Chapter 8

Finite Pointset Method and Preconditioners

8.1 Introduction to Preconditioners

The convergence rate of iterative methods depends on spectral properties of the coefficient matrix. Hence, one may attempt to transform the linear system into one that is equivalent in the sense that it has the same solution, but that has more favorable spectral properties of the coefficient matrix A. A preconditioner is a matrix that effects such a transformation. For instance, if a matrix M approximates the coefficient matrix A in some way, the transformed system

$$M^{-1}Ax = M^{-1}B \tag{8.1}$$

has the same solution as the original system $Ax = b$, but the spectral properties of its coefficient matrix $M^{-1}A$ may be more favorable. In devising a preconditioner, we are faced with a choice between finding a matrix M that approximates A, and for which solving a system is easier than solving one with A, or finding a matrix M that approximates A^{-1}, so that only multiplication by M is needed. Since using a preconditioner in an iterative method includes some extra cost, both initially for the setup, and per iteration for applying it, there is a trade-off between the cost of constructing and applying the preconditioner, and the gain in convergence speed. Certain preconditioners need little or no construction phase at all (for instance the SSOR preconditioner), but for others, such as incomplete factorizations, there can be substantial work involved.

Now, we shall start with different preconditioners.

8.1.1 Jacobi Preconditioner

The simplest preconditioner consists of just the diagonal of the matrix

$$m_{i,j} = \begin{cases} a_{i,i} & \text{if } i = j; \\ 0 & \text{otherwise.} \end{cases}$$

This is known as the Jacobi preconditioner. It is possible to use this preconditioner without using any extra storage beyond that of the matrix itself. However, division operations are usually quite costly, so in practice storage is allocated for the reciprocals of the matrix diagonal. This strategy applies to block Jacobi preconditioners.

8.1.2 Block Jacobi Preconditioner

Block versions of the Jacobi preconditioner can be derived by a partitioning of the variables. If the index set $S = \{1, 2, 3......, n\}$ is partitioned as $S = \bigcup_i S_i$ with the sets S_i mutually disjoint, then

$$m_{ij} = \begin{cases} a_{i,j}; & \text{if } i \text{ and } j \text{ are in the same index subset;} \\ 0; & \text{otherwise.} \end{cases}$$

The preconditioner is now a block-diagonal matrix. Natural choices for the partitioning is as follows:

- In problems with multiple physical variables per node, blocks can be formed by grouping the equations per node.
- In structured matrices, such as those from partial differential equations on regular grids, a partitioning can be based on the physical domain. Examples involve partitioning along lines in the $2D$ case, or planes in the $3D$ case.
- On parallel computers, it is natural to let the partitioning coincide with the division of variables over the processors.

8.1.3 Complete and Incomplete Factorization Preconditioner

A class of preconditioner is of the form complete factorization of coefficient matrix A. A broad class of preconditioners is based on incomplete factorizations of the coefficient matrix. We call a factorization incomplete if, during the factorization process, certain fill elements, nonzero elements in the factorization in positions where the original matrix had a zero, have been ignored. Such a preconditioner is then given in factored form $M = LU$ where L is a lower triangular matrix and U is an upper triangular matrix. The efficiency of the preconditioner depends on how well M^{-1} approximates A.

Creating an incomplete factorization

Incomplete factorizations are the first preconditioners for which there is a non-trivial creation stage. Incomplete factorizations may break down (attempted division by zero pivot) or result in indefinite matrices (negative pivots) even if the full factorization of the same matrix is guaranteed to exist and yield a positive definite matrix.

An incomplete factorization is guaranteed to exist for many factorization strategies if the original matrix is an M-matrix. This was originally proved by Meijerink and Van der Vorst (see [VB00]). Also see the references [BQ76], [Man80] and [Vor81] for more details.

In cases where pivots are zero or negative, strategies have been proposed such as substituting an arbitrary positive number (see [Ker78]), or restarting the factorization on $A + \alpha I$ for some positive value of α (see [Man80]).

An important consideration for incomplete factorization preconditioners is the cost of the factorization process. Even if the incomplete factorization exists, the number of operations involved in creating it is at least as much as for solving a system with such a coefficient matrix, so the cost may equal that of one or more iterations of the iterative method. On parallel computers, this problem is aggravated by the generally poor parallel efficiency of the factorization. Such factorization cost can be amortized, if the same preconditioner will be used for several linear systems.

8.1.4 Point Incomplete Factorization

The most common type of incomplete factorization is based on taking a set S of matrix positions, and keeping all positions outside this set equal to zero during the factorization. The resulting factorization is incomplete in the sense that the fill is suppressed.

