Preconditioning Techniques Analysis for CG Method

Huaguang Song

Department of Computer Science
University of California, Davis
hso@ucdavis.edu

Abstract

Matrix computation issue for solve linear system equation $Ax = b$ has been researched for years. There are many iterative techniques for solving linear system. For large sparse liner system, the convergence rate is always a concern for researchers, we can significantly improve the convergent rate by applying appropriate preconditoner. In this project, I am analysis and evaluate preconditioned conjugate gradient method with three different preconditioners, the Jacobi, Gauss-Seidel and incomplete Cholesky factorization preconditioner.

I. Introduction

Matrix computing arises in the solution of almost all linear and nonlinear systems of equations. As the computer power upsurges and high resolution simulations are attempted, a method can reach its applicability limits quickly and hence there is a constant demand for new and fast matrix solvers.

The congugate gradient iteration is one of the most important methods in scientific computation. It applies to symmetric positive definite systems and, for those systems. Since the convergence of an iterative method depends on the eigenvalues of the coefficient matrix, it is often advantageous to use a preconditioner that transforms the system to one with a better distribution of eigenvalues. Therefore, preconditioning is the key to a successful iterative solver.

In this report, there preconditioner has been studied and analyzed. The Jacobi, Gauss-Seidel and incomplete Cholesky factorization preconditioner. The convergence rate, computation time and error rate of the preconditioned conjugate gradient method with different preconditoner has been evaluated.

II. Background

The Conjugate Gradient (CG) method was originally proposed in 1952 by Hestenes et al.[6]. It is for solving large scale symmetric definite linear system of equations. The general idea of the algorithm is shown in Algorithm 1.

A. Preconditioning

Definition: Let $Ax = b$ is the linear system to be solved and M a matrix that is easy to invert. If *Cond* $(M^{-1}A)$ is much smaller than *Cond* (A), then M is called a *preconditioning* of A. The system $M^{-1}Ax = M^{-1}b$ has the same solution as $Ax = b$ and is said to be a *preconditioned* system.

Algorithm 1: Conjugate Gradient (CG)

```
$\mathbf{x}_0 = \mathbf{0}, \quad \mathbf{r}_0 = \mathbf{b}, \quad \mathbf{p}_1 = \mathbf{r}_0 = \mathbf{b}.$
For $k = 1, \ldots, n-1$:
    $\alpha_k = \frac{(\mathbf{r}_{k-1}, \mathbf{r}_{k-1})}{(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k)}$
    $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{A}\mathbf{p}_k$
    $\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha_k \mathbf{p}_k$
    $\beta_k = \frac{(\mathbf{r}_k, \mathbf{r}_k)}{(\mathbf{r}_{k-1}, \mathbf{r}_{k-1})}$
    $\mathbf{p}_{k+1} = \mathbf{r}_k + \beta_k \mathbf{p}_k$
```

Follows C. T. Kelley[7]

Since A is symmetric positive definite, we will choose a matrix M that is symmetric positive definite and approximates A. Then, $M^{-1}A$ should have a smaller condition number, and we apply the CG method to the preconditioned system in the hope that the CG iteration will converge faster. Of course, in addition to wanting *Cond* $(M^{-1}A) \ll$ *Cond* (A), M must be chosen so that $M^{-1}A$ is symmetric positive definite. Also, there must be a means of quickly and accurately computing $M^{-1}A$ either implicitly or explicitly.

B. Choosing the Preconditioner

There are seven types of preconditioners and they can be divided roughly into three categories.

Type 1	Matrix splitting preconditioner
Type 2	Approximate inverse preconditioner
Type 3	Multilevel (approximate inverse) preconditioner
Type 4	Recursive Schur complements preconditioner
Type 5	Matrix splitting and Approximate inverses
Type 6	Recursive Schur complements preconditioner
Type 7	Implicit wavelet preconditioner

Table 1: List of preconditioners types

I.	Preconditioners designed for general classes of matrices
II.	Preconditioners designed for broad classes of underlying problems
III.	Preconditioners designed for a specific matrix or underlying problems

Table 2: Three categories of preconditioners

There is an extensive literature concerning preconditioning that includes the CG method as well as other iterations [1, 2, 3]. I will introduce three commonly used techniques, the Jacobi, Gauss-Seidel and incomplete cholesky factorization, for preconditioning the CG method.

