LINEAR SYSTEM SOLVERS: SPARSE ITERATIVE METHODS

Henk A. Van der Vorst
Mathematical Institute
University of Utrecht
Utrecht, the Netherlands
e-mail: vorst@math.ruu.nl

Tony F. Chan[1]
Department of Mathematics
University of California at Los Angeles
Los Angeles, CA
e-mail: chan@math.ucla.edu

ABSTRACT

In this chapter we will present an overview of a number of related iterative methods for the solution of linear systems of equations. These methods are so-called Krylov projection type methods and they include popular methods as Conjugate Gradients, Bi-Conjugate Gradients, LSQR and GMRES. We will sketch how these methods can be derived from simple basic iteration formulas, and how they are interrelated.
Iterative schemes are usually considered as an alternative for the solution of linear sparse systems, like those arising in, e.g., finite element or finite difference approximation of (systems of) partial differential equations. The structure of the operators plays no explicit role in any of these schemes, and the operator may be given even as a rule or a subroutine.
Although these methods seem to be almost trivially parallellizable at first glance, this is sometimes a point of concern because of the inner products involved. We will consider this point in some detail.
Iterative methods are usually applied in combination with so-called preconditioning operators in order to further improve convergence properties. This aspect will receive more attention in a separate chapter in the same volume.

[1]The work of this author was partially supported by the National Science Foundation under contract ASC 92-01266, the Army Research Office under contracts DAAL03-91-G-0150 and subcontact from U. Tenn. DAAL03-91-C-0047, and ONR under contract ONR-N00014-92-J-1890.

1 Iterative methods versus direct methods

The present state of the art in numerical methods is that direct methods can be used as black boxes. This is by far not the case for iterative methods, at least not if we do not know about specific properties of the matrix of the linear system to be solved. And even then it is no trivial matter to decide when to stop the iteration process and to obtain a reasonable estimate of the approximation error in the result. Therefore, we start with some very global analysis of the complexity of Gaussian elimination and an iterative solution method for a model problem. This may serve as a motivation for the further study of iterative methods.

Gaussian elimination leads to fill-in, this makes the method often expensive. Usually large sparse matrices are related to some grid or network. In a 3D situation this leads typically to a bandwidth $\sim n^{\frac{2}{3}}$ ($= m^2$ and $m^3 = n$, $1/m$ the gridsize).

The number of flops is then typically $\mathcal{O}(nm^4) \sim n^{2\frac{1}{3}}$ (Golub and Van Loan, 1989) (Note: we assume that the sparsity structure is not particularly regular so that special computational variants can not be employed). If one has to solve many systems with different right-hand sides, then the costs for solving each system will vary like $n^{\frac{5}{3}}$.

If we assume that the matrix is symmetric positive definite then the Conjugate Gradient (CG) iteration method (which we will describe in more detail later on) can be used safely. The error reduction per iteration step of CG is $\sim \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$, with $\kappa = \|A\|_2 \|A^{-1}\|_2$ (Golub and van Loan, 1989).

For discretized second order pde's, over grids with gridsize $\frac{1}{m}$ we typically see $\kappa \sim m^2$. Hence, for such 3D problems we have that $\kappa \sim n^{\frac{2}{3}}$. For an error reduction of ϵ we must have that

$$\left(\frac{1 - \frac{1}{\sqrt{\kappa}}}{1 + \frac{1}{\sqrt{\kappa}}} \right)^j \approx (1 - \frac{2}{\sqrt{\kappa}})^j \approx e^{-\frac{2j}{\sqrt{\kappa}}} < \epsilon,$$

where j is the number of iteration steps to be carried out. Hence, it follows for j that

$$\Rightarrow \quad j \sim -\frac{\log \epsilon}{2} \sqrt{\kappa} \approx -\frac{\log \epsilon}{2} n^{\frac{1}{3}}.$$

If we assume the number of flops per iteration to be $\sim fn$ (f stands for the number of nonzeros per row of the matrix and the overhead per unknown introduced by the iterative scheme) then the number of flops necessary to achieve an error reduction by ϵ is $\sim -fn^{\frac{4}{3}} \log \epsilon$.

Conclusion: If we have to solve one system at a time, then for large n, or small f, or modest ϵ, **Iterative methods may be preferable.** If we have to solve many similar systems with different right-hand side, and if we assume their number to be so large that the costs for constructing the decomposition of A is relatively small per system, then it seems likely that for 2D problems direct methods may be more efficient, whereas for 3D problems this is still doubtful, since the flops count for a direct solution method varies like $n^{\frac{7}{3}}$, and the number of flops for the iterative solver (for the model situation) varies like $n^{\frac{4}{3}}$.

Example: The above given arguments are quite nicely illustrated by observations made by Simon (Simon, 1989). He expects that by the end of this century we will have to solve repeatedly linear problems with some 5×10^9 unknowns. For what he believes to be a model problem at that time, he has estimated the CPU time required by the most economic direct method, available at present, as $520,040$ years, provided that the computation can be carried out at a speed of 1 TFLOP. On the other hand, he estimates the CPU time for preconditioned conjugate gradients, assuming still a processing speed of 1 TFLOPS, as 575 seconds. Though we should not take it for granted that in particular a very sparse preconditioning part can be carried out at that high processing speed (for the direct solver this is more likely), we see that the differences in CPU time requirements are gigantic, indeed (we will come to this point in more detail).
Also the requirements for memory space for the iterative methods are typically smaller by orders of magnitude. This is often the argument for the usage of iterative methods in 2D situations, when flop counts for both classes of methods are more or less comparable.
Finally it should be noted that iterative methods can exploit good initial guesses, e.g., in time dependent problems. The preconditioner can often be chosen to adapt to the machine architecture.

Remarks:

- For special problems special methods may be faster: multigrid, fast poisson solvers.
- Storage considerations are often in favour of iterative methods.

- For matrices that are not positive definite symmetric the situation can be more problematic: it is often difficult to find the proper iterative method or a suitable preconditioner. However, for projection type methods, like GMRES, Bi-CG, CGS, and Bi-CGSTAB we often see that the flops counts vary as for CG.

