Preconditioning for sparse matrices with applications

Auke van der Ploeg

Rijksuniversiteit Groningen

Preconditioning for sparse matrices with applications

Proefschrift

ter verkrijging van het doctoraat in de
Wiskunde en Natuurwetenschappen
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus Dr. S.K. Kuipers
in het openbaar te verdedigen op
vrijdag 4 februari 1994
des namiddags te 4.00 uur

door

Auke van der Ploeg

geboren op 23 oktober 1965
te Zwagerveen

Aan mijn ouders
Aan Marleen

Contents

0. General introduction

Mathematical models play an ever increasing role in the development of science and technology, as they offer the possibility to simulate, understand and optimize reality in computing laboratory. They can be used in a large variety of applications, e.g. fluid flows and air flows, mass and temperature distributions, pollution of air and water, electrical and gravitational interactions, to name only a few. Since physical experiments are often very difficult and costly, physical problems are often represented by partial differential equations which have to be solved by discrete numerical methods. The computation of an accurate approximation to the original continuous problem will usually lead to systems of linear equations in which the number of unknowns can be very large: currently, for many practical three-dimensional applications, the size of the linear systems is typically of the order of 1-100 million unknowns. Fortunately, the coefficient matrices arising in such computations are very sparse, i.e, only a very small part of the entries of A differs from zero. In this thesis, we consider the solution of x from the large sparse system of linear equations $Ax = b$ for a real non-singular $N \times N$ matrix A and a given vector b.

Elimination methods have as drawback that, in general, the number of non-zero entries strongly increases during the elimination process. Therefore, these methods are often too costly in terms of computer storage and CPU-time. As a consequence, sparse systems of linear equations are often solved by iterative methods which generate a sequence of approximations $x^{(n)}$ which hopefully converges rapidly towards the exact solution x. One of the more popular methods is the conjugate gradient method, developed in 1952 by Hestenes and Stiefel [18] for the case of symmetric, positive definite systems. In exact arithmetic, the method can be considered as a direct method, because the solution is obtained in at most N iteration steps. In the 1970s the method was considered as an iterative method [34], and generalised for the non-symmetric case, resulting in a large variety of methods. In general, the speed of convergence of these methods strongly depends on the eigenvalue distribution of the coefficient matrix. For example, for the conjugate gradient method it can be proven that it is very important that the spectral condition number is small, and that the extreme eigenvalues are well separated [47]. Therefore, a conjugate gradient-like method is often applied to the system

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

instead of to the original system $Ax = b$. The non-singular matrix C is called the preconditioner. It should be chosen in such a way that the preconditioned matrix $C^{-1}A$ is close to the identity matrix, and the system $Cy = d$ for given d can be solved much easier than the original system. In general, the application of the preconditioner will increase both the storage requirements and the computational labour in each iteration step. Nevertheless, it is often possible to solve the preconditioned system in far fewer iteration steps than the unpreconditioned system, so that the *total* amount of computational work is lowered. This thesis deals with the construction of preconditioners for systems of linear equations as they occur in a number of practical problems. Primarily, we consider preconditioning techniques based on an incomplete decomposition of A, because of their simple definition and high efficiency. In this technique, an LU-decomposition is constructed in which the sparsity of the factors L and U is preserved by ignoring some or all elements causing fill-in additional to that of A. Meijerink and van der Vorst [25] have shown existence and uniqueness of the incomplete decomposition for an important class of problems and for an arbitrary choice of the sparsity pattern of $L + U$. They showed that the CG method gives excellent results when combined with an incomplete Choleski-decomposition as preconditioner (ICCG).

Outline of the thesis.
This thesis consists of seven chapters, arranged in chronological order.

In the first chapter, some well-known iterative techniques for solving sparse systems of linear equations are described and compared with each other, including some conjugate gradient-like methods. Also described are preconditioning techniques based on incomplete decompositions that can optimally exploit a banded sparsity pattern of the coefficient matrix. In [25], these sort of preconditioners have been considered in more detail. The techniques are tested on a number of problems, including some examples with non-symmetry coming from dominating convective parts in the partial differential equation.

In many practical problems the coefficient matrix does not have a banded structure. Realistic physical problems often involve complicated geometries. In general, this leads to a very complicated non-zero structure of A. But even if the original matrix has a banded structure, it is sometimes advantageous to perform a renumbering of the unknowns, thus generating a coefficient matrix with an irregular structure (see, for example, Chapter 6). Therefore, in Chapter 2 a data structure is described for non-symmetric matrices with an arbitrary sparsity pattern, and an algorithm is described for the construction of an incomplete LU-decomposition based on a drop tolerance. In this technique, a splitting $A = LU + R$ is made, which has the property that all elements of R are in absolute value less than a parameter ε. When the system of linear equations stems from the discretization of a partial differential equation with varying coefficients, or when stretched grids are used, the matrix entries can vary strongly in absolute value. Therefore, we expect advantages of incomplete decompositions based on a drop tolerance. These techniques are demonstrated on the same test problems as used in Chapter 1. Parts of Chapter 2 have been presented in [42] and [43].

The third chapter gives an example of a situation in which these techniques can be applied. It describes a numerical model for the simulation of the temperature distribution in a block of concrete during the hardening process. The model leads to a non-stationary partial differential equation for the temperature distribution. In order to solve this PDE, a system of linear equations has to be solved several times, in which a small part of the matrix has a very irregular sparsity pattern due to some special boundary conditions. Some special-purpose preconditioning techniques are described in which this small part of the matrix can be treated very efficiently. These techniques are compared to the more general preconditioning technique of Chapter 2 which makes no restriction with respect to the sparsity pattern of A. Several methods for choosing the sparsity pattern of the factors L and U are compared with each other. Some results of Chapter 3 have also been presented in [42] and [43].

On supercomputers it is not only important that the preconditioner reduces the number of iteration steps. It is also very important that the application of the preconditioner is suitable for vectorization and parallelization. There has been a major research effort in developing techniques which can be implemented efficiently on supercomputers (see, for example, [12, 37] and the references quoted therein). When an incomplete factorization is used, a large number of lower- and upper-triangular systems has to be solved. Since algorithms for solving such systems are highly sequential by nature, they often form the bottleneck in implementations on vector and parallel computers. Therefore, in Chapter 4 we describe some possibilities for implementation of these algorithms on supercomputers. The resulting incomplete decompositions are compared with a number of other preconditioning techniques like polynomial preconditioning, which lend themselves more naturally for implementation on supercomputers. The systems of linear equations which arise in a numerical model for the Boussinesq equations are used as test problems. Some results of Chapter 4 have also been presented in [45].

In Chapter 5 we consider the linear systems arising in the computation of flows governed by the incompressible Navier-Stokes equations. The numerical solution of these equations for realistic problems is very demanding; it requires the best of the available computing power and numerical algorithms. The linear systems arising in these computations are typically of the form

$$\begin{bmatrix} I\frac{d}{dt} + M & G \\ D & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

in which the block G is approximately equal to $-D^T$. The submatrix M often contains a dominating skew-symmetric part, which severely complicates the numerical solution. Moreover, since the submatrix in the lower-right corner contains only zero elements, a straightforward sparse incomplete decomposition is not possible. Therefore, we consider the equivalent system $QAx = Qb$, in which Q can be regarded as a pre-preconditioner. Its function is to obtain a linear system for which one can easily construct a proper preconditioner. The construction of an incomplete LU-decomposition of QA can be done in a similar way as the construction of L and U based on a drop tolerance as described in Chapter 2. Some parts of Chapter 5 have also been presented in [46].

The rate of convergence of most iterative methods deteriorates when the mesh is refined. For many problems, one can prove that the multigrid method has the optimum order of complexity. However, due to the required proper smoothers and the restriction and prolongation operators at each level, the implementation of multigrid techniques for practical problems is much more complicated than that of most conjugate gradient-like methods. In Chapter 6, we describe an incomplete decomposition which is based on the same basic idea as in multigrid methods: many iterative methods can eliminate high-frequency errors very effectively, but they are inefficient at eliminating long-wavelength errors. A couple of iteration steps results in an approximation with a smooth error.

This error can therefore be well corrected on a coarser grid. Solving the equations on the coarse grid gives the two-grid method. Applying this idea recursively on coarser and coarser grids leads to the multigrid method. The new preconditioning technique described in Chapter 6 uses a partition of the unknowns based on a similar sequence of grids as in multigrid. Renumbering the unknowns according to this partition enables us to construct an incomplete LU-decomposition which can be used in eliminating effectively both high- and low-frequency errors. The factors L and U are obtained from an incomplete decomposition based on a drop tolerance as described in Chapter 2. Results are presented from the above method applied to a large number of test problems described in the literature, including the test problems of Chapter 1. For many problems, the new preconditioning technique shows grid-independent convergence when combined with a conjugate gradient-like method. In many cases, the preconditioning technique of Chapter 6 is superior to the techniques of Chapter 1 and 2. More research is required in order to study the possibilities for applying the ideas of this chapter to the systems of linear equations of Chapters 3-5. Chapter 6 is a joint work with E.F.F. Botta and F.W. Wubs. It is also available as a technical report [44].

In the last chapter, some conclusions are drawn, and some suggestions for future research are made.

1. Comparison of some iterative methods for large sparse systems of linear equations

1.1 Introduction

Solution of large sparse systems of linear equations continues to be a major research area with widespread application. The numerical solution of time-dependent or non-linear partial differential equations repeatedly requires the solution of linear systems of the form

$$Ax = b \tag{1.1}$$

where A is a large sparse $N \times N$ matrix and b a given vector. One way to solve (1.1) is by using a direct method, for example, by the construction of an LU-decomposition. In this approach an upper-triangular part U and a lower-triangular part L are constructed in such a way that $A = LU$. The system (1.1) can be solved in two steps using the factors L and U subsequently. One major drawback of this strategy is the fact that L and U are not sparse due to fill-in during the factorization process, so that computer storage demands are very high. Moreover, the computational work for constructing the factors L and U increases strongly with the dimension of the problem. As a result, systems like (1.1) are often solved with iterative methods. Iterative techniques have a number of advantages over direct solution: first of all, they often can exploit the sparsity of the coefficient matrix and secondly, the accuracy of the solution can be controlled more easily. This is important, especially when the linear system solver is used as an inner-iteration method, for example, within some Newton-like method for solving a system of non-linear equations. Solving the systems of linear equations too accurately at the beginning of the outer-iteration process would result in a waste of computer time. Moreover, when the linear system stems from a discretization of some partial differential equation, it is not practical to determine an approximate solution in which the error is less than the error coming from the discretization.

In Section 1.2, we will describe some well-known iterative techniques for solving linear systems. They may be classified into stationary and gradient methods. First we consider the stationary methods of Jacobi and Gauss-Seidel. These are the earliest iterative methods, and they are based on a splitting of the coefficient matrix. For many practical problems the Gauss-Seidel method can be accelerated by using relaxation techniques, leading, for example, to the well-known successive over-relaxation (SOR) method. We also consider the method of steepest descent and the conjugate gradient (CG) method. The convergence behaviour of the latter can strongly be improved by using preconditioning techniques. The preconditioned CG method is considered as one of the most efficient methods for solving linear systems in which the coefficient matrix is symmetric positive definite. Even when A is not symmetric, it is often possible to use CG-like methods to solve (1.1). In this chapter we will describe both Bi-CGSTAB [50] and GMRES(M) [38].

In Section 1.4 some preconditioning techniques based on incomplete decompositions are described for matrices with a diagonal structure. These matrices arise, for example, in the discretization of a steady convection-diffusion equation on a rectangular domain. In some cases the quality of the preconditioner can be improved by adding fill-in to the diagonal as suggested by Gustafsson [17].

Section 1.5 presents numerical results for symmetric and non-symmetric test problems, including two problems in which the coefficient matrix stems from the discretization of a steady convection-diffusion equation with dominating convective parts. In one of these problems upwind differencing is used for the convective parts, and in the other one these terms are discretized using central differences.

1.2 Some stationary iterative methods

First we recall some commonly used definitions.

Definition 1 *A real $N \times N$ matrix A is said to be diagonally dominant if*

$$a_{ii} \geq \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}| \qquad \textit{for all } i = 1, \ldots, N$$

The matrix is said to be strictly diagonally dominant, when '$\geq$' is replaced by '$>$'.

Definition 2 *A real matrix A is an M-matrix, if it is non-singular, its entries $a_{ij} \leq 0$ for $i \neq j$, and all entries of A^{-1} are not negative.*

In setting up linear equations for the numerical solution of partial differential equations, it is frequently known from physical properties or mathematical reasoning that the coefficient matrix is an M-matrix.

Definition 3 *Suppose that A is a real matrix. The pair of matrices (Q, H) is called a splitting of A if Q is non-singular, and*

$$A = Q - H$$

The splitting is called regular if all entries of both Q^{-1} and H are not negative.

In this section we consider the linear system (1.1) for non-singular A. First we consider stationary iterative methods based on the splitting (Q, H) defined by

$$Qx^{(k+1)} = Hx^{(k)} + b \tag{1.2}$$

In general, the convergence will be fast when H is small in some sense. For example, if $H = 0$ we have that $Q = A$, and we obtain $x^{(1)} = A^{-1}b$. Hence only one step of the iteration is required. Obviously, this choice is impractical, because it requires the inverse of the coefficient matrix. The splitting $A = Q - H$ should be chosen in such a way that Q resembles A as closely as possible, but for a given vector d we must be able to solve $Qy = d$ efficiently. Suppose that the iteration error $v^{(k)}$ is defined by the difference $x - x^{(k)}$. By rewriting (1.1) as $Qx = Hx + b$ and subtracting (1.2) from this equation, we obtain

$$Qv^{(k+1)} = Hv^{(k)}$$

and with induction one can show that

$$v^{(k)} = (Q^{-1}H)^k v^{(0)}$$

The matrix $Q^{-1}H$ is called the iteration matrix. One can prove that the iterative method defined by (1.2) converges if and only if the spectral radius $\rho(Q^{-1}H)$ is less than 1 (see, for example, [52]). In [52] it is also proved that when (Q, H) is a regular splitting of an M-matrix A, the iterative method (1.2) converges.

Some choices of the splitting (Q, H).
Most of the methods described in this section will only converge if the coefficient matrix is diagonally dominant. In such cases it is convenient to assume that the diagonal of the matrix has been scaled to unity. We assume that the coefficient matrix can be expressed as

$$A = I - L_A - U_A \tag{1.3}$$

where I is the identity matrix,

$$L_A = - \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ a_{N,1} & \dots & & a_{N,N-1} & 0 \end{bmatrix} \tag{1.4}$$

and

$$U_A = - \begin{bmatrix} 0 & a_{12} & a_{13} & \dots & & a_{1,N} \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & & a_{N-2,N-1} & a_{N-2,N} \\ 0 & \dots & 0 & & 0 & a_{N-1,N} \\ 0 & \dots & 0 & & 0 & 0 \end{bmatrix} \tag{1.5}$$

The *Jacobi* iteration is described by (1.2) with Q equal to the identity matrix, hence

$$x^{(k+1)} = (L_A + U_A)x^{(k)} + b$$

The Jacobi iteration matrix M_J is equal to $L_A + U_A$, and, as follows from (1.3), this is equal to $I - A$. This implies that each eigenvalue μ_i of M_J is related to an eigenvalue γ_i of A by $\mu_i = 1 - \gamma_i$.

If the variables are not corrected simultaneously but in the sequence $i = 1, 2, \dots, N$, then $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$ are already available when $x_i^{(k+1)}$ is determined. In the *Gauss-Seidel* iteration these revised values are used instead of the original values, giving the faster iterative method

$$(I - L_A)x^{(k+1)} = U_A x^{(k)} + b$$

At each step of this iteration process a lower-triangular system has to be solved.

For large systems of equations the convergence of the Gauss-Seidel iteration may still be slow. In many cases the convergence behaviour is regular, and it is possible to improve the rate of convergence by an acceleration technique. In the *successive over-relaxation method* (SOR) an over-relaxation is carried out regularly at every iteration step, which leads to the iteration formula

$$(I - \omega L_A)x^{(k+1)} = [(1-\omega)I + \omega U_A]x^{(k)} + \omega b$$

where ω is the relaxation parameter. In the special case when $\omega = 1$, SOR reduces to the Gauss-Seidel iteration.
The iteration matrix M_{SOR} is equal to

$$(I - \omega L_A)^{-1}[(1-\omega)I + \omega U_A]$$

It is not possible, in general, to derive a simple relationship between the eigenvalues of this matrix and those of A. However, Young [59] has discovered a class of matrices, having the so-called property A, for which the convergence rate of SOR can be directly related to that of the Jacobi iteration: suppose that $\mu_i \neq 0$ is an eigenvalue of the Jacobi matrix, and λ_i is an eigenvalue of M_{SOR}. If $\omega \neq 0$, these eigenvalues will be related through

$$(\lambda_i + \omega - 1)^2 = \lambda_i \mu_i^2 \omega^2 \tag{1.6}$$

From (1.6) it follows that the convergence rate of the Gauss-Seidel iteration ($\omega = 1$) is twice that of the Jacobi iteration. Furthermore, from (1.6) it can be deduced that when all eigenvalues of the Jacobi matrix are real, and when μ_{max} is the largest eigenvalue of the Jacobi matrix, the optimal over-relaxation parameter can be expressed as

$$\omega_{opt} = \frac{2}{1 + [1 - \mu_{max}^2]^{\frac{1}{2}}}$$

leading to a spectral radius of M_{SOR} equal to $\omega_{opt} - 1$.

Suppose that A stems from a standard discretization of the Poisson equation $-\Delta u = f$ on the square $[0,1] \times [0,1]$ with Dirichlet boundary conditions everywhere. Furthermore suppose that a uniform grid is used with mesh size $h = 1/M$ in both directions (in the sequel of this chapter this problem will be referred to as the Poisson-model problem). One can prove that the eigenvalues of the Jacobi iteration matrix are given by

$$\mu_{pq} = \frac{\cos(p\pi/M) + \cos(q\pi/M)}{2}, \qquad p = 1, \ldots, M-1; \qquad q = 1, \ldots, M-1$$

The largest eigenvalue is obtained for $p = q = 1$, so that

$$\mu_{max} = \cos(\pi/M) \tag{1.7}$$

By using a Taylor expansion of the right-hand side of this equation we obtain the following approximations to the spectral radii.

$$\begin{array}{ll} 1 - \frac{1}{2}\,\pi^2/M^2, & \text{for Jacobi} \\ 1 - \pi^2/M^2, & \text{for Gauss-Seidel} \\ 1 - 2\pi/M, & \text{for SOR} \end{array}$$

Using these equations one can easily deduce that with Jacobi and Gauss-Seidel, the work necessary to obtain the solution of $Ax = b$ with a specified accuracy is $O(N^2)$. With SOR the amount of work is $O(N^{3/2})$, which is favourable. However, for realistic problems it may be difficult to obtain the optimal relaxation parameter, and, in many cases, another choice for ω than ω_{opt} can significantly deteriorate the convergence rate of SOR.

There are many other stationary methods of the form (1.2), for example, SLOR and SSOR. These methods are not described here. Instead, we will now consider some gradient methods.

1.3 Some conjugate gradient-like methods

Suppose that the coefficient matrix of the equations is symmetric and positive definite. This implies that the operator $(\cdot,\cdot)_A : R^N \times R^N \longrightarrow R$ defined by

$$(x, y)_A = (x, Ay)$$

is an inner product, and $\|\cdot\|_A : R^N \longrightarrow R$ defined by

$$\|y\|_A = (y, Ay)^{\frac{1}{2}}$$

defines a norm. Finding the solution of $Ax = b$ is equivalent to finding y for which $\|x-y\|_A^2$ is zero, and because

$$\|x - y\|_A^2 = (x - y, Ax - Ay) = (x, Ax) + (y, Ay) - 2(y, Ax)$$

this is equivalent to minimalization of the quadratic form

$$J[y] = \tfrac{1}{2}\ (y, Ay) - (y, b)$$

Suppose that $x^{(k)}$ is an approximation of the solution at stage k of the iteration process. The process consists now of choosing a search direction $z^{(k)}$ and minimizing $J[y]$ in this direction. This means that $x^{(k+1)}$ is chosen equal to $x^{(k)} + \alpha_k z^{(k)}$, where α_k is chosen in such a way that

$$\frac{d}{d\alpha_k} J[x^{(k)} + \alpha_k z^{(k)}] = 0$$

From this equation we obtain that α_k can be expressed as

$$\alpha_k = \frac{(z^{(k)}, r^{(k)})}{(z^{(k)}, Az^{(k)})}$$

in which the residual vector $r^{(k)}$ is defined by

$$r^{(k)} = b - Ax^{(k)}$$

Both the methods of steepest descent and conjugate gradients follow this strategy. The methods differ only in the choice of the search direction $z^{(k)}$. In the steepest descent procedure this direction is chosen to be the direction of the maximum gradient of $J[y]$ at the point $x^{(k)}$. One can show that this direction is equal to that of $r^{(k)}$. Hence the method of steepest descent can be represented by

$$x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)}, \qquad \text{with } \alpha_k = \frac{(r^{(k)}, r^{(k)})}{(r^{(k)}, Ar^{(k)})}$$

This iteration scheme is of the form (1.2) with $Q = (1/\alpha_k)I$.

In the conjugate gradient method, which was developed in 1952 by Hestenes and Stiefel [18], the search directions are chosen in such a way that at each iteration step the dimension of the problem is reduced by 1. This can be accomplished by choosing $z_0, z_1, \dots$ orthogonal with respect to A and hence satisfying the condition

$$(z^{(k)}, Az^{(j)}) = 0, \qquad j = 0, 1, \dots, k-1 \tag{1.8}$$

The conjugate gradient algorithm is described by Algorithm 1.1.

Algorithm 1.1. The conjugate gradient algorithm.

```
$r^{(0)} := b - Ax^{(0)}$;
$z^{(0)} := r^{(0)}$;
$\rho_0 := (r^{(0)}, r^{(0)})$;
FOR $k := 0, 1, 2, \dots$ DO
  BEGIN
    $\alpha_k := \rho_k/(z^{(k)}, Az^{(k)})$;
    $x^{(k+1)} := x^{(k)} + \alpha_k z^{(k)}$;
    $r^{(k+1)} := r^{(k)} - \alpha_k Az^{(k)}$;
    $\rho_{k+1} := (r^{(k+1)}, r^{(k+1)})$;
    $\beta_k := \rho_{k+1}/\rho_k$;
    $z^{(k+1)} := r^{(k+1)} + \beta_k z^{(k)}$
  END;
```

It can be shown by induction that the sequences of vectors $r^{(k)}$ and $z^{(k)}$ generated by this algorithm satisfy (1.8), and, in addition, the following relationships are satisfied:

$$(r^{(k)}, z^{(j)}) = (r^{(k)}, r^{(j)}) = 0, \qquad j = 0, 1, \dots, k-1,$$

$$(r^{(k)}, z^{(k)}) = (r^{(k)}, r^{(k)}) \qquad \text{and} \qquad (r^{(k)}, Az^{(k)}) = (z^{(k)}, Az^{(k)})$$

When rounding errors do not affect the computation, the orthogonality relations imply that the correct solution is obtained in at most N steps. Therefore, the conjugate gradient method was first considered as a *direct* method for solving systems of linear equations. However, when N iteration steps are required, the method is very costly compared with other methods. In many situations, far more than N steps are required due to rounding errors. Therefore, it is better to consider the CG method as an *iterative* method for solving sparse systems of linear equations.

Convergence of the CG method.
From the orthogonal relationships it follows that after k steps of the CG method the approximate solution $x^{(k)}$ is constructed in such a way that the $A-$norm of the error is minimized over all u with $u - x^{(0)}$ in the Krylov subspace

$$K_k(A; r^{(0)}) = \text{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{k-1}r^{(0)}\}$$

Hence

$$\|x^{(k)} - x\|_A = \min_{u \in x^{(0)} + K_k(A; r^{(0)})} \|u - x\|_A \tag{1.9}$$

The right-hand side of this equation can be written as

$$\min_{P_k \in \Pi_{k-1}} \|P_k(A)(x - x^{(0)})\|_A \tag{1.10}$$

where Π_{k-1} is the set of all polynomials P_k of degree $\leq k-1$ such that $P_k(0) = I$. Suppose that $x - x^{(0)}$ can be expressed in the eigenvector component form

$$x - x^{(0)} = \sum_{j=1}^{N} \sigma_j v_j$$

where v_j are the eigenvectors of A with corresponding eigenvalues λ_j. Then $P_k(A)(x - x^{(0)})$ can be expressed as

$$\sum_{j=1}^{N} \sigma_j P_k(A) v_j = \sum_{j=1}^{N} \sigma_j P_k(\lambda_j) v_j \tag{1.11}$$

In order to minimize the $A-$norm of this vector, the value $|P_k(\lambda_j)|$ should be as small as possible for all eigenvalues λ_j of A. The polynomial for which $P_k(\lambda) = 0$ for all λ is not possible, because we have as condition that $P_k(0) = 1$. Since the CG method produces the optimal polynomial, the convergence rate of the CG method can be irregular, and is strongly influenced by the location of the eigenvalues. For instance, if the eigenvalues are clustered into 10 groups, the 10th-order polynomial will have zeroes within or very close to these groups, giving a large error reduction at the 11th iteration step. It is possible to use Chebychev polynomials to give bounds for the convergence rate of the CG method based on the fact that they cannot produce better reductions of the error than the optimal polynomials generated by the CG method. In this way one can prove the well-known upper bound on the residual vector [3]:

$$\|x - x^{(k)}\|_A \leq \left(\frac{(\text{cond}(A))^{\frac{1}{2}} - 1}{(\text{cond}(A))^{\frac{1}{2}} + 1} \right)^k \|x - x^{(0)}\|_A \tag{1.12}$$

Herein cond(A) is the spectral condition number of the coefficient matrix. In practice, the rate of convergence can be considerably faster: when the extreme eigenvalues are relatively well separated from the remainder of the spectrum, the rate of convergence improves during the iteration process. This phenomenon is called superconvergence [47].

If the CG method is used to solve the linear system coming from the Poisson-model problem mentioned earlier, the amount of work to solve the linear system is approximately the same as with SOR, and the only advantage of CG over SOR is that no relaxation parameters have to be chosen. However, it appears to be possible to improve the convergence rate of CG drastically by using a preconditioner (see Section 1.4 for more details).

Some iterative methods for non-symmetric systems.
When the matrix is not symmetric, one possible iterative method is obtained by applying the CG method to the system

$$A^TAx = A^Tb$$

This approach does not lead to an efficient iterative method. First of all, the amount of computational work per iteration step almost doubles, because two matrix-vector products are required. Secondly, the spectral condition number of A^TA may be much larger than that of A. If the coefficient matrix is symmetric, cond(A^TA) is the square of cond(A), and about twice as many iteration steps are required as when the CG method is applied to $Ax = b$. When A is not symmetric, this may be even worse. For example, the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$$

has both eigenvalues equal to one, while

$$A^TA = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

has eigenvalues $3 \pm 2\sqrt{2}$, and its spectral condition number is approximately 34.

In the Bi-Conjugate Gradients (Bi-CG) method, proposed by Fletcher [15], two sequences of residual vectors $r^{(k)}$ and $\tilde{r}^{(k)}$ are generated by simple recurrence relations similar to those in the CG method. In many situations this approach is successful, although there is no solid theoretical basis for the convergence behaviour of the Bi-CG method.

At the $k-$th step of the Bi-CG algorithm, a polynomial P_k is generated in such a way that $r^{(k)} = P_k(A)r^{(0)}$ and $\tilde{r}^{(k)} = P_k(A^T)\tilde{r}^{(0)}$, so that

$$(\tilde{r}^{(k)}, r^{(k)}) = (P_k(A^T)\tilde{r}^{(0)}, P_k(A)r^{(0)}) = (\tilde{r}^{(0)}, P_k^2(A)r^{(0)})$$

Apparently, the operator $P_k(A)$ transforms the vector $r^{(0)}$ into a small vector $r^{(k)}$, so that it might be a good idea to apply $P_k(A)$ once more. This is the basis for the so-called Conjugate Gradient Squared (CGS) method [39]. For many problems, CGS converges about twice as fast as Bi-CG. An additional advantage of CGS over Bi-CG is that it requires no matrix-vector multiplication with A^T. However, the convergence behaviour can be quite irregular, in particular when the starting vector is close to the solution. Since a linear system solver is often used as an inner-iteration process within an outer-iteration process, this is no rare occasion. A very unpleasant side effect of this phenomenon is that the accuracy of the solution delivered by CGS can be degraded by the effect of rounding errors. An improved version of CGS called Bi-CGSTAB has recently been introduced by

Algorithm 1.2. The Bi-CGSTAB algorithm.

```
$r^{(0)} := b - Ax^{(0)}$;
$p$ is chosen in such a way that $(p, r^{(0)}) \neq 0$, e.g., $p := r^{(0)}$;
$\rho_0 := 1$; $\alpha := 1$; $\omega_0 := 1$;
$v^{(0)} := 0$; $q^{(0)} := 0$;
FOR $k := 1, 2, 3, \ldots$ DO
  BEGIN
    $\rho_k := (p, r^{(k-1)})$;
    $\beta := (\rho_k \alpha)/(\rho_{k-1}\omega_{k-1})$;
    $q^{(k)} := r^{(k-1)} + \beta(q^{(k-1)} - \omega_{k-1}v^{(k-1)})$;
    $v^{(k)} := Aq^{(k)}$;
    $\alpha := \rho_k/(p, v^{(k)})$;
    $s := r^{(k-1)} - \alpha v^{(k)}$;
    $t := As$;
    $\omega_k := (t, s)/(t, t)$;
    $x^{(k)} := x^{(k-1)} + \alpha q^{(k)} + \omega_k s$;
    IF $x^{(k)}$ is accurate enough THEN quit;
    $r^{(k)} := s - \omega_k t$
  END;
```

van der Vorst [50]. Bi-CGSTAB has a more regular convergence behaviour than CGS. It can be implemented as shown in Algorithm 1.2.

In many iterative methods for solving a linear system $Ax = b$, the idea is to solve the system projected onto the Krylov subspace $K_k(A; r^{(0)})$. When A is symmetric and positive definite, it is possible to form the projected system by a simple three-term recursion. This is fundamental for the efficiency of the conjugate gradient method. When the coefficient matrix is not symmetric, such a simple recursion is not possible. The GMRES method [38] solves the system projected onto the Krylov subspace. At the $k-$th iteration step of GMRES, $x^{(k)}$ is the vector that minimizes $\|b - Ay\|_2$ over all $y \in x^{(0)} + K_k(A; r^{(0)})$. This has the advantage that the 2-norm of the residual vector does not increase when the iteration process proceeds, and with exact arithmetic the method terminates within N steps. In Chapter 5 we will see that GMRES can be used successfully for solving the systems of linear equations that arise in the computation of flows governed by the incompressible Navier-Stokes equations.

At every step of GMRES we have to form an orthonormal basis $\{v^{(1)}, v^{(2)}, \ldots, v^{(k)}\}$ for $K_k(A; r^{(0)})$, which can be done by using Arnoldi's method. If we write $V^{(k)}$ for the matrix with columns $\{v^{(1)}, v^{(2)}, \ldots, v^{(k-1)}\}$, this leads to

$$AV^{(k)} = V^{(k+1)}H^{(k)}$$

where $H^{(k)}$ is an $(k+1) \times k$ upper-Hessenberg matrix. A disadvantage of GMRES is that all the vectors $v^{(k)}$ have to be stored in memory, and the construction of an orthonormal basis for the Krylov subspace becomes more complex with increasing k. Therefore, GMRES is restarted after M steps, so that one obtains an *iterative* method GMRES(M) instead of a *direct* method. GMRES(M) can be described by the scheme presented in Algorithm 1.3. In this scheme solve(x, k) represents the following computations:

compute y as the solution of $Hy = \tilde{g}$, where the upper-triangular part of H has $h_{i,j}$ as its elements, and $\tilde{g}$ represents the first k components of g;
$x := x^{(0)} + \sum_{i=1}^{k} y[i]v^{(i)}$;

Algorithm 1.3. The GMRES(M) algorithm.

```
REPEAT
    $r^{(0)} := b - Ax^{(0)}$;
    $v^{(1)} := r^{(0)}/\|r^{(0)}\|_2$;
    $g := \|r^{(0)}\|_2(1, 0, 0, \ldots)^T$;
    FOR $k := 1$ TO $M$ DO
        BEGIN
            $h_{i,k} := (Av^{(k)}, v^{(i)}), \qquad i = 1, 2, \ldots, k$;
            $\tilde{v}^{(k+1)} := Av^{(k)} - \sum_{i=1}^{k} h_{i,k}v^{(i)}$;
            $h_{k+1,k} := \|\tilde{v}^{(k+1)}\|_2$;
            $v^{(k+1)} := \tilde{v}^{(k+1)}/h_{k+1,k}$;
            Apply $J_1, J_2, \ldots, J_{k-1}$ to $(h_{1,k}, h_{2,k} \ldots h_{k+1,k})$;
            Construct $J_k$;
            Apply $J_k$ to $(h_{1,k}, h_{2,k} \ldots h_{k+1,k})$ and $g$;
            IF $g[k+1]$ is small enough THEN
                BEGIN
                    solve($x, k$);
                    quit
                END
        END; {$k$-loop}
    solve($x^{(0)}, M$)
END;
```

The Givens rotation J_k acting on the $k-$th and $(k+1)-$th component of a vector w is defined as follows:

$$(J_k w)[k] = c_{k+1}w[k] - s_{k+1}w[k+1], \qquad \text{and} \qquad (J_k w)[k+1] = s_{k+1}w[k] + c_{k+1}w[k+1]$$

where c_{k+1} and s_{k+1} are defined by

$$c_{k+1} = h_{k,k}/(h_{k,k}^2 + h_{k+1,k}^2)^{\frac{1}{2}}\ , \qquad \text{and} \qquad s_{k+1} = -h_{k+1,k}/(h_{k,k}^2 + h_{k+1,k}^2)^{\frac{1}{2}}$$

It can be proved that the $(k+1)-$th component of g is equal to the 2-norm of the residual vector $b - Ax$, where x is obtained from solve(x, k). In the scheme above the orthogonalization is performed using the modified Gram-Schmidt algorithm. It is important that all computations are performed in double precision, because there may be a loss of orthogonality among the $v^{(k)}$'s due to the presence of round-off errors. Therefore, Walker [54] proposed to use Householder transformations for computing an orthonormal basis of the Krylov subspace.

