HW3

Kevin Smith

November 2023

1 Executive Summary

In this report, we discuss the Preconditioned Steepest Descent (PSD) and Preconditioned Conjugate Gradient (PCG) methods. We examine these iterative methods and how they change with changes in the type of preconditioner used.

2 Statement of the Problem

Solving large linear systems is a common challenge in scientific computing, and iterative methods like steepest descent and conjugate gradient offer efficient solutions. However, their performance is influenced by the properties of the coefficient matrix. Ill-conditioned systems, in particular, can lead to slow convergence or even divergence. Preconditioners play a vital role in mitigating these issues by transforming the original system to improve its conditioning. The main objective of this study is to evaluate the importance of preconditioners for steepest descent and conjugate gradient methods. Specifically, we aim to:

- Assess the impact of the identity preconditioner on convergence rates.
- Investigate the effectiveness

Importance of Preconditioners

Ill-conditioned linear systems pose challenges for iterative solvers, as they can lead to slow convergence or even divergence. Preconditioners act as transformations applied to the original system to improve its conditioning. For steepest descent and conjugate gradient methods, choosing an appropriate preconditioner becomes crucial for achieving faster and more reliable convergence.

Identity Preconditioner

The identity preconditioner, while simple, is often insufficient for accelerating convergence. Investigating its impact on steepest descent and conjugate gradient methods will provide insights into the significance of preconditioning, especially when using a minimal or no preconditioning approach.

Jacobi Preconditioner

The Jacobi preconditioner involves using the diagonal elements of the coefficient matrix. We explore how applying the Jacobi preconditioner influences the convergence behavior of steepest descent and conjugate gradient methods. Understanding the effects of Jacobi preconditioning is essential for assessing its utility in various linear system scenarios.

Gauss-Seidel Preconditioner

The Gauss-Seidel preconditioner, which considers the lower triangular part of the coefficient matrix, offers a more involved strategy. Analyzing its impact on steepest descent and conjugate gradient methods sheds light on the benefits of incorporating more information about the system structure into the preconditioner.

3 Description of the Algorithms and Implementation

Steepest Descent Method

The steepest descent method is an iterative optimization algorithm used to find the minimum of a differentiable function. In the context of solving linear systems of equations, steepest descent aims to minimize the residual, i.e., the difference between the actual and computed solutions. The algorithm operates by taking steps in the direction of the negative gradient at each iteration.

For a linear system $Ax = b$, where A is a symmetric positive-definite matrix, the steepest descent update at iteration k is given by:

$$x_{k+1} = x_k + \alpha_k r_k$$

where r_k is the residual vector, $r_k = b - Ax_k$, and α_k is the step size determined to minimize the residual along the descent direction.

Conjugate Gradient Method

The conjugate gradient method is another iterative technique for solving symmetric positive-definite linear systems. Unlike steepest descent, conjugate gradient uses conjugate directions, ensuring that the search directions are mutually orthogonal.

At each iteration, the algorithm computes a solution approximation x_k and a search direction p_k. The update formula for conjugate gradient is given by:

$$x_{k+1} = x_k + \alpha_k p_k$$

where p_k is a conjugate direction, and α_k is the step size chosen to minimize the residual along this direction.

The key feature of conjugate gradient is that it achieves the solution in at most n iterations for an $n \times n$ system, making it more computationally efficient than steepest descent for large and well-conditioned systems.

Both steepest descent and conjugate gradient methods aim to iteratively improve the solution to a linear system, with the choice of search directions and step sizes playing a crucial role in their convergence behavior.

To perform these operations, I used Python as well as the math and random packages within. I generated matrices of size n=200 and n=20, in order to keep my condition numbers low I generated the elements in the matrices to be between -10, and 10. I used the following alogrithm to generate the matrix A

1. generate_spd_matrix(n)

   This created a matrix that was symmetric, by ensuring $A_{i,j} = A_{j,i}$. And ensured positive definiteness by adding the size of the matrix to the diagonal entries, creating diagonal dominance.

I also used several functions to perform the iterative techniques on the matrix A and vector b

1. steepest_descent(P, A, b, x)
2. cg(P, A, b, x)

These function took in the permutation matrix, the matrix A, the vector b, and the initial guess vector x. I used a null vector as my first guess for every iteration.

