower triangular), and U (strictly upper triangular).

In theory, these two preconditioners should be good enough for our numerical tests, but as discussed in the Numerical Results section, the gain from them are relatively minor. So, the SSOR preconditioner mentioned in [1] was also implemented. In addition, the choice $\omega = 1$ was used and SSOR is reduced to the Symmetric Gauss-Seidel (SGS) iteration

$$\begin{aligned} M_{SGS} &= (D + L)D^{-1}(D + U) \\ &= \hat{L}\hat{U} \end{aligned}$$

where

$$\hat{L} = (D+L)D^{-1} = I - LD^{-1}$$
$$\hat{U} = D + U$$

and $\hat{L}$ is a unit lower triangular matrix with the main diagonal and $\hat{U}$ is a an upper triangular matrix with the main diagonal.

From above, we can also see that M_{SGS} forms an approximate LU decomposition. So luckily, it should be relatively simple and inexpensive to use with a forward and backward solve. For example, to solve line 5 in Algorithm 1 above, we have $w_j = M_{SGS}^{-1} A v_j$ where

$$\begin{aligned} \text{foward solve} \quad & (I - LD^{-1})z = Av_j \\ \text{backward solve} \quad & (D+U)w_j = z \end{aligned}$$

Finally, as a general implementation to solve these systems of simple linear equations with any choice of M, the built-in solver was used in `Python`. Since sparse matrices are used with the `scipy.sparse` package, the sparse format of the matrices are used were made use of to speedup the solve (and due to time constraints since solving this modified system is not the focus of this project).

Theoretical Results

Note: Since the emphasis for this project is to look at the application and practical performance of GMRES, there will not be many theoretical results to discuss. However, the important properties related to the Krylov subspace and GMRES will be shown below.

Adapting the Proposition 6.9 given by [1], we have some useful properties of the plane rotations that allow us to check for the norm of the residual at each iteration with personal comments on how they help with the implementation.

Let $\bar{R}_m, \bar{g}_m = (\gamma_1, \ldots, \gamma_{m+1})^T$ be the corresponding matrix $\bar{H}_m$ and the right-hand side vector where $\bar{H}_m$ is transformed into an upper triangular form. We can let R_m, g_m be the $m \times m$ upper triangular matrix and m dimensional vector obtained from $\bar{R}_m, \bar{g}_m$ by deleting the last row and component, respectively.

1. *The rank of AV_m is equal to the rank of R_m. In particular, if $r_{mm} = 1$, then A must be singular*

   Note: useful for checking if the matrix A tested during implementation is singular, but following the linear system outlined above, this should not be an issue

2. *The vector y_m which minimizes $\left\|\beta e_1 - \bar{H}_m y\right\|_2$ is given by*

$$y_m = R_m^{-1} g_m$$

   Note: that is why we could modify what we are minizing in the GMRES algorithm provided above and still retain the capability to check the norm of the residual at each step

3. *The residual vector at step m satisfies*

$$b - Ax_m = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1})$$

   and as a result,

$$\|b - Ax_m\|_2 = |\gamma_{m+1}|$$

   Note: this means getting the norm of the residual at each iteration would ***basically not*** add any additional runtime to the GMRES algorithm (besides doing the plane rotation itself)

Numerical Results

For the code portion, a comparison of the performance between the following methods is planned:

- *LU* factorization (using the built-in `scipy.sparse.linalg.splu`)
- Conjugate Gradient (CG) (***not enough time – maybe cover in the slides***)

- GMRES
- Preconditioned GMRES
- Preconditioned CG (***not enough time – maybe cover in the slides***)

As discussed in office hours, the LU factorization could be used as a baseline to calculate the error with each iteration of the Krylov subspace methods (and thus determine the convergence). However, since the exact solution to either sets of boundary conditions mentioned above (fixed or variable α and β for Dirichlet boundary conditions) is known from [1] and [2], the exact solution may be used instead of the approximation given by LU factorization. Either way, since the main focus of the project is on GMRES, built-in functions will be used to find the LU factorization.