There exists a whole family of methods to solve linear systems in which the coefficient matrix is not symmetric. For a description of a large number of these methods see, for example, [8].

1.4 Preconditioning

From (1.12) it follows that the CG method performs well if the spectral condition number is small. This suggests that we precondition the original system of linear equations in such a way that the spectral condition number is decreased as much as possible. This means that we apply a CG-like method to the preconditioned system

$$C^{-1}Ax = C^{-1}b \tag{1.13}$$

The non-singular matrix C is called the preconditioner. There is a wide choice of preconditioners, see, for example, [8, 37]. The matrix C should have the following properties.

1. It should be a proper approximation of A, so that $C^{-1}A$ resembles the identity matrix. The most important quality of a preconditioner is to reduce the spectral condition number of the preconditioned matrix $C^{-1}A$.
2. The preconditioner should be cheap to compute, and it should be possible to solve the system $Cy = d$ for given d in $\mathrm{O}(N)$ operations.
3. It should not require a large amount of storage.

Matrices resulting from a discretization of partial differential equations usually are very sparse. For matrices which are diagonally dominant, a very effective way of obtaining a proper preconditioner is to proceed with an LU-decomposition, but to preserve the sparsity in the factors L and U by ignoring some or all elements causing fill-in additional to that of A. The matrix C is then chosen equal to LU. In this way one obtains the splitting $(LU, -R)$, where $R = A - LU$ is the so-called residual matrix. A CG-like method can be applied to the preconditioned system

$$U^{-1}L^{-1}Ax = U^{-1}L^{-1}b$$

or to the system

$$L^{-1}AU^{-1}\tilde{x} = L^{-1}b, \qquad \tilde{x} = Ux$$

We can describe the sparsity pattern of the matrix $L + U$ using the set P defined as

$$P = \{(i,k)|\text{specification of the positions } (i,k) \text{ where fill-in is allowed.}\} \tag{1.14}$$

Meijerink and van der Vorst [25] have proved that when A is an M-matrix, a unique incomplete decomposition exists for every choice of P that contains the main diagonal, and the incomplete decomposition process is more stable than the complete decomposition without pivoting. The splitting $(LU, -R)$ is regular in that case.

In this chapter, we will focus on iterative methods for matrices with a diagonal structure as shown in Fig. 1.1. These matrices arise, for example, from the discretization of the

Figure 1.1: Sparsity pattern for a five-point finite-difference operator over a rectangular grid.

steady convection-diffusion equation

$$-\Delta u(x,y) + d(x,y)\frac{\partial}{\partial x}u(x,y) + e(x,y)\frac{\partial}{\partial y}u(x,y) = f(x,y) \tag{1.15}$$

on a rectangular domain. If we perform an incomplete LU-decomposition in which all elements of L and U causing fill-in additional to that of A are neglected, we obtain the same sparsity pattern for $L+U$ as for the coefficient matrix. In many cases the number of iteration steps can be reduced by allowing fill-in not only in the bands shown in this figure, but also in a number of extra bands [25, 26]. However, taking the same sparsity pattern for $L+U$ as for A has a number of advantages. First of all, this choice gives better possibilities for vectorization and parallelization and secondly, it enables us to implement the matrix-vector multiplication efficiently using the so-called Eisenstat implementation [14].

Implementation on supercomputers.
On supercomputers it is not only important that the preconditioner reduces the number of iteration steps necessary to fulfil a certain stopping criterion. It is also very important that the application of the preconditioner is suitable for vectorization and parallelization. Algorithms to solve the lower- and upper-triangular systems are highly sequential by nature. However, when A represents a five-point finite-difference operator over a rectangular grid, and $L+U$ has the same sparsity pattern as A, it is possible to improve the performance on supercomputers by changing the order of computation. Suppose that x is solved from $Lx = y$. The components of x only depend on the previously computed neighbours in the west and south direction. Hence the unknowns corresponding to the grid points along a diagonal of the grid depend only on the unknowns corresponding to the previous diagonal. Therefore, if the unknowns are computed in the order corresponding to these diagonals, then for each diagonal all components of x can be computed independently. This technique is a special case of the more general technique known as level-scheduling (see, for example, [12] and Section 4.2).

There are many other ways to obtain efficient implementations of preconditioned CG-like methods on supercomputers, especially when the linear system comes from the discretization of (1.15), and $L+U$ has the same sparsity pattern as the coefficient matrix.

See, for example, [12, 37].

Eisenstat implementation.
It can be shown that if both A and $L+U$ have a sparsity pattern as shown in Fig. 1.1, only corrections on the main diagonal are performed during the construction of L and U. This implies that computer storage demands are low, because the preconditioner only requires one extra diagonal. Another advantage is that we are able to implement the matrix-vector multiplication efficiently as suggested by Eisenstat [14]. Suppose that the preconditioner LU can be written as

$$(D - L_A)(D - U_A)$$

where L_A and U_A are defined by (1.4) and (1.5). It is no restriction to assume that $D = I$, because we can scale the original system with D in such a way that LU takes the form

$$(I - L_A)(I - U_A)$$

Note that this scaling does not necessarily imply that the diagonal elements of A are all equal to 1. Suppose that after the scaling the diagonal of A is D_A. Now the preconditioned matrix $L^{-1}AU^{-1}$ can be represented by

$$\begin{aligned}
&(I - L_A)^{-1}(D_A - L_A - U_A)(I - U_A)^{-1} = \\
&(I - L_A)^{-1}(I - L_A + D_A - 2I + I - U_A)(I - U_A)^{-1} = \\
&(I - U_A)^{-1} + (I - L_A)^{-1}(D_A - 2I)(I - U_A)^{-1} + (I - L_A)^{-1}
\end{aligned}$$

which implies that $x = L^{-1}AU^{-1}y$ for given y can be computed as follows:

$$\begin{aligned}
&v := (I - U_A)^{-1}y; \\
&x := (I - L_A)^{-1}[y + (D_A - 2I)v] + v;
\end{aligned}$$

This computation requires only solving two triangular systems plus the multiplication of the diagonal matrix $(D_A - 2I)$ with v plus two vector additions. Hence the multiplication with the preconditioned matrix costs about the same as with the original coefficient matrix.

Gustafsson's modification.
In the so-called modified incomplete decomposition, which has been suggested by Gustafsson [17], a splitting $(LU, -R)$ is made in which fill-in outside the sparsity pattern of L and U is added on the main diagonal. As a result R has row sums zero:

$$\sum_{j=1}^{N} r_{ij} = 0, \text{ for } 1 \leq i \leq N \tag{1.16}$$

Suppose that A is a symmetric M-matrix, and L results from a modified incomplete Choleski-decomposition of A as described above. It can be shown that R has non-positive elements outside the main diagonal. Hence from Gerschgorin's theorem it follows that R is positive semidefinite. From $L^{-1}AL^{-T} = I + L^{-1}RL^{-T}$ it then follows that all eigenvalues

λ of the preconditioned matrix satisfy $\lambda \geq 1$. Thus the spectral condition number of the preconditioned matrix is bounded by its spectral radius.

When the coefficient matrix is symmetric, this preconditioning technique is referred to as MIC (Modified Incomplete Choleski), and when this technique is combined with the CG method, this is indicated with MICCG. Gustafsson [17] has shown that, in many cases, the modification of the diagonal leads to a reduction of the condition number of the preconditioned matrix from $\mathrm{O}(h^{-2})$ to $\mathrm{O}(h^{-1})$, where h is the characteristic mesh size. Surprisingly, in many cases this does not lead to a reduction in the number of iteration steps. In [49] an example is given for which the convergence behaviour of MICCG is worse than that of ICCG. This can be explained by the fact that when (1.16) is satisfied exactly, the iteration process may suffer from an unpredictable behaviour of the largest eigenvalue(s) of the preconditioned matrix (see, for example, [30]). Therefore, much attention has been given to strategies in which perturbations are introduced. In [5] a relaxed form of a modified incomplete decomposition is described in which the corrections are multiplied with a factor α which varies from 0.0 to 1.0. Taking α equal to 0.0 leads to the unmodified incomplete Choleski-decomposition, and taking α equal to 1.0 to the modified incomplete decomposition. In practice, it is difficult to choose the parameter α optimal, because the number of CG iteration steps depends very critically on α. In [49] another technique is used in combination with MICCG. Herein a modified incomplete decomposition of the matrix $A + \zeta h^2 D_A$ is used as a preconditioner. We have used these perturbations of the main diagonal only when we make a *modified* incomplete decomposition of the coefficient matrix.

1.5 Numerical experiments

In order to demonstrate the techniques described in the previous section, we show some results for three different problems. All numerical experiments have been carried out in double precision on an HP-720 workstation, and with the iterative method applied to the preconditioned system $L^{-1}AU^{-1}\tilde{x} = L^{-1}b$, where $\tilde{x} = Ux$. In all cases, the initial solution was equal to the null-vector, and the main diagonal of both L and U was scaled to unity, thus saving a number of floating point operations when solving the lower- and upper-triangular systems. The efficient Eisenstat implementation was used whenever it was possible.

Example 1. The first example shows the results of the Poisson-model problem described earlier with Dirichlet boundary conditions everywhere. The Poisson equation was discretized over a rectangular $(M + 2) \times (M + 2)$-grid with constant mesh size $1/(M + 1)$. The number of unknowns is M^2. The right-hand side was chosen in such a way that the exact solution vector is equal to $(1, 2, 3, 4, \ldots, N)^T$. We used SOR, ICCG and MICCG to solve the resulting system of linear equations. During the incomplete Choleski-decomposition all entries generating fill-in additional to that of A were neglected. The value of the optimum relaxation parameter for SOR can be calculated from (1.6) and (1.7), and can be expressed as

$$\omega_{opt} = \frac{2}{1+\sin\left(\pi/(M+1)\right)}$$

The stopping criterion for SOR was

$$\|x^{(n+1)} - x^{(n)}\|_2 < 10^{-10}\|b - Ax^{(0)})\|_2$$

and the CG iteration process was stopped when

$$\|L^{-1}(b - AL^{-T}\tilde{x}^{(n)})\|_2 < 10^{-10}\|L^{-1}(b - AL^{-T}\tilde{x}^{(0)})\|_2$$

where $\tilde{x}^{(n)} = L^T x^{(n)}$. In order to improve the efficiency of MICCG, we applied small perturbations of the main diagonal. Before the modified incomplete Choleski-decomposition was made, all diagonal elements were multiplied with a factor $1 + \zeta h^2$, in which $\zeta = 10$ appeared to be a proper choice. The results are summarised in Fig. 1.2, which shows the number of flops per unknown required to fulfil the stopping criterion against M. Fig. 1.2 clearly demonstrates the O($N^{3/2}$)-complexity of SOR: the amount of work per unknown is a linear function of $M = N^{1/2}$. With ICCG the computational complexity is also O($N^{3/2}$), although ICCG performs much better that SOR. Modification of the diagonal as suggested by Gustafsson results in a very efficient preconditioner. For this problem, it is possible to prove that the computational complexity of MICCG is O($N^{5/4}$) [17].

As mentioned in Section 1.4, it is possible to implement (M)ICCG efficiently on supercomputers by computing the unknowns in the order corresponding to diagonals of the grid. It appeared that the linear system can be solved at a speed of approximately 600 Mflop/sec on a NEC SX-3 with one processor, which has a peak performance of 2.75 Gflops/sec.

For the Poisson-model problem with Dirichlet boundary conditions, the number of MICCG iteration steps is not very sensitive to the parameter ζ. However, when Neumann boundary conditions are used, the number of iteration steps depends more clearly on the choice of ζ. In order to illustrate this, Fig. 1.3 shows the results of the MICCG method for various choices of ζ in case the coefficient matrix results from the Poisson-model problem with Neumann boundary conditions everywhere. We used a constant mesh size of $1/(M-1)$ in both directions, leading to a system with again M^2 unknowns. The matrix is singular, because the level of the solution is not fixed. In [21] it is shown that it is preferable to apply CG to the original singular system, instead of eliminating the singularity by keeping the level of the solution fixed during the iteration process. One can show that when ζ is larger than 0, all diagonal elements of the factor L are positive. If ζ is 0 and a standard, lexicographical ordering is used, the last diagonal element of L is zero. When this is the case, this element is replaced by 1.0. From the results of Fig. 1.3 it follows that ζ should be chosen larger than 0. For this problem, a proper choice for ζ is 10.

In Chapter 6 we will describe a preconditioning technique with which we can obtain grid-independent convergence for the test problem considered here: the number of CG iteration steps combined with this preconditioner does not increase when the mesh is refined.

Figure 1.2: Numerical results for Example 1 with Dirichlet boundary conditions.

Figure 1.3: MICCG for $\zeta = 0$, 1, 10 and 100. Example 1, Neumann boundary conditions.

Example 2. This example shows the effect of dominating first-order derivatives in a partial differential equation. The system of linear equations comes from a five-point finite-difference discretization of the steady convection-diffusion equation

$$-\epsilon\Delta u(x,y) + d(x,y)\frac{\partial}{\partial x}u(x,y) + e(x,y)\frac{\partial}{\partial y}u(x,y) = 0 \tag{1.17}$$

on the square $[0,1]\times[0,1]$ with Dirichlet boundary conditions everywhere. The diffusion coefficient $\epsilon = 10^{-5}$, and the functions $d(x,y)$ and $e(x,y)$ are defined as

$$d(x,y) = 4x(x-1)(1-2y), \qquad e(x,y) = -4y(y-1)(1-2x)$$

Equation (1.17) was discretized on a rectangular grid with constant mesh size $1/(M+1)$ in both directions. Using upwind finite differences for the first-order derivatives, the coefficient matrix becomes a non-symmetric M-matrix. The number of unknowns is M^2. In [11] this problem was used as a test problem for a multigrid code. It follows from the numerical experiments in [11], that it is a difficult problem. This is due to the presence of a turning point.

The incomplete decompositions of A are denoted by ILUkl, where the parameters k and l give the sparsity pattern of the factors L and U. For example, if we write ILU23 then $L+U$ has the non-zero pattern as shown in Fig. 1.4. The sparsity pattern of $L+U$ was always chosen to be symmetric.

Figure 1.4: Sparsity pattern of $L+U$ denoted by ILU23.

In order to demonstrate the effect of the preconditioner, Table 1.1 shows the number of iteration steps of Bi-CGSTAB combined with ILUkl for various values of k and l. Only the results of the unmodified ILU-decompositions are shown, because during the construction of a MILUkl-decomposition small or negative elements on the main diagonal are generated, which causes the construction of L and U to break down.

Each step of Bi-CGSTAB requires two matrix-vector products with the preconditioned matrix $L^{-1}AU^{-1}$ plus the computation of a number of inner products and vector updates. The number of flops per unknown for one iteration step is approximately $40 + 8(k+l)$, except when the efficient Eisenstat implementation is possible. In that case one step of Bi-CGSTAB requires only 44 flops per unknown. As a stopping criterion we used

$$\|L^{-1}(b - AU^{-1}\tilde{x}^{(n)})\|_2 < 10^{-10}\|L^{-1}(b - AU^{-1}\tilde{x}^{(0)})\|_2 \tag{1.18}$$

If all entries causing fill-in additional to that of A are neglected, the resulting ILU11-preconditioner is poor. For small values of M, the convergence of Bi-CGSTAB combined with ILU11 is slow, and for $M \geq 150$ we observed that the iteration process stagnates. If more diagonals are allowed in the ILU-decomposition, the efficiency and robustness of the method can be improved strongly. Increasing l in the ILUkl-decomposition drastically reduces the number of iteration steps of Bi-CGSTAB.

Table 1.1: Number of Bi-CGSTAB iterations when combined with ILUkl.

M	ILU11	ILU12	ILU13	ILU23	ILU14
32	27	15	12	12	9
64	55	32	22	22	17
128	215	79	47	46	36
256	stagnation	284	156	153	87

Increasing k has hardly any effect. In order to understand this, part of the matrix $L + U - I$ is shown in Fig. **??** for $M = 32$, and for L and U coming from an ILU24-decomposition. The size of a dot represents the absolute value of the corresponding matrix entry. The fill-in entries below the sub-diagonal and above the super-diagonal are very small in absolute value. Such a fill-in entry is formed by the product of an entry near the lower-most diagonal and one near the upper-most diagonal. Since we use an upwind discretization of a dominating convective part, one of these entries is likely to be small in absolute value, and the same holds for their product.

Example 3. Our third example is the discretized steady convection-diffusion equation

$$-\Delta u(x,y) + 1000x^3 \frac{\partial}{\partial x} u(x,y) - 1000y^3 \frac{\partial}{\partial y} u(x,y) = f(x,y)$$

on the square $[0,1] \times [0,1]$ with Dirichlet boundary conditions everywhere. For the discretization we used a rectangular grid with constant mesh size $1/(M+1)$ in both directions, and central differences for all derivatives, resulting in a system of linear equations with M^2 unknowns. The coefficient matrix is not necessarily an M-matrix, because the mesh-Péclet numbers can be larger than 2. Since the system is non-symmetric, it was solved with preconditioned Bi-CGSTAB. Table 1.2 shows the number of iteration steps required to satisfy the stopping criterion (1.18) for various choices of k and l. The results of MILUkl-preconditioning cannot be shown, because even for $M = 128$ the construction of the incomplete decomposition breaks down due to the generation of small elements on the main diagonal. Even for small values of M, when the mesh-Péclet number is much larger

Table 1.2: Number of preconditioned Bi-CGSTAB iterations for Example 3.

M	ILU11	ILU12	ILU13	ILU23	ILU24
32	30	20	14	10	9
64	40	23	16	14	12
128	69	39	29	24	21
256	164	86	60	54	44

than 2, the construction of the unmodified ILU-decomposition does not break down. The number of iteration steps decreases considerably when k and l are increased. However, one should realize that one iteration step of Bi-CGSTAB combined with ILUkl requires approximately $40 + 8(k + l)$ flops per unknown. Thus the work per unknown for one iteration step increases with k and l. Moreover, for $k = l = 1$ it is possible to use the efficient Eisenstat implementation. Therefore, Fig. 1.5 shows the number of *flops per unknown* against the number of unknowns. From the results of this figure it follows that the

Figure 1.5: Number of flops per unknown for Example 3 on various grids.

ILUkl-decomposition improves when l increases. The ILU13-decomposition is approximately as efficient as the ILU23-preconditioner. The results of ILU24-preconditioning are not shown in Fig. 1.5, because it appeared that these results are approximately the same as those of ILU23-preconditioning.

1.6 Conclusions

We have made a comparison between some iterative methods for solving large sparse systems of linear equations. When the matrix is symmetric and positive definite, the system of equations can be solved very efficiently with the conjugate gradient method. The CG method generates a polynomial that is optimal in the sense that the $A-$weighted 2-norm of the error is minimized over the Krylov subspace. The performance can be improved considerably by using a (modified) incomplete Choleski-decomposition, leading to the so-called (M)ICCG-methods. When the coefficient matrix is not symmetric, the system $Ax = b$ can be solved with CG-like methods like Bi-CGSTAB and GMRES(M).

Some techniques have been demonstrated on a number of test problems, including problems in which the coefficient matrix stems from the discretization of a steady convection-diffusion equation with dominating convective parts. When the coefficient matrix stems from the discretization of a Poisson equation, the MIC-decomposition in which the sparsity pattern of $L + L^T$ is taken the same as that of A leads to a very effective preconditioner. However, when the coefficient matrix stems from the discretization of a PDE with dominating convective parts, the construction of the MILU-decomposition may break down.

The last two test problems are examples of a situation in which both the robustness and efficiency of the preconditioner can be improved by allowing some extra fill-in during the construction of the incomplete LU-decomposition.

2. Incomplete LU-decompositions for matrices with arbitrary sparsity patterns

2.1 Introduction

In the previous chapter we have seen that the convergence rate of conjugate gradient-like methods can strongly be improved by using an incomplete LU-decomposition of A as a preconditioner. As mentioned in Chapter 1, the sparsity pattern of the factors L and U can be described by the set P defined by (1.14). The choice of the splitting to be used is of crucial importance. This is also concluded in [20]: "The most important conclusion that emerges again and again is that nothing can replace a good preconditioning."

Realistic physical problems often involve complicated geometries. If a preprocessor is used to partition the domain into a number of subdomains, the node numbering can be very irregular, which causes a very complicated non-zero structure of the coefficient matrix. We will therefore describe a preconditioning technique which has no restriction with respect to the sparsity pattern of A. Section 2.2 describes how a matrix with an arbitrary sparsity pattern can be stored in memory and how to construct an incomplete LU-decomposition with prescribed sparsity pattern P.

The problem is to find the set P for which the factors L and U are sparse, but also in such a way that the preconditioner LU resembles A as much as possible. In practice, the sparsity pattern of L and U is often taken to be the same as that of the original matrix regardless of the sparsity pattern of A, or of the size of the elements. When the matrix has a sparsity pattern as shown in Fig. 1.1, we can often construct a better approximation of A by allowing fill-in not only in the bands shown in this figure, but also in a number of extra bands [25, 26]. The results of Example 2 in Section 1.5 show that the extra bands must be chosen in such a way that they contain elements which are relatively large in absolute value. This suggests that a proper sparsity pattern for the factors L and U is obtained by allowing only entries which are in absolute value greater than a certain threshold parameter ε [1, 29]. In other words, all entries which are neglected should be small in absolute value. In Section 2.3 we describe how the set P can be chosen in such a way that a splitting $(LU, -R)$ is obtained in which all elements of the residual matrix $R = A - LU$ satisfy

$$|r_{ij}| < \varepsilon \text{ for } 1 \leq i, j \leq N \qquad \text{and} \qquad r_{ij} = 0 \text{ for } (i, j) \in P$$

Since L and U are computed dynamically, and their number of non-zero entries depends on the preset threshold parameter ε, the available storage for an initial choice of ε may be too small. During the elimination process this has to be checked, and if necessary ε must be increased.

Section 2.4 describes how symmetry of the matrix can be exploited, and in Section 2.5 some results are presented for the test problems described in Section 1.5, and for a system of linear equations coming from the finite-element package Sepran.

2.2 Non-symmetric matrices with arbitrary sparsity patterns

First, the data structure is described to store an $N \times N$-matrix A with arbitrary sparsity pattern. This involves the use of three arrays: $coA[1 \dots NzeroA]$ contains all the non-zero entries of A stored row by row, $jcoA[1 \dots NzeroA]$ contains all the column numbers of these entries, and $begA[1 \dots N+1]$ is an array of pointers: $begA[i]$ gives the address in coA of the first non-zero entry in row number i of A. The last entry of $begA$ is added to enable us to point to the last entry in row N of A: this entry is given by $coA[begA[N+1]-1]$. For example, the matrix

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & a_{45} \\ 0 & a_{52} & 0 & 0 & a_{55} \end{bmatrix}$$

can be represented by the coA, $jcoA$ and $begA$ arrays as

- coA: $[a_{11}, a_{13}, a_{21}, a_{22}, a_{23}, a_{33}, a_{41}, a_{44}, a_{45}, a_{52}, a_{55}]$
- $jcoA$: $[1, 3, 1, 2, 3, 3, 1, 4, 5, 2, 5]$
- $begA$: $[1, 3, 6, 7, 10, 12]$

The code for the matrix-vector multiplication $b := Ay$ using this storage scheme is given by Algorithm 2.1.

Algorithm 2.1. The matrix-vector multiplication $b := Ay$.

```
FOR $i := 1$ TO $N$ DO
   BEGIN
      $b[i] := 0$;
      FOR $v := begA[i]$ TO $begA[i+1]-1$ DO
         $b[i] := b[i] + coA[v] \times y[jcoA[v]]$
   END;
```

The incomplete decomposition is based on Doolittle's method for constructing a *complete* LU-decomposition. In this technique, a decomposition $A = LU$ is made in such a way that all diagonal elements of L are equal to one. We can rewrite $A = LU$ as

$$a_{ik} = \sum_{j=1}^{\min(i,k)} l_{ij} u_{jk} \tag{2.1}$$

This gives the following explicit formulas for l_{ik} and u_{ik}:

$$l_{ik} = (a_{ik} - \sum_{j=1}^{k-1} l_{ij} u_{jk}) / u_{kk} \qquad \text{for } i > k \tag{2.2}$$

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} l_{ij} u_{jk} \qquad \text{for } i \leq k \tag{2.3}$$

Suppose that both L and U are stored in the two-dimensional array $\{lu_{ij} | 1 \leq i, j \leq N\}$, and lu_{ij} is equal to a_{ij} for $1 \leq i, j \leq N$. Algorithm 2.2 computes the complete LU-decomposition using (2.2) and (2.3).

Algorithm 2.2. The complete LU-decomposition.

```
FOR $i := 2$ TO $N$ DO
    {The following statement computes row $i$ of the $L/U$-array.}
  FOR $j := 1$ TO $i - 1$ DO
    BEGIN
      $lu_{ij} := lu_{ij} / lu_{jj}$;
      FOR $k := j + 1$ TO $N$ DO $lu_{ik} := lu_{ik} - lu_{ij} \times lu_{jk}$
    END;
```

After this algorithm has been executed, the elements of L and U are stored as

$$l_{ij} = 0 \text{ and } u_{ij} = lu_{ij} \qquad \text{for } i < j$$

$$u_{ij} = 0 \text{ and } l_{ij} = lu_{ij} \qquad \text{for } j < i$$

$$l_{ii} = 1 \text{ and } u_{ii} = lu_{ii}$$

When making an *incomplete* LU-decomposition, we again only store the non-zero entries of L and U. We need another extra help array $diLU[1 \ldots N]$ which points to the diagonal elements of U in the array $coLU$. Suppose that the non-zero structure P of L and U is stored in $jcoLU$ and $begLU$, and $lu_{ij} = a_{ij}$ for $(i, j) \in P$. Algorithm 2.3 calculates the incomplete decomposition. The boolean variable $modify$ is $false$ for the unmodified incomplete decomposition, and $true$ for the modified version in which the row sums of the residual matrix R are equal to zero. The modifications of the main diagonal are temporarily stored in $coLU[0]$. The array $point[1 \ldots N]$ is an array of integers which point to the entries in L and U of row i. Before the algorithm starts, all the entries of $point$ must have been initialised to zero.

Algorithm 2.3. The incomplete LU-decomposition.

```
FOR $i := 2$ TO $N$ DO
   BEGIN
         {$right$ and $di$ are integer variables which point to the last}
         {and the diagonal element of row $i$ respectively.}
      $right := begLU[i+1]-1$; $di := diLU[i]$;
         {The modifications of the main diagonal are stored in $coLU[0]$.}
      $coLU[0] := 0$;
      FOR $v := begLU[i]+1$ TO $right$ DO $point[jcoLU[v]] := v$;
         {$v$ points to $lu_{ij}$ in Algorithm 2.2.}
      FOR $v := begLU[i]$ TO $di-1$ DO
         BEGIN
            $j := jcoLU[v]$;
               {The next statement is the equivalence of}
               {$lu_{ij} := lu_{ij}/lu_{jj}$; in Algorithm 2.2.}
            $coLU[v] := coLU[v]/coLU[diLU[j]]$;
               {$w$ points to $lu_{jk}$ from Algorithm 2.2.}
            FOR $w := diLU[j]+1$ TO $begLU[j+1]-1$ DO
               BEGIN
                  $k := point[jcoLU[w]]$;
                     {The next statement is the equivalence of}
                     {$lu_{ik} := lu_{ik} - lu_{ij} \times lu_{jk}$; in Algorithm 2.2.}
                  $coLU[k] := coLU[k] - coLU[v] \times coLU[w]$
               END {$w$-loop.}
         END; {$v$-loop.}
      IF $modify$ THEN $coLU[di] := coLU[di] + coLU[0]$;
      FOR $v := begLU[i]+1$ TO $right$ DO $point[jcoLU[v]] := 0$
   END; {$i$-loop.}
```

2.3 The sparsity pattern based on a drop tolerance

When solving the system $Ax = b$ using the splitting $(LU, -R)$, we consider the system $(LU)^{-1}Ax = (LU)^{-1}b$. The preconditioned matrix $(LU)^{-1}A$ has to resemble the identity matrix I as closely as possible. Since we have that $(LU)^{-1}A = (LU)^{-1}[LU + R] = I + (LU)^{-1}R$, the matrix $(LU)^{-1}R$ should be small in some sense. Theorem 1 states that $(LU)^{-1}$ is a proper approximation to A^{-1} if and only if $\|(LU)^{-1}R\|$ is small for some matrix norm $\| \; \|$.

Theorem 1 *Let $(LU, -R)$ be a splitting of the non-singular $N \times N$-matrix A where $R = A - LU$, and the product LU is non-singular. Then*

$$\frac{\|(LU)^{-1}R\|}{cond(A)} \le \frac{\|(LU)^{-1} - A^{-1}\|}{\|A^{-1}\|} \le \|(LU)^{-1}R\| \tag{2.4}$$

where $cond(A)$, the condition number of A, is defined as the product of $\|A\|$ and $\|A^{-1}\|$. If, in addition, x is the solution of $Ax = b$, and $\tilde{x}$ satisfies $LU\tilde{x} = b$, then

$$\frac{\|x - \tilde{x}\|}{\|x\|} \le \|(LU)^{-1}R\| \tag{2.5}$$

Proof: we can rewrite $(LU)^{-1}R$ as follows.

$$(LU)^{-1}R = (LU)^{-1}(A - LU) = (LU)^{-1}A - I = [(LU)^{-1} - A^{-1}]A \tag{2.6}$$

so that

$$\|(LU)^{-1}R\| \le \|A^{-1} - (LU)^{-1}\|\|A\|$$

By dividing the left- and right-hand sides by $\|A^{-1}\|\|A\|$, one obtains the first inequality of (2.4). The second inequality follows from equation (2.6).

$$(LU)^{-1}R = [(LU)^{-1} - A^{-1}]A \qquad \Longleftrightarrow$$
$$(LU)^{-1}RA^{-1} = (LU)^{-1} - A^{-1} \qquad \Longrightarrow$$
$$\|(LU)^{-1} - A^{-1}\| \le \|(LU)^{-1}R\|\|A^{-1}\|$$

After division by $\|A^{-1}\|$, we obtain the desired inequality. Finally we prove (2.5).

$$x - \tilde{x} = A^{-1}b - (LU)^{-1}b = (A^{-1} - (LU)^{-1})Ax$$

Together with (2.6) we get $x - \tilde{x} = -(LU)^{-1}Rx$. Taking the norm leads to (2.5), which completes the proof of Theorem 1. $\square$

The following theorem states that making $\|R\|$ small will have a positive effect on the condition number of $(LU)^{-1}A$.

Theorem 2 *Let $(LU, -R)$ be a splitting of the non-singular matrix A, and $\|A^{-1}R\| < 1$. Then the matrix LU is non-singular, and*

$$cond[(LU)^{-1}A] \le \frac{1 + \|A^{-1}R\|}{1 - \|A^{-1}R\|} \tag{2.7}$$

Proof: suppose LUx is equal to the null vector 0. With this we obtain

$$LUx = 0 \Longleftrightarrow (A - R)x = 0 \Longleftrightarrow (I - A^{-1}R)x = 0$$

so that

$$\|A^{-1}Rx\| = \|x\| \Longrightarrow \|x\| \leq \|A^{-1}R\|\|x\|$$

Together with $\|A^{-1}R\| < 1$ the last inequality implies that $\|x\|$ is equal to zero, thus $x = 0$. This proves that LU is non-singular. We also have that

$$\text{cond}[(LU)^{-1}A] = \text{cond}[(A - R)^{-1}A] = \text{cond}[(I - A^{-1}R)^{-1}]$$

$$= \|(I - A^{-1}R)^{-1}\|\|I - A^{-1}R\| \leq \|(I - A^{-1}R)^{-1}\|(1 + \|A^{-1}R\|)$$

By Theorem 7.11 of Atkinson [2], $\|(I - A^{-1}R)^{-1}\| \leq 1/(1 - \|A^{-1}R\|)$. This completes the proof of Theorem 2. □

Theorem 1 and 2 show that it is important that the residual matrix is small in some sense. In the following, we try to achieve this by making the entries of R small in magnitude. The set P follows from the construction of a splitting $(LU, -R)$ in which $R = A - LU$ satisfies

$$|r_{ij}| < \varepsilon \text{ for } 1 \leq i, j \leq N \qquad \text{and} \qquad r_{ij} = 0 \text{ for } (i,j) \in P \tag{2.8}$$

Herein ε is a threshold parameter which has to be chosen in advance. When neglecting the effect of roundoff errors, we make an exact decomposition of the matrix $A - R$.

The construction of the matrices L and U is as follows. First the matrix is scaled in such a way that

$$\sum_{j=1}^{N} |a_{ij}| = 1 \qquad \text{for } 1 \leq i \leq N \tag{2.9}$$

Next, the incomplete decomposition is constructed row by row. Given the first $i-1$ rows of L and U, we construct row i of L and U from $R = A - LU$ as

$$r_{ik} = a_{ik} - \sum_{j=1}^{\min(i,k)} l_{ij}u_{jk} \tag{2.10}$$

Suppose l_{ij} has been calculated for $j < k$. If $k < i$ one obtains from (2.10)

$$r_{ik} + l_{ik}u_{kk} = a_{ik} - \sum_{j=1}^{k-1} l_{ij}u_{jk}, \qquad k < i \tag{2.11}$$

If the absolute value of the right-hand side of this equation is less than ε, fill-in on position (i,k) is neglected; otherwise l_{ik} is calculated from (2.11) together with $r_{ik} = 0$. With $l_{ii} = 1$, u_{ii} can be calculated from

$$u_{ii} = a_{ii} - \sum_{j=1}^{i-1} l_{ij}u_{ji} \tag{2.12}$$

If $k > i$ one obtains from (2.10) together with $l_{ii} = 1$

$$r_{ik} + u_{ik} = a_{ik} - \sum_{j=1}^{i-1} l_{ij} u_{jk}, \qquad k > i \tag{2.13}$$

If the absolute value of the right-hand side of (2.13) is greater than or equal to ε, we demand r_{ik} to be zero; otherwise fill-in on position (i, k) is neglected.

Note that if we want to make a modified incomplete decomposition, the fill-in entries which are neglected in the above algorithm must be added to the main diagonal.