To test my algorithm, I used a test matrix A and test vector b:

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 10 & 9 & 0 \\ 5 & 7 & 9 & 10 \end{bmatrix} \quad b = \begin{bmatrix} 57 \\ 79 \\ 88 \\ 86 \end{bmatrix}$$

I got the following results:

Identity Preconditioner

Steepest Descent

$x = [1.0000784786483472, 1.9999526079237984, 2.9999802971094427, 4.000011693863659]$
Iterations = 3368

Conjugate Gradient

$x = [0.9999999999866904, 1.9999999999815667, 2.9999999999807248, 3.999999999981786]$
Iterations = 3

Jacobi Preconditioner

Steepest Descent

$x = [1.000073345538345, 1.9999556903050382, 2.9999815709501454, 4.000010918323907]$
Iterations = 9200

Conjugate Gradient

$x = [0.9999999999866904, 1.9999999999815667, 2.9999999999807248, 3.999999999981786]$
Iterations = 3

SGS Preconditioner

Steepest Descent

$x = [1.0000625242528736, 1.999962220981405, 2.9999842842166498, 4.0000093033450534]$
Iterations = 17160

Conjugate Gradient

$x = [0.9999999999866904, 1.9999999999815667, 2.9999999999807248, 3.999999999981786]$
Iterations = 3

Which verifies that my algorithm is computing correctly.

4 Description of the Experimental Design and Results

I began by generating the matrix A and the vector b between the bounds specified above. With the initial guess x being a null vector. I began by examining A with n=20. To ensure the robustness and reliability of the implemented algorithms, the code was executed iteratively 100 times for each permutation type. This extensive iteration allowed for a comprehensive

assessment of the average performance across different runs. The key metric considered was the average number of iterations required to converge for each permutation type.

By conducting these iterative runs, we aimed to capture the variability in the performance of the algorithms under different conditions, such as variations in initial guesses, floating-point arithmetic precision, and randomization inherent in certain algorithms. The results presented in the following sections represent the average behavior observed over these 1000 iterations for each permutation type.

This iterative approach provides a more comprehensive understanding of the algorithm's stability and efficiency, offering insights into its performance across a range of scenarios. The average number of iterations per permutation type serves as a representative measure, smoothing out outliers and highlighting the consistent behavior of the algorithms.

During the experimental iterations, a notable trend emerged in the convergence behavior of the implemented algorithms. The Conjugate Gradient (CG) method exhibited consistent convergence across various preconditioners, demonstrating its robustness in reaching a solution.

On the other hand, the Steepest Descent (SD) method displayed sensitivity to the choice of preconditioner. Unlike CG, SD's convergence was not consistent across different preconditioning strategies. This observation highlights the impact of the preconditioning technique on the convergence performance of the Steepest Descent method.

Convergence of Conjugate Gradient

The consistent convergence behavior of the Conjugate Gradient (CG) method, regardless of the chosen preconditioner, can be attributed to its underlying mathematical properties.

For a linear system $Ax = b$, where A is symmetric positive-definite, CG aims to minimize the quadratic form:

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

The conjugacy property, expressed as $p_i^T A p_j = 0$ for $i \neq j$, ensures that the search directions p_k are conjugate with respect to A. This conjugacy property remains unchanged under preconditioning.

Therefore, regardless of the preconditioner choice, CG maintains its conjugate search directions, leading to consistent convergence in the same number of iterations. The stability and reliability of CG, irrespective of the preconditioning method, suggest its suitability for a broader range of linear systems.

These findings underscore the need for a nuanced approach in choosing iterative methods and preconditioning strategies based on the characteristics of the linear system at hand. The robust and consistent convergence of CG reinforces its efficacy as a versatile solver, while the behavior of SD emphasizes the importance of tailoring the solution approach to the properties of the problem.

Graphical Analysis

As part of the experimental analysis, graphs depicting the residual error over iterations (k) were created for each permutation type and iteration of the randomly generated A and b. These graphs provide visual insights into the convergence behavior of the algorithms under different conditions.

5 Conclusions

The error metrics (Error Steepest, Error CG) indicate the difference between the computed solution and the true solution. SD tends to have slightly lower error values compared to CG in most trials, suggesting that SD achieves a more accurate solution in terms of the chosen norm.