Note: instead of using the exact solution given in [1] and [2], the built-in general solver for sparse matrices `scipy.sparse.linalg.spsolve` was used instead to generalize the matrix A used (for future modifications and usage).

Then for the other two main methods (CG and GMRES), their algorithms will be coded up from scratch (CG should be fairly straightforward).

Note: ran out of time, so CG will not be discussed in this paper (maybe in the slides if time permits).

The code will be written in `Python` to code up the methods and plot the convergence with errors and runtime (with and without preconditioning). The linear system mentioned above will be sparse (some tridiagonal matrix for A) so sparse formats will be used with $scipy.sparse$ with CSR format, mainly for the built-in solvers (i.e., general solver and LU factorization).

Brief Implementation Details

For array sizes, the values $m = \{400, 800, 1600, 3200, 6400, 12800\}$ were used (i.e., smaller array sizes) to speed up debugging and testing. Also, the values $c = \{0.5, 1, 10, 100, 1000\}$ were also tested along with the preconditioners mentioned above for GMRES (i.e., Jacobi (JA), Gauss-Seidel (GS), SGS). The particular c value corresponds to the backward difference mentioned above where $c = Rh$. While testing out the linear system, b was initially chosen as a parameter to test various cases (scaling with m). However, there were inconsistent convergence rates and whether GMRES converged at all with this inconsistent scaling with b (consequently affecting R and c). This consequently did not give fine grain control over when GMRES would converge so c was chosen as a parameter instead and various values were tested.

Also, the max number of iterations was capped at 100 with a relatively low tolerance of `1e-14` to keep the runtime manageable on our local machine.

However, since this created a large number of combinations with these three parameters, only a few plots of importance are given in this report. The reasons for these particular choices and which values were used for the three parameters will be discussed in the following sections.

Please see the code for implementation details, but the `MATLAB` code given in [7] was referenced, especially concerning the iteration step of the plane rotations since [1] did not provide an accompanying psuedocode with the explanation on progressively applying the rotation. Also, a `zip` file containing all the generated plots will also be provided.

Residual Plots

For the residual plots shown below with increasing iteration number, only values $c = 0.5, 1, 10$ are shown at a fixed array size $m = 400$. The residual plots across the different array sizes showed similar behavior so $m = 400$ was arbitrarily chosen. However, for the c values, they gave very distinct plots. Before continuing, it should be noted that values of c greater than 10 yielded similar results as $c = 10$ so that is why the rest of the c values were not shown below for the residual plots.

The plots only show GMRES since the built-in solvers does not give us the residuals and CG was skipped due to time constraints.

For $c = 0.5$, a strange case occurred where a sub-linear convergence rate was encountered for the unpreconditioned GMRES and all preconditioners tested. Looking at how the residual changes, it is not the issue of oscillating solutions as mentioned above (schema-specific) so it may be due to the lack of a diagonally dominant matrix A or some other factor. The SGS preconditioner does for some reason start to gain momentum around 90 iterations where the residual starts as the worst of all the cases and significantly increases the rate of convergence.

Then for $c = 1$, another strange thing happens with the SGS preconditioner where only one dot is shown. When looking at the final norm of the residual with the true solution (based on the general solver) and the output given from the program, SGS actually was able to compute the true solution after the first iteration and exited GMRES very early (when the Arnoldi iteration returns a 0 at the last element that should be nonzero in the kth column of $\bar{H}$ at the kth iteration). Besides this tested value of $c = 1$, there was no other instances where GMRES exited early due to finding the exact solution early. Besides that, we can see that JA and GS perform similarly in these past two plots and will continue to be a pattern going forward.

Finally, for $c = 10$, the more general behavior is shown where SGS starts off with a faster convergence rate from the start and there is a more noticeable difference (though, still not much) between the performance of unpreconditioned GMRES and using JA or GS as a preconditioner. Again, there is not much difference in performance for JA and GS (the residual curves overlap almost exactly in the plot with very little difference in the final norm of the residual with the true solution). Also, the gain from JA and GS from unconditioned GMRES is rather small at just 1 iteration, which when taking into account the overhead of solving the $M^{-1}Ax = M^{-1}y$ system at each iteration, the runtime will be higher, as will be discussed in the following discussion.