2.4 Symmetric matrices with general sparsity pattern

When the coefficient matrix A is sparse and symmetric, it is possible to exploit both the sparsity and the symmetry. To store A we again need three arrays coA, $jcoA$ and diA. The integer array $diA[0 \dots N]$ points to the diagonal elements of the matrix. The first entry of diA is used to point to the first entry of the first row: this entry is given by $coA[diA[0] + 1]$. For example, the symmetric matrix

$$A = \begin{bmatrix} a_{11} & a_{21} & 0 & a_{41} & 0 \\ a_{21} & a_{22} & 0 & 0 & a_{52} \\ 0 & 0 & a_{33} & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & a_{54} \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{bmatrix}$$

is stored as

- coA: $[a_{11}, a_{21}, a_{22}, a_{33}, a_{41}, a_{44}, a_{52}, a_{54}, a_{55}]$
- $jcoA$: $[1, 1, 2, 3, 1, 4, 2, 4, 5]$
- diA: $[0, 1, 3, 4, 6, 9]$

The storage requirements for both A and the incomplete decomposition can roughly be halved compared with the non-symmetric case. In order to demonstrate how this storage scheme works, Algorithm 2.4 gives the matrix-vector multiplication $b := Ay$ for a symmetric matrix A.

Following the technique described in Section 2.3, we can make a splitting $(LDL^T, -R)$ in which $R = A - LDL^T$ again satisfies (2.8). Suppose that rows $1 \dots i-1$ of L and D have been constructed, thus for $k < i$ the elements l_{kj}, $1 \le j < k$ and d_{kk} are known. Then row i of L and D can be constructed from

$$r_{ik} = a_{ik} - \sum_{j=1}^{k} l_{ij} d_{jj} l_{kj} \qquad \text{for } k \le i \tag{2.14}$$

The CPU-time required for this symmetric decomposition is roughly half the time needed for the incomplete LU-decomposition for non-symmetric matrices.

Algorithm 2.4. The matrix-vector multiplication $b := Ay$ using the symmetric storage scheme for A.

```
FOR $i := 1$ TO $N$ DO
  BEGIN
    $b[i] := coA[diA[i]] \times y[i]$;
    FOR $v := diA[i-1]+1$ TO $diA[i]-1$ DO
      BEGIN
        $b[i] := b[i] + coA[v] \times y[jcoA[v]]$;
        $b[jcoA[v]] := b[jcoA[v]] + coA[v] \times y[i]$
      END
  END;
```

2.5 Numerical results

The techniques described in Section 2.3 have been developed for matrices with an arbitrary sparsity pattern. Hence when A has a banded sparsity pattern as shown in Fig. 1.1, these methods are not as efficient as the special purpose methods of Chapter 1. However, the difference in efficiency is not very large. In order to demonstrate this, the first three examples are the same as in Section 1.5. We used the same discretization and stopping criteria as in Section 1.5 for these examples. The last example demonstrates that the techniques described in this chapter can be used, even when the sparsity pattern of the coefficient matrix is very irregular. When the sparsity pattern of the factors L and U was chosen in such a way that all elements of the residual matrix are in absolute value smaller than ε, this is indicated with (M)ILU(ε) in the non-symmetric case, and with (M)IC(ε) in the symmetric case. All computations were performed in double precision arithmetic on an HP-720 workstation. In all examples the main diagonals of both L and U were scaled to unity, thus saving N multiplications each time the preconditioner is applied.

Example 1. This is the Poisson-model problem described in Section 1.5 with Neumann boundary conditions everywhere. The Poisson equation was discretized over a rectangular $M \times M$-grid with constant mesh size $1/(M-1)$, leading to a symmetric system of linear equations with M^2 unknowns. Fig. 2.1 shows the results of the conjugate gradient method combined with MIC(ε)-preconditioning for various choices of ε. For comparison, this figure also shows the results of the standard MIC-decomposition of Section 1.4, in which the sparsity pattern of $L + L^T$ is the same as that of A. From these results we conclude that the preconditioning technique of Section 2.3 is almost as efficient as standard MICCG combined with the efficient Eisenstat implementation. Note that the latter implementation can only be applied to matrices with a banded sparsity pattern.

The choice of the parameter ε is not very critical for the CPU-time required for the iteration process. This is illustrated by the results of the fourth column of Table 2.1, which show the CPU-times in seconds for the iteration process for $M = 256$, and for various choices of ε. Table 2.1 also lists the number of iteration steps and the number of

Figure 2.1: Results of MICCG(ε) for Example 1 for various choices of ε.

non-zero entries in L divided by the number of unknowns. The fifth column shows the CPU-time required for the construction of the incomplete decomposition, and the last column shows the number of flops per unknown necessary for the iteration process. The

Table 2.1: Numerical results for MICCG(ε) for Example 1 with $M = 256$.

	# it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
$\varepsilon = 0.1$	81	4.0	32.1	1.5	2511
$\varepsilon = 0.02$	69	5.0	31.3	1.8	2415
$\varepsilon = 0.01$	58	7.0	32.0	3.0	2494
$\varepsilon = 0.002$	41	13.8	32.7	6.9	2878

number of CG iteration steps decreases with decreasing ε, but for small ε one iteration step becomes relatively expensive. Consequently, the number of flops per unknown varies only slightly with ε.

Example 2. The second test problem is the same as Example 2 of Chapter 1. For convenience, the partial differential equation has been written down below once again.

$$-10^{-5}\Delta u(x,y) + d(x,y)\frac{\partial}{\partial x}u(x,y) + e(x,y)\frac{\partial}{\partial y}u(x,y) = 0$$

Herein

$$d(x, y) = 4x(x - 1)(1 - 2y), \text{ and } \qquad e(x, y) = -4y(y - 1)(1 - 2x)$$

This PDE was discretized on a rectangular grid with constant mesh size $1/(M+1)$ in both directions. Using upwind finite differences for the first-order derivatives, the coefficient matrix becomes a non-symmetric M-matrix. Fig 2.2 shows the results of Bi-CGSTAB combined with ILU(ε)-preconditioning for several choices of ε.

Figure 2.2: Numerical results of ILU(ε)-preconditioning for Example 2.

From the results we conclude that for this linear system, decreasing ε gives much better incomplete decompositions, although the computer storage needed for the factors L and U increases: for $M = 256$ and $\varepsilon = 0.001$, we observed that the average number of non-zero entries in one row of $L + U$ is 24.6, and the preconditioned system of equations requires 19 iteration steps of Bi-CGSTAB to fulfil the stopping criterion.

In Section 1.5, it appeared to be advantageous to increase l in the ILUkl-decomposition. Thus allowing more fill-in in the neighbourhood of the outer-most diagonals reduces the number of Bi-CGSTAB iteration steps very effectively. The incomplete LU-decomposition based on a drop tolerance automatically gives more fill-in in this area. This is illustrated by Fig. **??**, which shows a part of the matrix $L + U - I$ for $M = 32$ and $\varepsilon = 0.01$.

Example 3. The third test problem is the same as Example 3 of Chapter 1. It concerns the solution of the PDE

$$-\Delta u(x, y) + 1000x^3 \frac{\partial}{\partial x} u(x, y) - 1000y^3 \frac{\partial}{\partial y} u(x, y) = f(x, y)$$

on the square $[0,1] \times [0,1]$ with Dirichlet boundary conditions. The discretization is described in Section 1.5. The system of linear equations was solved with Bi-CGSTAB combined with ILU(ε) as a preconditioner. The results of modified incomplete decompositions cannot be shown, because the construction of L and U breaks down if the modifications of the diagonal are performed.

Table 2.2 shows the numerical results for $M = 256$. The second column shows the number of iteration steps of Bi-CGSTAB, and the third column presents the average number of entries in $L + U$ per row. This number increases when ε decreases, but the number of iteration steps decreases very strongly. As a result, the number of flops per unknown required for Bi-CGSTAB decreases considerably, which is shown in the last column of Table 2.2. A drawback of choosing ε small is of course that computer storage demands are relatively high. In Chapter 6 we will describe a very effective preconditioning technique, which does not require a large amount of storage.

Table 2.2: Numerical results for ILU(ε) for Example 3 with $M = 256$.

	# it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
$\varepsilon = 0.1$	105	5.9	109	2.05	6258
$\varepsilon = 0.01$	42	11.6	59.4	3.1	3461
$\varepsilon = 0.001$	14	29.0	36.5	10.2	2128

Fig. 2.3 shows the 10-logarithm of the Euclidean norm of the residual $b - Ax^{(n)}$ versus the number of flops per unknown for various choices of ε. From these results it follows that decreasing ε reduces the work per unknown considerably. For small ε the convergence behaviour is relatively smooth.

Example 4. The next example concerns the solution of a system of linear equations in which the coefficient matrix has a very irregular sparsity pattern coming from an unstructured grid. The linear system comes from a finite-element discretization of the PDE

$$-\Delta u(x,y) + x\frac{\partial}{\partial x}u(x,y) + y\frac{\partial}{\partial y}u(x,y) = 0$$

We consider two different meshes leading to 2589 and 6765 unknowns. The domain under consideration together with the boundary conditions and the coarsest mesh are shown in Fig. ??. The finest mesh was obtained after a refinement of the coarsest mesh. The discretization was performed with the finite-element package Sepran using quadratic isoparametric triangles. Part of the coefficient matrix arising after the discretization on the coarsest mesh is shown in Fig. ??. The sparsity pattern is very irregular, and the number of entries in each row of A is larger than in the previous examples. Table 2.3 shows the numerical results of Bi-CGSTAB combined with MILU(0.001)-preconditioning. As a stopping criterion we used

$$\|L^{-1}(b - AU^{-1}\tilde{x}^{(n)})\|_2 < 10^{-8}\|L^{-1}(b - AU^{-1}\tilde{x}^{(0)})\|_2$$

Figure 2.3: Convergence behaviour of Bi-CGSTAB for Example 3.

The last two columns in this table show the CPU-time measured in seconds for the iteration process and for the construction of the factors L and U. For comparison, it

Table 2.3: Numerical results for Example 4.

# unknowns	# it.	# nonz/row	CPU-time it.	CPU-time prec.
2589	8	27	0.69	0.29
6765	13	29	2.70	0.80

is interesting to mention that the direct solver of Sepran needed 1.71 seconds for the smallest system, and 11.17 seconds for the largest system of linear equations. The CPU-time needed for the construction of the preconditioner grows linearly with the number of entries in $L + U$. It appears that Bi-CGSTAB combined with MILU(ε)-preconditioning can exploit very well even the irregular sparsity pattern of the coefficient matrix.

2.6 Conclusions

In Section 2.2 methods have been described to construct preconditioners for non-symmetric M-matrices with an arbitrary sparsity pattern. This allows the use of complicated geometries and an irregular node numbering. Even in many cases when A is not an M-matrix, one can obtain proper preconditioners with the methods described in Section 2.2.

It is then necessary to take measures in order to prevent the construction of the incomplete decomposition from breaking down due to the generation of small or negative diagonal elements. Methods in which the sparsity pattern of L and U is based on a threshold parameter give excellent preconditioners, and the CPU-time required for the construction of the incomplete decomposition is small compared with the time needed for the iteration process. From the numerical experiments we conclude that the CPU-time required for the incomplete decomposition grows linearly with the number of non-zero entries.

If the coefficient matrix is symmetric, it is possible to exploit this symmetry, even when the sparsity pattern is very irregular. The CPU-time for the construction of the preconditioner can roughly be halved, and the storage requirements for both A and the incomplete decomposition can be lowered.

3. Preconditioning techniques for non-symmetric matrices with application to temperature calculations of cooled concrete

3.1 Introduction

In this chapter we describe a model for the computation of the temperature distribution in a large block of concrete. During the hydration process the temperature rises to such a level that serious damage to the construction can occur due to expansion. Therefore, extra cooling is necessary during the hardening process. The model leads to a non-stationary partial differential equation for the temperature distribution. In order to solve this PDE, a system of linear equations has to be solved several times, in which the largest part of the coefficient matrix is a symmetric M-matrix with a regular sparsity pattern as shown in Fig. 3.1. As will be explained in Section 3.2, a relatively small part of the matrix has

Figure 3.1: Sparsity pattern for a seven-point finite-difference operator over a rectangular grid.

an irregular sparsity pattern due to some special boundary conditions arising from the extra cooling of the concrete.

When the geometry of the block of concrete is complex, the computation of the temperature distribution requires the solution of linear systems in which the sparsity pattern of the coefficient matrix is irregular. Therefore, we consider some methods to construct an incomplete LU-decomposition in which no restriction is made with respect to the non-zero structure. In the previous chapter we have shown how to construct an incomplete LU-decomposition with a prescribed sparsity pattern P defined by (1.14). When the sparsity pattern of A is irregular, there are several possibilities for a proper construction of the set P. Gustafsson proposed the following. First consider the standard incomplete LU-decomposition, i.e. $P = \{(i,j) | a_{ij} \neq 0\}$. Then extend P with positions (i,j) where the product LU has non-zero elements, and eventually repeat this a few more times [17]. This technique has been tested extensively by Langtangen [23].

Another approach determines the elements in P during the elimination. P is described implicitly by allowing only entries which are in absolute value greater than a certain drop tolerance [1, 29]. The incomplete LU-decomposition based on a drop tolerance has been described in more detail in Section 2.3.

In Section 3.2 the model of the hardening of concrete is described, and in Section 3.3 we describe how to construct a preconditioner which takes advantage of the symmetry and the banded non-zero pattern of the major part of A. Section 3.4 compares some choices of the set P, using the concrete cooling problem of Section 3.2 as a test problem. The fact that no restriction is made with respect to the sparsity pattern of A enables us to perform a renumbering of the unknowns. In Section 3.5 this renumbering is chosen in such a way that a large part of the unknowns can be eliminated in advance.

3.2 The hardening of concrete

As a numerical example, we choose the simulation of the temperature distribution in a large rectangular block of concrete. During the hardening of the concrete a substantial amount of heat is produced. When the volume of the block is large in comparison with its surface, the heat exchange between the concrete and air is not sufficient to keep the temperature below a critical level. This can lead to cracks at vital places, so that cooling is necessary during the hardening process. Therefore, cooling water is pumped through pipes which lie inside the concrete. Spijkstra [40] and van Diepen [51] have done research on the simulation of the temperature distribution in such a cooled block of concrete. They considered a block as shown in Fig. 3.2. The plane $y = 0.4$ contains one cooling pipe which

Figure 3.2: The concrete block with cooling pipes. The dimensions are given in meters.

is folded eight times as shown in Fig. 3.2. The cooling pipe can contain 10 litre of water. At the bottom of the block of concrete, 500 litre of cooling water per hour is pumped into the cooling pipe. The plane $y = 0.4$ is a symmetry plane, so that we only have to consider the part $0 \leq y \leq 0.4$. The temperature T_c in the concrete obeys the following heat-conduction equation

$$-\mathrm{div}(\lambda\,\mathrm{grad}\,T_c) + \rho c \frac{\partial}{\partial t} T_c = q \tag{3.1}$$

where λ is the thermal conductivity, ρ is the density, and c is the specific heat. The generation of heat per second for a unit volume due to the hydration process is given by the temperature-dependent function q. At $z = 0$ and $z = 5.15$ the block is completely isolated leading to the Neumann boundary condition $\partial T_c/\partial n = 0$. At the symmetry plane $y = 0.4$ we have the same boundary condition. The cooling of the concrete at the cooling pipes is described by the following Robin boundary condition

$$-\lambda \frac{\partial T_c}{\partial n} = \alpha_{cw}(T_c - T_{cw})$$

The temperature of the cooling water is given by T_{cw}, and α_{cw} is the transfer coefficient between the concrete and the cooling water. At all the other boundaries the heat exchange between air and the concrete is described by a similar boundary condition

$$-\lambda \frac{\partial T_c}{\partial n} = \alpha_a (T_c - T_a)$$

with T_a the prescribed air temperature, and α_a the transfer coefficient between the concrete and air.

Suppose that ξ represents a position along the cooling pipe. In [40] it is shown that the temperature of the cooling water at ξ can be calculated from the following equation

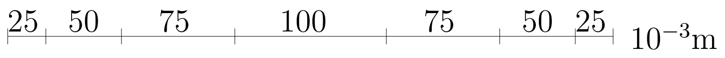
$$T_{cw}(\xi) = \int_0^{\xi} \frac{2\pi r [T_c(s) - T_{cw}(s)]\alpha_{cw}}{\rho_w c_w S_p v} ds + T_{cw}(0) \qquad (3.2)$$

where c_w is the specific heat of water, ρ_w is the density, S_p and r represent the surface and the radius respectively of the cross section of the cooling pipe, and v is the velocity of the cooling water in the pipe.

At the beginning of the hardening process ($t = 0$), a uniform temperature distribution of 20 °C is assumed in the concrete, and $T_{cw}(0)$ is set to 5 °C. Suppose that $T_c^{(n)}$ and $T_{cw}^{(n)}$ represent the temperature of the concrete and cooling water respectively after a certain time $t = t_n$. For the discretization in time of (3.1) we use the backward-Euler scheme given by

$$-\text{div}(\lambda \, \text{grad} \, T_c^{(n+1)}) + \rho c \frac{T_c^{(n+1)} - T_c^{(n)}}{\Delta t} = q^{(n)}$$

The first term in the left-hand side is discretized using a standard seven-point central-difference scheme. We use an equidistant grid in the $x-$ and $z-$direction with mesh size $h_x = 5.5$ meter and $h_z = 0.05$ meter, respectively. The mesh size in the $x-$direction can be relatively large, because the temperatures vary only very little in this direction. In the $y-$direction we use a non-uniform grid in which the mesh size varies as



æ

Suppose ξ_1 and ξ_2 represent two successive points at the cooling pipe. Equation (3.2) is discretized according to the trapezoidal rule as

$$\frac{T_{cw}^{(n+1)}(\xi_2) - T_{cw}^{(n+1)}(\xi_1)}{\xi_2 - \xi_1} = \frac{\pi r \alpha_{cw} [T_c^{(n+1)}(\xi_2) - T_{cw}^{(n+1)}(\xi_2) + T_c^{(n+1)}(\xi_1) - T_{cw}^{(n+1)}(\xi_1)]}{\rho_w c_w S_p v}$$

Suppose that $X_c^{(n+1)}$ is the vector with components equal to $T_c^{(n+1)}$ in the grid points, and the components of $X_{cw}^{(n+1)}$ give the temperature of the cooling water $T_{cw}^{(n+1)}$ at the discretization points of the cooling pipes. After discretization one obtains the following system of linear equations for the unknowns $X_c^{(n+1)}$ and $X_{cw}^{(n+1)}$, in which the coefficient matrix is a non-symmetric M-matrix.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_c^{(n+1)} \\ X_{cw}^{(n+1)} \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \qquad (3.3)$$

Herein B_1 and B_2 are both known vectors, and A_{11} is a large symmetric M-matrix with a sparsity pattern as shown in Fig. 3.1, and with dimensions 4160×4160. The block A_{12} has dimensions 4160×45, and it has the following block structure

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ H_{12} \end{bmatrix} \tag{3.4}$$

where the block H_{12} has dimensions 520×45. The block A_{21} has the block structure

$$\begin{bmatrix} 0 & \cdots & 0 & H_{21} \end{bmatrix}$$

where H_{21} is a 45×520 block. The sparsity pattern of H_{12} and H_{21} is very irregular, and $H_{12} \neq H_{21}^T$. The block A_{22} is a non-symmetric matrix with dimensions 45×45.

At every time step we have to solve a system of linear equations as described above.

3.3 A special-purpose method

In reference [25] and [26] one can find how to construct by incomplete Choleski decomposition a very effective preconditioner for A_{11}. Suppose that LL^T is such an incomplete decomposition of A_{11}. Then the following matrix will be used as a preconditioner for A:

$$\begin{bmatrix} LL^T & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L & 0 \\ A_{21}L^{-T} & A_{22} - A_{21}L^{-T}L^{-1}A_{12} \end{bmatrix} \begin{bmatrix} L^T & L^{-1}A_{12} \\ 0 & I \end{bmatrix} \tag{3.5}$$

The incomplete decompositions of A_{11} are denoted by ICklm, where the parameters k, l and m give the sparsity pattern of the factor L. For example, if we write IC213 then L^T has the non-zero pattern as shown in Fig. 3.3. Similarly, MICklm denotes the modified incomplete Choleski decomposition of A_{11}, in which the diagonal is modified as described in Section 1.4.

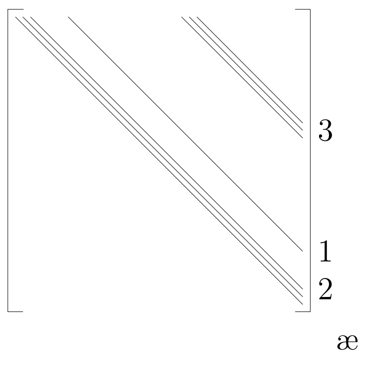


Figure 3.3: Incomplete factor L^T denoted by IC213.

To compute column k of $A_{21}L^{-T}L^{-1}A_{12}$, we first compute the corresponding column of $L^{-1}A_{12}$. From the block structure of A_{12}, which is shown in (3.4), it is obvious that only the last 520 entries can be non-zero. In order to be able to compute column k of $A_{21}L^{-T}L^{-1}A_{12}$, we then calculate the last 520 entries of column k of $L^{-T}L^{-1}A_{12}$. Once these entries are calculated, column k of $A_{21}L^{-T}L^{-1}A_{12}$ can be calculated very rapidly, because the block H_{21} in A_{21} is extremely sparse. We also need the complete LU-decomposition of $A_{21}L^{-T}L^{-1}A_{12}$. This is no problem, because the dimensions of this block are 45×45, which is very small compared with the dimensions of A.

The preconditioners combined with Bi-CGSTAB are demonstrated using the concrete cooling problem of Section 3.2 as a test problem. The temperature distribution in the block of concrete and the cooling water has been calculated after one time step of 3 hours. The results are compared with SOR and SLOR. The iteration processes of the latter two were stopped when

$$\|x^{(k+1)} - x^{(k)}\|_\infty < 10^{-6}\|b\|_\infty$$

When SOR or SLOR is used, it appears to be advantageous to use a much smaller relaxation parameter for the water temperatures than for the temperatures in the concrete. Only results with the best choices of these parameters we could find are shown. Preconditioned Bi-CGSTAB was stopped when

$$\|C^{-1}b - C^{-1}Ax^{(k)}\|_\infty < 10^{-6}\|C^{-1}b\|_\infty$$

The results are summarised in Table 3.1. The third column gives the norm of the relative residual

$$\frac{\|C^{-1}b - C^{-1}Ax^{(k)}\|_\infty}{\|C^{-1}b\|_\infty}$$

at the end of the iteration process. The last column gives the CPU-time needed for the construction of the preconditioner, including the computation of the complete LU-decomposition of $A_{22} - A_{21}L^{-T}L^{-1}A_{12}$. The computations were performed on both an Olivetti PC XP1 with both an Intel 80386 processor and an 80387 as numeric coprocessor, and on a CONVEX C2 of which we used only one processor. CPU-times measured on these machines are given in seconds. The number of flops per unknown for one iteration step of Bi-CGSTAB combined with (M)ICklm is approximately $48 + 8(k + l + m)$. Since A_{11} has a banded non-zero pattern, its entries can be stored in four arrays. The matrix-vector multiplication vectorizes very well on the CONVEX, because the non-zero entries of A_{11} can be addressed directly in this way. With a proper preconditioner, Bi-CGSTAB converges very rapidly, despite the irregular sparsity pattern and non-symmetry of a small part of the matrix. For this problem Bi-CGSTAB is considerably faster than SOR and SLOR. Moreover, S(L)OR has as drawback that it requires the choice of two relaxation parameters, whereas preconditioned Bi-CGSTAB does not require these sort of parameters. There is no difference in the rate of convergence between IC101 and IC111. This is caused by the fact that the mesh size in the $x-$direction is very large compared with the mesh sizes in the other directions. As a consequence, the coupling in the $x-$direction is very weak, and the corresponding band of A contains elements which are very small in absolute value. Table 3.1 shows that MICklm performs better than ICklm, especially when not many non-zero entries are allowed in the factor L.

Table 3.1: S(L)OR in comparison with preconditioned Bi-CGSTAB.

Method/ preconditioner	Number of iterations	Relative residual	Time for iteration		Time for preconditioning	
			PC	CONVEX	PC	CONVEX
SOR	49	7.1×10^{-7}	28.0	1.30		
SLOR	41	8.2×10^{-7}	27.7	1.29		
IC111	9	5.9×10^{-7}	27.3	0.77	0.71	0.022
IC101	9	5.9×10^{-7}	24.7	0.61	0.55	0.019
MIC101	6	1.6×10^{-7}	17.2	0.44	0.71	0.025
IC102	5	2.9×10^{-7}	15.6	0.42	0.93	0.028
MIC102	4	4.8×10^{-8}	12.9	0.35	0.99	0.033
IC103	4	4.5×10^{-7}	13.6	0.40	1.15	0.038
MIC103	3	3.3×10^{-7}	10.8	0.32	1.32	0.045
IC203	3	7.2×10^{-7}	11.5	0.39	1.54	0.050
MIC203	2	8.2×10^{-7}	8.6	0.30	1.70	0.060
IC204	3	1.7×10^{-7}	12.0	0.45	1.87	0.060
MIC204	2	2.3×10^{-7}	9.0	0.34	2.20	0.072

The preconditioning technique described in this section is no longer efficient when the part with an irregular sparsity pattern becomes too large, because in that case, the LU-decomposition of $A_{22} - A_{21}L^{-T}L^{-1}A_{12}$ becomes very time-consuming. In the next section some techniques are described that can be used for the preconditioning of non-symmetric matrices with an irregular sparsity pattern.

The calculated temperature distribution.
It appears that the temperature of the concrete reaches a maximum value after approximately 15 hours. The calculated temperature distribution in the symmetry plane $y = 0.4$ is shown in Fig. 3.4. The brightness corresponds to the temperature in degrees °C as shown in the left-most picture. As can be expected, at the bottom of the block the cooling is very effective, because the temperature of the cooling water is relatively low in this area. The highest level of the temperature is reached at the upper surface of the block.

3.4 Comparison of some sparsity patterns of $L + U$

In this section three different choices for the sparsity pattern of $L + U$ are compared with each other. The numerical results will show that the methods of this section are not as efficient as the special-purpose method of Section 3.3, which is due to the fact that no assumption is made about the non-zero pattern of A. Hence they can be used, for example, when the geometry of the block of concrete is more irregular, or when the number of cooling pipes increases.

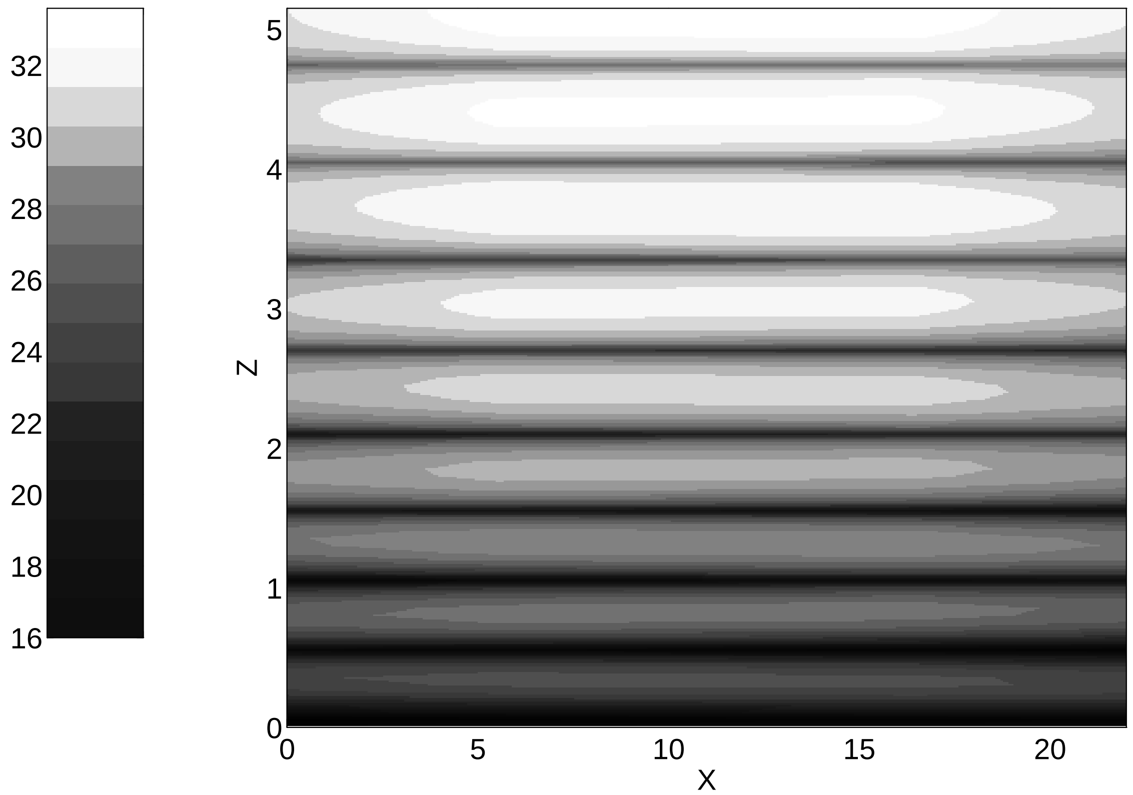


Figure 3.4: The temperature distribution in the plane $y = 0.4$ after 15 hours.

(i) Using the sparsity pattern of A^m.
Suppose that the main diagonal of A is scaled to one. Since A is strictly diagonally dominant, the spectral radius of the Jacobi matrix $I - A$ is less than 1. Hence the geometric series

$$A^{-1} = [I - (I - A)]^{-1} = I + (I - A) + (I - A)^2 + (I - A)^3 + \dots$$

converges. From this expression we learn that its truncation to m terms has a sparsity pattern equal to that of A^m. This suggests the following choice for the set P:

$$P_m = \{(i,k) | (A^m)_{ik} \neq 0\}$$

It is possible to determine this set without actually computing A^m: for given i, the set $\{k | (i,k) \in P_m\}$ can be obtained without having to perform multiplications. Row i of L and U is calculated according to Algorithm 2.3, after the set $\{k | (i,k) \in P_m\}$ has been determined. Only those $(i,k) \in P_m$ for which $lu_{ik} \neq 0$ are added to the set P.

(ii) ILU(l)-decomposition.
This technique has been suggested by Gustafsson [17] and extensively tested by Langtangen [23]. An incomplete decomposition ILU(l) is constructed as follows. ILU(0) is the incomplete LU-decomposition with the same sparsity pattern as that of A. Suppose P_{l-1} gives the sparsity pattern of ILU($l-1$), and L and U are the lower- and upper-triangular matrices at level $l-1$. P_l is then defined by the sparsity pattern of the matrix product LU (this can be determined without multiplications).

(iii) ILU(ε)-decomposition.
This is the technique which is described in Section 2.3, in which the set P is chosen in such a way that one obtains a splitting $(LU, -R)$ in which $R = A - LU$ satisfies

$$|r_{ij}| < \varepsilon, \text{ for } 1 \leq i,j \leq N \qquad \text{and} \qquad r_{ij} = 0 \text{ for } (i,j) \in P \tag{3.6}$$

The techniques (i) and (ii) do not use the size of the elements of A. When A is not ill-conditioned, we can improve the preconditioner in the following way. First, the matrix is scaled in such a way that

$$\sum_{j=1}^{N} |a_{ij}| = 1, \text{ for } 1 \leq i \leq N \tag{3.7}$$

Then those pairs (i,j) for which $|a_{ij}|$ is small can be neglected during the construction of the set P as described under (i) or (ii). This idea appeared to be successful for the problem described in Section 3.2. However, neglecting elements which are small in absolute value can be dangerous when A is ill-conditioned, because in that case, neglecting such entries can strongly influence the spectrum of the matrix.

We have used the methods described under (i)-(iii) to construct an incomplete LU-decomposition for the coefficient matrix in (3.3). The dimension of this matrix is 4205, and the number of non-zero entries is 26542.

Table 3.2 shows the results obtained with a preconditioner which has the sparsity pattern of A^m. The second column shows how the average number of non-zero entries per row in $L + U$ varies with the parameter m. The third column shows how many iteration steps were needed to make $\|C^{-1}b - C^{-1}Ax^{(k)}\|_\infty$ smaller than $10^{-6}\|C^{-1}b\|_\infty$, and the fourth column shows an estimate of the number of flops per unknown necessary for Bi-CGSTAB. The work necessary for the construction of the preconditioner is not included in this number, because it appeared possible to use the same preconditioner during several time steps, and the CPU-time for the iteration process is therefore more important than the CPU-time needed for the incomplete decomposition. The last column gives the CPU-time needed for finding the non-zero pattern of L and U plus the time needed for the computation of the entries of L and U. Again these CPU-times are given in seconds and measured on both an Olivetti PC XP1 and a CONVEX C2. Note that no effort is made in this chapter to construct computer programs that vectorize on the latter. The matrix-vector multiplication, for example, does not vectorize because we use indirect addressing for the non-zero entries of A.

Table 3.2: Non-zero pattern of L and U is the same as that of A^m.

m	Number of entries/unknown	Number of iter.	Number of flops/unknown	Time for iteration		Time for ILU-decomposition	
1	6.3	9	620	35.4	1.86	3.6	0.24
2	11	6	530	30.3	1.69	18.6	1.29
3	26	5	730	41.9	2.59	56.4	3.99

The results of Table 3.2 show that the number of non-zero entries increases strongly when m increases from 2 to 3, but the number of Bi-CGSTAB iteration steps decreases only slightly. As a result, the number of flops/unknown necessary for the iteration process increases, and it is therefore not practical to choose a larger value for m than 2. The last column shows that the CPU-time needed for the construction of the factors L and U increases strongly with m.

Tables 3.3 and 3.4 show the corresponding results of ILU(l)- and ILU(ε)-decomposition, respectively. The second row of Table 3.3 shows similar results as the second

Table 3.3: ILU(l)-preconditioning.

l	Number of entries/unknown	Number of iter.	Number of flops/unknown	Time for iteration		Time for ILU-decomposition	
0	6.3	9	620	35.4	1.86	3.6	0.24
1	11	6	530	30.3	1.69	9.6	0.63
2	19	5	610	34.6	2.06	22.8	1.51

row of Table 3.2, although the CPU-time needed for the incomplete decomposition is less

Table 3.4: ILU(ε)-preconditioning.