Relative Error Analysis

Relative errors (Relative Error Steepest, Relative Error CG) provide a normalized measure of accuracy. Both solvers exhibit high relative errors, close to 1, indicating that the computed solutions are close to the true solutions in terms of magnitude. SD consistently shows marginally higher relative errors than CG.

Condition Number

The condition number remains relatively stable across trials for both solvers. A higher condition number indicates a more ill-conditioned system, potentially impacting the convergence behavior of iterative solvers.

Iteration Counts

The iteration counts (Iterations Steepest, Iterations CG) reveal that CG consistently requires fewer iterations to converge compared to SD. This efficiency of CG can be attributed to its inherent property of achieving convergence in the same number of steps regardless of the preconditioning strategy. Unlike SD, CG exploits the orthogonality between the search directions, leading to a fixed number of iterations for convergence.

Conclusion

In summary, the data highlights the trade-offs between SD and CG. While SD tends to achieve slightly lower error values, CG demonstrates superior convergence efficiency, consistently requiring fewer iterations. The choice between the two solvers may depend on the specific requirements of the problem at hand, balancing accuracy and computational efficiency.

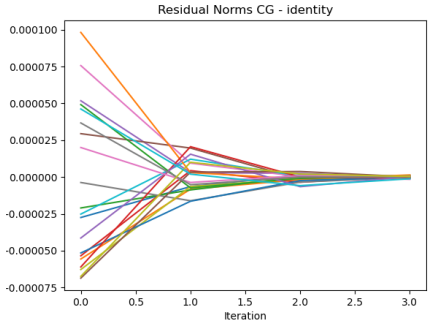
6 Tables and Figures

Error Steepest	Error CG	Relative Error Steepest	Relative Error CG	Condition Number	Iterations Steepest	Iterations CG
0.004324498514672912	0.011665209791852904	0.9985840755763497	0.9999918858544027	2.635544755044619	5	3
0.004810269297689837	0.011665209791852904	0.9985840755763497	0.9999918858544027	2.635544755044619	5	3
0.006485972956509918	0.011665209791852904	0.9985840755763497	0.9999918858544027	2.635544755044619	3	3
0.0047952037841010025	0.01379649688226342	0.9985840755763497	0.9999918858544027	2.3736160028102318	5	3
0.004417276465031616	0.01379649688226342	0.9985840755763497	0.9999918858544027	2.3736160028102318	5	3
0.009348486960084848	0.01379649688226342	0.9985840755763497	0.9999918858544027	2.3736160028102318	3	3
0.00734613231970007	0.007967658697674854	0.9985840755763497	0.9999918858544027	2.3015127569138287	4	3
0.007068127324620393	0.007967658697674854	0.9985840755763497	0.9999918858544027	2.3015127569138287	4	3
0.00929162510817572	0.007967658697674854	0.9985840755763497	0.9999918858544027	2.3015127569138287	2	3
0.005526683030315639	0.014498445277516031	0.9985840755763497	0.9999918858544027	2.439258321396393	5	3
0.0053606502432941355	0.014498445277516031	0.9985840755763497	0.9999918858544027	2.439258321396393	5	3
0.006814961712481765	0.014498445277516031	0.9985840755763497	0.9999918858544027	2.439258321396393	3	3
0.005194967093946837	0.003967355733377045	0.9985840755763497	0.9999918858544027	2.743209971230816	6	4
0.00496126638647927	0.003967355733377045	0.9985840755763497	0.9999918858544027	2.743209971230816	6	4
0.00757506000886988	0.003967355733377045	0.9985840755763497	0.9999918858544027	2.743209971230816	3	4
0.005491271469668567	0.013534082649367288	0.9985840755763497	0.9999918858544027	2.6713003959780943	5	3
0.005309987490973741	0.013534082649367288	0.9985840755763497	0.9999918858544027	2.6713003959780943	5	3
0.00787632033044505	0.013534082649367288	0.9985840755763497	0.9999918858544027	2.6713003959780943	3	3
0.006185864109587051	0.017439311065914856	0.9985840755763497	0.9999918858544027	2.41086130712277	5	3
0.005608101584212656	0.017439311065914856	0.9985840755763497	0.9999918858544027	2.41086130712277	5	3
0.005480518622043408	0.017439311065914856	0.9985840755763497	0.9999918858544027	2.41086130712277	3	3