Jacobi and Gauss-Seidel preconditioners are two simple preconditioners, which can be implemented easily. These two preconditioners are derived from iterative methods respectively known as the Jacobi and the symmetric Gauss-Seidel methods, respectively we thus call them the Jacobi and Gauss-Seidel preconditioners.

Jacobi preconditioner is the simplest preconditioner consists of just the diagonal of the matrix *A*:

$$M_{i,j} = \begin{cases} a_{i,j} \; if \; i = j \\ 0 \;\; otherwise \end{cases} \quad M_J\text{=diag(A)}$$

It is very simple and cheap, and might improve certain problems but usually insufficient.

The Gauss-Seidel preconditioner is another simple preconditioner that consists of the lower triangular part of matrix *A*, where $M_{GS} = D + L$. However, the matrix M_{GS} is not symmetric, which means it is not appropriate for the CG algorithm. Therefore, by combined effect of a forward sweep using the lower triangular component of *A* followed by a backward sweep using the upper component of *A* can form the symmetric Gauss-Seidel:

$$\mathrm{M}_{SGS} = (D + L)D^{-1}(D + U)$$

Since *M* is symmetric positive definite, the Cholesky factorization guarantees there is an upper triangular matrix *R* such that $M = R^T R$. We will use this factorization to obtain an equivalent system $Ax = b$, where *A* is symmetric positive definite, so the CG method applies.

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

$$(R^T R)^{-1} Ax = (R^T R)^{-1}x$$

$$(R^T)^{-1} A(R)^{-1} Rx = (R^T)^{-1}b$$

$$((R^{\mathrm{T}})^{-1} AR^{-1} Rx = (R^{\mathrm{T}})^{-1}b$$

System (1) is equivalent to the original system $Ax = b$.

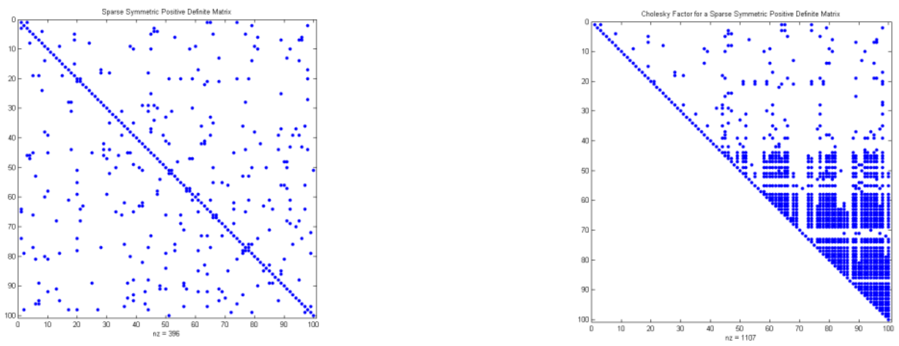
$$\bar{A}\bar{x} = \bar{b}, where\ \bar{A} = (R^{-1})^T AR^{-1}, \bar{x} = Rx, \bar{b} = (R^{-1})^T b \qquad (1)$$

That fact that $\bar{A}$ is positive definite because

$$x^{\mathrm{T}} (R^{-1})^{\mathrm{T}} AR^{-1}x = (R^{-1}x)^{\mathrm{T}} A\ (R^{-1}x)>0,\ \mathrm{x}\neq 0.$$

The CG method applies to $\bar{A}\bar{x} = \bar{b}$. After obtaining $\bar{x}$, we can find x by solving $Rx = \bar{x}$.

As we know, iterative methods are applied primarily to large, sparse systems. However, the Cholesky factor R used in system (1) is usually less sparse than M. Figure 1(a) shows the distribution of nonzero entries in a 100×100 sparse matrix, and (b) shows the distribution in the Cholesky factor. Note the significant loss of zeros in the Cholesky factor.