The question remains how well iterative methods can take advantage of modern computer architectures. From Dongarra's linpack benchmark (Dongarra, 1990) it may be concluded that the solution of a dense linear system can (in principle) be computed with computational speeds close to peak speeds on most computers. This is already the case for systems of, say, order 50000 on parallel machines with as many as 1024 processors.
In sharp contrast with the dense case are the computational speeds reported for the preconditioned as well as the unpreconditioned conjugate gradient method: ICCG and CG, respectively (Dongarra and Van der Vorst, 1992). A test problem was taken, generated by discretizing a three-dimensional elliptic partial differential equation by the standard 7-point central difference scheme over a three-dimensional rectangular grid, with 100 unknowns in each direction ($m = 100$, $n = 1,000,000$). The observed computational speeds for several machines (1 processor in each case) are given in Table 1, and we see that the actual speeds are often modest compared with the peak speeds. The reason for this is mainly that only few flops can be carried out per data that has to be moved from memory to, e.g., the fast vector registers, and therefore, considerable time is spent relatively in data movement.

2 A basic iterative method

A very basic idea, that leads to many effective iterative solvers, is to to split the matrix of a given linear system as the sum of two matrices, one of which is a matrix that would have led to a system that can easily be solved. The most simple splitting we can think of is $A = I - (I - A)$. Given the linear system $Ax = b$, this splitting leads to the well-known Richardson iteration:

$$x_{i+1} = b + (I - A)x_i = x_i + r_i.$$

Table 1: Speed in Megaflops for 50 Iterations of the Iterative Techniques

Machine	optimized ICCG Mflops	Scaled CG Mflops	Peak Performance Mflops
NEC SX-3/22 (2.9 ns)	607	1124	2750
CRAY Y-MP C90 (4.2 ns)	444	737	952
CRAY 2 (4.1 ns)	96.0	149	500
IBM 9000 Model 820	39.6	74.6	444
IBM 9121 (15 ns)	10.6	25.4	133
DEC Vax/9000 (16 ns)	9.48	17.1	125
IBM RS/6000-550 (24 ns)	18.3	21.1	81
CONVEX C3210	15.8	19.1	50
Alliant FX2800	2.18	2.98	40

Multiplication by $-A$ and adding b gives

$$b - Ax_{i+1} = b - Ax_i - Ar_i$$

or

$$r_{i+1} = (I - A)r_i = (I - A)^{i+1} r_0 = P_{i+1}(A) r_0,$$

or, in terms of the error

$$A(x - x_{i+1}) = P_{i+1}(A) A(x - x_0)$$

$$\Rightarrow \quad x - x_{i+1} = P_{i+1}(A)(x - x_0).$$

In these expressions P_{i+1} is a (special) polynomial of degree $i+1$. Note that $P_{i+1}(0) = 1$.

Results obtained for the standard splitting can be easily generalized to other splittings, since the more general splitting $A = M - N = M - (M - A)$ can be rewritten as the standard splitting $B = I - (I - B)$ for the preconditioned matrix $B = M^{-1}A$. The theory of matrix splittings, and the analysis of the convergence of the corresponding iterative methods, is treated in depth in (Varga, 1962). We will not discuss this aspect here, since it is not relevant for our purposes at this stage. Instead of studying the basic iterative methods we will show how other more powerful iteration methods can be viewed as accelerated versions of the basic iteration methods. In the context of these accelarated methods, the matrix splittings become important in another way, since the matrix M of the splitting is often used to *precondition* the given system. That is, the iterative method is applied to, e.g., $M^{-1}Ax = M^{-1}b$.

From now on we will assume that $x_0 = 0$. This too does not mean a loss of generality, for the situation $x_0 \neq 0$ can through a simple linear transformation $z = x - x_0$ be transformed to the system

$$Az = b - Ax_0 = \tilde{b}$$

for which obviously $z_0 = 0$.

For the simple Richardson iteration it follows that

$$x_{i+1} = r_0 + r_1 + r_2 + \cdots + r_i = \sum_{j=0}^{i} (I - A)^j r_0$$

$$\in \{r_0, Ar_0, \ldots, A^i r_0\} \equiv K^{i+1}(A; r_0).$$

$K^{i+1}(A; r_0)$ is a subspace of dimension $i + 1$, generated by r_0 and A and is called the *Krylov subspace for* A *and* r_0. Apparently, the Richardson iteration delivers elements of Krylov subspaces of increasing dimension Note that the Richardson iteration generates a basis for the Krylov suspace, and this basis can be used to construct other approximations for the solution of $Ax = b$ as well.

3 The conjugate gradient method

In this section we will discuss various aspects of the popular conjugate gradient method. This method can be viewed as the archetype of various so-called Krylov subspace methods.

3.1 Sketch of the background

Of course, it would be desirable to have a method that could generate at low costs the x_i in the Krylov subspace for wich $\|x_i - x\|_2$ is mimimal, but this is not possible. However, if A is symmetric positive definite then $(x, y)_A \equiv (x, Ay)$ defines a proper innerproduct, and we may use this innerproduct for the mimimization: $\|x\|_A^2 = (x, Ax)$. So our aim is to find the $x_i \in K^i(A; r_0)$ for which $\|x_i - x\|_A$ is minimal:

$$\Rightarrow \quad x_i - x \perp_A K^i(A; r_0)$$

$$\Rightarrow \quad r_i \perp K^i(A; r_0).$$

In particular $r_1 \in \{r_0, Ar_0\}$. Assuming that $r_1 \neq \gamma r_0$ (it is easy to check that in that case r_0 is an eigenvector of A and the process could be stopped since the exact solution has then be obtained after only one iteration step), we see that $\{r_0, r_1\}$ form an orthogonal basis for $K^2(A; r_0)$.
By an induction argument we conclude that when the process does not find the exact solution at or before step i then

$$\{r_0, r_1, \ldots, r_i\} \text{ is an orthogonal basis for } \; K^{i+1}(A; r_0).$$

This leads to the idea to construct an orthogonal basis for the Krylov subspace, a basis of which is generated implicitly by the standard iteration anyway, and then to project $x_i - x$, with respect to the A-innerproduct, onto the Krylov subspace and to determine x_i from that.