ε	Number of entries/unknown	Number of iter.	Number of flops/unknown	Time for iteration		Time for ILU-decomposition	
0.1	4.5	9	550	31.9	1.60	2.7	0.14
0.05	6.1	6	410	23.8	1.25	3.4	0.19
0.01	8.4	5	390	22.7	1.24	5.1	0.29
0.005	11.4	3	280	16.4	0.93	7.7	0.44

than in Table 3.2. This is due to the fact that determining the sparsity pattern of A^m is relatively expensive. With ILU(l)-preconditioning, the number of flops per unknown increases when l increases from 1 to 2. Hence it is impractical to choose l larger than 1.

Table 3.4 shows that when ε decreases, the number of iteration steps reduces strongly, and although the number of non-zero entries in $L + U$ increases, the *total* amount of work necessary for preconditioned Bi-CGSTAB is lowered, which is demonstrated by the column that shows the number of flops per unknown. Comparing the CPU-times listed in Tables 3.3 and 3.2 with those of Table 3.4 leads to the conclusion that, for this type of problem, the use of a threshold parameter leads to the best preconditioners. This can be explained by the fact that the methods described under (*i*) and (*ii*) do not take the size of the elements of A into account.

From the difference between ICklm and MICklm shown in Table 3.1, it follows that the preconditioner can be improved by forcing the row sums of the residual matrix R to be zero. Therefore, Tables 3.5 and 3.6 give the results of MILU(l)- and MILU(ε)-preconditioning, respectively. Note that the sparsity pattern of L and U coming from a MILU(ε)-decomposition can be different from that of L and U coming from an ILU(ε)-decomposition. This is caused by the fact that the sparsity pattern is based on the absolute value of the entries in the residual matrix, and adding fill-in to the diagonal can influence the size of these entries.

Table 3.5: MILU(l)-preconditioning.

l	Number of entries/unknown	Number of iter.	Number of flops/unknown	Time for iteration		Time for ILU-decomposition	
0	6.3	7	480	27.6	1.48	3.9	0.29
1	11	4	360	20.8	1.13	10.9	0.72
2	19	4	500	28.6	1.56	27.1	1.74

From Table 3.5 we see again that it is impractical to choose the parameter l in the MILU(l)-decomposition larger than 1. The results of Table 3.6 again show that when ε decreases, the number of iteration steps reduces strongly, and the *total* amount of work necessary for preconditioned Bi-CGSTAB is lowered.

Table 3.6: MILU(ε)-preconditioning.

ε	Number of entries/unknown	Number of iter.	Number of flops/unknown	Time for iteration		Time for ILU-decomposition	
0.1	4.7	8	500	28.8	1.44	3.0	0.16
0.05	6.3	6	420	24.1	1.25	3.8	0.21
0.01	9.0	4	330	19.0	1.04	6.0	0.33
0.005	11.6	3	290	16.5	0.92	8.2	0.47

Comparing Tables 3.5 and 3.6 with Tables 3.3 and 3.4 respectively shows that the modification of the main diagonal can slightly decrease the number of iteration steps. However, the difference between the modified and unmodified decompositions is smaller than the difference between ICklm and MICklm in Table 3.1.

3.5 Renumbering and eliminating unknowns

In this section a renumbering of the unknowns is introduced in such a way that the matrix has the block structure

$$\begin{bmatrix} D & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{3.8}$$

where D is a diagonal matrix with dimension as large as possible. When the coefficient matrix arises from a standard seven-point discretization of a Poisson equation, one can use a checkerboard ordering to obtain a system of the form (3.8). In this section, we describe an algorithm that can be used in the more general case in which the sparsity pattern is irregular. Suppose *new* is a permutation vector which gives the new ordering which leads to the block structure (3.8). Given the non-zero structure of A, this permutation vector can be calculated fully automatically from Algorithm 3.1. Herein $fix1$ and $fix2$ are two boolean help arrays. These are initialized in such a way that $fix1[i] = fix2[i] = false$ for $1 \leq i \leq N$. The CPU-time necessary for the renumbering is very small as compared with the time needed for solving the system.

After the renumbering is performed, the vector x is split into two parts x_1 and x_2 in such a way that x_1 can be eliminated. The system $Ax = b$ can then be written as

$$x_1 = D^{-1}(b_1 - A_{12}x_2) \tag{3.9}$$

$$(A_{22} - A_{21}D^{-1}A_{12})x_2 = b_2 - A_{21}D^{-1}b_1 \tag{3.10}$$

Solving these equations only requires the iterative solution of (3.10), which has the advantage that during an iterative procedure like Bi-CGSTAB, the vectors which have to be stored in memory can be much shorter. Applying Algorithm 2.3 to the matrix (3.8) automatically gives an incomplete decomposition of the block $A_{22} - A_{21}D^{-1}A_{12}$ in the lower-right corner. Hence one can obtain a proper preconditioner for this block without actually computing the matrix product $A_{21}D^{-1}A_{12}$.

Algorithm 3.1. The renumbering necessary to obtain the block structure (3.8).

```
{First all components which are not directly related}
{to each other are numbered.}
$count := 0$;
FOR $k := 1$ TO $N$ DO
    BEGIN
      $check := fix1[k]$;
      FOR $v := begA[k]$ TO $begA[k+1]-1$ DO
            $check := check$ or $fix2[jcoA[v]]$;
      IF $not(check)$ THEN
         BEGIN
            $count := count + 1; new[k] := count; fix2[k] := true$;
            FOR $v := begA[k]$ TO $begA[k+1]-1$ DO $fix1[jcoA[v]] := true$
         END
   END; {$k$-loop}
   {The dimension of the diagonal block $D$ in (3.8) is given by $block$.}
$block := count$;
   {Now the rest of the components can be numbered.}
FOR $k := 1$ TO $N$ DO IF $not[fix2[k]]$ THEN
   BEGIN
      $count := count + 1; new[k] := count$
   END; {$k$-loop}
```

Table 3.7 shows the results which were obtained by applying this technique to the concrete cooling problem. It appeared that 2087 unknowns can be eliminated in advance, so that the dimension of the block $A_{22} - A_{21}D^{-1}A_{12}$ is 2118. The sparsity pattern of this block is much more irregular than the banded structure shown in Fig. 3.1. The last column shows the CPU-time required for the construction and incomplete decomposition of this block. The fifth column shows the time needed for the iteration process applied to (3.10) plus the time for the computation of x_1 from (3.9). Only the results of preconditioning techniques without Gustafsson's modification are reported here, because making the row sums of R equal to zero does not reduce the number of Bi-CGSTAB iteration steps. This is in agreement with the observations made by Langtangen [23], who concluded that MILU-decomposition can be inferior to ILU-decomposition for linear systems with complicated sparsity patterns.

Note that the number of rows is approximately halved, and since the number of entries per row is about the same as in Table 3.4, the storage requirements for the preconditioner are reduced considerably. The irregular sparsity pattern of the block $A_{22} - A_{21}D^{-1}A_{12}$ forms no problem for the construction of an ILU(ε)-decomposition as described in Chapter 2. From the results of Table 3.7 we conclude that decreasing ε reduces the number of Bi-CGSTAB iteration steps considerably. Further, elimination of unknowns as described

Table 3.7: Elimination of unknowns.

ε	Number of entries/unknown	Number of iter.	Number flops/unknown	Time for iteration		Time for ILU-decomposition	
0.05	6.4	6	290	18.0	1.10	4.5	0.25
0.01	9.4	3	180	11.0	0.68	5.9	0.34
0.005	10.2	3	180	11.3	0.70	6.3	0.36
0.001	16.2	2	160	9.8	0.61	10.0	0.57

above can be very successful for problems like the concrete cooling problem. From Table 3.7 we conclude that the choice of ε is not very critical for the total time required for both the decomposition and the iteration process.

Increasing the number of pipes.
To show that the methods also work well when the non-zero pattern is more irregular, we consider the situation in which the number of pipes in the concrete has increased strongly. We consider the block of concrete as shown in Fig. 3.2 with cooling pipes in the planes $y = 0.2$, $y = 0.4$ and $y = 0.6$. These cooling pipes are folded 34 times. Again the plane $y = 0.4$ is a symmetry plane. Hence we only have to consider the part $0 \le y \le 0.4$. The discretization in the $x-$ and $z-$direction is the same as in Section 3.2, but in the $y-$direction there are 8 intervals with the mesh size varying as

After discretization we obtain a system of linear equations in which the coefficient matrix has the same block structure as in (3.3), but now the dimension of A_{11} is 4680, and the dimension of A_{22} is 350. Hence the non-symmetric part of A with an irregular non-zero structure is much larger than in the original problem. As a consequence, the special-purpose method of Section 3.3 is not practical in this case, because the computation of the block $A_{22} - A_{21}L^{-T}L^{-1}A_{12}$ would become too expensive.

We observed that Bi-CGSTAB does not converge when only diagonal scaling is used as a preconditioner. We also used the technique of renumbering and eliminating unknowns as described above. It appeared that 2384 unknowns can be eliminated, so that the dimension of the coefficient matrix in (3.10) is 2646. The results are summarised in Table 3.8. It appears that with a proper choice of ε, the number of iteration steps is very small, even when the number of non-zero entries in $L+U$ is not very large. Again we see that decreasing ε leads to an increase of fill-in in the factors L and U, but the number of iteration steps decreases. The CPU-times for the iteration process show that, as long as ε is not chosen larger than 0.01, the choice of this parameter is not very critical for the total amount of work required for the iteration process.

Table 3.8: Elimination of unknowns. The number of pipes has increased.

ε	Number of entries/unknown	Number of iter.	Number of flops/unknown	Time for iteration		Time for ILU-decomposition	
0.1	3.9	15	640	46.3	2.75	4.5	0.27
0.05	6.2	10	480	34.7	2.12	5.4	0.34
0.01	9.9	6	340	24.7	1.53	7.7	0.48
0.005	11.0	5	300	21.8	1.36	8.5	0.54
0.001	17.6	4	300	21.7	1.38	13.8	0.89
0.0005	22.0	3	260	19.0	1.23	18.6	1.22

3.6 Conclusions

The concrete cooling problem described in Section 3.2 can be solved efficiently by conjugate gradient-like methods such as Bi-CGSTAB. The excellent preconditioners that can be constructed for this type of problem can optimally exploit the banded structure and symmetry of the largest part of the matrix. The non-symmetric part of A with an irregular sparsity pattern can be treated very efficiently by the block decomposition (3.5).

In Section 3.4 a number of methods has been described to construct preconditioners for non-symmetric M-matrices with a very irregular sparsity pattern, which allows complicated geometries and an irregular node numbering. Three different methods for constructing the sparsity pattern of $L+U$ have been described and compared with each other. Incomplete LU-decompositions in which this sparsity pattern is based on a threshold parameter give the best preconditioners, and the CPU-time required for the construction of the factors L and U is relatively small, not more than the time for one or two iteration steps with Bi-CGSTAB. It should be emphasized that no effort is made in this chapter to construct computer programs that vectorize.

Elimination of unknowns as described in Section 3.5 can be very successful for problems like the concrete cooling problem. With this technique, both the storage requirements and the CPU-time needed for solving the linear systems can be decreased.

æ

4. Vectorizable preconditioning techniques with application to the Boussinesq equations

4.1 Introduction

Recently, a new numerical model for the computation of fairly low and fairly long waves has been developed with the support of the Dutch Foundation for Engineering Sciences (STW). The model is used, among others, by Delft Hydraulics for wave computations in harbours and coastal regions [27]. In this chapter, we consider some possibilities for improving the efficiency of the method on vector and parallel computers. In [27] it is shown that the propagation of fairly low and fairly long waves can be described by the Boussinesq equations, which are written down below:

$$\begin{aligned}
\frac{\partial \zeta}{\partial t} &= -\frac{\partial}{\partial x}[G^T(h+\zeta)G(u)] - \frac{\partial}{\partial y}[G^T(h+\zeta)G(v)] \\
\frac{\partial u}{\partial t} &= -G(u)\frac{\partial}{\partial x}G(u) - G(v)\frac{\partial}{\partial y}G(u) - g\frac{\partial \zeta}{\partial x} \\
\frac{\partial v}{\partial t} &= -G(u)\frac{\partial}{\partial x}G(v) - G(v)\frac{\partial}{\partial y}G(v) - g\frac{\partial \zeta}{\partial y}
\end{aligned} \tag{4.1}$$

The variables u and v are the velocities in the $x-$ and $y-$direction, respectively, ζ is the water elevation, and h defines the bottom profile. The differential operator G is of the form $G = L_{1.9}^{-1}L_{0.9}$, where L_γ ($\gamma = 0.9$ or 1.9) is defined by:

$$\begin{aligned}
L_\gamma(u) = u \quad &- \quad \frac{1}{4}\gamma(h\frac{\partial^2}{\partial x^2}(hu) + h\frac{\partial^2}{\partial y^2}(hu)) + \frac{1}{12}\gamma(h^2\frac{\partial^2 u}{\partial x^2} + h^2\frac{\partial^2 u}{\partial y^2}) \\
&+ \quad \frac{1}{6}\gamma(h\frac{\partial h}{\partial x}\frac{\partial u}{\partial x} + h\frac{\partial h}{\partial y}\frac{\partial u}{\partial y})
\end{aligned} \tag{4.2}$$

The Boussinesq equations are a refinement of the shallow-water equations which arise if $G = I$. For the time discretization, the classical Runge-Kutta method is used, and for the space discretization fourth-order accurate finite differences. The application of G to u or v in the right-hand side requires the solution of a Helmholtz-type equation. Its solution forms the bottleneck of the numerical model when a vector computer, in our case a NEC SX-3, is used. Hence, we want to apply a suitable vectorizable algorithm for solving this system of linear equations, which we write in the general form

$$Ax = b \tag{4.3}$$

Here A is a large sparse non-singular matrix of order N, and b a given vector. Due to the fourth-order accurate finite differences, A can have nine non-zero elements per row. The coefficient matrix is constant in time, and at every time step a reasonable estimate $x^{(0)}$ of the solution is available. We want to solve the linear systems with a conjugate gradient-like method. To improve the rate of convergence, we replace (4.3) by the preconditioned system

$$C^{-1}Ax = C^{-1}b \tag{4.4}$$

The preconditioner C should again be such that, for given y, the calculation of the product $C^{-1}y$ does not require much CPU-time or computer storage. Since the coefficient matrix is constant in time, we are in the pleasant situation that the preconditioner needs to be constructed only once.

On parallel and vector computers, the bottleneck of an efficient implementation is often formed by the application of the preconditioner. In [12] and [37] an overview is given of several methods to implement iterative methods for solving (4.4) on supercomputers.

Most choices of C are based on an incomplete LU-decomposition of the coefficient matrix as described in Chapter 1. In order to obtain the product $z = C^{-1}y$ for given y, one has to solve the systems

$$Lx = y \tag{4.5}$$
$$Uz = x \tag{4.6}$$

successively. This forms the main difficulty in vectorizing or parallelizing iterative methods preconditioned by incomplete LU-decompositions, because algorithms for solving (4.5) and (4.6) are highly sequential by nature. In Section 4.2, we will consider some possibilities for implementation of these algorithms on supercomputers.

In many cases, the coefficient matrix has a diagonal structure. Hence the matrix-vector multiplication can be performed very efficiently on supercomputers. This motivates the so-called polynomial preconditioning. In this technique, a polynomial P_n of degree n is chosen in such a way that the preconditioned matrix $P_n(A)A$ has a more favourable eigenvalue distribution than A. In Section 4.3, some choices of the polynomial P_n will be described, which can be used for the systems of linear equations arising in the Boussinesq model. In Section 4.4, some results of numerical experiments with these techniques will be given. Section 4.5 describes an efficient implementation of polynomial preconditioning, which is especially useful for the linear systems arising in the Boussinesq model.

4.2 Incomplete LU-decompositions

Since algorithms for solving (4.5) and (4.6) both lead to the same vectorization and parallelization problems, we only consider algorithms for solving the lower-triangular system $Lx = y$.

In the method with which we started, the sparsity pattern of the factors L and U is based on a drop tolerance as described in Section 2.3. The sparsity pattern of L can be very irregular. Hence no restrictions are made with respect to the sparsity pattern, and we assume that L is stored row-wise in 3 arrays as described in Section 2.2. When the diagonal entries of L are equal to 1, the calculation of x from $Lx = y$ can be implemented as shown in Algorithm 4.1. A drawback of this technique is the indirect addressing, which

Algorithm 4.1. Sequential sweep for solving $Lx = y$.

```
FOR $i := 1$ TO $N$ DO
  BEGIN
    $h := y[i]$;
    FOR $v := begL[i]$ TO $begL[i+1]-1$ DO
        $h := h - coL[v] \times x[jcoL[v]]$;
    $x[i] := h$
  END; $\{i\text{–loop}\}$
```

forms a difficulty for vectorizing this algorithm. A second disadvantage is that the inner loop can be regarded as the calculation of an inner product with length $begL[i+1]-begL[i]$. For most vector computers this length is much too small to perform this calculation at an acceptable speed. In [12] and [37] several techniques have been proposed to get rid of the sequential nature of algorithms for solving (4.5).

Renumbering strategies.
One can select all unknowns that have no direct relationship with each other and number them first. This can be repeated for the remaining unknowns. After this renumbering, the coefficient matrix has the block structure

$$\begin{bmatrix} D & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where D is a diagonal matrix, and the block A_{22} has again a similar block structure. When the sparsity pattern of $L+U$ is chosen the same as that of the coefficient matrix, algorithms to solve (4.5) and (4.6) can be parallelized by exploiting this block structure. For the matrix arising after a standard five-point discretization of the Laplace equation on a rectangular grid, the strategy described above leads to the well-known red-black ordering. For more complicated geometries one can use more colours to decouple the unknowns in groups which are as large as possible. A disadvantage of this kind of reordering techniques is that they can increase the number of iteration steps required to solve the system of linear equations [13].

Partial vectorization.
Another way to get rid of the recursion in algorithms for solving (4.5) is to use a technique called partial vectorization. When solving $Lx = y$, the vectors x and y are partitioned as $(X_1, X_2, \ldots, X_m)$ and $(Y_1, Y_2, \ldots, Y_m)$ respectively, where X_i and Y_i represent groups of components. Suppose that, according to this partitioning, L has the block structure as shown in Fig. 4.1. The vector x can then be calculated in the following way:

$$X_1 = L_1^{-1} Y_1;$$

Figure 4.1: Block structure of L with partial vectorization.

```
FOR $i := 2$ TO $m$ DO
    $X_i = L_i^{-1}(Y_i - E_i(X_1, \dots, X_{i-1})^T)$;
```

The matrix multiplication with E_i can often be implemented efficiently on supercomputers. If the block size is small, the inverse of the blocks L_i can be calculated, thus the calculation of X_i can also be performed efficiently. A drawback of this technique is that for increasing block size, the calculation of the inverse of the blocks on the diagonal becomes time consuming. We therefore consider the following approach.

Level-scheduling.
In this technique, one looks for groups of unknowns which can be solved simultaneously. The first group consists of those components $x[j]$ which are independent of all other components, and the i-th group contains all unknowns which depend only on components from the first $i-1$ groups. After reordering the unknowns group by group, the matrix L has a block structure as shown in Fig. 4.1, where all the blocks L_i are diagonal blocks. A difference with the technique of reordering the unknowns, as described above, is that with level-scheduling the reordering is performed *implicitly*: only the order of the computations is changed, so that the results are the same as without level-scheduling.

Suppose that *nlev* is the number of groups, and *new* is an integer array which defines the group by group reordering. The integer $level[i]$, $i = 1 \dots nlev + 1$, points to the beginning of the i-th group in *new*. Solving the lower-triangular system can be implemented as shown in Algorithm 4.2. The integer *nlev* and the arrays *level* and *new* can be obtained

Algorithm 4.2. Solving $Lx = y$ using level-scheduling.

```
FOR $j := 1$ TO $nlev$ DO
   BEGIN
      FOR $i := level[j]$ TO $level[j+1] - 1$ DO
      BEGIN
         $k := new[i]$;
         $h := y[k]$;
         FOR $v := begL[k]$ TO $begL[k+1] - 1$ DO
                  $h := h - coL[v] \times x[jcoL[v]]$;
         $x[k] := h$
      END $\{i$–loop$\}$
   END; $\{j$–loop$\}$
```

fully automatically from the arrays *coL*, *jcoL* and *begL* using Algorithm 4.3. The integer arrays *freq* and *group* are two locally defined help arrays. Herein $group[i]$ gives the group number i of every component $x[i]$, and $freq$ is necessary to store the number of components contained in every group.

Algorithm 4.3. Calculation of arrays necessary for level-scheduling.

```
FOR $i := 1$ TO $N$ DO $freq[i] := 0$;
FOR $i := 1$ TO $N$ DO
  BEGIN
    $m_1 := 0$;
    FOR $v := begL[i]$ TO $begL[i+1]-2$ DO
          $m_1 :=\mathrm{MAX}(m_1, group[jcoL[v]])$;
    $group[i] := 1 + m_1$;
    $nlev :=\mathrm{MAX}(nlev, 1 + m_1)$;
    $freq[1 + m_1] := freq[1 + m_1] + 1$
  END; $\{i\text{−loop}\}$
$level[1] := 1$;
FOR $i := 1$ TO $nlev$ DO $level[i + 1] := level[i] + freq[i]$;
FOR $i := 1$ TO $nlev$ DO $freq[i] := 0$;
FOR $i := 1$ TO $N$ DO
  BEGIN
    $new[level[group[i]] + freq[group[i]]] := i$;
    $freq[group[i]] := freq[group[i]] + 1$
  END; $\{i\text{−loop}\}$
```

Since there are no recurrence relations within the i-loop of Algorithm 4.2, level-scheduling can be very successful on parallel architectures, especially when the block sizes are large. However, for an efficient implementation on vector computers, it is necessary to use another implementation of the i-loop in order to increase the vector length. In many practical problems, (almost) all the non-zero elements of the coefficient matrix A can be stored in a few diagonals, thus the matrix-vector multiplication can be performed at vector computers with near-peak performance. If one chooses the same non-zero structure for $L + U$, the i-loop of Algorithm 4.2 can be implemented more efficiently by using the diagonal structure of L. This choice for the sparsity pattern of $L+U$ leads to the so-called standard incomplete decomposition of A. In many situations, this means that L and U are equal to the strictly lower- and upper-triangular part of A, respectively, thus it is possible to use the efficient Eisenstat implementation as described in Section 1.4. In the linear systems arising from the Boussinesq model, this is not the case, which is due to the fill-in generated during the incomplete decomposition. However, one obtains a proper incomplete decomposition when only corrections on the diagonal and on the sub- and the super-diagonal are performed. Therefore, only 3 vectors of length N are needed to store the elements of L and U. To implement level-scheduling, in addition, we need two integer arrays of length N.

The technique described above is similar to the technique described in [33], which uses slightly overlapping blocks along the main diagonal.

4.3 Polynomial-like preconditioning

In this section various preconditioning techniques are described which only require matrix-vector multiplications, instead of solving lower- and upper-triangular systems.

Polynomial preconditioning.
The idea of polynomial preconditioning for solving $Ax = b$ is to consider the equivalent system $P_n(A)Ax = P_n(A)b$, where P_n is a polynomial of degree n such that the preconditioned matrix $P_n(A)A$ has a more favourable eigenvalue distribution than A. This has the advantage that only matrix-vector multiplications are necessary to implement the preconditioner. As mentioned earlier, this can be implemented very efficiently on supercomputers if (almost) all the non-zero elements of A can be stored in a few diagonals. To be efficient, the polynomial P_n should be chosen in such a way that the preconditioned matrix resembles the identity matrix in some sense. Suppose that all the eigenvalues of A are located in a certain domain $\mathcal{D}$ in the complex plane. Then a possible choice for P_n is the polynomial that minimizes

$$\max_{z\in\mathcal{D}} \|1 - zP_n(z)\| \tag{4.7}$$

A possible choice for the region $\mathcal{D}$ is a polygon that contains all the eigenvalues of A. When GMRES(M) is used as an iterative method for solving $Ax = b$, eigenvalue estimates can be obtained from the Hessenberg matrix which is generated during the iteration process. Details of this approach can be found in [37].

The Boussinesq model leads to systems of linear equations in which the coefficient matrix is diagonally dominant and almost symmetric. Therefore, we assume that after diagonal scaling, all eigenvalues lie in the interval $[0, 2]$. Minimizing (4.7) in this case leads to choosing the residual polynomial $1-\lambda P_n(\lambda)$ equal to a Chebychev polynomial. Another technique, which is described in [36], is to choose the so-called least squares polynomials on the interval $[0, 2]$. This approach has the advantage that it does not require accurate estimates of the eigenvalues of the coefficient matrix, and it is very easy to implement. The first five least squares polynomials for the interval $[0, 2]$ are given below [36]:

$$\begin{aligned}
&P_1(z) = 2.5 - z\\
&P_2(z) = 3.5 - 3.5z + z^2\\
&P_3(z) = 3.75 - 6.75z + 9z^2 - z^3\\
&P_4(z) = 55/16 - (77/8)z + 11z^2 - 5.5z^3 + z^4\\
&P_5(z) = 91/32 - (91/8)z + 19.5z^2 - (65/4)z^3 + 6.5z^4 - z^5
\end{aligned}$$

A major drawback of polynomial preconditioning is that on sequential machines it is outperformed by preconditioning techniques which are based on incomplete decompositions. Polynomial preconditioning is therefore only of interest for vector computers, where the matrix-vector multiplications can be performed at near-peak performance.

Smoothing matrices as preconditioners.

In this technique, the preconditioner C is chosen as $(P_k(D_x)P_k(D_y))^{-1}$, where P_k is a polynomial of degree k satisfying $P_k(0) = I$, and D_x and D_y are smoothing matrices with respect to x- and y-direction, respectively [41]. For example, D_x is defined by

$$D_x = \frac{1}{4}\begin{bmatrix} 2 & 1 & & & 0 \\ 1 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 2 & 1 \\ 0 & & & 1 & 2 \end{bmatrix}$$

and D_y is defined in a similar way. These matrices are to a large extent independent of the problem, and of a very simple form. Therefore, this preconditioning technique is very easy to implement and highly vectorizable.

It is also possible to use the smoothed Jacobi-method [10] as a preconditioner. In this technique, the preconditioner C is defined implicitly as follows. Suppose that $z^{(q)}$ is the solution of $Cz^{(q)} = d$. This vector is calculated from the following scheme:

$$\begin{aligned} z^{(0)} &= d \\ z^{(k+1)} &= z^{(k)} + \omega_k S_k(d - Az^{(k)}),\ k = 0,\dots,q-1 \end{aligned} \tag{4.8}$$

where $z^{(k)}$ is the $k-$th iterate, ω_k a relaxation parameter, and S_k is equal to $D_y^k D_x^k$. We have chosen ω_k independent of the $x-$ and $y-$coordinates. In [10] it is shown that a proper choice for this parameter is

$$\omega_k = \frac{2\rho(A)}{\rho(S_k A)}$$

4.4 Numerical experiments

In this section, we compare the various vectorizable preconditioning techniques for solving the linear systems arising in the Boussinesq model. These techniques are tested on an HP-720 workstation, a CONVEX C2 and on a NEC SX-3. On the latter two machines we used only one processor. The calculations are performed during four time steps in a fully developed wave field. During each evaluation of the right-hand side of (4.1), the equation

$$L_{1.9}B = L_{0.9}f \tag{4.9}$$

has to be solved four times, where $L_{1.9}$ and $L_{0.9}$ are defined in (4.2). Since a fourth-order Runge-Kutta method is used, such an equation has to be solved 16 times at each time step. Because two different orderings are used, the model works with the two coefficient matrices. To be precise, at each time step the following linear systems have to be solved:

$$A_u x = v_i,\ i = 1\dots 8 \tag{4.10}$$

and

$$A_v x = w_j,\ j = 1\dots 8 \tag{4.11}$$

where A_u and A_v are matrices which are constant in time. Discretizing (4.9), using a fourth-order accurate finite-difference scheme, leads to systems of linear equations $Ax = b$ in which the coefficient matrix has positive eigenvalues. The order of A is 32924, and there can be nine non-zero elements in one row. We use GMRES(M) as an iterative method, because it appears that with a proper preconditioner, GMRES(M) does not have to be restarted very often. Hence the number of matrix-vector products with the preconditioned matrix is minimized. On a supercomputer like the NEC SX-3, these form the bottleneck in solving the systems of linear equations. In order to minimize the effect of rounding errors in the orthogonalization process, we take a small value of M: $M = 10$. The following stopping criterion is used:

$$\|C^{-1}(b - Ax)\|_2 \leq 10^{-6}$$

We will first consider the matrix-vector multiplication, because its implementation is crucial for the efficiency of polynomial preconditioning. Next, we will present some results of incomplete LU-decompositions, both with and without level-scheduling, and of polynomial preconditioning. Finally, some results of smoothing techniques will be presented.

Implementation of the matrix-vector multiplication.
In our test case, the region is rectangular, and discretizing (4.1) leads to a coefficient matrix in which all non-zero elements are contained in nine diagonals. However, we want to be able to work with a more irregular domain in which the number of unknowns in the y-direction varies. In that case, indirect addressing is necessary for the matrix elements which come from the discretization in this direction. Therefore, each matrix-vector multiplication needs a gather and a scatter operation. If the diagonal is scaled to unity, it can be implemented as shown in Algorithm 4.4. Herein *indyx* is a permutation vector,

Algorithm 4.4. Matrix-vector multiplication $r := Ax$, version 1.
```
FOR $i := 1$ TO $N$ DO $px[indyx[i]] := x[i]$;
FOR $i := 1$ TO $N$ DO
   BEGIN
      $j := indyx[i]$;
      $r[i] := AMY[i] * x[i-2] + AMX[j] * px[j-2] +$
               $BMY[i] * x[i-1] + BMX[j] * px[j-1] + x[i] +$
               $DMY[i] * x[i+1] + DMX[j] * px[j+1] +$
               $EMY[i] * x[i+2] + EMX[j] * px[j+2]$
   END; $\{i\text{–loop}\}$
```

which can be used to number a vector from $y-$ to $x-$direction.

In Algorithm 4.4 the calculations may be slowed down by the presence of an indirect addressing in the loop with the multiplications. It may therefore be better to implement the algorithm as shown in Algorithm 4.5. In this implementation, the part with the multiplications can be performed at full vector speed, although one gather operation is required after r and r_2 have been calculated. The CPU-times for both implementations are listed in Table 4.1. The second implementation of the matrix-vector multiplication

Algorithm 4.5. Matrix-vector multiplication $r := Ax$, version 2.

```
FOR $i := 1$ TO $N$ DO $px[indyx[i]] := x[i]$;
FOR $i := 1$ TO $N$ DO
  BEGIN
    $r[i] := AMY[i] * x[i-2] + BMY[i] * x[i-1] +$
        $DMY[i] * x[i+1] + EMY[i] * x[i+2]$;
    $r_2[i] := AMX[i] * px[i-2] + BMX[i] * px[i-1] +$
        $DMX[i] * px[i+1] + EMX[i] * px[i+2]$
  END; $\{i$–loop$\}$
FOR $i := 1$ TO $N$ DO $r[i] := x[i] + r[i] + r_2[indyx[i]]$;
```

Table 4.1: CPU-times for the matrix-vector multiplication.

	Alg. 4.4	Alg. 4.5
HP-720	0.135	0.105
CONVEX	0.069	0.057
NEC SX-3	0.0022	0.0011

appears to be more efficient than the first one on all computers considered here. Especially on the NEC SX-3 this difference is significant: with the second implementation, it appears possible to gain approximately a factor two in speed. The matrix-vector multiplication is performed at a speed of approximately 500 Mflop/sec. Without the indirect addressing this can be more than three times faster. Hence in many situations, it is advisable to introduce 'dummy' equations in such a way that the domain under consideration becomes a rectangle, thus avoiding the need of indirect addressing.

MILU-preconditioning.
We have solved (4.10) and (4.11) with GMRES(10) combined with MILU(ε)-preconditioning. With this preconditioning technique, the storage requirements are relatively high. Therefore, the factors L and U are stored in single precision. From the numerical results it appeared that this has a negative effect on GMRES(10). When the factors L and U coming from a standard incomplete decomposition are stored in double instead of single precision, the average number of GMRES iteration steps decreases from 11.4 to 9.7. Therefore, when a standard incomplete decomposition is used, the preconditioner is stored in double precision. This does not require a large amount of storage, because only three vectors of length N have to be stored (see Section 4.2).

The results of GMRES(10) combined with several incomplete decompositions are summarised in Table 4.2. The first row shows the average number of matrix-vector products $y := (LU)^{-1}Ax$ required for solving one linear system. This number has been obtained during four time steps in a fully developed wave field. The second row shows the maximum number of entries in $L+U$ divided by the number of unknowns. The last three rows show the average CPU-times required for solving x from $Ax = b$. These times have been measured in seconds. MILU(ε)-preconditioning has not been combined with level-scheduling, because all elements in L and U have to be addressed indirectly. Hence it is not expected that this technique leads to a strong reduction in the CPU-time. For comparison, the last two columns show the performance of a standard incomplete decomposition, in which the sparsity pattern of $L + U$ is equal to that of A, both without and with level-scheduling.

Table 4.2: GMRES(10) combined with MILU(ε)- and standard preconditioning.

ε	0.05	0.02	0.01	0.005	standard, no l.s.	standard, with l.s.
# mat-vec. multiplications	11.1	8.3	7.0	6.5	9.7	9.7
# $NZ(L+U)$ per unknown	4.9	6.8	8.7	10.7	8.7	8.7
Times $x := A^{-1}b$, HP-720	4.2	3.3	2.9	2.9	3.9	4.1
Times $x := A^{-1}b$, CONVEX	5.1	4.7	4.7	5.0	6.4	2.1
Times $x := A^{-1}b$, NEC SX-3	0.47	0.43	0.44	0.47	0.65	0.077

The MILU(ε)-preconditioning very effectively reduces the number of matrix-vector multiplications required for solving the linear systems. If we choose ε equal to 0.05, the matrix $L + U$ is much more sparse than the coefficient matrix, and even then the average number of matrix-vector multiplications necessary for solving one linear system is relatively small. Without preconditioning this number is almost 4 times larger (see the first column of Table 4.3).

It is interesting to see that with MILU(ε), the average CPU-time for one solution process remains approximately constant. If ε is chosen smaller, the number of iteration steps decreases, but the application of the preconditioner requires more CPU-time, because the number of non-zero entries in L and U increases.

As can be expected, the performance of MILU(ε) as a preconditioner is poor on the two vector computers. The difference between the last two columns of Table 4.2 shows that the performance of a standard incomplete decomposition can be improved strongly by using level-scheduling.