The set S is usually chosen to encompass all positions (i, j) for which $a_{i,j} \neq 0$. A position that is zero in A but not so in exact factorization is called a *fill position*, and if it is outside S, the fill is said to be *discarded.* Often, S is chosen to coincide with the set of nonzero positions in A, discarding all fills. This factorization type is called the ILU(0) factorization: the Incomplete LU factorization of level zero.

The accuracy of the ILU(0) factorization may be insufficient to yield an adequate rate of convergence. More accurate Incomplete factorizations are often more efficient as well as more reliable. These more accurate factorizations will differ from ILU(0) by allowing some fill-in. In hierarchy, more accurate incomplete factorizations are ILU(1), ILU(2), ILU(3) and so on (see [Saa00]).

8.2 Iterative Methods and its Solvers

The term *iterative method* refers to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system at each step. There are two types of iterative methods, namely stationary methods and nonstationary methods.

- Stationary methods are older, simpler to understand and to implement, but usually not as effective. Jacobi method, Gauss-Seidal method, Successive Overrelaxation method and Symmetric Successive Overrelaxation method lies in the category of stationary methods.
- Nonstationary methods are a relatively recent development; their analysis is usually harder to understand, but they can be highly effective. The nonstationary methods, which we present here, are based on the idea of sequences of orthogonal vectors.

The rate at which an iterative method converges depends greatly on the spectrum of the coefficient matrix. Hence, iterative methods usually involve a second matrix that transforms the coefficient matrix into one with a more favorable spectrum. The transformation matrix is called a *preconditioner.* A good preconditioner improves the convergence of the iterative method, sufficiently to overcome the extra cost of constructing and applying the preconditioner. Indeed, without a preconditioner the iterative method may even fail to converge.

8.2.1 Nonstationary Iterative Method

Conjugate Gradient (CG) method

The conjugate gradient method derives its name from the fact that it generates a sequence of conjugate (or orthogonal) vectors. These vectors are the residuals of the iterates. They are also the gradients of a quadratic functional, the minimization of which is equivalent to solve the linear system. CG is an extremely effective method when the coefficient matrix is symmetric positive definite, since storage for only a limited number of vectors is required.

Minimum Residual (MINRES) method and Symmetric LQ (SYMMLQ) method

These methods are computational alternatives for CG for coefficient matrices that are symmetric but possibly indefinite. SYMMLQ will generate the same solution iterates as CG if the coefficient matrix is symmetric positive definite.

Generalized Minimal Residual (GMRES) method

The Generalized Minimal Residual method computes a sequence of orthogonal vectors (like MINRES), and combines these through a least-squares solve and update. However, unlike MINRES (and CG) it requires storing the whole sequence, so that a large amount of storage is needed. For this reason, restarted versions of this method are used. In restarted versions, computation and storage costs are limited by specifying a fixed number of vectors to be generated. This method is useful for general nonsymmetric matrices.

BiConjugate Gradient (BiCG) method

The Biconjugate Gradient method generates two CG-like sequences of vectors, one based on a system with the original coefficient matrix A, and one on A^T. Instead of orthogonalizing each sequence, they are made mutually orthogonal, or "bi-orthogonal". This method, like CG, uses limited storage. It is useful when the matrix is nonsymmetric and nonsingular; however, convergence may be irregular, and there is a possibility that the method will break down. BiCG requires a multiplication with the coefficient matrix and with its transpose at each iteration.

Quasi-Minimal Residual (QMR) method

The Quasi-Minimal Residual method applies a least-squares solve and update to the BiCG residuals, thereby smoothing out the irregular convergence behavior of BiCG, which may lead to more reliable approximations. In full glory, it has a look ahead strategy built in that avoids the BiCG breakdown. Even without looking ahead, QMR largely avoids the breakdown that can occur in BiCG. On the other hand, it does not effect a true minimization of either the error or the residual, and while it converges smoothly, it often does not improve on the BiCG in terms of the number of iteration steps.

Conjugate Gradient Squared (CGS) method

The Conjugate Gradient Squared method is a variant of BiCG that applies the updating operations for the A-sequence and the A^T-sequence both to the same vectors. Ideally, this would double the convergence rate, but in practice convergence may be much more irregular than for BiCG, which may sometimes lead to unreliable results. A practical advantage is that the method does not need the multiplications with the transpose of the coefficient matrix.

Biconjugate Gradient Stabilized (Bi-CGSTAB) method

The Biconjugate Gradient Stabilized method is a variant of BiCG, like CGS, but using different updates for the A^T-sequence in order to obtain smoother convergence than CGS.