We have seen that the r_j form an orthogonal basis for $K^i(A; r_0)$, but the next remarkable property is that they satisfy a 3-term recurrence relation when A is symmetric:

$$\alpha_{j+1} r_{j+1} = Ar_j - \beta_j r_j - \gamma_j r_{j-1}. \tag{1}$$

The proof is by induction.
The values of β_j and γ_j follow from the orthogonality of the residual vectors:

$$\beta_j = (r_j, Ar_j)/(r_j, r_j), \quad \text{and} \quad \gamma_j = (r_{j-1}, Ar_j)/(r_{j-1}, r_{j-1}).$$

The value of α_{j+1} determines the proper length of the new residual vector. From the relation $r_{j+1} = b - Ax_{j+1}$ and $x_{j+1} \in K^{j+1}(A; r_0)$ it follows that each residual can be written as r_0 plus powers of A times r_0. Comparing the coefficient for r_0 in the recurrence relation (1) leads to

$$\alpha_{j+1} + \beta_j + \gamma_j = 0.$$

We can view this 3-term recurrence relation slightly different as

$$Ar_j = \gamma_j r_{j-1} + \beta_j r_j + \alpha_{j+1} r_{j+1} \; ,$$

and if we consider r_j as being the j-th column of the matrix

$$R_i = (r_0, \ldots, r_{i-1})$$

then the recurrence relation says that A applied to a column of R_i results in the combination of three successive columns, or

$$AR_i = R_i \begin{pmatrix} \ddots & \ddots & & & \\ \ddots & \ddots & \gamma_j & & \\ & \ddots & \beta_j & \ddots & \\ & & \alpha_{j+1} & \ddots & \ddots \\ & & & \ddots & \ddots \end{pmatrix} + \alpha_i \begin{pmatrix} 0, 0, \ldots, r_i \end{pmatrix}$$

or

$$AR_i = R_i T_i + \alpha_i r_i e_i^T, \qquad (2)$$

in which T_i is an i by i tridiagonal matrix and e_i is the ith canonical unit vector in $\mathbb{R}^i$.

Since we are looking for a solution x_i in $K^i(A; r_0)$, that vector can be written as a combination of the basis vectors of the Krylov subspace, and hence

$$x_i = R_i y$$

(note that y has i components).
Furthermore, we have for x_i, for which the error in A-norm is minimal, that

$$R_i^T(Ax_i - b) = 0$$
$$\Rightarrow \quad R_i^T A R_i y - R_i^T b = 0.$$

Using equation (2) and the fact that r_i is orthogonal with respect to the columns of R_i we obtain

$$R_i^T R_i T_i y = \|r_0\|_2^2 e_1$$

Since $R_i^T R_i$ is a diagonal matrix with diagonal elements $\|r_0\|_2^2$, $\cdots$, $\|r_{i-1}\|_2^2$, we find the desired solution from

$$T_i y = e_1 \quad \Rightarrow \quad y \quad \Rightarrow \quad x_i = R_i y.$$

Note that so far we have only used the fact that A is symmetric and we have assumed that the matrix T_i is not singular. We will see later that this opens the possibility for several suitable iterative methods, among which the conjugate gradients method. The Krylov subspace method that has been derived here is known as the Lanczos method for symmetric systems (Lanczos, 1952).

The conjugate gradients method (Hestenes and Stiefel, 1954) is merely a variant on the above approach, which saves storage and computational effort. For, when solving the projected equations in the above way, we see that we have to save all columns of R_i throughout the process in order to recover the current iteration vector x_i. This can be done much more efficiently. If we assume that the matrix A is in addition positive definite then, because of the relation

$$R_i^T A R_i = R_i^T R_i T_i,$$

we conclude that T_i can be transformed by a rowscaling matrix $R_i^T R_i$ into a positive definite symmetric tridiagonal matrix (note that $R_i^T A R_i$ is positive definite for $y \in \mathbb{R}^{i+1}$). This implies that T_i can be LU decomposed without any pivoting:

$$T_i = L_i U_i,$$

with L_i lower unit bidiagonal and U_i upper bidiagonal, and this leads to the elegant short recurrences in the CG method. We omit the details of the precise derivation.
Note that the positive definiteness of A is only exploited as to guarantee the flawless decomposition of the implictly generated tridiagonal matrix T_i. Of course, one can construct a different decomposition of T_i when A is not positive definite. In SYMMLQ (Paige and Saunders, 1975) this is accomplished by an LQ decomposition, which also leads to an elegant method.
Still another possibillity is to minimize $\|Ax_i - b\|_2$ for x_i in the Krylov subspace. In the above approach this leads to a low dimensional least squares problem in which T_i is involved. The resulting method is known as MINRES (Paige and Saunders, 1975).

3.2 Computational notes

CG is most often used in combination with a suitable splitting $A = M - N$, and then M^{-1} is called the preconditioner. We will assume that M is also positive definite.
The following computational scheme represents preconditioned CG, for the solution of $Ax = b$ with preconditioner M^{-1}:

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0;$

Solve w_0 from $Mw_0 = r_0$;
$\rho_0 = (r_0, w_0)$
for $i = 0, 1, 2, ...$
  $p_i = w_i + \beta_{i-1}p_{i-1}$;
  $q_i = Ap_i$;
  $\alpha_i = \frac{\rho_i}{(p_i,q_i)}$
  $x_{i+1} = x_i + \alpha_i p_i$;
  $r_{i+1} = r_i - \alpha_i q_i$;
  if x_{i+1} accurate enough then quit;
  Solve w_{i+1} from $Mw_{i+1} = r_{i+1}$;
  $\rho_{i+1} = (r_{i+1}, w_{i+1})$;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end;

This formulation, which is quite popular, has the advantage that the preconditioner needs not be represented as the product of two factors, and it is also avoided to backtransform solutions and residuals, as is necessary when one applies CG to $L^{-1}AL^{-T}y = L^{-1}b$.

A very well-known upperbound for the iteration error is given by

$$\|x_i - x\|_A^2 \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2i} \|x_0 - x\|_A^2, \tag{3}$$

where κ is the condition number (the ratio of the largest and the smallest eigenvalue) of the matrix $M^{-1}A$.
A more formal presentation of CG can be found in (Golub and Van Loan, 1989).

3.3 Parallelism and data locality in CG

Most often, the conjugate gradients method is used in combination with some kind of preconditioning. Unfortunately, a popular class of preconditioners, based upon incomplete factorization of A, is not very well-suited for parallel implementation. We will discuss this aspect in a different chapter. Here we will assume that the preconditioner is chosen such that the parallelism in solving $My = z$ is comparable with the parallelism in computing Ap, for given p.
For CG it is also required that the preconditioner M^{-1} is symmetric positive definite. This aspect will play a role in our discussions since it shows how some properties of the preconditioner can be used sometimes to our advantage for an efficient implementation.