If we use MILU(ε)-preconditioning on the NEC SX-3 without level-scheduling, it appears that the speed at which (4.5) and (4.6) are solved is not more than 11 Mflop/sec. When level-scheduling is used, this speed can be increased to 100 Mflop/sec. Although this is a major improvement, it is still slow compared with the 500 Mflop/sec at which the matrix-vector multiplication is performed. This disappointing result can be explained by the indirect addressing in Algorithm 4.2.

Polynomial preconditioning.
Table 4.3 shows the results of GMRES(10) combined with a polynomial preconditioner $P_n(A)$ as described in Section 4.3. The average number of GMRES iteration steps required for solving one system of linear equations is listed in the first row. The second row gives the number of matrix-vector multiplications with A. The last three rows give the

average CPU-times measured in seconds for solving x from $Ax = b$. The diagonal of the coefficient matrix is scaled to unity, thus when $n = 0$, only diagonal scaling is used.

Table 4.3: Polynomial preconditioning.

n	0	1	2	3	4	5
# mat-vec. mult. $P_n(A)A$	42.3	24.5	17.2	13.1	10.3	8.3
# mat-vec. mult. A	42.3	49.0	51.7	52.6	51.3	49.9
Times $x := A^{-1}b$, HP-720	11.4	9.5	8.9	8.5	8.0	7.2
Times $x := A^{-1}b$, CONVEX	5.3	4.5	4.4	4.0	3.9	3.3
Times $x := A^{-1}b$, NEC SX-3	0.10	0.085	0.079	0.073	0.073	0.063

From the difference between the last column of Table 4.2 and 4.3 we conclude that on the NEC SX-3, the preconditioning with P_5 is slightly more effective than a standard MILU-decomposition combined with level-scheduling. Polynomial preconditioning does not reduce the number of matrix-vector multiplications with A, but it considerably reduces the amount of work necessary for the orthogonalization process of GMRES.

Smoothing as a preconditioning method.
We have tried to use the smoothing matrix $C = (P_k(D_x)P_k(D_y))^{-1}$ described in Section 4.3 as a preconditioner for GMRES. Unfortunately, the smoothing matrices did not reduce the number of iteration steps very effectively. This is probably due to the fact that the diagonal dominance of the coefficient matrices is rather strong, so that the effect of extra smoothing is not very large. For $k = 1$, it appeared possible to almost halve the number of iteration steps, but this is not better than using a polynomial preconditioner with degree 1. For $k > 1$, it is possible to implement the smoothing matrix without having to use more gather and scatter operations, but it appeared not possible to obtain a further reduction in the number of iteration steps.

We have also used the smoothed Jacobi-method [10] as a preconditioner. The second column of Table 4.4 gives the results of this technique for $q = 1$ and $\omega_0 = 4$ in (4.9). The smoothing technique has not been applied for higher values of q, because this requires a lot of programming effort, and it is not expected that this technique reduces the number of iteration steps strongly. Again, the average CPU-time necessary for solving one linear system is given. For comparison, the first column ($q = 0$) shows the results of only diagonal scaling. The last column shows the results of smoothing in only one direction,

Table 4.4: The smoothed Jacobi-method on the NEC SX-3.

	$q = 0$	$q = 1$	$C = D_x^{-1}$
# matrix-v. mult. with $C^{-1}A$	42.3	17.6	33.3
CPU-time $x := A^{-1}b$	0.10	0.094	0.083

e.g. C is equal to D_x^{-1}. This appears to be a proper choice, because no gather and scatter operations are needed, and the average number of matrix-vector multiplications reduces with more than 20 percent as compared with diagonal scaling. However, this technique is not competitive in comparison with polynomial preconditioning. The use of Jacobi-smoothing does not reduce the CPU-times as compared with diagonal scaling. This is probably due to the fact that the choice of the relaxation parameters is not optimal. In [10] it is reported that when an irregular bottom topography is used, much better results are obtained with a relaxation parameter which depends on the local depth.

4.5 An efficient implementation of polynomial preconditioners

From the results of the previous section, it appears that polynomial preconditioning is competitive in comparison with preconditioning techniques based on incomplete decompositions, especially on the NEC SX-3. This can be explained by the fact that the NEC SX-3 needs rather long vectors to achieve a reasonable speed, and the performance is relatively poor when indirect addressing is involved. In this section, we describe an efficient implementation of polynomial preconditioning, which is especially useful for the linear systems arising in the Boussinesq model. We will focus on the following aspects:

reduction of the number of gather and scatter operations, and of the number of floating point operations.

Reduction of the number of gather and scatter operations.
For an efficient implementation of polynomial preconditioners on the NEC SX-3, it is very important to minimize the negative effects of indirect addressing. With a straightforward implementation, every matrix-vector multiplication needs one gather and one scatter operation. It is possible to halve the number of these operations. We illustrate this with a polynomial of degree 3. The matrix A can be written as $I + D_1 + D_2$, in such a way that in Algorithm 4.5 the vectors r and r_2 are equal to D_1x and D_2x, respectively. In the following, we assume that the matrices D_1 and D_2 commute. Since these matrices come from a discretization of a second derivative, this is a reasonable assumption. With this assumption we obtain for the polynomial P_3 from Section 4.3

$$\begin{aligned}P_3(A) &= P_3(I + D_1 + D_2),\\ &= 0.5I - 0.75(D_1 + D_2) + 1.5(D_1^2 + D_2^2) - (D_1^3 + D_2^3)+\\ &\quad 3(D_1D_2 - D_1^2D_2 - D_1D_2^2),\\ &= 0.5I - 0.75D_2 + 1.5D_2^2 - D_2^3+\\ &\quad D_1[-0.75I + 3(D_2 - D_2^2) + D_1(1.5I - 3D_2 - D_1)].\end{aligned}$$

The grouping of the terms is done in such a way that the calculation of $r = P_n(A)x$ can be implemented efficiently as follows:

1. create a copy of x numbered in y-direction: $px[indyx[j]] = x[j]$, $j = 1 \dots N$,
2. compute $x_1 = D_2px$, $x_2 = D_2^2px$ and $x_3 = D_2^3px$ successively,
3. number the results in x-direction: $y_i[j] = x_i[indyx[j]]$, $j = 1 \dots N$, $i = 1 \dots 3$,
4. $t_1 = D_1x$,
5. $t_2 = D_1(1.5x - 3y_1 - t_1)$,
6. $t_3 = D_1(-0.75x + 3(y_1 - y_2) + t_2)$,
7. $r = 0.5x - 0.75y_1 + 1.5y_2 - y_3 + t_3$

Applying the polynomial $P_n(A)$ once only requires one scatter operation and n gather operations, instead of the needed n gather and n scatter operations for a straightforward application of $P_n(A)$. Below we give a list of the polynomials P_n for $n = 2, \dots, 6$, written in such a form that the approach described above can be followed:

$$\begin{aligned}P_2(A) &= I - 1.5D_2 + D_2^2 + D_1(-1.5I + 2D_2 + D_1)\\ P_3(A) &= 0.5I - 0.75D_2 + 1.5D_2^2 - D_2^3+\\ &\quad D_1(-0.75I + 3(D_2 - D_2^2) + D_1(1.5I - 3D_2 - D_1))\\ P_4(A) &= 0.3125I - 0.125D_2 + 0.5D_2^2 - 1.5D_2^3 + D_2^4+\\ &\quad D_1(-0.125I + D_2 - 4.5D_2^2 + 4D_2^3+\\ &\quad D_1(0.5I - 4.5D_2 + 6D_2^2+\\ &\quad D_1(-1.5I + 4D_2 + D_1)))\end{aligned}$$

$$\begin{aligned}
P_5(A) = {} & 0.21875I - 0.125D_2 - 0.25(D_2^2 + D_2^3) + 1.5D_2^4 - D_2^5 + \\
& D_1(-0.125I - 0.5D_2 - 0.75D_2^2 + 6D_2^3 - 5D_4 + \\
& D_1(-0.25I - 0.75D_2 + 9D_2^2 - 10D_2^3 + \\
& D_1(-0.25I + 6D_2 - 10D_2^2 + \\
& D_1(1.5I - 5D_2 - D_1)))) \\
P_6(A) = {} & 0.125I - 0.1875D_2 + 0.625D_2^3 - 1.5D_2^5 + D_2^6 + \\
& D_1(-0.1875I + 1.875D_2^2 - 7.5D_2^4 + 6D_2^5 + \\
& D_1(-15(D_2^3 + D_2^4) + 1.875D_2 + \\
& D_1(0.625I - 15D_2^2 + 20D_2^3 + \\
& D_1(-7.5I + 15D_2^2 + \\
& D_1(-1.5I + 6D_2 + D_1))))).
\end{aligned}$$

The polynomial P_6 is added, because after evaluating $P_6(I + D_1 + D_2)$, it appears that some terms cancel, and the polynomial is of a simpler form than one would expect, so that the corresponding preconditioner can be implemented more efficiently. For large values of n, one can obtain $P_n(A)$ easily with a symbolic manipulation package like Mathematica or Maple.

Reduction of the number of floating point operations.
So far, the polynomial preconditioning has been implemented in such a way that an iterative method is used for solving the system of linear equations $P_n(A)Ax = P_n(A)b$, where $P_n(A)$ is an approximation of A^{-1}. We will now consider the linear system $P_n(B)Ax = P_n(B)b$, in which B is a matrix with the following properties:

1. Bx is approximately equal to Ax for all $x \in R^N$.

2. The calculation of Bx can be implemented more efficiently than the matrix-vector multiplication with A.

The coefficient matrix stems from a *fourth*-order accurate finite-difference scheme of a Poisson-like equation, thus A can have nine non-zero elements per row. The idea in choosing a matrix B with the two properties mentioned above is to use a standard, *second*-order accurate discretization of the Poisson equation. This choice leads to a matrix with not more than five non-zero elements per row. One can show that with this choice of B, the calculation of $r = Bx$ can be performed as in Algorithm 4.5, in which the terms with $i-2$ and $i+2$ are omitted, and the corresponding matrix elements are added to the diagonal.

Numerical results.
The first row of Table 4.5 shows the average number of matrix-vector products $y = P_n(B)Ax$ required for solving one linear system. The second row gives the average CPU-time measured in seconds on a NEC SX-3 for solving one system of linear equations. Comparing the first row of this table with the first row of Table 4.3 leads to the conclusion that the number of matrix-vector multiplications has hardly increased. Hence the convergence behaviour of GMRES(10) is hardly influenced by the replacement of $P_n(A)$ by $P_n(B)$. It appeared that with the efficient form of the polynomial preconditioner, the matrix-vector multiplication with the preconditioner requires only half the CPU-time as

Table 4.5: Results of simplified polynomial preconditioning on the NEC SX-3.

n	2	3	4	5	6
# mat-vec. mult. $P_n(B)A$	17.2	13.2	10.3	8.5	7.6
CPU-time $x := A^{-1}b$	0.075	0.062	0.051	0.051	0.048

with the original form. However, the difference between the last row of Table 4.5 and 4.3 shows that the reduction in the CPU-times is not more than 19 percent. This is due to the fact that the orthogonalization process of GMRES(10) requires a large number of inner products and vector updates. Hence we conclude that the application of the preconditioner forms no longer the bottleneck in solving the linear systems.

4.6 Conclusions

In this chapter various vectorizable preconditioning techniques for the Boussinesq model have been compared with each other on three computers with different architectures. We have tested both polynomial-like preconditioners, and preconditioners based on incomplete LU-decompositions. For the latter we have used level-scheduling in order to improve the performance on supercomputers. So far, we only gave CPU-times of the solution process of the set of linear equations, which shows more clearly the effect of the various techniques. The impact of the most successful techniques on the CPU-time of the complete problem is given in Table 4.6: it shows the CPU-times measured in seconds necessary for four time steps in a fully developed wave field. For comparison, the first column shows the CPU-time of the technique with which we started, e.g. the technique in which the linear systems were solved with CGS combined with MILU(ε)-preconditioning, and with the matrix-vector multiplication implemented as shown in Algorithm 4.4. An asterisk ('*') indicates the 'winner' on a particular machine.

Table 4.6: Comparison of several strategies. CPU-times for four time steps.

iterative method	CGS	GMRES(10)	GMRES(10)	GMRES(10)
preconditioner	MILU(0.01) no l.s.	stand. MILU with l.s.	diagonal scaling	$P_6(B)$
Times, HP-720	205*	312	697	371
Times, CONVEX	438	146*	331	169
Time, NEC SX-3	56	5.5	7.4	3.7*

None of the suggested strategies appears to be a clear winner on all computers. However, standard MILU combined with level-scheduling is in all cases not worse than 1.5 times the best one. Hence we conclude that this type of preconditioner performs reasonably well on all computers considered here. The impact of level-scheduling on the

CPU-time is demonstrated clearly by the difference between the last two columns of Table 4.2.

When working with a scalar machine, one should minimize the number of operations. In that case, CGS in combination with MILU(ε)-preconditioning is a proper choice. On vector computers, one should minimize the number of matrix-vector products with the preconditioned matrix, because these form the bottleneck in solving the systems of linear equations. Therefore, on the CONVEX and the NEC SX-3, it is better to use GMRES(M) as an iterative method. This method has as drawback that much work has to be done for obtaining an orthonormal basis for the Krylov subspace. However, this extra work consists mainly of calculating inner products and vector updates, which can be vectorized very well.

On the CONVEX, GMRES(10) combined with standard MILU-decomposition and level-scheduling performs best. This technique is almost three times as fast as the technique with which we started.

On the NEC SX-3, the efficient implementation of polynomial preconditioning as described in Section 4.5 appears to be the most efficient preconditioner. It appears that GMRES(10) combined with the latter technique reduces the CPU-time with approximately a factor 15 as compared with the model with which we started. If we use a straightforward implementation of an incomplete LU-decomposition without level-scheduling, more than 98 percent of the CPU-time of the complete problem is necessary for solving the linear systems. With the efficient form of polynomial preconditioning, this reduces to 83 percent. The bottleneck is formed by the indirect addressing, which is needed for the matrix-vector multiplication. Hence in many situations, it is advisable to introduce 'dummy' equations in such a way that the domain under consideration becomes a rectangle, thus avoiding the need of indirect addressing.

We have also tested smoothing matrices and Jacobi-smoothing as preconditioners. With these methods, the number of iteration steps decreases, although they are not competitive in comparison with polynomial preconditioning.

It appears to be possible to make a modification of the Boussinesq model in such a way that one obtains systems of linear equations in which the coefficient matrix is symmetric positive definite [48]. These linear systems can be solved efficiently with the conjugate gradient method combined with the preconditioners described in this chapter.

5. Sparse matrix techniques for solving the incompressible Navier-Stokes equations

5.1 Introduction

The incompressible Navier-Stokes equations play an important role in a large variety of applications. For example, one can think of all kinds of water works and off-shore constructions, flows in cooling systems or in blood vessels, low-speed aerodynamics for car design, etc. The equations of motion and continuity are

$$\frac{\partial u}{\partial t} + u \cdot \bigtriangledown u = -\frac{1}{\rho} \bigtriangledown P + \nu \bigtriangledown^2 u \tag{5.1}$$

$$\bigtriangledown \cdot u = 0 \tag{5.2}$$

where u is the velocity, P the pressure, ρ the density and ν the kinematic viscosity. Suppose that x_1 and x_2 are vectors which contain the velocities and pressure respectively in the grid points. The system of equations which arises from the discretization of (5.1) and (5.2) is typically of the form

$$\begin{bmatrix} I\frac{d}{dt} + M(x_1) & G \\ D & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{5.3}$$

where $M(x_1)$ represents the convective and diffusive terms in the momentum equations, and D is the discrete divergence operator. The block G is approximately equal to $-D^T$. The numerical solution of (5.3) for realistic problems is demanding; it requires the best of the available computing power and numerical algorithms. Therefore, it will be clear that fast solution methods for this type of equations are urgently required. However, their solution is non-trivial, because of three aspects:

(a) The presence of the zero-matrix in the lower-right corner which -in physical terms- is related to an infinite propagation speed of acoustical waves, implies that the coefficient matrix in (5.3) is certainly not an M-matrix, and hence a straightforward sparse incomplete decomposition is not possible.

(b) The submatrix $M(x_1)$ often contains a dominating skew-symmetric part (for example, if central differences are used for the convective terms), which severely complicates the numerical solution approaches.

(c) The set of equations is non-linear.

System (5.3) is rarely solved as a whole, and it is very common to take as a starting point

$$\begin{bmatrix} I\frac{d}{dt} + M(x_1) & G \\ DM(x_1) & DG \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ Db_1 - \frac{d}{dt}b_2 \end{bmatrix} \tag{5.4}$$

Next, an iterative procedure can be built where the non-linear terms in the pressure equation are put in the right-hand side, and the velocity is set equal to the value of the previous iterate (pressure-correction approach). In this way one obtains a Poisson equation for the pressure, which can be solved by a standard Poisson-solver (multigrid, ICCG, SOR, SLOR, ADI, etc.). For the momentum equation a non-symmetric matrix (often not an M-matrix) is obtained, which can also be solved by iterative methods. Both iteration processes are often heavily intertwined. Well-known methods of this type for the steady-state problem are SIMPLE [32] and its refinements. During the iterative process, the non-linearity can be taken into account immediately.

In this chapter, we follow a different approach. In order to get rid of the poor coupling between continuity and momentum equation, we want to solve the complete system (5.3) as a whole. In the past this approach has rarely been considered, because the complete matrix requires a large amount of computer storage. Nowadays, the available storage has increased considerably, and this approach becomes more and more attractive, as a better convergence may be expected.

Due to the non-linearity, a system of linear equations $Ax = b$ has to be solved repeatedly. These systems are often solved with iterative methods like GMRES [38] or Bi-CGSTAB [50], which are described in more detail in Section 1.3. In this context, such a method for solving the linear systems can be looked at as an inner-iteration process within the Newton-like method which is used to deal with the non-linearity. Iterative solution of the linear equations has a number of advantages over direct solution: (*i*) it enables us to exploit the sparsity of the matrices involved; and (*ii*) it provides a means of controlling the accuracy of the solution of the linear systems.

To increase the rate of convergence of the inner-iteration process, one can use an incomplete LU-decomposition of A as a preconditioner. Chin et al. [9] describe a preconditioned conjugate gradient method for solving the whole system of linear equations. Their preconditioning technique is based on an incomplete LU-decomposition, and special attention is given to the numbering of the unknowns in such a way that the fill-in during the incomplete decomposition process is minimized.

A straightforward construction of an incomplete LU-decomposition of the coefficient matrix in (5.3) always leads to an enormous fill-in. Therefore, we consider the equivalent system $QAx = Qb$, in which Q can be regarded as a pre-preconditioner. Its function is to obtain a linear system for which one can easily construct a proper preconditioner. In Section 5.2 a choice for Q is specified which improves with mesh refinement. Also a method is described to construct a preconditioner for the matrix QA. No restriction is made with respect to the sparsity pattern of A. Hence the methods can be applied in case of irregular geometries. A similar type of preconditioning is used by Wittum in a multi-grid context calling it *transforming smoothers* [57, 58].

In Section 5.3, we present various computations of the laminar flow over a backward-facing step. It is shown that with an upwind discretization, the technique described above

Figure 5.1: Location of u, v and p when we use a staggered grid.

works fine, but with central differences, the number of inner-iteration steps increases strongly. Therefore, in Section 5.4 we describe an alternative preconditioning technique for the matrix QA. With this technique, one can also use central differences for the convective terms, and still obtain a reasonable rate of convergence of the inner-iteration process. In this respect, it is favourable to multigrid approaches where central differences are problematic.

In Section 5.5 the most successful preconditioning techniques described in this chapter are used to calculate the flow in a furnace with burning waste and in a driven cavity. The driven cavity problem is a well-known test problem known from the literature [9, 16].

5.2 Solution strategy

We consider flows in two dimensions governed by the incompressible Navier-Stokes equations. The time-independent flow is governed by the equations

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{5.5}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) \tag{5.6}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{5.7}$$

where u and v are the velocities in the x-and y-direction respectively, P is the pressure, ρ is the density and ν is the kinematic viscosity. For incompressible flow, the function ρ is constant, and we determine the unknown function $p = P/\rho$ instead of P. In the sequel of this chapter, p will be referred to as the pressure for it is equal to P multiplied with a constant. Equations (5.5)-(5.7) are discretized on staggered grids with u, v and p defined on different locations as shown in Fig. 5.1 and described in the MAC (Marker and Cell) method [55]. Herein '$\rightarrow$' and '$\uparrow$' show where u and v respectively are defined.

Picard iteration.
To deal with the non-linearity of the equations (5.5) and (5.6), we have to use an iterative method. In order to obtain systems of linear equations which are relatively easy to solve, we first consider the Picard iteration. Suppose $u^{(n)}$, $v^{(n)}$ and $p^{(n)}$ are the results after n iteration steps. Then $u^{(n+1)}$, $v^{(n+1)}$ and $p^{(n+1)}$ are obtained from the equations

$$u^{(n)}\frac{\partial}{\partial x}u^{(n+1)} + v^{(n)}\frac{\partial}{\partial y}u^{(n+1)} = -\frac{\partial}{\partial x}p^{(n+1)} + \nu(\frac{\partial^2}{\partial x^2}u^{(n+1)} + \frac{\partial^2}{\partial y^2}u^{(n+1)}) \qquad (5.8)$$

$$u^{(n)}\frac{\partial}{\partial x}v^{(n+1)} + v^{(n)}\frac{\partial}{\partial y}v^{(n+1)} = -\frac{\partial}{\partial y}p^{(n+1)} + \nu(\frac{\partial^2}{\partial x^2}v^{(n+1)} + \frac{\partial^2}{\partial y^2}v^{(n+1)}) \qquad (5.9)$$

$$\frac{\partial}{\partial x}u^{(n+1)} + \frac{\partial}{\partial y}v^{(n+1)} = 0 \qquad (5.10)$$

At every step of the Picard iteration a linear system of the form

$$\begin{bmatrix} C_1 & 0 & G_x \\ 0 & C_2 & G_y \\ D_x & D_y & 0 \end{bmatrix} \begin{bmatrix} x_u^{(n+1)} \\ x_v^{(n+1)} \\ x_p^{(n+1)} \end{bmatrix} = \begin{bmatrix} b_u^{(n+1)} \\ b_v^{(n+1)} \\ b_p^{(n+1)} \end{bmatrix} \qquad (5.11)$$

has to be solved, where G_x and G_y are approximately equal to $-D_x^T$ and $-D_y^T$, respectively. In the sequel of this chapter we take x_u and x_v together as x_1, and write (5.11) as

$$\begin{bmatrix} A_{11} & G \\ D & 0 \end{bmatrix} \begin{bmatrix} x_1^{(n+1)} \\ x_2^{(n+1)} \end{bmatrix} = \begin{bmatrix} b_1^{(n+1)} \\ b_2^{(n+1)} \end{bmatrix}$$

With an upwind finite-difference scheme for the convective terms, and a standard discretization for the diffusive terms, the block A_{11} becomes an M-matrix, so that the construction of an incomplete decomposition of it is straightforward (see Meijerink and van der Vorst [25]). D_x and D_y are discretizations of the first-order derivatives in the x- and y-direction.

Solving the system of linear equations.
In this chapter no restriction is made with respect to the sparsity pattern of the blocks A_{11}, G and D. Therefore, all the non-zero elements have to be addressed indirectly. The matrix is stored as described in Section 2.2. To solve (5.11), one can use an iterative method suitable for non-symmetric systems. Herein $x_1^{(n)}$ and $x_2^{(n)}$ can be taken as starting values for $x_1^{(n+1)}$ and $x_2^{(n+1)}$.

The coefficient matrix is singular, because the level of the pressure is not fixed. One way to overcome this problem is by fixing one entry of the solution, deleting the corresponding rows and columns of A, adjusting the right-hand side, and solving the resulting system. We have followed a different approach: we subtract from the right-hand side its orthogonal projection onto the null space of A, and solve the resulting system of linear equations with a conjugate gradient-like method. This approach gives a slightly better rate of convergence. (see, for example, [21]).

The preconditioning technique.
The rate of convergence of the inner-iteration process strongly depends on the eigenvalue distribution of the coefficient matrices. Therefore, the inner-iteration process is applied to the equivalent system

$$C^{-1}\begin{bmatrix} I & 0 \\ 0 & C_q \end{bmatrix}\begin{bmatrix} A_{11} & G \\ D & 0 \end{bmatrix}\begin{bmatrix} x_1^{(n+1)} \\ x_2^{(n+1)} \end{bmatrix} = C^{-1}\begin{bmatrix} I & 0 \\ 0 & C_q \end{bmatrix}\begin{bmatrix} b_1^{(n+1)} \\ b_2^{(n+1)} \end{bmatrix} \tag{5.12}$$

One can think of the matrix

$$\begin{bmatrix} I & 0 \\ 0 & C_q \end{bmatrix}$$

to be a sort of pre-preconditioner. In the sequel of this chapter we will write this matrix as Q. The matrix C_q herein has to be chosen in such a way that one obtains a coefficient matrix QA for which it is relatively easy to construct a proper preconditioner. The non-singular matrix C should have the following properties:

1. C approximates the block
$$QA = \begin{bmatrix} A_{11} & G \\ C_qD & 0 \end{bmatrix}$$
so that the preconditioned matrix $C^{-1}QA$ has its eigenvalues clustered near 1.

2. For a given vector d we can solve $Cy = d$ in $\mathrm{O}(N)$ operations.

3. C has low storage requirements.

Since the coefficient matrix in (5.11) is singular, there will always be an eigenvalue of the preconditioned matrix which is equal to zero. Hence in this case we will require all the eigenvalues of $C^{-1}QA$ except one to be clustered near 1.

If one could succeed in finding a matrix C_q such that C_qD is equal to DA_{11}, the matrix C should be an approximation of

$$\begin{bmatrix} A_{11} & G \\ DA_{11} & 0 \end{bmatrix}$$

Since the block A_{11} is an M-matrix, we can easily construct an incomplete decomposition L_1U_1 of it, which gives the following possible choice for C

$$C = \begin{bmatrix} I & 0 \\ D & L_2U_2 \end{bmatrix}\begin{bmatrix} L_1U_1 & G \\ 0 & I \end{bmatrix} \approx \begin{bmatrix} L_1U_1 & G \\ DL_1U_1 & 0 \end{bmatrix} \tag{5.13}$$

where L_2U_2 is an approximation of -DG. The incomplete decomposition of this block is based on a drop tolerance which depends on the difference between C_qD and DA_{11}. Since the matrix product DG does not depend on the solution, the factors L_2 and U_2 need to be calculated only once. The construction of L_2 and U_2 is relatively easy, because DG is approximately equal to the matrix which arises after a standard discretization of the Laplace equation. The factors L_1 and U_1 are adapted only during the first few iteration steps of the outer-iteration process.

The choice of the pre-preconditioner.
In practice, it is not possible to find a matrix C_q in such a way that C_qD actually *equals* DA_{11}, but one can try to construct C_q in such a way that C_qD *resembles* the matrix DA_{11} as closely as possible. The choice of C_q is based on the following considerations. The blocks C_1 and C_2 in (5.11) are discretizations at u- and v-points respectively of the same differential operator

$$u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} - \nu(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})$$

Hence C_1 and C_2 are approximately the same. Since D_x and D_y are discretizations of first-order derivatives in the x- and y-direction respectively, and the differential operators approximately commute when u and v are not strongly varying, the choice of C_q equal to C_1 or C_2 gives the desired property. This choice has the nice property that the pre-preconditioner will get better when the mesh is refined, because in that case, the matrix blocks will be more accurate discretizations of the differential operators, and the choice of C_q is based on the fact that these operators approximately commute. In this chapter C_q was taken equal to C_1.

5.3 Numerical results

To demonstrate the methods of the previous section, we have calculated the time-independent flow in a two-dimensional channel over a backward-facing step. This so-called backward-facing step problem has become a well-known model problem to test the accuracy and efficiency of incompressible Navier-Stokes solvers [28]. The domain under consideration is shown in Fig. 5.2. Herein $h = 2$, $H = 3$, $l_{tot} = 44$ and $l = 6$ meter. At

Figure 5.2: Geometry of the backward-facing step problem.

the entrance of the channel a parabolic velocity profile was prescribed:

$$u(y) = \frac{4}{h^2}(h - y)y$$

At the end of the channel the following boundary conditions were used:

$$\frac{\partial u}{\partial x} = 0 \text{ and } v = 0$$

These boundary conditions are based on the hypothesis that the channel behind the step is long enough for the flow to become parallel, and not to change anymore along the x-direction. At all the other boundaries the impermeability and no-slip conditions were used: $u = v = 0$.

Tables 5.1 and 5.2 show the results of the preconditioning technique described earlier, when Bi-CGSTAB and GMRES(M) respectively were used as inner-iteration process for solving the system of linear equations. From the results it appeared that the value of M should not be chosen too small, because otherwise GMRES(M) converges only very slowly. In many cases, a reasonable value for M appeared to be 20. The Reynolds number, which is defined by $Re = U_{MAX}(H - h)/\nu$, was taken equal to 500. Equations (5.5) to (5.7) were discretized on an equidistant grid using an upwind finite-difference scheme for the convective terms. The grid was refined several times by reducing the mesh sizes in both the $x-$ and y-direction. The first column shows the number of internal grid points. This number should be multiplied by three in order to obtain the number of degrees of freedom. The second column gives the average number of non-zero entries required for the preconditioner per grid point, the third column the total number of matrix-vector multiplications with the preconditioned matrix, and the fourth column the number of Picard iteration steps. The CPU-time for the complete calculation is given in seconds. All computations were performed on an HP-720 work station. In order to make a comparison with other solvers easier, we also give the number of flops per grid point required for the complete calculation. The Picard iteration was terminated when the maximum norm of the difference between two succeeding iteration steps was less than 10^{-4}. As starting vector for the Picard iteration we took the null vector. Suppose that j is the index of the inner-iteration method. As a stopping criterion for Bi-CGSTAB and GMRES(20) the maximum norm of the residual was required to decrease with a factor *tol*:

$$\|C^{-1}Q(Ax_j^{(n+1)} - b^{(n+1)})\|_\infty \leq tol\|C^{-1}Q(Ax_0^{(n+1)} - b^{(n+1)})\|_\infty$$

Herein *tol* is a parameter which has to be chosen in advance. For small Reynolds numbers, the outer-iteration process converges very rapidly, and to take full advantage of this rate of convergence, the solution obtained from the inner-iteration process has to be very accurate, thus *tol* has to be chosen relatively small.

Table 5.1: Upwind finite differences, Bi-CGSTAB and Picard iteration. Re=500.

# grdps	# entries per grdp	# mat-vec. mult.	# Picard iter.	CPU-time	# flops/grdp $\times 10^6$
504	46.3	234	23	4.3	0.051
1134	49.9	330	32	15.2	0.080
2016	52.2	562	41	47.8	0.143
3150	53.4	742	46	100.4	0.191

The number of non-zero elements required for preconditioning is approximately 17 times the number of degrees of freedom. This is very low compared with the number of non-zero elements in a complete LU-decomposition.

The number of GMRES iteration steps required per outer-iteration step is less than 20. Hence it is not necessary to restart the inner-iteration process. Since GMRES minimizes the 2-norm of the residual over the Krylov subspace, the number of matrix-vector multiplications required per Picard iteration step is minimized. From the difference between

Table 5.2: Upwind finite differences, GMRES(20) and Picard iteration. Re=500.

# grdps	# entries per grdp	# mat-vec. mult.	# Picard iter.	CPU-time	# flops/grdp $\times 10^6$
504	45.2	153	23	3.5	0.042
1134	49.9	262	37	16.2	0.086
2016	52.8	372	42	45.9	0.137
3150	54.7	478	43	106.8	0.203

the third column of Table 5.1 and of Table 5.2, it follows that with Bi-CGSTAB as linear system solver, the number of matrix-vector multiplications is much larger. However, with GMRES(20) much work has to be done for obtaining an orthonormal basis for the Krylov subspace. As a consequence, on a scalar machine GMRES(20) is approximately as expensive as Bi-CGSTAB.

On a vector machine the results can be quite different. This is illustrated by the results of Table 5.3 which show the CPU-times in seconds measured on a CONVEX C2. Only one processor of this machine is used.

Table 5.3: CPU-times measured on a CONVEX C2. Upwind discretization. Re=500.

# grid points	504	1134	2016	3150
Bi-CGSTAB	16.1	53.3	163.1	323.5
GMRES(20)	12.4	47.3	116.7	226.7

The computations of GMRES(M) consist mainly of calculating inner products, which can be vectorized very well. A drawback of GMRES(M) is that computer storage demands become higher with increasing M. Unfortunately, the value of M has to be chosen relatively large, because otherwise the convergence behaviour is poor, or GMRES(M) may not converge at all.

5.3.1 Newton's method

From Table 5.2 one can see that the number of Picard iteration steps is quite large. Therefore, we want to solve the discretized incompressible Navier-Stokes equations with Newton's method [31]. This is one of the most commonly used methods for solving large systems of non-linear equations. It takes full advantage of first and second derivative information, and possesses ideal characteristics of local convergence. However, the coefficient matrix arising in the Newton iteration, the Jacobian, contains more non-zero elements than the coefficient matrix of the linearised system which arises in the Picard iteration. Moreover, it is more difficult to construct a proper preconditioner for the Jacobian, because the matrix need not be diagonally dominant, even if we use upwind finite differences. An alternative is to use an approximation of the Jacobian for which it is easier to construct an incomplete decomposition. However, this approach may deteriorate the quadratic rate of convergence of Newton's method. Therefore, in this section we will consider a preconditioning method for the full Jacobian.

The convective term

$$v^{(n+1)}\frac{\partial}{\partial y}u^{(n+1)}$$

is linearised as

$$v^{(n)}\frac{\partial}{\partial y}u^{(n+1)} + v^{(n+1)}\frac{\partial}{\partial y}u^{(n)} - v^{(n)}\frac{\partial}{\partial y}u^{(n)} \tag{5.14}$$

This is based on the following observations. Suppose that $v^{(n+1)} = v^{(n)} + O(\delta)$, and

$$\frac{\partial}{\partial y}u^{(n+1)} = \frac{\partial}{\partial y}u^{(n)} + O(\delta)$$

then

$$v^{(n+1)}\frac{\partial}{\partial y}u^{(n+1)} \approx v^{(n)}\frac{\partial}{\partial y}u^{(n+1)} + v^{(n+1)}\frac{\partial}{\partial y}u^{(n)} - v^{(n)}\frac{\partial}{\partial y}u^{(n)} + O(\delta^2)$$

The other convective terms are linearised similarly. The term $v^{(n+1)}$ in (5.14) should be approximated in the '$\rightarrow$-points' in Fig. 5.1. By taking for $v^{(n+1)}$ the value defined in the '$\uparrow$-points', we make an error which is $\mathrm{O}(h)$, where h is the largest mesh size. As long as one uses an upwind discretization for the convective terms, the local discretization error is also $\mathrm{O}(h)$. Hence this approach does not affect the accuracy of the solution very much.