(a) (b)

Figure 1.: Cholesky factorization of a sparse symmetric positive definite matrix

In the incomplete Cholesky factorization preconditioning method, we form $M = R^T R$ from a modified Cholesky factor of A with the hope that the condition number of $M^{-1}A$ is considerably smaller than that of A. If an element a_{ij} off the diagonal of A is zero, the corresponding element $\overline{r_{ij}}$ is set to zero. The factor returned, $\bar{R}$, has the same distribution of nonzeros as A above the diagonal. The incomplete Cholesky factorization algorithm (See Algorithm 3) is a simple modification of the original Cholesky algorithm(See Algorithm 2).

III. Algorithm

Given a symmetric positive definite $A \in R^{n\times n}$, the following algorithm computes a lower triangular $R \in R^{n\times n}$ such that $A = RR^T$. For all $i \geq j$, $R(i, j)$ overwrites $A(i, j)$.

Algorithm 2: Cholesky Factorization

```
for $k = 1 : n$
    $A(k,k) = \sqrt{A(k,k)}$
    for $i = k+1 : n$
        $A(i,k) = A(i,k)/A(k,k)$
    end
    for $j = k+1 : n$
        for $i = j : n$
            $A(i,j) = A(i,j) - A(i,k)A(j,k)$
        end
    end
end
```

Follows Gene H. Golub[8]

Given a symmetric positive definite $A \in R^{n \times n}$, an easy but effective way to determine an incomplete Cholesky H that approximates R is to step through the above Cholesky reduction setting h_{ij} to zero if the corresponding a_{ij} is zero. Pursuing this with Cholesky factorization we obtain Algorithm 3. In our experiment, for simplicity purpose, I choose IC(0), which means incomplete Cholesky factorization with 0 fill.

Algorithm 3: Incomplete Cholesky Factorization

```
for $k = 1 : n$
    $A(k,k) = \sqrt{A(k,k)}$
    for $i = k+1 : n$
        if $A(i,k) \neq 0$
            $A(i,k) = A(i,k)/A(k,k)$
        end
    end
    for $j = k+1 : n$
        for $i = j : n$
            if $A(i,j) \neq 0$
                $A(i,j) = A(i,j) - A(i,k)A(j,k)$
            end
        end
    end
end
```

Follows Gene H. Golub[8]

By combining the CG and incomplete Cholesky factorization algorithms, we obtain the following PCG algorithm.

Algorithm 4: Preconditioned Conjugate Gradients (PCG)

```
        $\mathbf{x}_0 = \mathbf{0}, \quad \mathbf{r}_0 = \mathbf{b}$; solve $\mathbf{M}\mathbf{z}_0 = \mathbf{b}$; $\mathbf{p}_1 = \mathbf{z}_0$
For $k = 1, \ldots, n-1$ :
        $\alpha'_k = \frac{(\mathbf{z}_{k-1}, \mathbf{r}_{k-1})}{(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k)}$
        $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{A}\mathbf{p}_k$
        $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$
        solve $\mathbf{M}\mathbf{z}_k = \mathbf{r}_k$
        $\beta'_k = \frac{(\mathbf{z}_k, \mathbf{r}_k)}{(\mathbf{z}_{k-1}, \mathbf{r}_{k-1})}$
        $\mathbf{p}_{k+1} = \mathbf{z}_k + \beta'_k \mathbf{p}_k,$
```

Follows C. T. Kelley[7]

In this project, I implemented the PCG algorithm with three preconditioners, the implementation are very straight forward.

[x, numIter] = myPCG(A, b, v0, M1, M2, tol, maxiter)

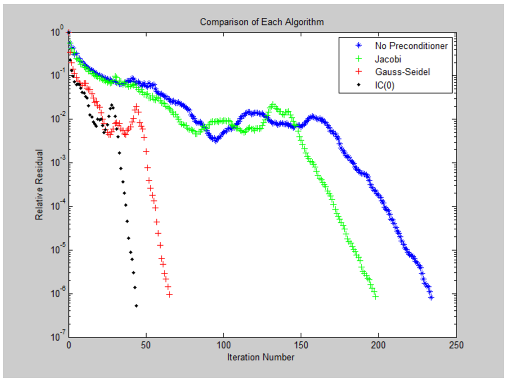
The function will take the following as input: SPD matrix *A*, right-hand side Nx1 column vector *b*, Nx1 start vector (initial guess) *v0*, preconditioner *M1* and *M2*, break condition *tol*, and maximum number of iterations to perform *maxiter*. It will return *x*, the solution vector, and number of actually performed iterations *numIter*.