The scheme for preconditioned CG is given in Section 3.2. Note that in that scheme the updating of x and r can only start after the completion of the innerproduct required for α_i. Therefore, this innerproduct is a so-called synchronization point: all computation has to wait for completion of this operation. One can try to avoid such synchronization points as much as possible, or formulate CG in such a way that synchronization points can be taken together.
Since on a distributed memory machine communication is required to assemble the innerproduct, it would be nice if we could proceed with other useful computation while the communication takes place. However, as we see from our CG scheme, there is no possibility to overlap this communication time with useful computation. The same observation can be made for the updating of p, which can only take place after the completion of the innerproduct for β_i.
In our formulation of CG there are two such synchronization points, namely the computation of both innerproducts.
We consider here a variant of CG (Demmel et al, 1993), in which there is possibility to overlap all of the communication time with useful computations. This variant is just a rescheduled version of the original CG scheme, and is therefore precisely as stable. The key trick in this approach is to delay the updating of the solution vector by one iteration step.
It is assumed that M^{-1} can be written as $M^{-1} = (LL^T)^{-1}$.

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0; \alpha_{-1} = 0;$
$s = L^{-1}r_0;$
$\rho_0 = (s, s);$
for $i = 0, 1, 2, ...$
    $w_i = L^{-T}s;$ (0)
    $p_i = w_i + \beta_{i-1}p_{i-1};$ (1)
    $q_i = Ap_i;$ (2)
    $\gamma = (p_i, q_i);$ (3)
    $x_i = x_{i-1} + \alpha_{i-1}p_{i-1};$ (4)
    $\alpha_i = \frac{\rho_i}{\gamma};$ (5)
    $r_{i+1} = r_i - \alpha_i q_i;$ (6)
    $s = L^{-1}r_{i+1};$ (7)
    $\rho_{i+1} = (s, s);$ (8)
    if r_{i+1} *small enough* **then** (9)
        $x_{i+1} = x_i + \alpha_i p_i$

quit;
$\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end i;

The advantage of this scheme is that all communication for the inner products can be overlapped with other useful computation:

1. The communication required for the assembly of the innerproduct in (3) can be overlapped with the update for x (which could have been done in the previous iteration step).
2. The assembly of the innerproduct in (8) can be overlapped with the computation in (0). Also step (9) usually requires information such as the norm of the residual, which can be overlapped with (0).

For a further discussion on the above scheme, as well as on other approaches, see (Demmel et al, 1993). Vector processing and parallel computing aspects of CG are discussed in more depth in (Dongarra et al, 1991) and (Ortega, 1988).

4 Nonsymmetric problems

There are essentially three different ways to solve unsymmetric linear systems, while maintaining some kind of orthogonality between the residuals:

1. Solve the normal equations $A^T Ax = A^T b$ with conjugate gradients
2. Make all the residuals explicitly orthogonal in order to have an orthogonal basis for the Krylov subspace
3. Construct a basis for the Krylov subspace by a 3-term biorthogonality relation

4.1 Normal equations

The first solution seems rather obvious. However, it may have severe disadvantages because of the squaring of the condition number. This has as effects that the solution is more susceptible to errors in the right-hand side and that the rate of convergence of the CG procedure is much slower as for a comparable symmetric system with a matrix with the same condition number as A. Moreover, the amount of work per iteration step, necessary for the matrix vector product, is doubled.

Several proposals have been made to improve the numerical stability of this rather robust approach. The most well-known is by Paige and Saunders (Paige and Saunders, 1982) and is based upon applying the Lanczos method to the auxiliary system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

Clever execution of this delivers in fact the factors L and U of the LU-decomposition of the tridiagonal matrix that would have been delivered when carrying out the Lanczos procedure with $A^T A$. The resulting method is known as LSQR.

4.2 GMRES

The second approach is to form explicitly an orthonormal basis for the Krylov subspace. Since A is not symmetric we have no longer a 3-term recurrence relation for that purpose and the new basis vector has to be made explicitly orthonormal with respect to all the previous vectors:

$$v_1 = \frac{1}{\|r_0\|_2} r_0,$$

$$h_{i+1,i} v_{i+1} = A v_i - \sum_{j=1}^{i} h_{j,i} v_j.$$

As in the symmetric case this can be exploited in two different ways. The orthogonality relation can either be written as

$$A V_i = V_i H_i + h_{i+1,i} v_{i+1} e_i^T, \tag{4}$$

after which the projected system, with a Hessenberg matrix instead of a tridiagonal matrix as in the symmetric case, can be solved (non-symmetric CG, GENCG, FOM, Arnoldi's method), or it can be written as

$$A V_i = V_{i+1} \bar{H}_i, \tag{5}$$

after which the projected system, with an $i+1$ by i upper Hessenberg matrix can be solved as a least squares system. In GMRES (Saad

and Schultz, 1986) this is done by the QR method using Givens rotations in order to annihilate the subdiagonal elements in the upper Hessenberg matrix $\bar{H}_i$.
The first approach (based upon (4)) is similar to the conjugate gradient approach (or SYMMLQ), the second approach (based upon (5)) is similar to the conjugate directions method (or MINRES).