The discretization according to (5.14) leads to a system of linear equations in which the coefficient matrix A has the block structure

$$\begin{bmatrix} C_1 & D_1 & G_x \\ D_2 & C_2 & G_y \\ D_x & D_y & 0 \end{bmatrix}$$

where D_1 and D_2 are both diagonal matrices with entries depending on the value of $\partial u^{(n)}/\partial y$ and $\partial v^{(n)}/\partial x$, respectively. Those diagonal matrices are not present in the linear system (5.11). Another important difference with (5.11) is that the block

$$A_{11} = \begin{bmatrix} C_1 & D_1 \\ D_2 & C_2 \end{bmatrix}$$

need not be an M-matrix. Hence during the construction of an incomplete decomposition of A_{11}, a large amount of fill-in is generated, or the construction may even break down. Therefore, we make an incomplete decomposition of the block

$$\tilde{A}_{11} = \begin{bmatrix} \tilde{C}_1 & 0 \\ 0 & \tilde{C}_2 \end{bmatrix} \tag{5.15}$$

Herein $\tilde{A}_{11}$ comes from the upwind discretization of the convective terms and a standard discretization of the diffusive terms. Of course, it is not practical to make a very accurate incomplete decomposition of $\tilde{A}_{11}$, because in that case the quality of the preconditioner would be determined completely by the difference between A_{11} and $\tilde{A}_{11}$.

An inherent problem is how much precision is required in the solution coming from the iteration process in each step of Newton's method. Since the Newton equations are based on a Taylor series expansion near the solution, it is not desirable to solve the linearised system very accurately when far away from the solution of the system of non-linear equations. At the beginning of the solution process, a reasonable estimate of the search direction may be as effective as the exact search direction itself. Solving the linear systems too accurately results in a waste of computer time. As the solution is approached, the search directions have to be be determined more accurately. The inner-iteration process is stopped when

$$\|C^{-1}Q(Ax_j^{(n+1)} - b^{(n+1)})\|_\infty \leq tol\|C^{-1}Q(Ax_0^{(n+1)} - b^{\ (n+1)})\|_\infty$$

At the later stages of the Newton iteration *tol* is decreased in order to take full advantage of the quadratic rate of convergence.

To test this technique we consider the same problem as in Table 5.1 and use Newton instead of Picard iteration. The results are shown in Table 5.4.

Table 5.4: Upwind finite differences, Bi-CGSTAB and Newton iteration. Re=500.

# grdps	# entries per grdp	# mat-vec. mult.	# Newton iter.	CPU-time	# flops/grdp $\times 10^6$
504	35.4	242	6	3.8	0.045
1134	38.1	336	7	13.4	0.071
2016	39.1	394	8	29.7	0.089
3150	39.1	520	9	62.4	0.119

By comparing these results with those of Table 5.1, we conclude that the number of outer-iteration steps decreases considerably. However, with Newton's method each iteration step requires more inner-iteration steps. This is due to the fact that we make an incomplete decomposition of $\tilde{A}_{11}$ instead of A_{11}, which can deteriorate the quality of the preconditioner. Moreover, in order to take advantage of the quadratic rate of convergence of the outer-iteration process, the parameter *tol* has to be chosen smaller, which increases the number of inner-iteration steps.

5.3.2 Central differences for the convective terms

We will now consider the more realistic case in which a second-order discretization for the convective terms is used. Consider the following three grid points with corresponding function values ϕ_-, ϕ_0 and ϕ_+:

In the second grid point we want to have a second-order accurate discretization of $\partial\phi/\partial x$. One can use the first term in the right-hand side of the following formula

$$\frac{\partial\phi}{\partial x} = \frac{\phi_+ - \phi_-}{h_- + h_+} - \tfrac{1}{2}\,(h_+ - h_-)\frac{\partial^2\phi}{\partial x^2} - \tfrac{1}{6}\,\frac{h_+^3 + h_-^3}{h_- + h_+}\frac{\partial^3\phi}{\partial x^3} + \dots \tag{5.16}$$

On first sight the error term with $(h_+ - h_-)$ looks a first-order term, but when the grid is obtained from a transformation $x_i = f(\xi_i)$, where the points ξ_i form an equidistant grid with mesh-size δ, we have

$$h_+ = x_+ - x_0 = \delta f'(\xi_0) + \tfrac{1}{2}\,\delta^2 f''(\xi_0) + \dots$$

$$h_- = x_0 - x_- = \delta f'(\xi_0) - \tfrac{1}{2}\,\delta^2 f''(\xi_0) + \dots$$

and one can readily see that the difference $(h_+ - h_-)$ is in fact a second-order term. The term $u\partial u/\partial x$ will therefore be discretized as

$$u_0\frac{u_+ - u_-}{h_- + h_+}$$

The other convective terms are discretized similarly. For an exponential grid with a fixed stretching rate $h_+/h_- \neq 1$, the local truncation error is only of first order. However, in [53] it is shown that the discretization (5.16) produces acceptable solutions, even when other discretization schemes, which have a second-order local truncation error, produce inacceptable results.

To obtain the stationary solution of the incompressible Navier-Stokes equations, we use the Picard iteration defined by (5.8)-(5.10). Discretizing these equations as described above leads to the system of linear equations

$$\begin{bmatrix} C_1 & 0 & G_x \\ 0 & C_2 & G_y \\ D_x & D_y & 0 \end{bmatrix} \begin{bmatrix} x_u^{(n+1)} \\ x_v^{(n+1)} \\ x_p^{(n+1)} \end{bmatrix} = \begin{bmatrix} b_u^{(n+1)} \\ b_v^{(n+1)} \\ b_p^{(n+1)} \end{bmatrix}$$

in which the blocks C_1 and C_2 have positive elements on the diagonal, because the convective term does not reduce the main diagonal. However, when the mesh Péclet numbers are larger than 2, these blocks are not M-matrices. Therefore, we again make an incomplete decomposition of (5.15) instead of the coefficient matrix.

Table 5.5 shows the results for a Reynolds number of 500. The Picard iteration was combined with GMRES(20) as inner-iteration method. To make sure that the mesh Péclet numbers are small in those regions where the flow is strongly varying, we used a non-uniform grid with the smallest mesh sizes in the recirculation zone. The grid uses the lines $x_i = f(\xi_i)$, where the points ξ_i form a uniform grid on [-1,3], and f was chosen in such a way that $f(-1) = 0$, $f(0) = 6$, $f(3) = 44$ and f varies only slowly in the neighbourhood of 0. In the $y-$direction the mesh size is constant. The CPU-times are given in seconds, and they were measured on an HP-720 workstation.

When the number of inner-grid points is 352, GMRES(20) does not convergence. This can probably be explained by the fact that, for a coarse grid, the pre-preconditioning

Table 5.5: Central differences, GMRES(20) and Picard iteration. Re=500.

# grdps	# entries per grdp	# mat-vec. mult.	# Picard iter.	CPU-time	# flops/grdp $\times 10^6$
352					
792	35.1	3502	85	142.4	1.08
1408	38.2	3265	92	265.4	1.13
2200	40.6	3819	106	560.8	1.53

technique does not work very well. The construction of an incomplete decomposition of C_1 and C_2 does not fail, even when these blocks are not M-matrices. The rate of convergence of both the inner- and outer-iteration method is very slow. Every step of the outer-iteration process needs more than 35 matrix-vector multiplications, so that GMRES has to be restarted at least once. In order to improve the convergence rate of the inner-iteration process, one has to use a better preconditioning technique.

5.4 Improving the preconditioner

Again we use the pre-preconditioner which has been introduced earlier. In this section, we try to develop a preconditioner which is a more accurate approximation of the coefficient matrix

$$\begin{bmatrix} A_{11} & G \\ C_qD & 0 \end{bmatrix} \tag{5.17}$$

than the matrix product in (5.13). Assuming that L_1U_1 is a proper incomplete decomposition of A_{11}, and L_2U_2 is an incomplete decomposition of $-C_qD(L_1U_1)^{-1}G$, the matrix product

$$\begin{bmatrix} L_1U_1 & 0 \\ C_qD & L_2U_2 \end{bmatrix} \begin{bmatrix} I & (L_1U_1)^{-1}G \\ 0 & I \end{bmatrix} \approx \begin{bmatrix} L_1U_1 & G \\ C_qD & 0 \end{bmatrix}$$

is a proper preconditioner for (5.17). In practice, it is very difficult to construct a preconditioner for the block $-C_qD(L_1U_1)^{-1}G$, because $(L_1U_1)^{-1}$ is not available. Instead a block $\hat{D}$ is constructed which is approximately equal to $C_qD(L_1U_1)^{-1}$. Since C_q is chosen in such a way that C_qD is approximately equal to DA_{11}, $\hat{D}$ will be similar to the block D. The construction of an incomplete decomposition of $-\hat{D}G$ is expected to be relatively cheap, because this block is similar to the coefficient matrix arising from a standard discretization of the Laplace equation.

Solving $\hat{D}$ approximately from $C_qD = \hat{D}L_1U_1$ can be done in two steps. First the block $\tilde{D}$ is solved from

$$C_qD = \tilde{D}U_1 \tag{5.18}$$

and next $\hat{D}$ is solved from

$$\tilde{D} = \hat{D}L_1 \tag{5.19}$$

The matrix $\tilde{D}$ can be constructed row by row in a similar way as the construction of an ILU(ε)-decomposition, as is described in Section 2.3. Suppose that the elements $\tilde{d}_{ij}$ have been constructed for $j <k$. We can rewrite (5.18) as follows:

$$rl_{ik} = \sum_{j=1}^{k} \tilde{d}_{ij} u_{jk}$$

in which rl_{ik} represents the (i, k)-element of the matrix product $C_q D$, and u_{ik} represents the corresponding entry of U_1. Assuming that all diagonal entries of U_1 are equal to 1, the last equation can be written as

$$\tilde{d}_{ik} = rl_{ik} - \sum_{j=1}^{k-1} \tilde{d}_{ij} u_{jk} \tag{5.20}$$

To be able to work with large coefficient matrices, the block $\tilde{D}$ should be sparse. This can be realized by solving (5.20) only approximately in the following way. Suppose d_{max} is the maximum absolute value of the entries in the i-th row of $C_q D$. When the absolute value of the right-hand side of (5.20) is less than εd_{max}, the entry $\tilde{d}_{ik}$ is neglected. Herein ε is a threshold parameter which has to be chosen in advance. Solving $\hat{D}$ from (5.19) can be done similarly.

As soon as row i of $\hat{D}$ has been constructed, row i of L_2 and U_2 can be calculated, where $L_2 U_2$ is an incomplete decomposition of $-\hat{D}G$. This can be done in a similar way as described in Section 2.3.

The construction of an incomplete decomposition of $-\hat{D}G$ as described above does not require the matrices $\hat{D}$ and $\tilde{D}$ to be stored in memory: the construction of row i of L_2 and U_2 only requires the corresponding rows of $\hat{D}$ and $\tilde{D}$.

In order to increase the diagonal dominance of $\tilde{A}_{11}$, we use a time-stepping approach. The terms $\partial u/\partial t$ and $\partial v/\partial t$ are discretized according to the backward-Euler scheme, and to deal with the non-linearity, we use Picard iteration. The time step is chosen large enough in such a way that the number of outer-iteration steps hardly increases by using the time-stepping approach.

The preconditioning technique described in this section makes no restriction with respect to the sparsity pattern of the blocks C_1 and C_2. As a consequence, all non-zero elements have to be addressed indirectly. Hence the construction of the preconditioner cannot be vectorized. Fortunately, the preconditioner has to be constructed only during the first few iteration steps of (5.8)-(5.10), because the coefficient matrix hardly changes at the later stages of this iteration process.

Tables 5.6 and 5.7 show the results of this technique applied to the system of equations coming from the backward-facing step problem for Reynolds numbers 150 and 500. The second column gives the average number of entries per grid point required for the preconditioning technique, including the entries of the incomplete decompositions for both A_{11} and $-\hat{D}G$. We only list the results of GMRES(20); Bi-CGSTAB performs approximately as well. For Re = 150 we performed some calculations on a very fine grid in order to enable us to make a straightforward comparison with some other solvers.

Table 5.6: Central differences. GMRES(20). Re=150. Preconditioner of this section.

# grdps	# entries per grdp	# mat-vec. mult.	# Picard iter.	CPU-time	# flops/grdp $\times 10^6$
352	56.1	119	21	3.3	0.056
792	63.0	148	22	10.1	0.077
1408	66.5	168	22	22	0.094
2200	67.9	192	22	41	0.112
5632	69.0	271	22	165	0.176
8800	76.0	336	22	354	0.241
12672	87.0	369	24	585	0.277

Table 5.7: Central differences. GMRES(20). Re=500. Preconditioner of this section.

# grdps	# entries per grdp	# mat-vec. mult.	# Picard iter.	CPU-time	# flops/grdp $\times 10^6$
352	55.1	815	64	17.5	0.30
792	65.5	536	77	35.9	0.27
1408	69.7	555	77	76.4	0.33
2200	72.3	656	88	139	0.38
5632	74.3	916	87	551	0.59

From the results it follows that the preconditioning technique described in this section strongly reduces the number of matrix-vector multiplications required at each step of the Picard iteration. However, the number of entries in the preconditioner has increased with almost a factor 2 compared with the results of Table 5.5.

Comparison with other solvers.
In [28] a comparison is found of various computations of the laminar flow over a backward-facing step with Reynolds number 50 and 150. Some authors also gave results for Re=500. In order to compare the techniques described in [28] with the preconditioning techniques of this section, we have tried to summarise the results of [28] for Re=150 in Table 5.8. These results can be compared with those of Table 5.6. It should be mentioned that Table 5.8 can only give a rough idea of the cost of a particular method, because it is very difficult to obtain information about exact stopping criteria, which compiler was used, how well the algorithms vectorize etc. The second column shows which formulation

Table 5.8: Results for the backward-facing step for Re=150 obtained from [28].

name:	method	computer:	CPU-time	# grdps	# flops/ grdp $\times 10^6$
Becker	ψ-ω FD upwind	Cyber 174	120	1701	0.1
Toumi	ψ-ω FD central	UNIVAC 1110	360	1782	0.07
Schkalle	ψ-ω FD central	Cyber 175	108	3725	0.1
Schkalle	ψ-ω FD central	Cyber 175	178	14553	0.04
Durst	uvp FD upwind	Univac 1100	589	1200	0.19
Durst	uvp FD central	Univac 1100	592	1200	0.19
Segal	uvp FE central	AMDAHL 470	180	752	0.3
Roose	uvp FE central	CDC 750/170	6000	1623	4.0
Donea	uvp FE central	AMDAHL 470	1100	580	1.0
Cliffe	uvp FD central	CRAY 1	50	2240	0.4
Cliffe	uvp FE central	CRAY 1	360	11441	0.6

of the equations was used. When the primitive variable formulation was used, this is indicated with ‘uvp’, and when the method was based on the vorticity-stream function formulation, this is indicated with $\psi - \omega$. Further, FD and FE indicate the use of either finite-differences or finite-element methods. The fourth column shows the CPU-time in seconds on the machine shown in the third column, the fifth column shows the number of grid points, and the last column shows the number of flops per grid point. This number has been obtained by first making an estimate of the speed of the specific computer, and by next calculating the total number of flops from the results of the fourth column. As mentioned before, the results of the last column are only *estimates* of the cost of a particular method.

From this table we conclude that the most efficient solvers are based on the vorticity-stream function formulation. This can be explained by the fact that in two dimensions this formulation leads to two unknown functions instead of three, and the coefficient matrix does not have a zero-matrix in the lower-right corner, as is the case in the primitive variable formulation. However, in three dimensions the vorticity-stream function formulation leads to six unknown functions, and therefore this approach is only attractive for calculating two-dimensional flows.

By comparing the results of Table 5.6 with those of Table 5.8, we conclude that the method of Section 5.4 is almost as efficient as the methods based on the $\psi - \omega$ formulation. This is also demonstrated by the results of Fig. 5.3 which show the number of flops per grid point required for the methods of Tables 5.6 and 5.8 using central differences.

Figure 5.3: Flops per grid point for several methods using central differences.

5.5 Some other test problems

In order to test the preconditioning techniques of this chapter, we have calculated the flow in two other examples. The first example concerns the calculation of the flow in a furnace, and the last example is a well-known test problem taken from the literature.

5.5.1 The flow in a furnace with burning waste

This test problem comes from the Institute of Environmental and Energy Technology of TNO from the Netherlands. It concerns the simulation of air flow in a furnace which is used to burn waste. As the combustion process is so complex, and takes place in a poorly mixed mixture of gases, the combustion is often incomplete, so that carbon monoxide or other harmful products are formed. In order to improve the combustion process, air is injected into the furnace through nozzles which are located above the burning waste. The purpose of these air jets is to enhance the mixing of incomplete burnt gases. As a lot of

Figure 5.4: Geometry for the second test problem.

air (up to 40% of the total air in the furnace) is injected through the nozzles, and the jet velocity is high (up to 75 m/s), the flow pattern in the furnace is strongly influenced by the jets.

We have calculated the flow of a simplified version of the above furnace problem, consisting of a part of the furnace with only one jet emanating from a corner of the domain. The incompressible Navier-Stokes equations have been solved in the domain shown in Fig. 5.4. The area in the neighbourhood of the nozzle has been enlarged as shown in the left-most picture. The zero-stress boundary condition can be written as

$$\begin{bmatrix} \frac{1}{2}\,\nu(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}) \\ -p+\nu\frac{\partial v}{\partial y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The effect of turbulence was modelled with a uniform eddy-viscosity which was taken equal to $0.02\delta|v_{jet}-v_{in}|$ (see, for example, [24]). Herein δ gives the width of the jet, and v_{in} gives the speed of air at the bottom of the furnace. This is the Prandtl mixing length model. In this way we obtain an eddy-viscosity of $0.17\text{m}^2/\text{sec}$. This is very large compared with the kinematic viscosity of air which is equal to $0.145\times 10^{-4}\text{m}^2/\text{sec}$. The incompressible Navier-Stokes equations were discretized on a 102×202-grid, with constant mesh-size in the horizontal direction. We used central differences for all derivatives and a time-stepping approach as described in Section 5.4, which leads to a system of non-linear equations of the form (5.3) with 60000 unknowns. This system was solved with Picard iteration combined with GMRES(20) as inner-iteration method. It appeared that with $\nu = 0.17\text{m}^2/\text{sec}$, a stationary solution can be found, whereas for smaller values of ν this is not possible. As a stopping criterion for the Picard iteration we used

$$\|x_1^{(n)} - x_1^{(n-1)}\|_\infty \leq 10^{-5}$$

On an HP-720 workstation, the calculation took 29 minutes of CPU-time and approximately 0.5×10^6 flops per grid point. Fig. 5.5 and 5.6 show the calculated flow and pressure respectively. These figures show that in the recirculation zone the pressure is lower. The boundary conditions in the neighbourhood of the jet and in the corners are discontinuous, which is reflected in non-physical behaviour of the pressure in these areas. However, the influence of these discontinuities is restricted to the direct neighbourhood of the corners.

5.5.2 The driven cavity problem

The driven cavity problem has become a well-known problem to test solvers for the incompressible Navier-Stokes equations. The problem consists in calculating the flow field in a square cavity whose top wall moves with a constant velocity U. The domain under

Figure 5.5: The calculated flow in the furnace.

Figure 5.6: The calculated pressure in the furnace.

Figure 5.7: Geometry for the driven cavity problem.

consideration and boundary conditions are shown in Fig. 5.7. The Reynolds number is defined by $Re = U/\nu$. We used a time-stepping approach as described at the end of Section 5.4.

The preconditioning technique of Section 5.4 has as advantage that it works, even when central differences are used for the convective terms. However, this technique has as drawback that it is rather complex: one has to describe how (5.18) and (5.19) are approximately solved, which requires the choice of a number of parameters. Moreover, the technique of Section 5.4 requires more computer memory than the preconditioning technique described by (5.12) and (5.13). An alternative method to obtain second-order accuracy is to represent the convective terms via a first-order accurate upwind-difference scheme, including its second-order accurate term in the right-hand side as a deferred correction [22]. The convective terms are discretized as

$$u_j^{(n)}\frac{u_j^{(n+1)} - u_{j-1}^{(n+1)}}{h_x} + u_j^{(n)}\frac{u_{j-1}^{(n)} - 2u_j^{(n)} + u_{j+1}^{(n)}}{2h_x} \qquad u_j^{(n)} \geq 0$$

or

$$u_j^{(n)}\frac{u_{j+1}^{(n+1)} - u_j^{(n+1)}}{h_x} - u_j^{(n)}\frac{u_{j-1}^{(n)} - 2u_j^{(n)} + u_{j+1}^{(n)}}{2h_x} \qquad u_j^{(n)} < 0$$

This scheme has the property that for a converged solution, e.g. when $u_j^{(n+1)} = u_j^{(n)}$, it reduces to a central-difference scheme, which is second-order accurate, while it uses the coefficient matrix of an upwind finite-difference scheme. Thus, we can use the relatively simple preconditioning technique described by (5.12) and (5.13). We have used this technique for calculating the flow for the Reynolds numbers 3200, 5000, 7500 and 10000. We used a non-uniform 130×130-grid, thus the number of internal grid points is 16384. In the neighbourhood of the boundaries the mesh was slightly refined in such a way that the smallest mesh size is 0.0035. The grid was obtained from a transformation $x_i = f(\xi_i)$, where the points ξ_i form an equidistant grid, and the function f is shown in Fig. 5.8.

When $Re = 3200$, we took the solution for $Re = 1000$ as starting vector for the Picard iteration, and the solution obtained in this way was used as starting vector for $Re = 5000$. In this way we proceeded until the solution for $Re = 10000$ was obtained.

The outer-iteration process was stopped when the maximum norm of the difference between two succeeding iteration steps was less than 10^{-6}. If one is only interested in the primary vortex, a larger value of 10^{-5} can be chosen, which reduces the CPU-time strongly. However, when the secondary vortices in the corners have to be calculated, the stopping criterion of the outer-iteration process should be chosen sufficiently small. As inner-iteration method we used GMRES(30). It was never necessary to restart GMRES. The results are listed in Table 5.9. Note that here the first column gives the Reynolds number, and not the number of grid points. It should be mentioned that we do not take

Figure 5.8: The function which is used for mesh refinement.

advantage of the special geometry of the problem. For this particular problem, the CPU-time can be reduced strongly by using the geometry and the vorticity-stream function formulation [16]. Hence the method described in this chapter is not as efficient as the method described in [16].

Table 5.9: Results for the driven cavity problem on a 130×130-grid.

Re	# entries per grdp	# mat-vec. mult.	# Picard iter.	CPU-time	# flops/grdp $\times 10^6$
3200	45.1	852	38	1577	0.58
5000	44.2	779	36	1399	0.51
7500	43.9	968	45	1742	0.64
10000	43.0	1980	103	3401	1.24

Fig. 5.9 shows that the secondary vortex in the lower-right corner is very small. Hence grid refinement near the boundaries is required in order to be able to represent such details on a 130×130-grid. The results agree perfectly well with the results obtained by Ghia et al. obtained on a uniform 257×257-grid [16]. Fig. 5.10 and 5.11 show respectively the streamlines and the calculated pressure for $Re = 10000$. The size of the secondary vortex increases with the Reynolds number, and it appears that in the lower-left corner there is a secondary vortex as well.

5.6 Conclusions

In this chapter, a preconditioner for the full system of equations arising after linearization of the incompressible Navier-Stokes equations is described. A straightforward sparse

Figure 5.9: The calculated flow for $Re = 3200$.

incomplete decomposition for this system is not possible. Therefore, we use a pre-preconditioner. After applying this pre-preconditioner to the system, a sparse incomplete decomposition of the resulting linear system can be made, which results in savings on CPU-time and storage compared with direct solution of the linear systems. The preconditioning technique performs well for both upwind and central differences. When upwind differences are used, the system of equations can be solved very efficiently by using Newton's method as an outer-iteration method, and by the relatively simple preconditioning technique described by (5.12) and (5.13). When central differences are used, it is better to use the preconditioning technique described in Section 5.4. However, this technique has as drawback that it requires more storage. An alternative is to use the discretization suggested by Khosla and Rubin [22], which is second-order accurate, while it uses the coefficient matrix of an upwind finite-difference scheme.

We used both Bi-CGSTAB and GMRES(M). On scalar machines both are equally efficient, although GMRES(M) has the drawback of requiring more storage. However, on vector computers, GMRES(M) is more attractive, because a large part of the computations consists of calculating inner products, which vectorizes quite well. The techniques described in this chapter have no restriction with respect to the sparsity pattern. Hence, far more complicated geometries than those of the test problems used in this chapter can be handled. The other side of this generality is that indirect addressing is unavoidable, which is disadvantageous on some vector computers. However, for regular geometries we can do without indirect addressing, and good performance on all types of computers is

Figure 5.10: The calculated flow for $Re = 10000$.

Figure 5.11: The calculated pressure for $Re = 10000$.

expected.

6. Grid-independent convergence based on preconditioning techniques

6.1 Introduction

Today numerical calculations are no longer restricted to a class of simple problems, but cope with complicated simulations and complex geometries. In many situations, the accuracy of the numerical solution is determined by the limited amount of computer power and memory. Therefore, in this chapter we will focus on fast iterative methods for solving large sparse systems of linear equations like MICCG and algebraic multigrid. Gustafsson has shown that for several problems the CPU-time using MICCG is $\mathrm{O}(N^{5/4})$ in two dimensions and $\mathrm{O}(N^{7/6})$ for 3D-problems, where N is the total number of unknowns. Multigrid methods perform even better, and for a large class of problems they have an optimal order of convergence: the amount of work and storage is proportional to the number of unknowns N. However, due to the required proper smoothers and the restriction and prolongation operators at each level, the implementation of multigrid techniques for practical problems is much more complicated than that of MICCG. Here we look for a combination of these properties: an incomplete LU-decomposition so that the preconditioned system

$$(LU)^{-1}Ax = (LU)^{-1}b$$

can be solved with the optimal computational complexity $\mathrm{O}(N)$ by iterative methods such as Bi-CGSTAB [50] or GMRES [38]. The basic idea behind this preconditioning technique is the same as in multigrid methods. Many iteration methods can eliminate high-frequency errors very effectively, but they are inefficient at eliminating long-wavelength errors. Multigrid techniques use coarser grids in order to remove the low-frequency errors effectively. In Section 6.2 a preconditioning technique is described which uses a partition of the unknowns based on the sequence of grids in multigrid. After a renumbering of the unknowns according to this partition, L and U are obtained from an incomplete decomposition based on a drop tolerance as described in Chapter 2. In this technique a splitting $(LU, -R)$ of A is made, in which the elements r_{ij} of the residual matrix $R = A - LU$ satisfy

$$|r_{ij}| \leq \varepsilon_{ij} \text{ for } 1 \leq i,j \leq N$$

Herein ε_{ij} is a drop tolerance parameter which should be chosen carefully in order to obtain a proper incomplete decomposition of A. Once ε_{ij} is chosen, the factors L and U can be constructed as described in Section 2.3. This construction makes no restriction with respect to the sparsity pattern of A, and the computational complexity for the building of the incomplete decomposition is O(N).

In Section 6.3 results are presented from the above method applied to a large number of test problems described in the literature. These results show the robustness of the method. Some theoretical results concerning the convergence properties will be given in Section 6.4.

6.2 The preconditioning technique

Before an incomplete decomposition of A is made, a renumbering of the unknowns based on the multigrid idea is performed. Consider a sequence of nested grids $\Omega_1, \Omega_2, \ldots, \Omega_\gamma$, where $\Omega_\gamma \subset \Omega_{\gamma-1} \cdots \subset \Omega_1$. If all grids are uniform, Ω_m has mesh size $2^{m-1}h$, where h is the mesh size of the finest grid Ω_1. The set of unknowns at the m-th level is now defined by $W_m = \Omega_m \backslash \Omega_{m+1}$, where $\Omega_{\gamma+1} = \emptyset$. If the numbering within the levels is lexicographical, and if Dirichlet boundary conditions are used, we obtain for the inner grid points of a rectangular 8×8-grid with constant mesh size:

1	2	3	4	5	6
7	*28*	8	*29*	9	*30*
10	11	12	13	14	15
16	*31*	17	**36**	18	*32*
19	20	21	22	23	24
25	*33*	26	*34*	27	*35*

The points with numbers 1 to 27 belong to the first level W_1. Similarly, the two sets of points *28* to *35* and **36** belong to W_2 and W_3 respectively (note that the number of inner grid points in one direction does not necessarily have to be a power of 2). In Appendix A an algorithm for the generation of such a numbering is given in the more general case where the mesh is not uniform. Numbering the unknowns as described above results in a system of linear equations which can be written as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{6.1}$$

where x_1 is the vector containing the unknowns of the first level W_1, and x_2 those of the second grid Ω_2. This partitioning of the matrix can be repeated for the matrix in the lower-right corner until we arrive at the coarsest grid.

Remark 1. In order to obtain factors L and U which enable us to make an efficient implementation of the statement $y := (LU)^{-1}z$ on supercomputers, it is advantageous to use an appropriate choice for the ordering of unknowns within each level. For example, one can use a red-black ordering.

Fig. **??** shows the sparsity pattern of the coefficient matrix arising after a standard discretization of the Poisson equation on a uniform 10×10-grid with Neumann boundary conditions everywhere. The unknowns have been numbered level for level as described above, and within the separate levels we have used a red-black ordering. The size of the dots represents the absolute value of the corresponding matrix entries. The dashed lines are added in order to show the block structure of A, which arises after renumbering the unknowns as described above.

The preconditioning technique consists now of making a splitting $A = LU + R$ in which the elements r_{ij} all satisfy $|r_{ij}| \leq \varepsilon_{ij}$. We will show that it is advantageous to choose the drop tolerance small for the block in the lower-right corner. Suppose that ε_{ij} can be chosen in such a way that LU has the block structure

$$\begin{bmatrix} L_1U_1 & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ A_{21}U_1^{-1} & A_{22} - A_{21}U_1^{-1}L_1^{-1}A_{12} \end{bmatrix} \begin{bmatrix} U_1 & L_1^{-1}A_{12} \\ 0 & I \end{bmatrix} \qquad (6.2)$$

This implies that the residual matrix R has the block structure

$$\begin{bmatrix} L_1U_1 - A_{11} & 0 \\ 0 & 0 \end{bmatrix}$$

The vector Rx only contains components of the first level. With this type of preconditioner all low-frequency errors are eliminated immediately, and the iterative method only has to remove high-frequency errors with a wavelength in the order of the mesh size. As mentioned before, this can be done very effectively. In Section 6.4 we will prove that for a discretized Laplace operator, the choice (6.2) leads to a preconditioned matrix with a condition number which is bounded by 2. Of course, it is not realistic to use (6.2) as a preconditioner, because this requires the inverse of the block $A_{22} - A_{21}U_1^{-1}L_1^{-1}A_{12}$, but it is possible to choose ε_{ij} in such a way that one obtains a residual matrix with relatively small elements r_{ij} in the lower-right corner.

In the following we describe our choice for ε_{ij}. Suppose that A is obtained from a standard discretization of a Poisson equation in two dimensions on a uniform rectangular grid. After a renumbering of the unknowns, as mentioned before, we consider the corresponding block partitioning of the matrix A. The drop tolerance ε_{ij} is kept constant within each of the diagonal blocks, e.g. $\varepsilon_{ij} = \varepsilon^{(m)}$. Starting with $\varepsilon^{(1)}$ in the first diagonal block, corresponding with level W_1, we let the drop tolerance decrease by multiplying with a positive factor $c < 1$ at each new level. In the lower-triangular part ε_{ij} is chosen equal to ε_{ii}, and in the upper-triangular part it follows from symmetry. Fig. 6.1 shows the drop tolerance near the diagonal blocks corresponding with W_{m-1} and W_m. Herein $\varepsilon^{(m)} = c\varepsilon^{(m-1)} = c^{m-1}\varepsilon^{(1)}$. For most problems the choice of $\varepsilon^{(1)}$ and c is not very critical. In 2D-problems $c = 0.2$ is a reasonable choice, but in 3D-problems the optimal value for c is smaller. The above choice can also be used very well for different mesh sizes h and k

Figure 6.1: The drop tolerance for isotropic problems.

in horizontal and vertical direction, respectively, as long as h and k have the same order of magnitude. When $h \ll k$ or $k \ll h$, it is much better to choose ε_{ij} as

$$\varepsilon_{ij} = \varepsilon \frac{h^2 k^2}{(h^2 + k^2)\rho_{ij}^2} \tag{6.3}$$

Herein ρ_{ij} is the distance between the two grid points with numbers i and j, and ε is a parameter which has to be chosen in advance (the numerical experiments will demonstrate that good results are obtained with $\varepsilon \approx 0.2$). In Section 6.4 we will motivate this choice for the drop tolerance.

When a non-uniform grid is used, we propose the following choice for the drop tolerance

$$\varepsilon_{ij} = \varepsilon \frac{h_{ij}^2 k_{ij}^2}{(h_{ij}^2 + k_{ij}^2)\rho_{ij}^2} \tag{6.4}$$

where h_{ij} and k_{ij} are the minimum of the mesh sizes in respectively horizontal and vertical direction at points i and j. In the special case where a uniform grid is used with the same mesh size in the horizontal and vertical direction, (6.4) gives similar results as when ε_{ij} is decreased by multiplying with a factor c at each new level as described above. In that case, it is cheaper to choose ε_{ij} as shown in Fig. 6.1, because this choice does not require the distance ρ_{ij} between the points i and j.