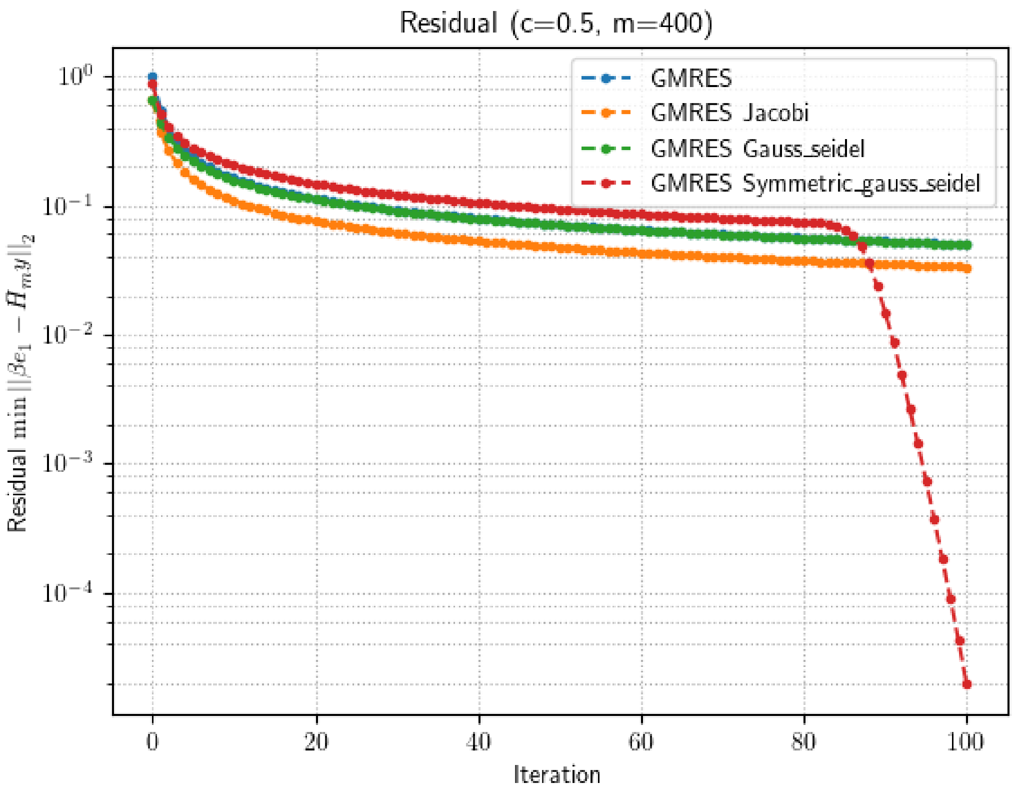

Figure 1: Residual plot with $c = 0.5, m = 400$

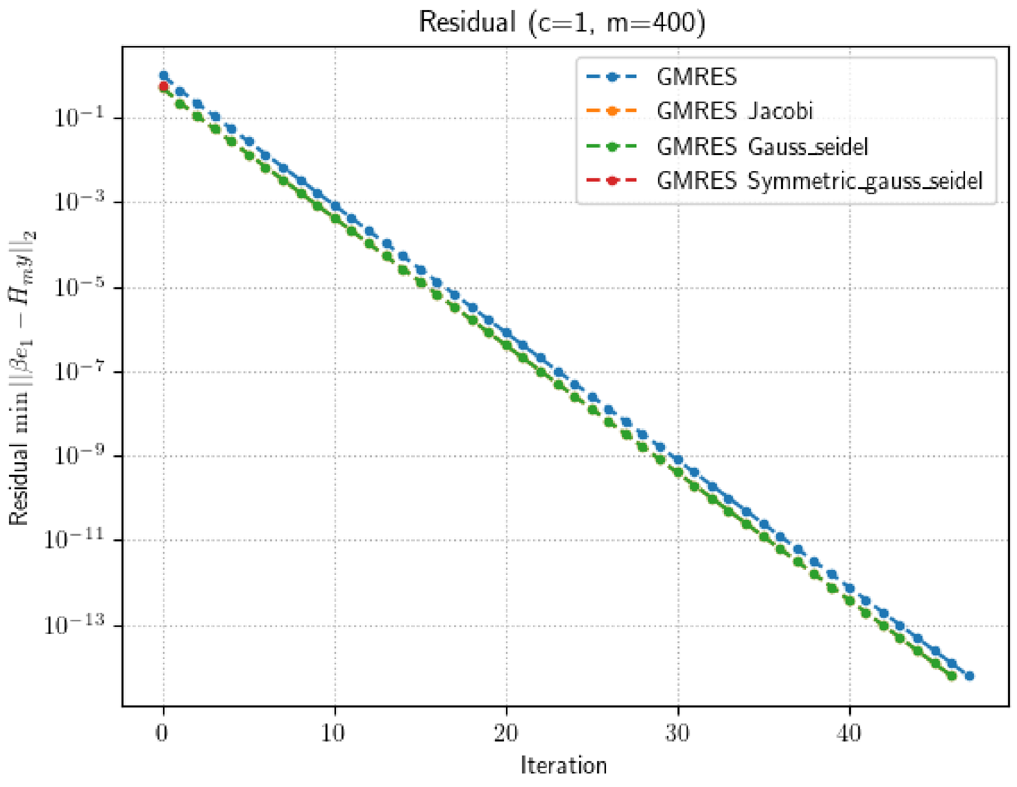

Figure 2: Residual plot with $c = 1, m = 400$

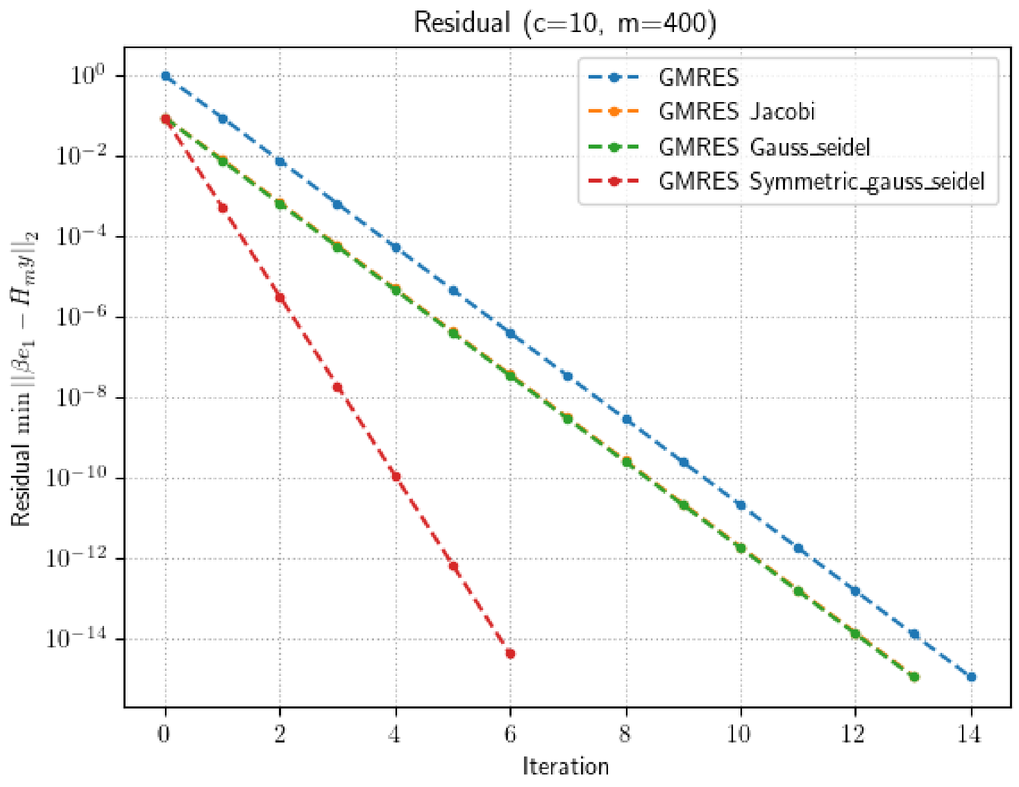

Figure 3: Residual plot with $c = 10, m = 400$

Runtime Plots

For the runtimes, a bar graph is produced to accommodate the split in total runtime between the normal GMRES operations and the time to complete the preconditioning step.