In order to avoid excessive storage requirements and computational costs for the orthogonalization, GMRES is usually restarted after each m iteration steps: GMRES(m). The costs for the orthogonalization in GMRES(m) are $\mathcal{O}(m^2)$. The proper choice of m is a very delicate problem which we will not discuss here.
Below we give a scheme for GMRES(m) for solving $Ax = b$, with a given preconditioner M:

```
x_0 is an initial guess;
for j = 1, 2, ....
      Solve r from Mr = b - Ax_0;
      v_1 = r/||r||_2;
      s := ||r||_2 e_1;
      for i = 1, 2, ..., m
          Solve w from Kw = Av_i;
          for k = 1, ..., i                  orthogonalization of w
               h_{k,i} = (w, v_k);           against v's, by modified
               w = w - h_{k,i} v_k;          Gram-Schmidt process
          end k;
          h_{i+1,i} = ||w||_2;
          v_{i+1} = w/h_{i+1,i};
          apply J_1, ..., J_{i-1} on (h_{1,i}, ..., h_{i+1,i});
          construct J_i, acting on i-th and (i+1)-st component
          of h_{.,i}, such that (i+1)-st component of J_i h_{.,i} is 0;
          s := J_i s;
          if s(i+1) is small enough then (UPDATE(x~, i); quit);
      end i;
      UPDATE(x~, m);
end j;
```

In this scheme UPDATE($\tilde{x}, i$) replaces the following computations:

Compute y as the solution of $Hy = \tilde{s}$, in which
the upper i by i triangular part of H has $h_{i,j}$ as

its elements (in least squares sense if H is singular),
$\tilde{s}$ represents the first i components of s;
$\tilde{x} = x_0 + y_1 * v_1 + y_2 v_2 + ... + y_i v_i$;
s_{i+1} equals $\|b - A\tilde{x}\|_2$;
if this component is not small enough
then $x_0 = \tilde{x}$;
else quit;

4.2.1 Parallel aspects of GMRES

Similar to CG, and other iterative schemes, the major computation intensive kernels are matrix-vection computations (with A and M), innerproducts and vector updates. All of these operations are almost trivially parallelizable, but, again, on distributed memory machines problems are likely to come from the innerproducts in the orthogonalization procedure (they act as synchronization points). In this part of the algorithm, one new vector: Av_j, is orthogonalized against the previously built orthogonal set v_1, v_2, ... , v_j. This leads typically to vector vector operations (BLAS1), and therefore we do not expect optimal performance on machines with distributed memory or memory hierarchy.
The obvious way to obtain more possibilities for improved parallelism and data locality is to generate a basis for the Krylov subspace first: v_1, Av_1, ..., $A^m v_1$, and to orthogonalize this set afterwards: m-step GMRES(m) (Chronopoulos and Kim, 1990). This approach does not lead to an increase in computational work, and the numerical instability due to the generation of a possibly near-dependent set is not necessarily a drawback. In any case, the resulting set, after orthogonalization, is the basis of some subspace, and the residual is then minimized over that subspace. However, if one wants to get results as close as possible to standard GMRES(m), one may try to generate a better starting set of basis vectors (i.e., more independent) by computing v_1, $y_2 = p_1(A)v_1$, ..., $y_{m+1} = p_m(A)v_1$, in which the p_j are suitable chosen j-th degree polynomials. In (De Sturler, 1991) it is claimed that Chebyshev polynomials led to satisfactory results. Anyhow, the m-step approach does not seem to lead to serious instability problems, since the method is restarted after each cycle of m steps.

After having generated a suitable starting set, we still have to

Table 2: Speed-ups for GMRES

Processor grid	speed-up standard GMRES	Speed-up with comb. innerp.
$10 \star 10$	77	98
$14 \star 14$	100	181
$17 \star 17$	114	213
$20 \star 20$	108	223

orthogonalize it. Now that the basis for the Krylov subspace is available, the orthogonalization is restructured as:

```
for k = 2, ..., m+1
    for j = k, ..., m+1                  orthogonalize y_k, ..., y_{m+1}
        y_j = y_j - (y_j, v_{k-1}) v_{k-1};   w.r.t. v_{k-1}
    end j;
    v_k = y_k / ||y_k||_2;
end k
```

Note that all the innerproducts in the inner loop can be done independent of each other, and that all message passing for these inner products can be combined (leading to less overhead). In (De Sturler, 1991) it is shown how this can be done with modified Gram-Schmidt, whilst avoiding communication times that cannot be overlapped.
In Table 2 we see some speed-ups as observed on a Parsytec Machine for GMRES(50). The linear system of order $60,000$ came from 5-point discretization of a second order partial differential equation over a rectangular grid in 2D.

4.2.2 <u>GMRES in combination with other schemes</u>

In (Van der Vorst and Vuik, 1994) it is shown how GMRES can be combined (or rather preconditioned) with other iterative schemes. The iteration steps of GMRES (or GCR) are called outer iteration steps, while the iteration steps of the preconditioning iterative method are referred to as inner iterations. The combined method is called GMRES$\star$, where $\star$ stands for any given iterative scheme; in the case of GMRES as the inner iteration method, the combined scheme is called GMRESR.

Similar schemes have been proposed recently. In FGMRES (Saad, 1993) the update directions for the approximate solution are preconditioned, whereas in GMRES$\star$ the residuals are preconditioned. The latter approach offers more control over the reduction in the residual, in particular break-down situations can be easily detected and remedied.
The GMRES$\star$ algorithm can be described by the following computational scheme:

> x_0 is an initial guess; $r_0 = b - Ax_0$;
> for $i = 0, 1, 2, 3, ...$
>   Let $z^{(m)}$ be the approximate solution of $Az = r_i$
>   obtained after m steps of an iterative method.
> $c = Az^{(m)}$ (often available from the iterative method)
>   for $k = 0, ..., i-1$
> $\alpha = (c_k, c)$
> $c = c - \alpha c_k$
> $z^{(m)} = z^{(m)} - \alpha u_k$
> $c_i = c/\|c\|_2$; $u_i = z^{(m)}/\|c\|_2$
> $x_{i+1} = x_i + (c_i, r_i)u_i$
> $r_{i+1} = r_i - (c_i, r_i)c_i$
>   if x_{i+1} is accurate enough then quit
> end

A sufficient condition to avoid break-down in this method ($\|c\|_2 = 0$) is that the norm of the residual at the end of an inner iteration is smaller than the right-hand residual: $\|Az^{(m)} - r_i\|_2 < \|r_i\|_2$. This can easily be controlled during the inner iteration process.
The idea behind this combined iteration scheme is that we explore parts of high-dimensional Krylov subspaces, hopefully localizing the same approximate solution that full GMRES would find over the entire subspace, but now at much lower computational costs. Alternatives for the inner iteration could be either one cycle of GMRES(m), since then we have also locally an optimal method, or some other iteration scheme, like for instance BiCGSTAB (Section 4.3.2).
It may seem questionable whether a method like BiCGSTAB should lead to success in the inner iteration. This method does not satisfy a useful global minimization property and large part of its effectiveness comes from the underlying BiCG algorithm, which is based on bi-orthogonality relations. This means that for each outer iteration the

inner iteration process has to build a bi-orthogonality relation again. By the nature of these kind of Krylov processes the largest eigenvalues and their corresponding eigenvector components quickly do enter the process after each restart, and hence it may be expected that much of the work is lost in rediscovering the same eigenvector components in the error over and over again, whereas these components may already be so small that further reduction in those directions in the outer iteration is waste of time, since it hardly contributes to a smaller norm of the residual.
This heuristic way of reasoning may explain in part our rather disappointing experiences with, e.g., BiCGSTAB as the inner iteration process for GMRES$\star$.