LSQR implementation of Conjugate Gradients on the Normal Equations

LSQR method for solving linear system is based on least square sense. The details of these methods are given in [Saa00].

8.2.2 Solvers for Numerical Methods

In this section, we shall present the solvers for iterative methods. We shall give details of solver, based on one iterative method. The details for other method iterative solvers are same.

Solver for MINRES method

- $x = minres(A, b)$ attempts to find a minimum norm residual solution x to the system of linear equations $Ax = b$. The $n \times n$ coefficient matrix A must be symmetric but need not be positive definite. It should be large and sparse. The column vector b must have length n. If $minres$ converges, a message to that effect is displayed. If $minres$ fails to converge after the maximum number of iterations or halts for any reason, a warning message is printed displaying the relative residual $\frac{\|b-Ax\|}{\|b\|}$ and the iteration number at which the method stopped or failed.

- $minres(A, b, tol)$ specifies the tolerance of the method. If tol is [ ], then $minres$ uses the default value, $1e-6$.
- $minres(A, b, tol, maxit)$ specifies the maximum number of iterations. If $maxit$ is [ ], then $minres$ uses the default, $\min(n, 20)$.
- $minres(A, b, tol, maxit, M)$ and $minres(A, b, tol, maxit, M_1, M_2)$ use symmetric positive definite preconditioner M or $M = M_1 * M_2$. If M is [ ] then $minres$ applies no preconditioner.
- $minres(A, b, tol, maxit, M_1, M_2, x_0)$ specifies the initial guess. If x_0 is [ ], then $minres$ uses the default, an all-zero vector.
- $[x, flag] = minres(A, b, ...)$ also returns a convergence flag. The meaning of flag obtained is given in Table 8.1 Whenever flag is not 0, the solution x returned is that with minimal

Flag	Convergence
0	$minres$ converged to the desired tolerance tol within $maxit$ iterations.
1	$minres$ iterated $maxit$ times but did not converge.
2	preconditioner M was ill-conditioned.
3	$minres$ stagnated. (Two consecutive iterates were the same.)
4	One of the scalar quantities calculated during $minres$ became too small or too large to continue computing.

Table 8.1: Meaning of flag

norm residual computed over all the iterations. No messages are displayed if the flag output is specified.

- $[x, flag, relres] = minres(A, b, ...)$ also returns the relative residual $\frac{\|b-Ax\|}{\|b\|}$. If flag is 0, $relres \quad \leq \quad tol$.
- $[x, flag, relres, iter] = minres(A, b, ...)$ also returns the iteration number at which x was computed, where $0 \leq \quad iter \quad \leq \quad maxit$.

8.2.3 Solvers for Another Method

$symmlq$ method

$x = symmlq(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x. The $n \times n$ coefficient matrix A must be symmetric but need not be positive definite. It should also be large and sparse.

$gmres$ method

$x = gmres(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x. The $n \times n$ coefficient matrix A must be square and should be large and sparse.

$bicgstab$ method

$x = bicgstab(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x. The $n \times n$ coefficient matrix A must be square and should be large and sparse.

qmr method

$x = qmr(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x. The $n \times n$ coefficient matrix A must be square and should be large and sparse.

cgs method

$x = cgs(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x. The $n \times n$ coefficient matrix A must be square and should be large and sparse.

pcg method

$x = pcg(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x. The $n \times n$ coefficient matrix A must be symmetric and positive definite, and should also be large and sparse.

lsqr method

$x = lsqr(A, b)$ attempts to solve the system of linear equations $Ax = b$ for x if A is consistent, otherwise it attempts to solve the least squares solution x that minimizes $\| b - A * x \|$. The $m \times n$ coefficient matrix A need not be square but it should be large and sparse.

8.3 Results and Discussion

We have consider the following test examples in $1D$ and $2D$ for our analysis. The examples are as follows:

- Dirichlet problem in $1D$
$$\frac{d^2y}{dx^2} + 4\frac{dy}{dx} + 4y = \exp x; \;\; y(0) = 1, \;\; y(1) = 0. \tag{8.2}$$
- Neumann problem in $1D$
$$\frac{d^2y}{dx^2} + 4\frac{dy}{dx} + 4y = x^2 + 1; \;\; \frac{dy}{dx}(0) = 0, \;\; \frac{dy}{dx}(1) = 0 \tag{8.3}$$
- Mixed problem in $1D$
$$\frac{d^2y}{dx^2} + 4\frac{dy}{dx} + 4y = \exp x; \;\; y(0) = 1, \frac{dy}{dx}(1) = 0 \tag{8.4}$$
- Dirichlet problem in $2D$
Let Ω be a domain.
$$\Omega = \{(x, y)| - 1 < x < 1; \;\; and \;\; - 1 < y < 1\}$$
$$u_{xx} + u_{yy} = 4; \;\; on \;\; \Omega \tag{8.5}$$
$$u = x^2 + y^2; \;\; on \quad \partial\Omega \tag{8.6}$$
- Neumann problem in $2D$
Let Ω be a domain.
$$\Omega = \{(x, y)|0 < x < 1, \;\; and \;\; 0 < y < 1\}$$
$$u_{xx} + u_{yy} = -\cos(\pi x), \;\; on \;\; \Omega \tag{8.7}$$
$$\frac{\partial u}{\partial \mathbf{n}} = 0, \;\; on \;\; \partial\Omega \tag{8.8}$$