Once the drop tolerance ε_{ij} is given, the incomplete LU-decomposition of A can be constructed as described in Section 2.3. During the construction of the factors L and U all elements which are neglected are lumped on the main diagonal as suggested by Gustafsson [17]. When constructing an incomplete Choleski-decomposition of a symmetric matrix, it is possible to exploit the symmetry as described in Section 2.4. The calculation time for the construction of the preconditioner can roughly be halved, and the storage requirements for both A and the incomplete decomposition can be made much lower. Fig. **??** shows the sparsity pattern of the matrix $L + L^T$ arising after the construction of an incomplete Choleski-decomposition of the coefficient matrix shown in Fig. **??**. The largest part of the factor L is very sparse, but in the lower-right corner the amount of fill-in increases, due to the decrease in drop tolerance. The residual matrix R is shown in Fig. **??**. Since all black points of the red-black ordering can be eliminated exactly, this matrix contains non-zero entries only in the lower-right corner.

6.3 Numerical experiments

In order to demonstrate the preconditioning technique described in the previous section, we show the results of seven different situations. Examples 3 to 5 were taken from [50], and problem 6 was taken from [11]. All numerical experiments were carried out in double precision on an HP-720 workstation, and with the iterative method applied to the preconditioned system $L^{-1}AU^{-1}\tilde{x} = L^{-1}b$, where $\tilde{x} = Ux$. In all cases, the initial solution was some random vector.

It appeared to be advantageous to number the unknowns in the separate groups according to a red-black ordering. This numbering has also the advantage that the resulting method can be implemented more efficiently on supercomputers. This means that the best ordering for scalar computers is also optimal for vector and parallel computers. Unless mentioned otherwise, the preconditioning technique was combined with the conjugate gradient method when the matrix is symmetric, and with Bi-CGSTAB when the matrix is non-symmetric. When the sparsity pattern of the factors L and U was chosen in such a way that all elements of the residual matrix are in absolute value less than ε, this is indicated with (M)ILU(ε). When the technique described in the previous section was used, this is indicated with NGIC(ε) or NGILU(ε), where NGIC stands for Nested Grids Incomplete Choleski. When NGIC(ε) was combined with the conjugate gradient method, this is indicated with NGICCG(ε). In the first 6 examples, NGILU and NGIC were implemented as shown in Fig. 6.1, where the parameter c was chosen equal to 0.2, except for the discretized Poisson equation in three dimensions, where c was chosen equal to 0.05. Only in the last example, where a non-uniform grid was used, we used the drop tolerance given by (6.4).

When the sparsity pattern of the matrix $L + U$ was taken the same as that of A, this is indicated with *standard* (M)ILU. When A stems from a discretization of a steady convection-diffusion equation in two or three dimensions, and a standard (M)ILU decomposition is used, the matrix-vector multiplication $y := L^{-1}AU^{-1}\tilde{x}$ can be implemented efficiently using the Eisenstat implementation (see Section 1.4), so that this matrix-vector multiplication requires approximately $11N$ flops in two dimensions and $15N$ flops for 3D-problems. This efficient implementation was used whenever it was possible.

Example 1. The first example shows the results of solving a Poisson equation on the unit square $[0, 1] \times [0, 1]$ with Neumann boundary conditions everywhere. This problem is of interest in computational fluid dynamics for calculating the pressure. In Chapter 1 this example was also used as a test problem for standard MICCG. Since the level of the solution is not fixed, the coefficient matrix is singular. Therefore, the conjugate gradient method was implemented as described in [21]. Table 6.1 shows the convergence behaviour of the conjugate gradient method for the discretized Poisson equation over a rectangular $M \times M$-grid with constant mesh size $1/(M - 1)$. The number of unknowns is M^2. As a preconditioning technique we used NGIC(0.2). The second column shows the number of iteration steps required to fulfil the stopping criterion

$$\|L^{-1}(b - AL^{-T}\tilde{x}^{(n)})\|_2 < 10^{-6}\|L^{-1}(b - AL^{-T}\tilde{x}^{(0)})\|_2$$

where $\tilde{x}^{(n)} = L^T x^{(n)}$. The third column shows the number of non-zero elements in L divided by the number of unknowns. The fourth and fifth column show the CPU-times per unknown necessary for the iteration process and the construction of the preconditioner, respectively. These times are given in milli-seconds, and they have been measured on an HP-720 workstation. The last column shows the number of floating point operations (flops) per unknown necessary for the iteration process. From this table we conclude that both the number of iteration steps and the number of flops per unknown are approximately constant. The number of non-zero elements per row necessary for the preconditioner increases only slightly with M.

The results of the conjugate gradient method combined with several incomplete Choleski-decompositions are summarised in Fig. 6.2. It shows the number of flops per unknown

Table 6.1: Numerical results for the discretized Poisson equation, equidistant $M \times M$-grid.

M	# CG it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
32	8	5.4	0.049	0.049	319
64	9	5.6	0.073	0.049	364
128	9	5.8	0.078	0.051	372
256	9	5.9	0.081	0.055	376
512	9	6.0	0.083	0.059	380

Figure 6.2: Numerical results for a discretized Poisson problem.

which is necessary for the iteration process versus the number of unknowns. In order to improve the efficiency of MICCG, we applied the small perturbations of the main diagonal as described in [21]: before the modified incomplete Choleski-decomposition was made, all diagonal elements were multiplied with a factor $1 + 10h^2$. As shown in Example 1 of Chapter 1, these perturbations decrease the number of MICCG iteration steps considerably. The results of Fig. 6.2 clearly show the effect of choosing different incomplete Choleski-decompositions. With standard ICCG the number of flops per unknown grows very strongly with the number of unknowns. Standard MICCG performs much better, but the amount of work per unknown still grows with mesh-refinement. With NGICCG as described in Section 6.2, the number of flops per unknown is approximately constant.

In order to demonstrate the technique of Section 6.2 in 3 dimensions, Fig. 6.3 shows the corresponding results for a Poisson equation on the cube $[0, 1] \times [0, 1] \times [0, 1]$ with again Neumann boundary conditions everywhere. We used NGICCG with $\varepsilon = 0.5$, and at every

Figure 6.3: Numerical results for discretized Poisson problem in 3D.

new level the drop tolerance parameter was decreased by multiplying with a factor 0.05. As mentioned before, in the 3D-case the parameter c has to be chosen smaller than in two-dimensional problems. Again we see that with NGICCG the amount of work per unknown hardly increases with mesh refinement, but the difference with standard MICCG is less than in the two-dimensional case. First of all, this is due to the fact that the number of grid points in one direction is significantly smaller. Secondly, the number of flops using MICCG is $\mathrm{O}(N^{5/4})$ in two dimensions and $\mathrm{O}(N^{7/6})$ for 3D-problems. Thirdly, since A contains more diagonals than in the 2D-case, the advantage of the efficient Eisenstat implementation is greater than in two dimensions.

Example 2. Our second example is Example 3 from Chapter 1 and 2. We consider the discretized steady convection-diffusion equation

$$-\Delta u(x,y) + 1000x^3 \frac{\partial}{\partial x} u(x,y) - 1000y^3 \frac{\partial}{\partial y} u(x,y) = f(x,y)$$

on the square $[0,1] \times [0,1]$ with Dirichlet boundary conditions everywhere. For the discretization we used a rectangular grid with constant mesh size $1/(M{+}1)$ in both directions, and central differences for all derivatives. As a stopping criterion we used

$$\|L^{-1}(b - AU^{-1}\tilde{x}^{(n)})\|_2 < 10^{-10}\|L^{-1}(b - AU^{-1}\tilde{x}^{(0)})\|_2$$

The results of Bi-CGSTAB combined with NGILU(0.2) are shown in Table 6.2. Again the CPU-times per unknown are given in milli-seconds. The third column shows the number of non-zero entries in $L + U$ divided by the number of unknowns.

Table 6.2: Numerical results of NGILU(0.2)-preconditioning for Example 2.

M	# it.	# nonz./row	CPU-time it.	CPU-time prec.	flops/unknown
32	9	16.5	0.19	0.088	918
64	9	15.7	0.23	0.090	889
128	11	13.4	0.25	0.073	986
256	12	11.7	0.28	0.063	994
400	11	11.1	0.25	0.064	884

Even for small values of M, when the mesh-Péclet number is much larger than 2, the construction of the factors L and U does not break down. However, the number of non-zero entries in one row of $L + U$ is larger than in Table 6.1. For smaller mesh sizes the results are similar to those of Table 6.1. Again the number of iteration steps is approximately constant when the mesh is refined.

We compared standard ILU-preconditioning with ILUkl-preconditioning as described in Section 1.4. The results are summarised in Fig. 6.4 giving the number of flops per unknown against the number of unknowns. The results of the standard MILU-preconditioning cannot be shown, because even for $M = 128$ the construction of the incomplete decomposition breaks down due to the generation of small elements on the main diagonal. For $M = 256$ the construction of the MILU-decomposition can be completed, but Bi-CGSTAB combined with the resulting preconditioner does not converge.

We observed that the convergence behaviour of Bi-CGSTAB with standard ILU as preconditioner is very irregular, whereas with NGILU(0.2) this behaviour is much smoother. In case of a very irregular convergence behaviour, the accuracy of the calculated solution may be spoiled by cancellation effects ([50]).

Example 3. The third example shows the effect of discontinuous coefficients in the partial

Figure 6.4: Number of flops per unknown for Example 2 on various grids.

differential equation. The system of linear equations stems from a five-point finite-difference discretization of the partial differential equation

$$-\frac{\partial}{\partial x}(D(x,y)\frac{\partial}{\partial x}u(x,y)) - \frac{\partial}{\partial y}(D(x,y)\frac{\partial}{\partial y}u(x,y)) = 1$$

over the unit square, with Dirichlet boundary condition along $y = 0$, and Neumann conditions on the other boundaries. The function $D(x,y)$ is defined as

$$D(x,y) = 1000 \text{ for } 0.1 \le x, y \le 0.9 \qquad \text{and} \qquad D(x,y) = 1 \text{ elsewhere.}$$

The equation was discretized on a 151 × 151-grid, hence the number of unknowns N is equal to 22500. In [49] it is shown that for this problem the conjugate gradient method loses orthogonality among the residuals in a very early phase of the iteration process. As shown in [50], it is therefore better to use Bi-CGSTAB as an iterative method, instead of CG, although the coefficient matrix is symmetric positive definite. Fig. 6.5 shows the 10-logarithm of the Euclidean norm of the residuals $b - Ax^{(n)}$ versus the number of flops per unknown which is necessary for Bi-CGSTAB combined with various preconditioners. This figure shows that with standard IC or with MIC(0.005), the convergence behaviour can be irregular. From the results of Fig. 6.5 we conclude that, for this problem, Bi-CGSTAB combined with NGIC(0.05) is a very efficient method with a convergence behaviour which is relatively smooth.

Figure 6.5: Convergence behaviour of Bi-CGSTAB for Example 3.

Example 4. In this example, taken from [50, 56], we consider

$$-\Delta u(x,y) + (\frac{\partial}{\partial x}(a(x,y)u(x,y)) + a(x,y)\frac{\partial}{\partial x}u(x,y))/2 = 1$$

on the unit square, with Dirichlet boundary conditions along all boundaries, and $a(x,y) = 20e^{3.5(x^2+y^2)}$. The partial differential equation was discretized on a rectangular grid with constant mesh size 1/201 in both directions. All derivatives were discretized by central differences. Fig. 6.6 shows the 10-logarithm of the 2-norm of the vector $b - Ax^{(n)}$ versus the number of flops per unknown for Bi-CGSTAB combined with various preconditioners. Again we see that with standard (M)ILU, the convergence behaviour can be very irregular, whereas with NGILU(0.2), this behaviour is very smooth.

Example 5. The following test problem is a simplified aquifer problem and was also taken from [50]. The non-symmetric system of linear equations stems from the discretization of the steady convection-diffusion equation

$$-\frac{\partial}{\partial x}(A(x,y)\frac{\partial}{\partial x}u(x,y)) - \frac{\partial}{\partial y}(A(x,y)\frac{\partial}{\partial y}u(x,y)) + B(x,y)\frac{\partial}{\partial x}u(x,y) = F(x,y)$$

on the square $[0,1] \times [0,1]$. The diffusion coefficient function $A(x,y)$ is given in Fig. 6.7, in which the dashed area indicates the region in which $A(x,y) = 10000$. Further we have that the function $B(x,y)$ is defined by

$$B(x,y) = 2e^{2(x^2+y^2)}$$

Figure 6.6: Convergence behaviour of Bi-CGSTAB for Example 4.

Figure 6.7: The diffusion coefficient for Example 5.

and $F(x, y) = 0$ everywhere, except for the small square in the center, where $F(x, y) = 100$. We have Dirichlet boundary conditions along all boundaries as shown in Fig. 6.7. The partial differential equation was discretized on a rectangular grid with mesh size $1/M$, and we used central differences for all derivatives. The number of unknowns is $(M-1)^2$. Fig. 6.8 shows the number of flops per unknown necessary for Bi-CGSTAB versus the

Figure 6.8: Numerical results for Example 5.

number of unknowns. As a stopping criterion for the iteration process we used

$$\|L^{-1}(b - AU^{-1}\tilde{x}^{(n)})\|_2 < 10^{-8}\|L^{-1}(b - AU^{-1}\tilde{x}^{(0)})\|_2$$

Again the standard MILU-decomposition breaks down, and therefore no results of this preconditioning technique can be shown. The construction of the MILU(0.02)-preconditioner does not break down. The results of Fig. 6.8 clearly show the effect of the choice of the preconditioner. With standard ILU and with MILU(0.02) the number of flops per unknown increases very strongly with mesh-refinement, whereas with NGILU(0.2) the amount of work per unknown is approximately constant. With a mesh size of 1/320 in both directions, the average number of entries in one row of the matrix $L + U$ is 12.4, and no more than 11 iteration steps of Bi-CGSTAB are necessary in order to fulfil the stopping criterion.

Fig. 6.9 shows the convergence behaviour of preconditioned Bi-CGSTAB for a mesh size of 1/200, and again we note the relatively smooth convergence behaviour of the iterative method combined with NGILU(0.2).

Example 6. The following test problem is the same as Example 2 of Chapter 1 and

Figure 6.9: Convergence behaviour of Bi-CGSTAB for Example 5 on a 201 × 201-grid.

Chapter 2. For convenience, the partial differential equation has been written down below once again.

$$-10^{-5}\Delta u(x,y) + d(x,y)\frac{\partial}{\partial x}u(x,y) + e(x,y)\frac{\partial}{\partial y}u(x,y) = 0$$

with

$$d(x,y) = 4x(x-1)(1-2y), \qquad e(x,y) = -4y(y-1)(1-2x)$$

This PDE was discretized on a rectangular grid with constant mesh size $1/(M+1)$ in both directions. Using upwind finite differences for the first-order derivatives, the coefficient matrix becomes a non-symmetric M-matrix. Fig. 6.10 shows the convergence behaviour of Bi-CGSTAB combined with various preconditioners when a 202 × 202-grid is used. From the results one can see that Bi-CGSTAB combined with standard ILU stagnates. One obtains a better preconditioner by allowing more fill-in during the construction of the incomplete decomposition (ILU(0.01)). Again the technique described in Section 6.2 gives the best preconditioner: the convergence of Bi-CGSTAB combined with NGILU(0.1) is about twice as fast as when Bi-CGSTAB is combined with ILU(0.02).

Fig. 6.11 shows the number of flops per unknown necessary for the iteration process versus the number of unknowns. As a stopping criterion we used

$$\|L^{-1}(b - AL^{-T}\tilde{x}^{(n)})\|_2 < 10^{-8}\|L^{-1}(b - AL^{-T}\tilde{x}^{(0)})\|_2$$

From the results of Fig. 6.11 we conclude that with NGILU(0.1), the amount of work necessary for Bi-CGSTAB increases only slightly with mesh refinement.

Figure 6.10: Convergence behaviour of Bi-CGSTAB for Example 6 on a 202×202-grid.

Figure 6.11: Number of flops per unknown for Example 6 on various grids.

Table 6.3: Numerical results for Example 6 with NGILU(0.1), $c = 0.2$.

M	# it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
32	6	11.8	0.11	0.049	499
64	7	13.4	0.17	0.073	627
130	10	14.8	0.27	0.096	952
256	12	16.0	0.34	0.124	1200

Table 6.4: Numerical results for Example 6 with NGILU(0.05), $c = 0.2$.

M	# it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
32	5	14.1	0.12	0.068	462
64	8	16.4	0.22	0.105	813
130	7	18.3	0.23	0.134	764
256	11	20.0	0.36	0.183	1276

The numerical results of Bi-CGSTAB combined with NGILU(0.1) and NGILU(0.05) are shown in Table 6.3 and 6.4, respectively. The second column gives the number of iteration steps necessary to fulfil the stopping criterion. Columns 4 and 5 give the CPU-time per unknown necessary for the iteration process and the construction of the preconditioner respectively. Again these times are gives in milliseconds and they have been measured on an HP-720 workstation. The last column gives the number of flops per unknown necessary for the iteration process. From the difference between these tables we conclude that the choice of ε is not very critical for the CPU-time necessary for preconditioned Bi-CGSTAB. The results agree quite well with the results obtained in [11] with the best multigrid code.

Example 7. The convergence rate of multigrid algorithms based on point relaxation smoothers deteriorates for problems with strong anisotropies. Anisotropic discrete operators arise, for example, in problems in which the differential operator is discretized on highly stretched grids. Therefore, we tested the convergence of NGICCG for the system of linear equations which arises after the discretization of a Poisson equation on a highly stretched rectangular grid. We took the same test problem as in [6]. We consider $-\Delta u(x,y) = f(x,y)$ on the unit square with Neumann boundary conditions along $x = 0$ and $y = 0$, and Dirichlet boundary conditions along the other boundaries. An exponential stretching of an $(M+1) \times (M+1)$-grid was used in both coordinate directions. The stretching was done in such a way that the minimum mesh sizes occur near the boundaries with Neumann boundary conditions, and the largest mesh sizes occur near the boundaries $x = 1$ and $y = 1$ as shown in Fig. 6.12.

Table 6.5 lists the number of iteration steps of the conjugate gradient method for various choices of M and h_{max}/h_{min}. As a stopping criterion for the iteration process we

Figure 6.12: The geometry for Example 7.

Table 6.5: Numerical results for Example 7.

h_{max}/h_{min}	$M = 32$	$M = 64$	$M = 128$	$M = 256$
10	6 (9.5)	6 (9.7)	7 (9.7)	7 (9.8)
100	5 (10.8)	5 (11.5)	7 (11.2)	9 (11.2)
10^3	4 (12.1)	5 (16.3)	6 (15.7)	10 (15.1)
10^4	3 (11.6)	4 (15.6)	5 (22.0)	9 (25.1)
10^5	3 (11.0)	3 (15.3)	4 (21.7)	7 (28.4)
10^6	3 (10.8)	3 (15.0)	5 (22.4)	6 (31.0)

used

$$\|L^{-1}(b - AL^{-T}\tilde{x}^{(n)})\|_2 < 10^{-6}\|L^{-1}(b - AL^{-T}\tilde{x}^{(0)})\|_2$$

For the drop tolerance ε_{ij} we used (6.4) with $\varepsilon = 0.2$. The average number of entries in one row of L is given between brackets. When a strong refinement of the grid is used, the factor L contains more elements, but the number of CG iteration steps combined with the resulting preconditioner is very small.

6.4 Preliminary analysis

In this section, we take a closer look on the condition number of the preconditioned matrix in case the coefficient matrix A stems from a standard discretization of a Poisson equation on a rectangular $M \times M$-grid with constant mesh size h. Hence A has the five-point stencil

$$\frac{1}{4h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

We want to construct an incomplete Choleski-decomposition of A in such a way that the preconditioned matrix $L^{-1}AL^{-T}$ has a condition number as small as possible. Although we only consider the case when A is symmetric and positive (semi-)definite, the results of Section 6.3 show that the preconditioning technique is of interest for a much broader class of problems.

We will see that for a certain choice of the drop tolerance ε_{ij}, the condition number of $L^{-1}AL^{-T}$ behaves like $\mathrm{O}(h^{-\alpha})$ with $\alpha < \frac{1}{2}$, and that this particular choice of ε_{ij} is far from optimal. First we will show that, even if the largest part of L is chosen very sparse, we still can construct a preconditioner in such a way that cond($L^{-1}AL^{-T}$)≤ 2.

The grid points are divided into 4 parts: the red and black points of the first level, and the red and black points of the second level. For example:

$$\begin{array}{cccccccccccccc}
\ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ \\
\circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star \\
\ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ \\
\circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet \\
\ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ \\
\circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star \\
\ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ \\
\circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet & \circ & \star & \circ & \bullet \\
\ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ & \ast & \circ
\end{array}$$

First all '$\circ$' are numbered, then all '$\ast$', all '$\bullet$' and all '$\star$'-points. Suppose that the main diagonal of A is scaled to unity, then according to this partition, A has the block structure

$$\begin{bmatrix} I_1 & A_{12} & A_{13} & A_{14} \\ A_{21} & I_2 & 0 & 0 \\ A_{31} & 0 & I_3 & 0 \\ A_{41} & 0 & 0 & I_4 \end{bmatrix} \tag{6.5}$$

where $A_{ij}^T = A_{ji}$, and I_j is an identity matrix. We want to make a complete Choleski-decomposition LL^T of the matrix $A - R$ in such a way that L is sparse. The residual matrix R is taken so small that LL^T resembles A. We consider a choice of LL^T which leads to a residual matrix with the block structure

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & R_{22} & 0 & 0 \\ 0 & 0 & R_{33} & 0 \\ 0 & 0 & 0 & R_{44} \end{bmatrix} \tag{6.6}$$

After eliminating all '$\circ$'-points we obtain the Schur-complement

$$S_2 = \begin{bmatrix} I_2 - A_{21}A_{12} & -A_{21}A_{13} & -A_{21}A_{14} \\ -A_{31}A_{12} & I_3 - A_{31}A_{13} & -A_{31}A_{14} \\ -A_{41}A_{12} & -A_{41}A_{13} & I_4 - A_{41}A_{14} \end{bmatrix}$$

It can easily be verified that $A_{31}A_{13} = \frac{1}{4}\, I_3$ and $A_{41}A_{14} = \frac{1}{4}\, I_4$, hence

$$S_2 = \begin{bmatrix} I_2 - A_{21}A_{12} & -A_{21}A_{13} & -A_{21}A_{14} \\ -A_{31}A_{12} & \frac{3}{4}\, I_3 & -A_{31}A_{14} \\ -A_{41}A_{12} & -A_{41}A_{13} & \frac{3}{4}\, I_4 \end{bmatrix}$$

By lumping all off-diagonal elements onto the diagonal, the first diagonal block $I_2 - A_{21}A_{12}$ is approximated by the diagonal matrix $\frac{1}{2}\, I_2$. This implies that $R_{22} = \frac{1}{2}\, I_2 - A_{21}A_{12}$ with row sums zero. After this modification of the Schur-complement we can easily eliminate all '$\ast$'-points. The next Schur-complement becomes

$$S_3 = \begin{bmatrix} \frac{3}{4}\, I_3 - 2A_{31}A_{12}A_{21}A_{13} & -A_{31}A_{14} - 2A_{31}A_{12}A_{21}A_{14} \\ -A_{41}A_{13} - 2A_{41}A_{12}A_{21}A_{13} & \frac{3}{4}\, I_4 - 2A_{41}A_{12}A_{21}A_{14} \end{bmatrix}$$

It can be shown that $2A_{41}A_{12}A_{21}A_{13} = A_{41}A_{13}$, thus S_3 can be rewritten as

$$S_3 = \begin{bmatrix} \frac{3}{4}\, I_3 - 2A_{31}A_{12}A_{21}A_{13} & -2A_{31}A_{14} \\ -2A_{41}A_{13} & \frac{3}{4}\, I_4 - 2A_{41}A_{12}A_{21}A_{14} \end{bmatrix}$$

Just as before, we approximate the first diagonal block $\frac{3}{4}\, I_3 - 2A_{31}A_{12}A_{21}A_{13}$ by the diagonal matrix $\frac{1}{2}\, I_3$. Hence $R_{33} = \frac{1}{4}\, I_3 - 2A_{31}A_{12}A_{21}A_{13}$ with row sums zero. After eliminating all '•'-points, it follows that R_{44} is the residual matrix in the approximation of the Schur-complement

$$S_4 = \tfrac{3}{4}\, I_4 - 2A_{41}A_{12}A_{21}A_{14} - 8A_{41}A_{13}A_{31}A_{14}$$

It can be shown that this block is a sparse, positive semidefinite matrix with the nine-point stencil

$$\frac{1}{32}\begin{bmatrix} & & -1 & & \\ & -3 & & -3 & \\ -1 & & 16 & & -1 \\ & -3 & & -3 & \\ & & -1 & & \end{bmatrix}$$

One way to proceed is to lump the entries -1/32 on the main diagonal, make a block decomposition of the approximate Schur-complement in the same way as in (6.5), and then make an incomplete decomposition as described above for A. This is exactly the nested recursive two-level decomposition method described by Axelsson and Eijkhout [4]. They have proved that if the incomplete Choleski-decomposition is made in this way, the condition number of $L^{-1}AL^{-T}$ behaves like $\mathrm{O}(h^{-0.69})$, where h is the mesh size.

For a system coming from the discretization on an $M \times M$-grid the so-called Repeated Red-Black (RRB)-method), which has been described by Brand in [7], follows the same strategy for the first ${}^2\log M$ levels. When the level number exceeds ${}^2\log M$, the drop tolerance is set to zero. In [7] it has been proven that when A stems from the discretization of a Poisson equation on a uniform rectangular grid with Dirichlet boundary conditions, this strategy results in an incomplete Choleski-decomposition which is such that the condition number of $L^{-1}AL^{-T}$ behaves like $\mathrm{O}(h^{-\alpha})$, with $\alpha < \frac{1}{2}$. This is illustrated by the results of Table 6.6 giving the number of CG iterations steps combined with RRB-preconditioning necessary to fulfil the stopping criterion

$$\|L^{-1}(b - AL^{-T}\tilde{x}^{(n)})\|_2 < 10^{-6}\|L^{-1}(b - AL^{-T}\tilde{x}^{(0)})\|_2$$

We used Neumann boundary conditions everywhere. The mesh size is equal to $1/(M - 1)$, and the number of unknowns is M^2. The fourth and fifth column give the CPU-time per unknown for the iteration process and the incomplete Choleski-decomposition, respectively. These times are measured in milli-seconds on an HP-720 workstation. With RRB-preconditioning, one does not obtain grid-independent convergence. In [4] the nested two-level decomposition method is combined with nested polynomial approximations in order to obtain a method of optimal order of computational complexity. Reusken [35] uses a slightly different lumping strategy, and combines the incomplete decomposition with a multigrid technique in order to obtain grid-independent convergence. In this chapter, we

Table 6.6: Numerical results for the model problem with RRB-preconditioning.

M	# CG it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
32	10	5.0	0.049	0.039	375
64	13	5.1	0.103	0.051	485
128	13	5.2	0.104	0.047	491
256	16	5.2	0.128	0.055	599
512	17	5.2	0.138	0.064	638

follow a different approach: we make a more accurate incomplete Choleski-decomposition of the Schur-complement S_4. Table 6.7 shows the results when this block is approximated with an NGIC(0.01)-decomposition as described earlier. At every additional level the drop tolerance is decreased by multiplying with a factor $c = 0.2$ (this is the standard choice for c which is used in most of the numerical experiments in Section 6.3).

Table 6.7: Numerical results with an NGIC(0.01)-decomposition of S_4.

M	# CG it.	# nonz/row	CPU-time it.	CPU-time prec.	flops/unknown
32	8	5.6	0.049	0.049	327
64	8	6.0	0.071	0.061	341
128	8	6.2	0.073	0.061	348
256	8	6.3	0.077	0.071	352
512	8	6.4	0.081	0.080	355

In the sequel of this chapter we write $A_1 \preceq A_2$ if the matrix $A_2 - A_1$ is positive semidefinite, and $A_1 \prec A_2$ if $A_2 - A_1$ is positive definite.

Theorem 3 *Suppose that the non-singular matrix A stems from a standard discretization of the Poisson equation on a rectangular grid with constant mesh size, and that after renumbering of the unknowns and scaling, A has the block structure shown in (6.5). Furthermore, suppose that LL^T is an incomplete Choleski-decomposition of A, and $R = A - LL^T$ has the block structure*

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & R_{22} & 0 & 0 \\ 0 & 0 & R_{33} & 0 \\ 0 & 0 & 0 & R_{44} \end{bmatrix}$$

with $R_{22} = \frac{1}{2}\, I_2 - A_{21}A_{12}$, and $R_{33} = \frac{1}{4}\, I_3 - 2A_{31}A_{12}A_{21}A_{13}$. If, in addition, R_{44} satisfies

$$R_{44} \preceq \tfrac{1}{4}\, I_4 - 2A_{41}A_{12}A_{21}A_{14} \qquad (6.7)$$

then all eigenvalues λ of $L^{-1}AL^{-T}$ satisfy $1 \le \lambda \le 2$.

REMARK 2. It can easily be shown that the block $\frac{1}{4}\, I_4 - 2A_{41}A_{12}A_{21}A_{14}$ can be represented by the five-point stencil

$$\frac{1}{32}\begin{bmatrix} -1 & & -1 \\ & 4 & \\ -1 & & -1 \end{bmatrix}$$

so that it follows that this block is positive semidefinite.

To prove Theorem 3 we need the following lemmas:

Lemma 1 *Let the incomplete Choleski-decomposition of the non-singular matrix A be given by $LL^T = A - R$. If $R \succeq 0$ and $A - \alpha R \succeq 0$ for $\alpha > 1$, then*

$$I \preceq L^{-1}AL^{-T} \preceq \frac{\alpha}{\alpha - 1} I$$

and hence the condition number of $L^{-1}AL^{-T}$ is bounded by $\alpha/(\alpha - 1)$.

REMARK 3. Suppose that A is an M-matrix, and L results from a modified incomplete decomposition of A as described in [17]. In that case R has non-positive elements outside the main diagonal and row sums zero. Hence from Gerschgorin's theorem it follows that the condition $R \succeq 0$ is fulfilled.

Proof of Lemma 1: first we prove that all eigenvalues λ of $L^{-1}AL^{-T}$ satisfy $1 \leq \lambda$. From $LL^T = A - R$ it follows that $L^{-1}AL^{-T} = I + L^{-1}RL^{-T}$, and together with $R \succeq 0$ we obtain $I \preceq L^{-1}AL^{-T}$.

$A - \alpha R \succeq 0$ implies that $A - \alpha(A - LL^T) \succeq 0$, hence $(1 - \alpha)L^{-1}AL^{-T} + \alpha I \succeq 0$, and this proves Lemma 1. $\Box$

Lemma 2 *Suppose that a symmetric matrix is partitioned as*

$$\begin{bmatrix} E & B \\ B^T & C \end{bmatrix}$$

where E and C are square and positive definite. This matrix is positive semidefinite if and only if $C \succeq B^TE^{-1}B$. It is positive definite if and only if $C \succ B^TE^{-1}B$.

The proof of this lemma can be found in [19]. As a consequence, we obtain

Lemma 3 *Suppose that E and C are square and positive definite. Then $C \succeq B^TE^{-1}B$ if and only if $E \succeq BC^{-1}B^T$, and $C \succ B^TE^{-1}B$ if and only if $E \succ BC^{-1}B^T$.*

Proof of Theorem 3.
Suppose that $1 < \alpha < 2$. Since R is positive semidefinite, it follows with Lemma 1 that $A - \alpha R \succeq 0$ is sufficient for having

$$I \preceq L^{-1}AL^{-T} \preceq \frac{\alpha}{\alpha - 1} I$$

It can be shown that, under the conditions of Theorem 3, the blocks $I_j - \alpha R_{jj}$ for $j = 1$, 2 and 3 are all positive definite. Hence we can apply Lemma 2 and 3 with $E = I_1$ to show that $A - \alpha R \succeq 0$ is equivalent to

$$A_{12}(I_2 - \alpha R_{22})^{-1}A_{21} + A_{13}(I_3 - \alpha R_{33})^{-1}A_{31} + A_{14}(I_4 - \alpha R_{44})^{-1}A_{41} \preceq I_1$$

In the following we will show that, under the conditions of Theorem 3, this holds. Since R_{22} is positive semidefinite and $\alpha < 2$, it follows that $\alpha R_{22} \preceq 2R_{22}$. This last equation can be rewritten as $2A_{21}A_{12} \preceq I_2 - \alpha R_{22}$, and by applying Lemma 3 it follows that this is equivalent to

$$A_{12}(I_2 - \alpha R_{22})^{-1}A_{21} \preceq \tfrac{1}{2}\, I_1$$

Therefore, it suffices to prove that

$$\begin{aligned} & A_{13}(I_3 - \alpha R_{33})^{-1}A_{31} + A_{14}(I_4 - \alpha R_{44})^{-1}A_{41} \preceq \tfrac{1}{2}\, I_1 \\ \Longleftrightarrow\quad & \begin{bmatrix} A_{13} & A_{14} \end{bmatrix} \begin{bmatrix} (I_3 - \alpha R_{33})^{-1} & 0 \\ 0 & (I_4 - \alpha R_{44})^{-1} \end{bmatrix} \begin{bmatrix} A_{31} \\ A_{41} \end{bmatrix} \preceq \tfrac{1}{2}\, I_1 \end{aligned}$$

From Lemma 3 it follows that this is equivalent to

$$2\begin{bmatrix} A_{31} \\ A_{41} \end{bmatrix} \begin{bmatrix} A_{13} & A_{14} \end{bmatrix} \preceq \begin{bmatrix} I_3 - \alpha R_{33} & 0 \\ 0 & I_4 - \alpha R_{44} \end{bmatrix}$$

Using $A_{31}A_{13} = \frac{1}{4}\, I_3$ and $A_{41}A_{14} = \frac{1}{4}\, I_4$, and by using the definition of R_{33}, this can be rewritten as

$$\begin{bmatrix} (\frac{1}{2} - \frac{1}{4}\,\alpha)I_3 + 2\alpha A_{31}A_{12}A_{21}A_{13} & -2A_{31}A_{14} \\ -2A_{41}A_{13} & \frac{1}{2}\, I_4 - \alpha R_{44} \end{bmatrix} \succeq 0 \tag{6.8}$$

It is sufficient to prove (6.8) for R_{44} replaced by its upperbound, so that it suffices to prove that

$$\tfrac{1}{4}\begin{bmatrix} (2-\alpha)I_3 + 8\alpha A_{31}A_{12}A_{21}A_{13} & -8A_{31}A_{14} \\ -8A_{41}A_{13} & (2-\alpha)I_4 + 8\alpha A_{41}A_{12}A_{21}A_{14} \end{bmatrix} \succeq 0 \tag{6.9}$$

It can be proven that this matrix can be represented by the stencil

$$\frac{1}{32}\begin{bmatrix} \alpha & -4 & \alpha \\ -4 & 16-4\alpha & -4 \\ \alpha & -4 & \alpha \end{bmatrix}$$

with meshsize $2h$. The eigenvalues of this matrix can be found by substituting a Fourier component $\exp\left(i(f_1 x + f_2 y)\right)$. Hence it follows that the eigenvalues are given by

$$\lambda_{k,l} = \tfrac{1}{32}\,(16 - 8(a+b) - 4\alpha + 4\alpha ab) \tag{6.10}$$

where $a = \cos k\pi 2h$ and $b = \cos l\pi 2h$. One can easily prove that all eigenvalues given by (6.10) are not negative, so that (6.9) holds. Hence we have shown that, for $1 < \alpha < 2$, the condition (6.7) is sufficient for having

$$I \preceq L^{-1}AL^{-T} \preceq \frac{\alpha}{\alpha - 1} I$$

This can be shown for α chosen arbitrarily close to 2, which completes the proof of Theorem 3. $\Box$

REMARK 4. Theorem 3 shows that, even if the largest part of L is chosen very sparse, we still can obtain an incomplete Choleski-decomposition such that cond$(L^{-1}AL^{-T})\leq 2$. However, it should be noted that the condition on R_{44} in Theorem 3 is difficult to check.