IV. Experiment Results:

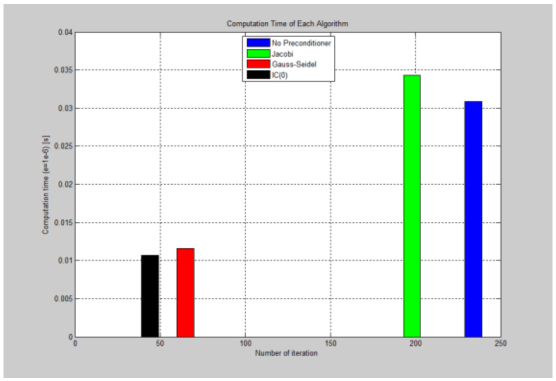
Name	Size	Properties
nos3	960 x 960, 8402 entries	real symmetric positive definite
bcsstk27	1224 x 1224, 28675 entries	real symmetric positive definite

Table 3: Testing matrix information

Table 3 list the testing matrix used in this experiment, all data can be downloaded from matrix market.



(a) Number of Iteration



(b) Computation Time

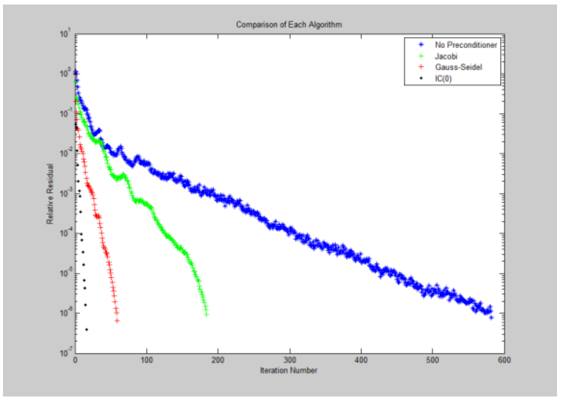
Figure 2. Comparison of each Algorithms for Matrix ‘nos3’

NumberIteration (No Preconditioner) = 234
NumberIteration (Jacobi) = 198
NumberIteration (Gauss-Seidel) = 65
NumberIteration (IC(0)) = 44

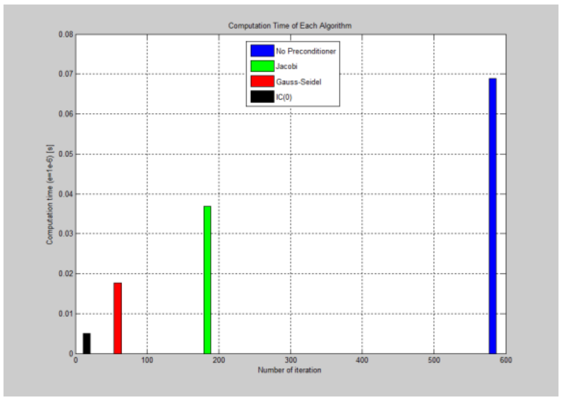
RunTime (No Preconditioner) = 0.0309 s
RunTime (Jacobi) = 0.0344 s
RunTime (Gauss-Seidel) = 0.0116 s
RunTime (IC(0)) = 0.0107 s

ApproximationError = $||(|X_{approx}-X_{true}|/|X|)||_2$, where X_{approx} is the approximation solution for PCG with each proconditioner, and X_{true} is the result calculated by A\b.