In these plots, both built-in solvers (general solver and LU) were included in the runtimes to see how our (naive) implementation compared to the optimized built-in solvers. The bars are color-coded but for clarity going from left to right, the bars correspond to the general sparse solver, LU, the unpreconditioned GMRES, then the preconditioners JA, GS, SGS in that order where the non-preconditioning time is on the bottom and the preconditioned time is stacked above.

For $c = 0.5$, we can see the relative time between the built-in solvers is quite large, which is confirmed with what we saw with the residual plot for $c = 0.5$. The convergence rate we observed was quite slow, and even when SGS started to pick up speed towards the end, the final norm of the residual was still modest at around `1e-5`.

Note: the bars are plotted on a `semilogy` plot with a log scale on the y-axis so the relative heights can be hard to determine (though, it allows the built-in runtimes to be put in the same plot without extreme scales).

Then for $c = 1$, we get to see the overall runtime of SGS with its very quick convergence after the first iteration. Unfortunately, the unpreconditioned time of SGS at just 1 iteration is still higher than the built-in solvers for all array sizes. The other two preconditioners show consistent behavior from what we observed from the runtime plots where the unpreconditioned time is comparable to unpreconditioned GMRES with the preconditioning adding a fair bit to the overall runtime. For this particular plot, it is easier to see that at $m = 12800$, the unpreconditioned time for these three cases are around $1e-1$ and with the preconditioning, it stands at around $2e-1$, which means it doubled the runtime. Even though the bar sizes may not look it, the preconditioning takes as much time as the normal steps of GMRES – not much gain either from what was shown in the residual plots.

Finally for $c = 10$, the runtimes of all the methods are faster in general (not hitting $1e - 1$ quite yet) and the gap between the built-in solvers and the GMRES implementations are closer, especially at the higher array sizes. From here, we can see that all of the GMRES implementations are now comparable in terms of runtime and makes comparisons a little bit easier since we can just look at the residuals without factoring in runtime to find the better performance. With SGS, even though the runtime is around the same order or magnitude (or slightly faster in some cases), the accuracy gain is quite substantial, relative to the other preconditioners as shown in the residual plot.