In (De Sturler and Fokkema, 1993) it is proposed to prevent the outer search directions explicitly to enter again in the inner process. This is done by keeping the Krylov subspace that is built in the inner iteration orthogonal with respect to the Krylov basis vectors generated in the outer iteration. The procedure works as follows.
In the outer iteration process the vectors c_0, ..., c_{i-1} build an orthogonal basis for the Krylov subspace. Let C_i be the n by i matrix with columns c_0, ..., c_{i-1}. Then the inner iteration process at outer iteration i is carried out with the operator A_i instead of A, and A_i is defined as

$$A_i = (I - C_i C_i^T)A. \tag{6}$$

It is easily verified that $A_i z \perp c_0, ..., c_{i-1}$ for all z, so that the inner iteration process takes place in a subspace orthogonal to these vectors. The additional costs, per iteration of the inner iteration process, are i innerproducts and i vector updates. In order to save on these costs, one should realize that it is not necessary to orthogonalize with respect to all previous c-vectors, and that "less effective" directions may be dropped, or combined with others. Suggestions have been made for such strategies. Of course, this orthogonalization approach should only be considered in cases where we see too little residual reducing effect in the inner iteration process in comparison with the outer iterations of GMRES$\star$.

4.3 Bi-conjugate gradients

The third class of methods arises from the attempt to construct a suitable set of basis vectors for the Krylov subspace by a three-term recurrence relation as in the symmetric case:

$$\alpha_{j+1} r_{j+1} = A r_j - \beta_j r_j - \gamma_j r_{j-1}. \tag{7}$$

When A is symmetric then this recurrence relation can be used to create an orthogonal set of vectors. By similar arguments as in the symmetric case we conclude that (7) can be used to generate a set of basis vectors r_0,...,r_{i-1} for $K^i(A; r_0)$, such that $r_j \perp K^{j-1}(A^T; r_0)$, or even more general,

$$r_j \perp K^{j-1}(A^T; \hat{r}_0),$$

since there is no explicit need to generate the Krylov subspace for A^T with r_0 as the starting vector.
If we let the basis vectors $\hat{r}_j$ for $K^i(A^T; \hat{r}_0)$ satisfy the same recurrence relation as the vectors r_j, i.e., with identical recurrence coefficients, then we see that

$$(r_k, \hat{r}_j) = 0 \quad \text{for} \quad k \neq j$$

(by a simple symmetry argument).
Hence, the sets $\{r_j\}$ and $\{\hat{r}_j\}$ satisfy a *biorthogonality* relation. Now we can proceed in a similar way as in the symmetric case:

$$A R_i = R_i T_i + \alpha_i r_i e_i^T, \tag{8}$$

but now we use the matrix $\hat{R}_i = [\hat{r}_0, \hat{r}_1, ..., \hat{r}_{i-1}]$ for the projection of the system

$$\hat{R}_i^T (A x_i - b) = 0,$$

or

$$\hat{R}_i^T A R_i y - \hat{R}_i^T b = 0.$$

Using (8) we find that y_i satisfies

$$\hat{R}_i^T R_i T_i y = (r_0, \hat{r}_0) e_1.$$

Since $\hat{R}_i^T R_i$ is a diagonal matrix with diagonal elements $(r_j, \hat{r}_j)$, we find, if all these diagonal elements are nonzero, that

$$T_i y = e_1 \quad \Rightarrow \quad x_i = R_i y.$$

This method is known as the Bi-Lanczos method (Lanczos, 1952).
We see that we get problems when a diagonal element of $\hat{R}_i^T R_i$ becomes (near) zero, this is referred to in literature as a serious (near)

breakdown. The way to get around this difficulty is the so-called Look-ahead strategy (Parlett et al, 1985), which comes down to taking a number of successive basis vectors for the Krylov subspace together and to make them blockwise bi-orthogonal.
Another way to avoid break-down is to restart as soon as a diagonal element gets small. Of course, this strategy looks surprisingly simple, but one should realise that at a restart the Krylov subspace, that has been built up so far, is thrown away, which destroys possibilities for faster (i.e., superlinear) convergence.

As for Conjugate Gradients, the LU decomposition of the tridiagonal system can be used to obtain short recurrences for the search direction and the residual vectors. This variant of Bi-Lanczos is usually called Bi-Conjugate Gradients (Bi-CG) (Fletcher, 1976).
Of course one can in general not be certain that an LU decomposition (without pivoting) of the tridiagonal matrix T_i exists, and this may lead also to break-down of the Bi-CG algorithm. This problem can be cured by the so-called composite step approach, which does a block pivot step occasionally (Bank and Chan, 1993). The same problem is also circumvented in the QMR variant.
The QMR method (Freund et al, 1992) relates to Bi-CG in a similar way as MINRES relates to CG, in the sense that a small least squares problem over the Krylov subspace is solved. The adjective Quasi in QMR (*Quasi Minimum Residual*) stems from the fact that in formulating the least squares problem one ignores the possible non-orthogonality of the basis vectors r_j. In its most robust form the QMR method is carried out on top of a look-ahead variant of the bi-orthogonal Lanczos method. Experiments show that QMR has a much smoother convergence behaviour than Bi-CG, but it is not essentially faster than Bi-CG (when the latter does not break down).

Note that for symmetric matrices Bi-Lanczos generates the same solution as Lanczos, provided that $\hat{r}_0 = r_0$, and under the same condition Bi-CG delivers the same iterands as CG for positive definite matrices. However, the Bi-orthogonal variants do so at the cost of two matrix vector operations per iteration step.