ErrorRate(No Preconditioner) = 1.1113e-05
ErrorRate(Jacobi) = 1.4706e-05
ErrorRate(Gauss-Seidel) = 2.4526e-05
ErrorRate(IC(0)) = 9.2194e-06



(a)Number of Iteration



(b) Computation Time

Figure 3. Comparison of each Algorithms for Matrix 'bcsstk27'

NumberIteration (No Preconditioner) = 582
NumberIteration (Jacobi) = 184
NumberIteration (Gauss-Seidel) = 59
NumberIteration (IC(0)) = 16

RunTime (No Preconditioner) = 0.0689 s
RunTime (Jacobi) = 0.0369 s
RunTime (Gauss-Seidel) = 0.0177 s
RunTime (IC(0)) = 0.0050 s

ErrorRate(No Preconditioner) = 0.0015
ErrorRate(Jacobi) = 3.2200e-04
ErrorRate(Gauss-Seidel) = 4.4557e-04
ErrorRate(IC(0)) = 9.2194e-06

Clearly, the preconditioning helped significantly, the number of iterations (shown in Figure 2) for PCG without preconditioner (the common CG method) took 234 iterations, with Jacobi preconditioner, number of iterations drop to 198, and it has a little improvement. And for Gauss-Seidel, the number of iterations is 65, and with incomplete Cholesky IC(0) factorization preconditioning method, it only took 44 iterations. Therefore, by apply appropriate preconditioner, the convergence rate increase significantly, where we can see in this case the best result is 44 iterations compare to 234 iteration where we have 582 v.s. 16 iterations shown in Figure 3, that's a huge improvement.

The running time is another issue for choosing a good preconditioner. As we can see from Figure 2 and Figure 3, the computation time is proportional to the number of iteration. The less the number of iteration the less computation time is required. Therefore, computation time for IC(0) < Gauss-Seidel < Jacobi < No Preconditioner for both testing case. This is also corresponding to the number of iteration where IC(0) < Gauss-Seidel < Jacobi < No Preconditioner. However, if we force the PCG run for fix number of iteration (say 200) for all three precondítoners we have the following results. As we can see IC(0) takes longer computation time the others. Jacobi is actually very fast.

RunTime (No Preconditioner) = 0.0428 s
RunTime (Jacobi) = 0.0578 s
RunTime (Gauss-Seidel) = 0.0849 s
RunTime (IC(0)) = 0.0867 s

The error rate is quite small even without preconditoner. In this experiment, I pick tolerance 1e-6 for all the precondioners. For both test case, the IC(0) has the smallest error rate, and all other three preconditoners are around the same.

V. Conclusion:

In this project, I have analysis and evaluate three preconditioners for preconditioned conjugate gradient method. I found the IC(0) preconditoner has the best convergence rate for both cases because it has the least number of iterations for convergence. The CG method without preconditoner has the largest number of iterations. The Jacobi preconditioner are simple and cheap, but not very effective, where Gauss-Seidel are in the middle between Jacobi and IC(0). Overall, the IC(0) preconditioner has the best performance results in number of iteration and computational time.

However, there are still many other preconditoners can be studied, such as parallelism, Coarse-grid approximations, and modified IC factorization. For different situation and matrix properties choose a good preconditioner is very importatent. It could significantly reduce the amount of computation time and allow fast convergence.

Acknowledgement

The author would like to thank Professor Zhaojun Bai for supporting useful materials regarding to this project.

References

[1] Biswa Nath Datta. *Numerical Linear Algebra and Applications*. SIAM, Philadelphia, 2 edition, 2010.

[2] Gene H. GolubGolub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3 edition, 1996.

[3] David S. Watkins. *Fundamentals of Matrix Computations*. Wiley, 3 edition, 2010.

[4] Barrett, R., M. Berry, T. F. Chan, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods,* SIAM, Philadelphia, 1994.

[5] Saad, Youcef and Martin H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, July 1986, Vol. 7, No. 3, pp. 856-869.

[6] Hestenes, Magnus R.; Stiefel, Eduard (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems". *Journal of Research of the National Bureau of Standards* 49 (6).

[7] C. T. Kelley, “Iterative Methods for Optimization (Frontiers in Applied Mathematics)”, *Society for Industrial and Applied Mathematics*; 1 edition (January 1, 1987)

[8] Gene H. Golub, Charles F. Van Loan, “Matrix Computations”, JHU Press, Oct 15, 1996