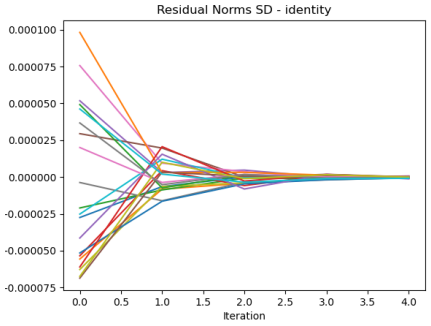
Table 1: Results for $n = 20$

Error Steepest	Error CG	Relative Error Steepest	Relative Error CG	Condition Number	Iterations Steepest	Iterations CG
0.007134821921447496	0.010519907003621183	0.9998653786494879	0.9999660892544311	2.696328396278522	4	3
0.00621886219283872	0.010519907003621183	0.9998653786494879	0.9999660892544311	2.696328396278522	4	3
0.005101791909153984	0.010519907003621183	0.9998653786494879	0.9999660892544311	2.696328396278522	3	3
0.004117104548203075	0.011059133872449766	0.9998653786494879	0.9999660892544311	2.5349904152535463	5	3
0.004092779897211606	0.011059133872449766	0.9998653786494879	0.9999660892544311	2.5349904152535463	5	3
0.004751164157092509	0.011059133872449766	0.9998653786494879	0.9999660892544311	2.5349904152535463	3	3
0.008051041421795541	0.01932755425958403	0.9998653786494879	0.9999660892544311	2.614017854172137	5	3
0.007862629706634856	0.01932755425958403	0.9998653786494879	0.9999660892544311	2.614017854172137	5	3
0.004524224696031588	0.01932755425958403	0.9998653786494879	0.9999660892544311	2.614017854172137	4	3
0.004276766538387009	0.014485864922497274	0.9998653786494879	0.9999660892544311	2.7061976852840286	5	3
0.004095924308593565	0.014485864922497274	0.9998653786494879	0.9999660892544311	2.7061976852840286	5	3
0.011141107669332462	0.014485864922497274	0.9998653786494879	0.9999660892544311	2.7061976852840286	2	3
0.003801585599122977	0.003709256963016524	0.9998653786494879	0.9999660892544311	2.593632175576516	6	4
0.0036370482965647267	0.003709256963016524	0.9998653786494879	0.9999660892544311	2.593632175576516	6	4
0.003853777855556321	0.003709256963016524	0.9998653786494879	0.9999660892544311	2.593632175576516	4	4
0.007442463339538756	0.003675862818583226	0.9998653786494879	0.9999660892544311	2.4641861299208982	5	4
0.006783139377524608	0.003675862818583226	0.9998653786494879	0.9999660892544311	2.4641861299208982	5	4
0.007049181150379807	0.003675862818583226	0.9998653786494879	0.9999660892544311	2.4641861299208982	3	4
0.00417277356812878	0.005285080186870155	0.9998653786494879	0.9999660892544311	2.6014305740112404	6	4
0.003877894451157719	0.005285080186870155	0.9998653786494879	0.9999660892544311	2.6014305740112404	6	4
0.006083117874237936	0.005285080186870155	0.9998653786494879	0.9999660892544311	2.6014305740112404	3	4