It is difficult to make a fair comparison between GMRES and Bi-CG. GMRES really minimizes a residual, but at the cost of increasing work for keeping all residuals orthogonal and increasing demands for memory space. Bi-CG does not minimize a residual, but often it has a comparable fast convergence as GMRES, at the cost of twice the amount of matrix vector products per iteration step. However,

the generation of the basis vectors is relatively inexpensive and the memory requirements are limited and modest. Several variants of Bi-CG have been proposed which increase the effectiveness of this class of methods in certain circumstances. These variants will be discussed briefly in coming subsections.
The algorithm for Bi-CG is, from the implementation point of view, very similar to CG, and problems with respect to parallelism are almost similar. For algorithms, other implementation issues, and further references we refer to Barret et al (Barret et al, 1994,DDSV).

4.3.1 CGS

For the bi-conjugate gradient residual vectors it is well-known that they can be written as $r_j = P_j(A)r_0$ and $\hat{r}_j = P_j(A^T)\hat{r}_0$, and because of the bi-orthogonality relation we have that

$$\begin{aligned}(r_j, \hat{r}_i) &= (P_j(A)r_0, P_i(A^T)\hat{r}_0) \\ &= (P_i(A)P_j(A)r_0, \hat{r}_0) = 0,\end{aligned}$$

for $i < j$.
The iteration parameters for bi-conjugate gradients are computed from innerproducts like these ones. Sonneveld (Sonneveld, 1989) observed that we can also construct the vectors $\tilde{r}_j = P_j^2(A)r_0$, using only the latter form of the innerproduct for recovering the bi-conjugate gradients parameters (which implicitly define the polynomial P_j). By doing so, it can be avoided that the vectors $\hat{r}_j$ have to be formed, nor is there any multiplication with the matrix A^T.
The resulting CGS method works in general very well for many unsymmetric linear problems. It converges often much faster than BI-CG (about twice as fast in some cases) and does not have the disadvantage of having to store extra vectors like in GMRES. However, CGS usually shows a very irregular convergence behaviour. This behaviour can even lead to cancellation and a spoiled solution (Sleijpen et al, 1994).
Some attempts have been made to improve the convergence behavior of CGS. For instance, in (Freund, 1993) the QMR approach is applied to CGS, which leads to TFQMR.
The break-down that may occur in Bi-CG due to a zero pivot in the LU factorization occurs also in CGS. This problem is addressed in (Chan and Szeto, 1994). The break-down situation in the underlying Lanczos process is considered in (Brezinski and Redivo-Zaglia, 1994).

Bi-CGSTAB

Bi-CGSTAB (Van der Vorst, 1992) is based on the following observation. Instead of squaring the Bi-CG polynomial, we can also construct other iteration methods, by which x_i are generated so that $r_i = \tilde{P}_i(A)P_i(A)r_0$ for other i^{th} degree polynomials $\tilde{P}$. An obvious possibility is to take for $\tilde{P}_j$ a polynomial of the form

$$Q_i(x) = (1-\omega_1 x)(1-\omega_2 x)...(1-\omega_i x), \tag{9}$$

and to select suitable constants ω_j. This expression leads to an almost trivial recurrence relation for the Q_i.
In Bi-CGSTAB ω_j in the j^{th} iteration step is chosen as to minimize r_j, with respect to ω_j, for residuals that can be written as $r_j = Q_j(A)P_j(A)r_0$.
Bi-CGSTAB can be viewed as the product of Bi-CG and repeated GMRES(1). Of course, other product methods can be formulated as well. Gutknecht (Gutknecht, 1993) has proposed BiCGSTAB2, which is constructed as the product of Bi-CG and GMRES(2).

Bi-CGSTAB2 and variants

As soon as the GMRES(1) part of Bi-CGSTAB (nearly) stagnates, then the Bi-CG part in the next iteration step cannot (or only poorly) be constructed.
Another dubious aspect of Bi-CGSTAB is that the factor Q_k has only real roots by construction. It is well-known that optimal reduction polynomials for matrices with complex eigenvalues may have complex roots as well. If, for instance, the matrix A is real skew-symmetric, then GMRES(1) stagnates forever, whereas GMRES(2), in which we minimize over two combined successive search directions, may lead to convergence, and this is mainly due to the fact that then complex eigenvalue components in the error can be effectively reduced.
This point of view was taken in (Gutknecht, 1992) for the construction of the BiCGSTAB2 method. In the odd-numbered iteration steps the Q-polynomial is expanded by a linear factor, as in Bi-CGSTAB, but in the even-numbered steps this linear factor is discarded, and the Q-polynomial from the previous even-numbered step is expanded by a quadratic $1 - \alpha_k A - \beta_k A^2$. For this construction the information from the odd-numbered step is required. It was anticipated that the introduction of quadratic factors in Q might help

to improve convergence for systems with complex eigenvalues, and, indeed, some improvement was observed in practical situations (see also (Pommerell, 1992)).

However, our presentation suggests a possible weakness in the construction of Bi-CGSTAB2, namely in the odd-numbered steps the same problems may occur as in Bi-CGSTAB. Since the results of the odd-numbered steps are necessary for the completion of the even-numbered steps, this may equally lead to unnecessary break-downs or poor convergence. In (Sleijpen and Fokkema, 1993) another, and even simpler approach was taken to arrive at the desired even-numbered steps, without the necessity of the construction of the intermediate Bi-CGSTAB-type step in the odd-numbered steps. In this approach the polynomial Q is constructed straight-away as a product of quadratic factors, without ever constructing any intermediate linear factor. As a result the new method BiCGSTAB(2) leads only to significant residuals in the even-numbered steps and the odd-numbered steps do not lead necessarily to useful approximations.

In fact, it was shown that the polynomial Q can also be constructed as the product of ℓ-degree factors, without the construction of the intermediate lower degree factors. The main idea is that ℓ successive Bi-CG steps are carried out, where for the sake of an A^T-free construction the already available part of Q is expanded by simple powers of A. This means that after the Bi-CG part of the algorithm vectors from the Krylov subspace $s, As, A^2s, ..., A^\ell s$, with $s = P_k(A)Q_{k-\ell}(A)r_0$ are available, and it is then relatively easy to minimize the residual over that particular Krylov subspace. There are variants of this approach in which more stable bases for the Krylov subspaces are generated (Sleijpen et al, 1994), but for low values of ℓ a standard basis satisfies, together with a minimum norm solution obtained through solving the associated normal equations (which requires the solution of an ℓ by ℓ system. In most cases BiCGSTAB(2) will already give nice results for problems where Bi-CGSTAB or BiCGSTAB2 may fail. Note, however, that in exact arithmetic, if no break-down situation occurs, BiCGSTAB2 would produce exactly the same results as BiCGSTAB(2) at the even-numbered steps.