Choice of the drop tolerance.
Now we want to motivate the choice of the drop tolerance. Though the approach is heuristic, more insight in the behaviour of the algorithm is obtained in the course of this derivation.

The criterion is based on Lemma 1, hence for some $\alpha > 1$ we try to find a condition which assures that $A - \alpha R$ is positive semidefinite. Our starting point will be the necessary condition for this requirement that

$$u_k^T(A - \alpha R)u_k \geq 0$$

where u_k is a normalized eigenvector of A. This condition is sufficient if the eigenvectors u_k are equal or sufficiently close to the eigenvectors of $A - \alpha R$. As we need indices below for other purposes, we denote u_k by q, and rewrite our condition in the form

$$\alpha q^T Rq \leq \lambda \tag{6.11}$$

where λ is the corresponding eigenvalue to q. Note that (6.11) is a strong requirement for low-frequency components. E.g., for the constant eigenvector with eigenvalue $\lambda = 0$, we see that Gustafsson's modification is necessary. We observed that R is of near block-diagonal form. Therefore, we restrict our analysis to the strict block-diagonal form

$$R = \begin{bmatrix} 0 & & & & & \\ & R_{22} & & & & \\ & & R_{33} & & & \\ & & & R_{44} & & \\ & & & & \ddots & \\ & & & & & R_{\gamma\gamma} \end{bmatrix} \tag{6.12}$$

Consider $q^T Rq$. From (6.12) it follows that

$$q^T Rq = \sum_{i=2}^{\gamma} q_i^T R_{ii} q_i$$

where, using the same partition, $q = [q_1, q_2, \ldots, q_\gamma]^T$. Next suppose that

$$q_i^T R_{ii} q_i \leq \mu_i q_i^T q_i$$

then

$$q^T Rq = \sum_{i=2}^{\gamma} q_i^T R_{ii} q_i \leq \sum_{i=2}^{\gamma} \mu_i q_i^T q_i$$

Now, for all i it holds that q_i is a restriction of q to a coarser grid. For a smooth eigenvector q, one can find a smooth eigenfunction $u(x, y)$ of the partial differential equation, such that the components of the vector q are given by the value of $u(x, y)$ in the grid points. In that case, both $q^T q$ and $q_i^T q_i$ represent, up to a factor, an approximation to the integral

$$\int_0^1 \int_0^1 u(x, y) dx dy$$

The ratio between these two approximations can be found by substituting a constant in the respective integration rules, thus it is approximately equal to the ratio of the lengths of the vectors q and q_i. Hence $q_i^T q_i \approx 2^{-i}$ (e.g. q_1 has about half the length of q). Furthermore, suppose that $\mu_i = \mu_0 \beta^i$ and $q_i^T q_i \leq 2^{-i}$, then

$$q^T R q \leq \mu_0 \sum_{i=2}^{\gamma} 2^{-i} \beta^i = \tfrac{1}{4}\, \mu_0 \beta^2 \sum_{i=2}^{\gamma} (\beta/2)^{i-2} = \tfrac{1}{4}\, \mu_0 \beta^2 \frac{1 - (\beta/2)^{\gamma-1}}{1 - \beta/2}$$

Now if $\beta < 2$ then for all γ

$$q^T R q \leq \frac{\mu_0 \beta^2}{4 - 2\beta}$$

Hence, under the assumptions made, condition (6.11) is satisfied if

$$\frac{\mu_0 \beta^2}{4 - 2\beta} < \lambda/\alpha,\ \alpha > 1$$

It should be noted that the condition $\beta < 2$ allows a growth of μ_i with i. Furthermore we observed that far enough away from the boundaries the blocks R_{ii} have a very regular structure and constant coefficients. This allows us to use Fourier analysis in order to obtain μ_i. Note also that for our model problem the eigenvectors q are in fact Fourier components with a special choice for the frequency.

Hence, we approximate R_{ii} by a linear combination of elementary symmetric finite-difference stencils. In our current algorithm all contributions to R_{ii} are balanced. More precise, define $\mu_i = \sum \delta_j$ where the sum is over all stencils, then the eigenvalues δ_j satisfy the condition

$$\delta_j < \delta_{\max} = \hat{\varepsilon} \lambda, \qquad \text{for all } j \tag{6.13}$$

Hence the stencil is allowed in R_{ii} if (6.13) holds, otherwise it should be included in the decomposition. In the numerical experiments we observed a moderate growth of fill-in with increasing i. Hence μ_i will grow with the number of significant stencils, but this increase is not so large that $\beta = \mu_{i+1}/\mu_i \geq 2$. In the following we will make condition (6.13) more specific.

Suppose that A stems from a standard five-point discretization of a Poisson equation on an $M_x \times M_y$-grid with mesh size h and k in the horizontal and vertical direction respectively. When the diagonal is scaled to unity, A has the five-point stencil

$$\frac{1}{2(h^2 + k^2)} \begin{bmatrix} & -h^2 & \\ -k^2 & 2(h^2 + k^2) & -k^2 \\ & -h^2 & \end{bmatrix}$$

The eigenvalues for a Fourier component $\exp(i(f_1x + f_2y))$ for this stencil are

$$\begin{aligned}\lambda_A(f_1, f_2) &= \frac{1}{2(h^2+k^2)}[2(h^2+k^2) - 2k^2\cos f_1h - 2h^2\cos f_2k] \\ &= \frac{2}{(h^2+k^2)}[k^2\sin^2(\tfrac{1}{2}\ f_1h) + h^2\sin^2(\tfrac{1}{2}\ f_2k)] \qquad (6.14)\end{aligned}$$

where $|f_1h|, |f_2k| < \pi$. Suppose that R_{ii} is the sum of stencils of the form

$$\epsilon\begin{bmatrix} -1 & & -1 \\ & 4 & \\ -1 & & -1 \end{bmatrix}$$

with mesh sizes nh and mk in the horizontal and vertical direction respectively. Here, n or m may be zero! Note that we may take the sum of the coefficients equal to zero, because the row sums of R are zero. The eigenvalue δ_j of this stencil is given by

$$\begin{aligned}\delta_j(f_1, f_2) &= 4\epsilon[1 - \cos f_1nh\cos f_2mk] \\ &= 8\epsilon[\sin^2(\tfrac{n}{2}\ f_1h) + \sin^2(\tfrac{m}{2}\ f_2k) - 2\sin^2(\tfrac{n}{2}\ f_1h)\sin^2(\tfrac{m}{2}\ f_2k)] \qquad (6.15)\end{aligned}$$

Now $\delta_j(f_1, f_2)$ can be bounded from above by

$$\begin{aligned}\delta_j(f_1, f_2) &\le 8\epsilon[\sin^2(\tfrac{n}{2}\ f_1h) + \sin^2(\tfrac{m}{2}\ f_2k)] \\ &\le 8\epsilon[n^2\sin^2(\tfrac{1}{2}\ f_1h) + m^2\sin^2(\tfrac{1}{2}\ f_2k)] \qquad (6.16)\end{aligned}$$

Note that in the first step equality occurs only if nf_1h or mf_2k is a multiple of π, and in the second step if $f_1 = f_2 = 0$. The second step holds only because of the limitations put on f_1h, f_2k given below equation (6.14).

Condition (6.13) is, in this specific case, $\delta_j(f_1, f_2) < \hat{\varepsilon}\lambda_A(f_1, f_2)$. This is certainly true if it holds for δ_j replaced by the upper bound (6.16), hence if

$$\epsilon < \frac{\hat{\varepsilon}(k^2\sin^2(\frac{1}{2}\ f_1h) + h^2\sin^2(\frac{1}{2}\ f_2k))}{4(h^2+k^2)(n^2\sin^2(\frac{1}{2}\ f_1h) + m^2\sin^2(\frac{1}{2}\ f_2k))}$$

In its turn this condition is certainly fulfilled if

$$\epsilon < \frac{\hat{\varepsilon}h^2k^2}{4(h^2+k^2)}\min\left(\frac{1}{(nh)^2}, \frac{1}{(mk)^2}\right)$$

We obtained slightly better results with the following choice

$$\epsilon < \frac{\hat{\varepsilon}h^2k^2}{4(h^2+k^2)(n^2h^2+m^2k^2)}$$

In our implementation with varying meshes we use

$$\varepsilon_{ij} = \varepsilon\frac{h_{ij}^2k_{ij}^2}{(h_{ij}^2+k_{ij}^2)\rho_{ij}^2} \qquad (6.17)$$

Herein h_{ij} and k_{ij} are the minimum of the mesh sizes in respectively horizontal and vertical direction at points i and j, ρ_{ij} is the distance between the two grid points with numbers i and j, and ε is a parameter which has to be chosen in advance. As mentioned before, in the special case of a constant mesh size in both directions, it is cheaper to choose ε_{ij} as shown if Fig. 6.1. From (6.17) it follows that c should be less than $\frac{1}{4}$.

Numerical verification of the condition number.
To obtain insight in the condition number of $L^{-1}AL^{-T}$, we have computed the maximum eigenvalue of this matrix, which is according to Lemma 1 equal to the spectral condition number, by an iterative method. The coefficient matrix results from a standard five-point discretization of the Poisson equation on a rectangular grid with constant mesh size $1/(M+1)$. We used Dirichlet boundary conditions everywhere. At every new level the drop tolerance ε was decreased by multiplying with $c = 0.2$. The results are summarised in Fig. 6.13 which shows the computed largest eigenvalue versus M. For comparison, we also give the results of the RRB-method. From these results we conclude that with the

Figure 6.13: Condition number of $L^{-1}AL^{-T}$ versus M.

RRB-method the condition number of $L^{-1}AL^{-T}$ increases only very slightly with mesh-refinement. With NGIC(0.2) and NGIC(0.1) the results are even better: the condition number hardly increases with mesh-refinement. The calculated largest eigenvalues of $L^{-1}AL^{-T}$ for various preconditioning techniques are listed in Table 6.8. The average number of entries in one row of L is given in brackets. From the results we conclude that when $c = 0.1$ and $\varepsilon = 0.2$, the difference between 2 and the largest eigenvalue behaves like $\mathrm{O}(h^2)$. From the difference between the last two columns of this table we conclude

Table 6.8: Calculated condition numbers of $L^{-1}AL^{-T}$.

M	RRB	NGIC	NGIC	NGIC
		$\varepsilon = c = 0.2$	$\varepsilon = 0.1$, $c = 0.2$	$\varepsilon = 0.2$, $c = 0.1$
32	2.39(5.0)	1.990(5.2)	1.577(6.9)	1.98992(6.4)
64	3.00(5.1)	2.162(5.5)	1.762(7.5)	1.99753(7.1)
128	3.73(5.2)	2.378(5.7)	1.898(7.8)	1.99939(7.7)
256	4.63(5.2)	2.532(5.9)	1.990(8.0)	2.000(8.1)
512	5.73(5.2)	2.647(6.0)	2.057(8.2)	2.000(8.3)
1024	7.07(5.2)	2.724(6.0)	2.102(8.2)	2.000(8.4)

that the choice of the parameters ε and c is not very critical.

6.5 Conclusions and discussion

In this chapter, a preconditioning technique is described which shows grid-independent convergence when combined with any conjugate gradient-like method. This technique is relatively easy to implement. Only an incomplete LU-decomposition of A has to be made. Essential in the method is the choice of a drop tolerance controlling the size of $R = A - LU$ and the ordering of the unknowns. The ordering is similar to that in multigrid approaches and makes it possible to construct an incomplete LU-decomposition which can be used in eliminating effectively both high- and low-frequency errors. For model problems, an expression for the drop tolerance has been derived in such a way that the condition number of the preconditioned matrix has a small upperbound independent of the dimension of the system. Similar expressions for less trivial problems seem to be possible. The method is demonstrated on a variety of well-known elliptic test problems described in the literature. These include strongly varying coefficients, advective terms and grid refinement. In all cases, the method is much cheaper than standard (M)ICCG. This difference is more pronounced for the really difficult problems and increases with the dimension. The convergence behaviour is, in contrast to that of standard (M)ICCG, always smooth, which is advantageous to the construction of stopping criteria when the linear solver is used as an inner-iteration method, for example, within some inexact Newton method.

The computational work consists of three parts: the construction of the preconditioner, its application, and the number of iterations steps. From the numerical experiments we conclude that the construction of the preconditioner grows linearly with the number of unknowns. Its application is linear with the fill-in. This fill-in is about a factor two larger than that of A, which is modest and comparable to that of a standard ILU-decomposition. With the present method an iteration step is significantly cheaper than one multigrid step, due to additional smoothing operations needed in the latter method. Some preliminary results show that the number of iterations steps is comparable, and hence, in many cases, the computational work of the present method is less than that of multigrid.

In our view, the results in this chapter show the potential of the introduced method. Of course, more analysis (e.g. rigorous convergence proof) and further optimization (e.g. choice of the drop tolerance and ordering of the unknowns) is needed. Moreover, implementations on modern computers which exploit the parallelism present in the algorithm must be studied. A different topic is the application of these ideas to problems which are far from elliptic, such as the Navier-Stokes equations. Research on these subjects is in progress.

A Renumbering of the grid points

Consider a non-uniform rectangular $M_x \times M_y$-grid. Let the set of all grid points be denoted by Ω_T, and suppose that this set is the Cartesian product of two sets X_T and Y_T, where X_T and Y_T contain all x- and y-coordinates respectively. The first step is to divide X_T into a number of subsets using a bisection-like method. The first set is given by

$$X_1 = \{x_1^{(1)}, x_2^{(1)}\}$$

where $x_1^{(1)}$ and $x_2^{(1)}$ are the smallest and the largest element of X_T, respectively. The second set is obtained from X_1 as follows

$$X_2 = X_1 \cup \{\xi \in X_T | x_1^{(1)} < \xi < x_2^{(1)}, \text{ and } |\xi - (x_1^{(1)} + x_2^{(1)})/2| \text{ is minimal.}\}$$

We continue in this manner in the following way: suppose that $X_k = \{x_1^{(k)}, \ldots, x_{n_k}^{(k)}\}$ is ordered in such a way that $x_1^{(k)} < x_2^{(k)} < \cdots x_{n_k}^{(k)}$. Next X_{k+1} is defined as follows.

$$X_{k+1} = X_k \cup \{\xi \in X_T | \text{there are two points } x_j^{(k)} \text{ and } x_{j+1}^{(k)} \text{ in } X_k \text{ such that}$$

$$x_j^{(k)} < \xi < x_{j+1}^{(k)}, \text{ and } |\xi - (x_j^{(k)} + x_{j+1}^{(k)})/2| \text{ is minimal}\}$$

This process is stopped until a set X_k is obtained which equals X_T, so that we construct the subsets $X_1 \ldots X_k$ in such a way that $X_1 \subset X_2 \cdots X_{k-1} \subset X_k = X_T$. Now we perform the same steps for the set Y_T, hence we construct the subsets $Y_1 \ldots Y_l$ in such a way that $Y_1 \subset Y_2 \cdots Y_{l-1} \subset Y_l = Y_T$.

Herewith the set of grid points Ω_T is divided into the subsets

$$\begin{aligned}
&\Omega_1 = X_1 \times Y_1,\\
&\Omega_2 = (X_2 \times Y_2) \backslash \Omega_1\\
&\vdots\\
&\Omega_n = (X_n \times Y_n) \backslash \Omega_{n-1}
\end{aligned}$$

Herein we assume that $X_{k+j} = X_T$ and $Y_{l+j} = Y_T$ for $j > 0$. We continue in this way until all grid points are in some subset Ω_i. Finally, the renumbering of the grid points consists now of first numbering all elements of Ω_n, then all elements of Ω_{n-1} and so on. In the special case of a constant mesh size, it is not necessary to look for the minimum of $|\xi - (x_j^{(k)} + x_{j+1}^{(k)})/2|$. Suppose that a grid with constant mesh size consists of the grid points

$$(x_i, y_j),\ 1 \leq i \leq M_x,\ 1 \leq j \leq M_y$$

Algorithm 6.1. Renumbering of the grid points of a uniform rectangular grid.

```
$N := M_x \times M_y$;
$count := 0$;
$st := 1$;
$tw := 2$;
REPEAT
   $switch := tw$;
   FOR $i := st$ STEP $st$ TO $M_x$ DO
      BEGIN IF $switch = st$ THEN $switch := tw$
                                  ELSE $switch := st$;
            FOR $j := st$ STEP $switch$ TO $M_y$ DO
               BEGIN $count := count + 1$;
                     $numb[i, j] := count$
               END {$j$-loop}
      END; {$i$-loop}
   $st := tw$;
   $tw := 2 * st$
UNTIL $count = N$;
```

In that case, the renumbering can be done efficiently by Algorithm 6.1. After this algorithm has been executed, the integer $numb[i, j]$ contains the new number of the point (x_i, y_j).

As mentioned before, we can choose a red-black ordering within each level, which enables us to make a more efficient implementation of the statement $y := (LU)^{-1}z$ on supercomputers. This can be done, for example, by defining those grid points (x_i, y_j) with $i + j$ even as the red points, and the remaining points as the black points. From numerical experiments it appeared that this renumbering within the separate levels has only little effect on the number of iteration steps.

7. Conclusions and suggestions for future research

In this thesis, we have considered the development of preconditioners for large sparse systems of linear equations $Ax = b$. Primarily, we have studied preconditioners based on incomplete decompositions.

In Chapter 1, we have shown that for many problems, conjugate gradient-like methods combined with a proper preconditioner perform significantly better than stationary iterative methods such as Jacobi, Gauss-Seidel and SOR. In Chapter 1 we also have considered some preconditioning techniques for matrices with a very regular sparsity pattern. When A is symmetric positive definite, it appeared that in most cases the system can be solved efficiently by the MICCG-method, in which the sparsity pattern of $L + L^T$ is the same as that of A. This regular sparsity pattern enables us to implement the MICCG-method efficiently on supercomputers, and to use the Eisenstat implementation for the matrix-vector multiplication.

When the coefficient matrix stems from a discretization of a steady convection-diffusion equation with dominating convective parts, both the robustness and efficiency of the preconditioner can be improved by allowing some extra fill-in in the factors L and U.

In the next chapter, we have described a preconditioning technique which can be used for non-symmetric matrices with an arbitrary sparsity pattern. This allows the use of complicated geometries and an irregular node numbering. Once the sparsity pattern of $L+U$ is prescribed, Algorithm 2.3 calculates the incomplete decomposition. In Chapter 2 we also have described an ILU-decomposition in which the sparsity pattern of $L + U$ is based on a threshold parameter ε. In this technique, which is denoted by ILU(ε)-preconditioning, a splitting $(LU, -R)$ of the scaled matrix is constructed, in such a way that all entries of R are in absolute value smaller than ε. When neglecting the effect of roundoff errors, we make an exact decomposition of the matrix $A - R$.

In Chapter 3, three different methods for constructing the sparsity pattern of $L + U$ have been described and compared with each other, using a model for the computation of the temperature distribution in a block of concrete. From several numerical experiments it appeared that ILU(ε)-preconditioning gives the best preconditioners. It appeared that with the latter technique, the construction of the preconditioner grows linearly with the number of unknowns. The choice of ε is not very critical, and all values in the range 0.001 to 0.01 perform well. Many practical problems (for example, time-dependent problems) require the solution of a large number of linear systems in which the coefficient matrices do not differ very much. In such cases, it is possible to use the same preconditioner several times, and choosing a relatively small value for ε can be favourable.

From several numerical experiments it appeared that ILU(ε)-preconditioning performs very well on scalar machines, because it strongly reduces the number of floating point operations. However, a drawback of this technique is that the sparsity pattern of L and U can be irregular, even when A has a very regular sparsity pattern. Hence all non-zero entries have to be addressed indirectly, which can be disadvantageous to the efficiency on supercomputers. More research is necessary for developing an incomplete decomposition in which the sparsity pattern of $L + U$ is based on a drop tolerance, but also chosen in such a way that indirect addressing can be avoided.

When the coefficient matrix has a regular sparsity pattern, an alternative to avoid indirect addressing is to use a so-called *standard* (M)ILU-decomposition, e.g. an incomplete decomposition in which the sparsity pattern of $L + U$ is taken the same as that of A. In many cases, this strategy can be combined with level-scheduling in order to deal with the recursive character of algorithms for solving the lower- and upper-triangular systems. In Chapter 4, we have compared this technique with MILU(ε)-preconditioning on three different computers: an HP-720, a Convex C2 and a NEC SX-3. The systems of linear equations arising in a numerical model for Boussinesq equations were used as test problems. The efficiency of a particular preconditioner strongly depends on the choice of the computer. On a scalar machine like the HP-720, (M)ILU(ε)-preconditioning performs best, because with this technique, the total number of floating point operations is relatively small. On the CONVEX and the NEC, this technique performs poorly, due to the indirect addressing. On these computers, level-scheduling strongly improves the efficiency of standard (M)ILU-decompositions, so that on these machines, it is much better to use the latter technique.

In Chapter 4, we also used polynomial preconditioning techniques, which lend themselves more naturally for implementation on supercomputers. On the Convex, a standard (M)ILU-decomposition combined with level-scheduling performs better than a polynomial preconditioner, whereas on the NEC SX-3, the latter appeared to be slightly more efficient than the first. However, standard (M)ILU combined with level-scheduling is a proper choice on *all* three computers, whereas on a sequential machine polynomial preconditioning performs poorly.

In Chapter 5, we have described a preconditioning technique for the full system of equations arising after linearization of the incompressible Navier-Stokes equations. Since a straightforward sparse incomplete decomposition for this system is not possible, we developed a pre-preconditioner. After applying this pre-preconditioner to the system, a sparse incomplete decomposition of the resulting linear system can be made in a similar way as the construction of the factors L and U of the ILU(ε)-preconditioning described in Chapter 2. This preconditioning technique performs well for both upwind and central differences.

No restriction is made with respect to the sparsity pattern of the coefficient matrix.

Hence this approach can be successful, even when the geometry of the domain under consideration is very complex. A drawback of this generality is that again indirect addressing is unavoidable, which is disadvantageous to an efficient implementation on vector and parallel computers. However, for simple geometries like a rectangle or an L-shaped domain, we can obtain a coefficient matrix with a regular sparsity pattern. More research is required for developing algorithms which can exploit this sparsity pattern, so that the preconditioner can be implemented efficiently on all types of computers.

In Chapter 6, a preconditioning technique is described which shows, in many cases, the optimal order of computational complexity when combined with any conjugate gradient-like method: the number of iterations steps does not increase with mesh refinement. This leads to a method which is much easier to implement than multigrid techniques, where proper smoothers and restriction and prolongation operators are required. The new preconditioning technique only requires an ordering of the unknowns based on the different levels of multigrid, the choice of a drop tolerance controlling the size of $R = A - LU$, and an incomplete LU-decomposition as described in Chapter 2. Therefore, this technique is referred to as Nested Grids ILU-decomposition (NGILU). The reordering enables us to choose the drop tolerance in such a way that both high- and low-frequency of the error can be eliminated.

For model problems, an expression for this drop tolerance has been derived in such a way that the condition number of the preconditioned matrix has a small upperbound independent of the dimension of the system. Several numerical experiments demonstrate that the NGILU-decomposition is of interest in much more general cases, for example, when the mesh size is far from constant, or when the PDE contains dominating convective parts. As is clearly demonstrated by Fig. 6.8 and 6.9, the convergence rate of Bi-CGSTAB combined with an NGILU-decomposition is excellent, even when the coefficients in the PDE are strongly discontinuous. The results show that the convergence behaviour of a CG-like method combined with NGILU is always smooth. This is advantageous to the construction of stopping criteria when the linear solver is used as an inner-iteration method, for example, within some inexact Newton method.

Of course, more analysis (e.g. rigorous convergence proof) and further optimization (e.g. choice of the drop tolerance and ordering of the unknowns) is needed. In the present method, the renumbering of the unknowns is based on a rectangular grid. An algorithm in which a similar renumbering is performed using only the sparsity pattern of A, so that it can be used in case of unstructured grids and irregular geometries, is worth further study. Also, more research is required to exploit the parallelism present in the algorithm, and to study the possibilities for using the ideas of Chapter 6 for solving the discretized incompressible Navier-Stokes equations.

æ

Bibliography

[1] M.A. Ajiz and A. Jennings. A robust incomplete cholesky-conjugate gradient algorithm. *Int. J. Num. Methods in Eng.*, 20:949–966, 1984.

[2] K.E. Atkinson. *An introduction to numerical Analysis.* John Wiley & Sons, Inc., 1978.

[3] O. Axelsson. Solution of linear systems of equations: Iterative methods. In V.A. Barker, editor, *Sparse Matrix Techniques: Copenhagen*, pages 1–51. Springer-Verlag, Berlin, 1977.

[4] O. Axelsson and V. Eijkhout. The nested recursive two-level factorization method for nine-point difference matrices. *SIAM J. Sci. Statist. Comput.*, 12:1373–1400, 1991.

[5] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Numer. Math.*, 48:479–498, 1986.

[6] E.F.F. Botta and F.W. Wubs. The convergence behaviour of iterative methods on severely stretched grids. *Int. J. Num. Methods in Eng.*, 36:3333–3350, 1993.

[7] Cl.W. Brand. An incomplete-factorization preconditioning using red-black ordering. *Numer. Math.*, 61:433–454, 1992.

[8] A.M. Bruaset. *Preconditioners for discretized elliptic problems.* PhD thesis, University of Oslo, 1992. Department of informatics.

[9] P. Chin, E.F. D'Azevedo, P.A. Forsyth, and W.P. Tang. Preconditioned conjugate gradient methods for the incompressible Navier-stokes equations. *Int. J. for Num. Methods in Fluids*, 15:273–295, 1992.

[10] E.D. de Goede. *Numerical methods for the three-dimensional shallow water equations on supercomputers.* PhD thesis, University of Amsterdam, 1992.

[11] P.M. de Zeeuw. Matrixdependent prolongations and restrictions in a blackbox multigrid solver. *Journal of Comp. and Appl. Math.*, 3:1–27, 1990.

[12] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Solving linear systems on vector and shared memory computers.* SIAM, 1991.

[13] I.S. Duff and G.A. Meurant. The effect of ordering on preconditioned conjugate gradient. *BIT*, 29:635–657, 1989.

[14] S.C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comput.*, 2:1–4, 1981.

[15] R. Fletcher. Conjugate gradient methods for indefinite systems. *Lecture notes in Mathematics*, 506, 1976.

[16] U. Ghia, K.N. Ghia, and C.T. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *J. of Comput. Phys.*, 48:387–411, 1982.

[17] I. Gustafsson. A class of 1:st order factorization methods. *BIT*, 18:142–156, 1978.

[18] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.

[19] R.A. Horn and C.R. Johnson. *Matrix Analysis.* Cambridge Universiy Press, 1985.

[20] C.P. Jackson and P.C. Robinson. A numerical study of various algorithms related to the preconditioned conjugate gradient method. *Int. J. Num. Methods in Eng.*, 21:1315–1338, 1985.

[21] E.F. Kaasschieter. Preconditioned conjugate gradients for solving singular systems. In A. Hadjidimos, editor, *Iterative methods for the solution of linear systems.* Elsevier, North-Holland, 1988. Reprinted from the Journal of Comp. and Appl. Math., Volume 24, Numbers 1 and 2.

[22] P.K. Khosla and S.G. Rubin. *J. of Comput. Phys.*, 48:387–411, 1982.

[23] H.P. Langtangen. Conjugate gradient methods and ILU-preconditioning of non-symmetric matrix systems with arbitrary sparsity patterns. *Int. J. for Num. Methods in Fluids*, 9:213–233, 1989.

[24] N.C. Markatos. Mathematical modelling of turbulent flow. *Appl. Math. Modelling*, 10:190–220, june 1986.

[25] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, 31:148–162, 1977.

[26] J.A. Meijerink and H.A. van der Vorst. Guidelines for the usage of incomplete decomposition in solving sets of linear equations as they occur in practical problems. *J. of Comput. Phys.*, 44:134–155, 1981.

[27] J. Mooiman and G.K. Verboom. A new Boussinesq model based on a positive definite Hamiltonian. In *International Conference on Computational methods in water resources. Denver*, pages 513–527, June 1992.

[28] K. Morgan, J. Periaux, and F. Thomasset. *Notes on numerical fluid mechanics. Volume 9.* Friedr. Vieweg and Sohn Braunschweig/Wiesbaden, 1984. Proceedings of a GAMM-workshop. Analysis of laminar flow over a backward facing step.

[29] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM. Trans. Math. Software*, 6:206–219, 1980.

[30] Y. Notay. Upper eigenvalue bounds and related modified incomplete factorization strategies. In R. Beauwens and P. de Groen, editors, *Iterative methods in linear algebra*. Elsevier, North-Holland, 1992. Proceedings of the IMACS-symposium.

[31] M. Papadrakakis and G. Pantazopoulos. A survey of quasi-Newton methods with reduced storage. *Int. J. Num. Methods in Eng.*, 36:1573–1596, 1993.

[32] S.V. Patankar. *Numerical heat transfer and fluid flow.* Hemisphere-McGraw Hill, 1980.

[33] G. Radicati and Y. Robert. Parallel conjugate gradient-like algorithms for solving nonsymmetric linear systems on a vector multiprocessor. *Parallel computing*, 11:223–239, 1989.

[34] J.K. Reid. On the method of Conjugate Gradients for the solution of large sparse systems of linear equations. In J.K. Reid, editor, *Large sparse sets of linear equations*, pages 231–254. Academic press, 1971.

[35] A. Reusken. Multigrid with matrix-dependent transfer operators for convection-diffusion problems. Technical Report RANA 93-10, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1993.

[36] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6:865–881, 1985.

[37] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Sci. Statist. Comput.*, 10(6):1200–1232, 1989.

[38] Y. Saad and M.H. Schultz. A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[39] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10:36–52, 1989.

[40] A. Spijkstra. Numerieke methoden voor de temperatuurberekening bij verhardend beton, 1987. Master's Thesis, University of Groningen.

[41] P.J. van der Houwen, C. Boon, and F.W. Wubs. Analysis of smoothing matrices for the preconditioning of elliptic difference equations. *Z. Angew. Math. Mech.*, 68:3–10, 1988.

[42] A. van der Ploeg. Preconditioning techniques for large sparse, non-symmetric matrices with arbitrary sparsity patterns. In R. Beauwens and P. de Groen, editors, *Iterative methods in linear algebra*, pages 173–179. Elsevier, North-Holland, 1992. Proceedings of the IMACS-symposium.

[43] A. van der Ploeg. Preconditioning techniques for non-symmetric matrices with application to temperature calculation of cooled concrete. *Int. J. Num. Methods in Eng.*, 35(6):1311–1328, 1992.

[44] A. van der Ploeg, E.F.F. Botta, and F.W. Wubs. Grid-independent convergence based on preconditioning techniques. Technical Report W-9310, Department of Mathematics, Groningen, 1993.

[45] A. van der Ploeg and F.W. Wubs. Vectorizable preconditioning techniques for solving the Boussinesq equations. In Ch. Hirsch, J.Periaux, and W. Kordulla, editors, *Computational fluid dynamics '92*, pages 481–488. Elsevier, North-Holland, 1992. Proceedings of the first European computational fluid dynamics conference.

[46] A. van der Ploeg and F.W. Wubs. The use of sparse matrix techniques for solving the incompressible Navier-Stokes equations. In W.H. Hackbusch and G. Wittum, editors, *Incomplete Decompositions (ILU)- Algorithms, Theory and Applications*, pages 113–121. Vieweg, Braunschweig, 1993. Proceedings of the eighth GAMM-seminar.

[47] A. van der Sluis and H.A. van der Vorst. The rate of convergence of conjugate gradients. *Numer. Math.*, 48:543–560, 1986.

[48] W.A. van der Veen and F.W. Wubs. A Hamiltonian approach to fairly low and fairly long gravity waves. Technical Report W-9314, Department of Mathematics, Groningen, 1993.

[49] H.A. van der Vorst. The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors. In O. Axelsson and L.Y. Kolotilina, editors, *Preconditioned Conjugate Gradient Methods*, volume 18. Springer Verlag, Berlin, 1990. Proceedings, Nijmegen, 1989.

[50] H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.

[51] G. van Diepen. Numerieke methoden voor niet-symmetrische lineaire stelsels bij de temperatuur berekening van door water gekoeld verhardend beton, 1988. Master's Thesis, University of Groningen.

[52] R.S. Varga. *Matrix iterative analysis.* Prentice Hall, 1962.

[53] A.E.P. Veldman and K. Rinzema. Playing with nonuniform grids. *J. of Engineering Math.*, 26:119–130, 1992.

[54] H.F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9:152–163, 1988.

[55] J.E. Welch, F.H. Harlow, J.P. Shannon, and B.J. Daly. The MAC method: a computing technique for solving viscous, incompressible, transient fluid-flow problems involving free surfaces, 1966. Los Alamos scientific laboratory report.

[56] O. Widlund. A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.

[57] G. Wittum. Multi-grid methods for Stokes and Navier-Stokes equations. *Numer. Math.*, 54:543–653, 1989.

[58] G. Wittum. On the convergence of multi-grid methods with transforming smoothers. *Numer. Math.*, 57:15–38, 1990.

[59] D.M. Young. *Iterative solution of large linear systems.* Academic Press, 1971.