- Helmholtz problem in $2D$
  Let Ω be a domain.

$$\begin{aligned}
\Omega &= \{(x,y) | -1 < x < 1, \ \ and \ \ -1 < y < 1\} \\
u_{xx} + u_{yy} + u &= 4 + x^2 + y^2, \ \ on \ \ \Omega \\
\frac{\partial u}{\partial \mathbf{n}} &= 2y, \ \ on \ \ y = -1, \\
\frac{\partial u}{\partial \mathbf{n}} &= -2y, \ \ on \ \ y = 1, \\
\frac{\partial u}{\partial \mathbf{n}} &= 2x, \ \ on \ \ x = -1, \\
\frac{\partial u}{\partial \mathbf{n}} &= -2x, \ \ on \ \ x = 1.
\end{aligned} \tag{8.9}$$

Using FPM technique explained in Chapter 7, the above examples of differential equations can be reduced in the linear system of the form (7.15). We have used different iterative methods like CGS, BICG, GMRES, QMR, LSQR to solve this system. These methods works for any square sparse coefficient matrix. We know that PCG, GMRES, SYMLQ works for any symmetric matrix. Even though, we have use these methods since we are not aware of the property of A. Our numerical results shows that these methods did not give convergence at all. It shows that the coefficient matrix which we have obtained is not symmetric matrix. In [Som04c], we have shown that Gaussian function is better for convergence analysis, so we shall consider the Gaussian function as a weight function. We have taken random numerical points in domain Ω. The points are 100 and 60 for one dimensions and two dimensions respectively. We have consider the support of weight function as 1. We have done the system preconditioned by the following preconditioners.

- Jacobi preconditioner
- Block Jacobi preconditioner of first kind (for one set S)
- Block Jacobi preconditioner of second kind (for second set S)
- Complete LU factorization of coefficient matrix A
- Incomplete LU factorization of level 0 with some drop tolerance $1e-6$
- Incomplete LU factorization of level 0 with some drop tolerance $1e-8$

The tolerance of the methods and maximum number of iterations which we consider are $1e-6$ and 1500 respectively.

8.3.1 Analysis without using Preconditioner for $1D$ and $2D$

Dirichlet problem

- Generalized Minimal Residual method takes only nine iteration to converge to the desired tolerance.
- Conjugate Gradient Squared(CGS) method, BiConjugate Gradient method, Quasi-Minimal Residual method and Least Square Residual method takes more than double iteration to converge to the desired tolerance in compare to Generalized Minimal Residual method.
- The maximum error between analytical solution and numerical solution is same for all cases.

Neumann problem

- Generalized Minimal Residual method does not converge to the desired tolerance within maximum number of iterations.
- BiConjugate Gradient method and Quasi-Minimal Residual method take slightly less iteration in compare to Conjugate Gradient Squared (CGS) method and Least Square Residual method.

Mixed problem

- Conjugate Gradient Squared (CGS) method and Generalized Minimal Residual method does not converge to the desired tolerance within maximum number of iterations.
- BiConjugate Gradient method and Quasi-Minimal Residual method have same number of iteration but Least Square Residual method have double iteration to converge to the desired tolerance.

8.3.2 Analysis with using Preconditioner for $1D$ and $2D$

Now, we shall present the convergence analysis of Dirichlet problem, Neumann problem, Mixed problem and Helmholtz problem

- All methods converge to desired tolerance.

- The preconditioner of the type Complete LU factorization in the preconditioned system take less iteration to converge to the desired tolerance.
- The preconditioner of the type ILU factorization of level 0 (with tolerance $1e-6$ of the factorization) take less iteration in compare to Jacobi preconditioner and block Jacobi preconditioner of two kinds. The hierarchy of the methods for the convergence to desired tolerance is GMRES, CGS, BICG and QMR.
- LSQR take more iteration to converge to desired tolerance.
- In compare to unpreconditioned system, preconditioned system takes less iteration to converge to desired tolerance.