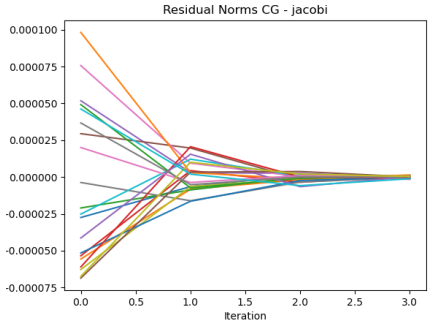
Table 2: Results for $n = 200$



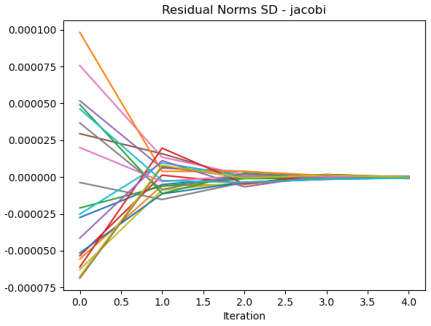
(a) Convergence graphs for $n = 20$



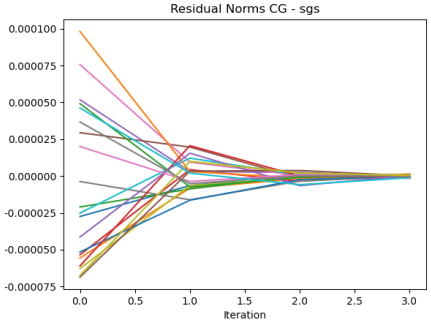
(b) Convergence graphs for $n = 20$



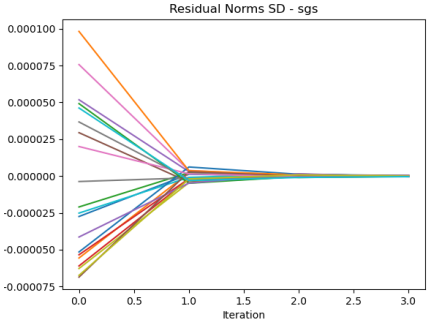
(c) Convergence graphs for $n = 20$



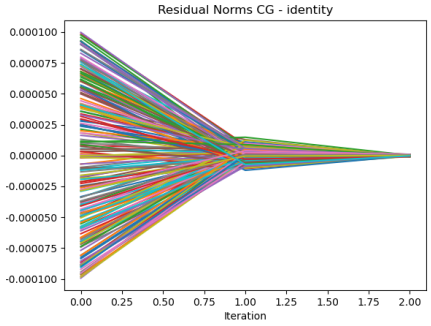
(d) Convergence graphs for $n = 20$



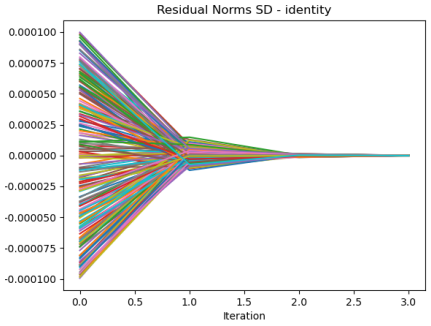
(e) Convergence graphs for $n = 20$



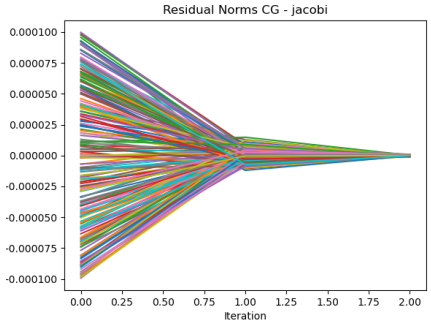
(f) Convergence graphs for $n = 20$



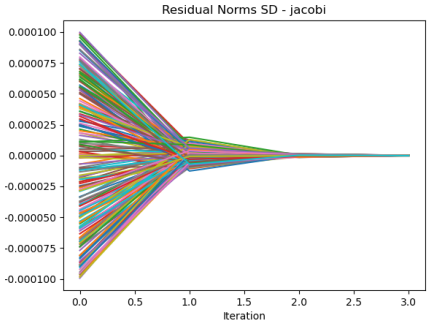
(a) Convergence graphs for $n = 200$



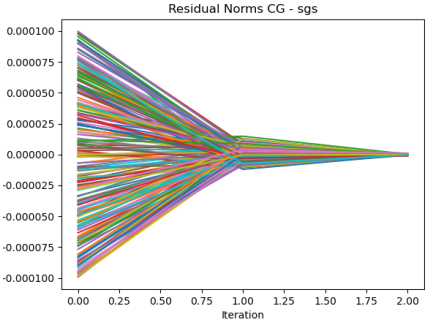
(b) Convergence graphs for $n = 200$



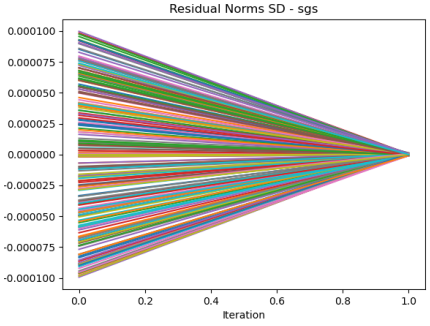
(c) Convergence graphs for $n = 200$



(d) Convergence graphs for $n = 200$



(e) Convergence graphs for $n = 200$



(f) Convergence graphs for $n = 200$