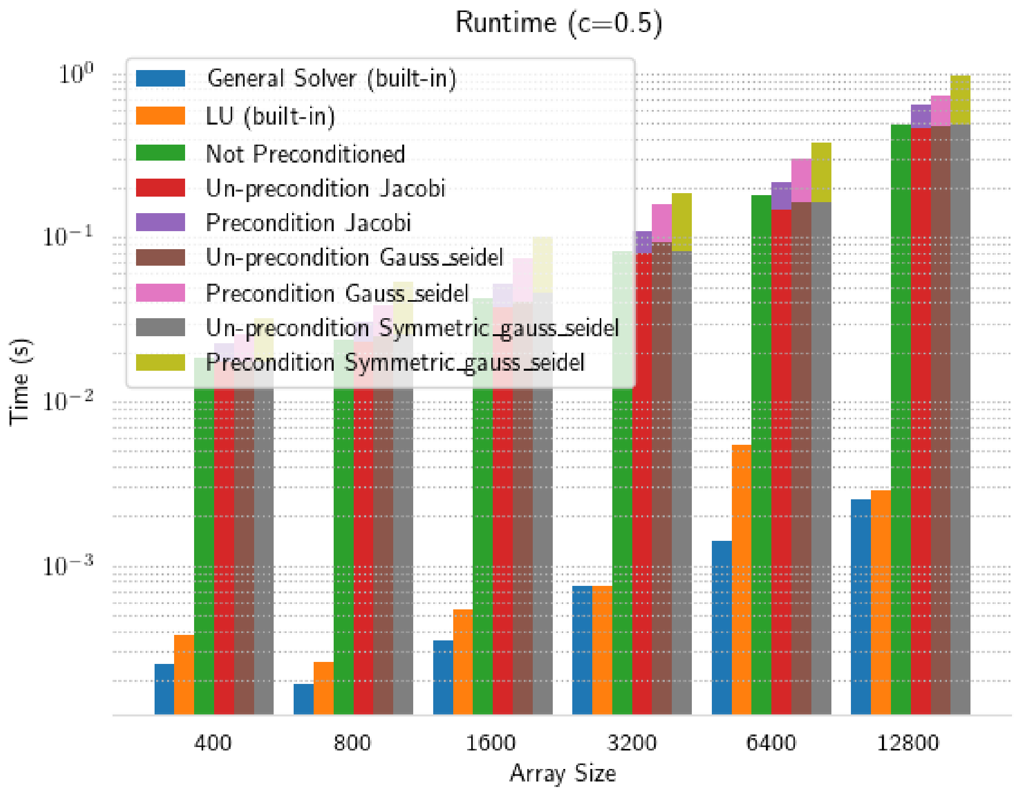

Figure 4: Runtime plot with $c = 0.5$

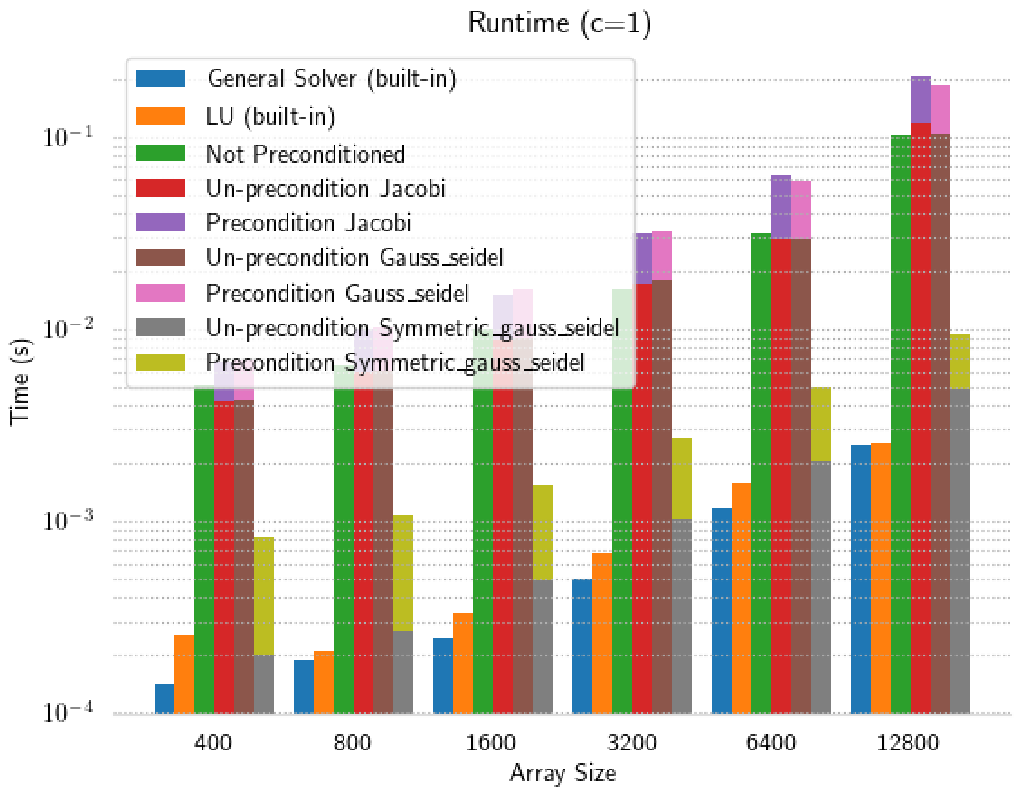

Figure 5: Runtime plot with $c = 1$

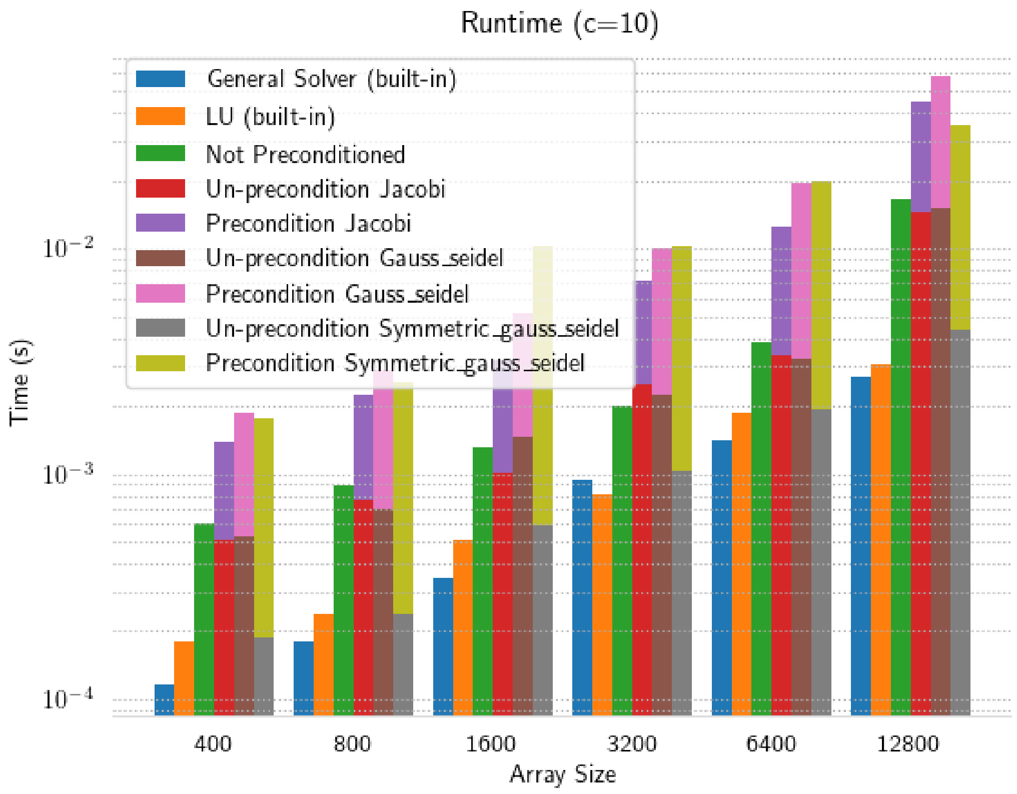

Figure 6: Runtime plot with $c = 10$

Note: the Results section of the report was rushed and more details could have been added, like convergence reference lines to check the rate of convergence and comparisons with other methods, like CG. However, with the time constraints, many ideas had to be scratched.

References

[1] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Second. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898718003. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

[2] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007. DOI: 10.1137/1.9780898717839. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898717839.

[3] J. Verschelde. "Krylov and Preconditioned GMRES." (Oct. 2022), [Online]. Available: http://homepages.math.uic.edu/~jan/mcs471/krylov.pdf.

[4] G. Fasshauer. "Preconditioning." (2006), [Online]. Available: http://www.math.iit.edu/~fass/477577_Chapter_16.pdf.

[5] A. J. Wathen, "Preconditioning," *Acta Numerica*, vol. 24, pp. 329–376, 2015, Found online copy here at the provided url. DOI: 10.1017/S0962492915000021. [Online]. Available: http://people.maths.ox.ac.uk/wathen/preconditioning.pdf.

[6] Wikipedia contributors. "Preconditioner — Wikipedia, the free encyclopedia." (2023), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Preconditioner&oldid=1141155872.

[7] Wikipedia contributors. "Generalized minimal residual method — Wikipedia, the free encyclopedia." (2022), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Generalized_minimal_residual_method&oldid=1126405909.