BiCGSTAB(2) can be represented by the following algorithm:

x_0 is an initial guess; $r_0 = b - Ax_0$;
$\hat{r}_0$ is an arbitrary vector, such that $(r, \hat{r}_0) \neq 0$,

e.g., $\hat{r}_0 = r$;

$\rho_0 = 1; u = 0; \alpha = 0; \omega_2 = 1;$

for $i = 0, 2, 4, 6, ...$

$\rho_0 = -\omega_2\rho_0$

even Bi-CG step: $\rho_1 = (\hat{r}_0, r_i); \beta = \alpha\rho_1/\rho_0; \rho_0 = \rho_1$

$u = r_i - \beta u;$

$v = Au$

$\gamma = (v, \hat{r}_0); \alpha = \rho_0/\gamma;$

$r = r_i - \alpha v;$

$s = Ar$

$x = x_i + \alpha u;$

odd Bi-CG step: $\rho_1 = (\hat{r}_0, s); \beta = \alpha\rho_1/\rho_0; \rho_0 = \rho_1$

$v = s - \beta v;$

$w = Av$

$\gamma = (w, \hat{r}_0); \alpha = \rho_0/\gamma;$

$u = r - \beta u$

$r = r - \alpha v$

$s = s - \alpha w$

$t = As$

GCR(2)-part: $\omega_1 = (r, s); \mu = (s, s); \nu = (s, t); \tau = (t, t);$

$\omega_2 = (r, t); \tau = \tau - \nu^2/\mu; \omega_2 = (\omega_2 - \nu\omega_1/\mu)/\tau;$

$\omega_1 = (\omega_1 - \nu\omega_2)/\mu$

$x_{i+2} = x + \omega_1 r + \omega_2 s + \alpha u$

$r_{i+2} = r - \omega_1 s - \omega_2 t$

if x_{i+2} accurate enough then quit

$u = u - \omega_1 v - \omega_2 w$

end

For distributed memory machines the innerproducts may cause communication overhead problems. We note that the BiCG steps are very similar to conjugate gradient iteration steps, so that we may consider similar tricks as for CG, to reduce the number of synchronization points caused by the 4 innerproducts in the Bi-CG parts. For an overview of these approaches see (Barret et al, 1994). If on a specific computer it is possible to overlap communication with communication, then the Bi-CG parts can be rescheduled as to create overlap possibillities:

1. the computation of ρ_1 in the even Bi-CG step may be done just before the update of u at the end of the GCR part.

2. The update of x_{i+2} may be delayed until after the computation

of γ in the even Bi-CG step.
3. The computation of ρ_1 for the odd Bi-CG step can be done just before the update for x at the end of the even Bi-CG step.
4. The computation of γ in the odd Bi-CG step has already overlap possibillities with the update for u.
For the GCR(2) part we note that the 5 innerproducts can be taken together, in order to reduce start-up times for their global assembling. This gives BiCGSTAB(2) a (slight) advantage over Bi-GSTAB. Furthermore we note that the updates in the GCR(2) may lead to more efficient code than for Bi-CGSTAB.
Recently a composite step Bi-CGSTAB2 method was proposed in (Chan and Szeto, 1994), which is very close to to BiCGSTAB(2), but which has the additional advantage that it also overcomes the 'pivot break-down' in the underlying Bi-CG process.

References

Bank, R. and Chan, T.F., 1993. "An analysis of the composite step biconjugate gradient method", *Numer. Math.*, **66**, p. 295.

Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. ,and Van der Vorst, H., 1994. "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", SIAM, Philadelphia, PA.

Brezinski, C. and Redivo-Zaglia, M., 1994. "Treatment of near breakdown in the CGS algorithm", to appear in *Numerical Algorithms.*

Chan, T.F. and Szeto, T., 1993. "A composite step conjugate gradient squared algorithm for solving nonsymmetric linear systems', Technical report CAM-93-27, UCLA, Department of Mathematics, Los Angeles. (to appear in *Numerical Algorithms, 1994*)

Chan, T.F. and Szeto, T., 1994. "Composite step product methods for solving nonsymmetric linear systems", Techical Report CAM-94-??, UCLA, Department of Mathematics, Los Angeles.

Chronopoulos, A.T. and Kim, S.K., 1990. "s-Step Orthomin and GMRES implemented on parallel computers," Technical Report 90/43R, UMSI, Minneapolis.

De Sturler, E., 1991. "A parallel restructured version of GMRES(m)," Technical Report 91-85, Delft University of Technology, Delft.

De Sturler, E. and Fokkema, D.R., 1993. "Nested Krylov methods and preserving the orthogonality," Preprint 796, Utrecht University, Department of Mathematics, Utrecht.

Demmel, J., Heath, M., and Van der Vorst, H., 1993. "Parallel linear algebra," *Acta Numerica 1993*, Cambridge University Press, Cambridge.

Dongarra, J.J., 1990. "Performance of various computers using standard linear equations software in a fortran environment," Technical Report CS-89-85, University of Tennessee, Knoxville.

Dongarra, J.J., Duff, I.S., Sorensen, D.C., and Van der Vorst, H.A., 1991. "Solving Linear Systems on Vector and Shared Memory Computers," SIAM, Philadelphia, PA.

Dongarra, J.J. and Van der Vorst, H.A., 1992. "Performance of various computers using standard sparse linear equations solving techniques," *Supercomputer*, **9(5)**, p. 17.

Fletcher, R., 1976. "Conjugate gradient methods for indefinite systems," *Lecture Notes Math.*, **506**, p. 73. Springer-Verlag, Berlin–Heidelberg–New York.

Freund, R.W., Golub, G.H., and Nachtigal, N.M., 1992. "Iterative solution of linear systems," *Acta Numerica 1992*, Cambridge University Press, Cambridge.

Freund, R.W., 1993. "A transpose-free quasi-minimum residual algorithm for non-Hermitian linear systems", *SIAM J. Sci. Comput.*, **14**, p. 470.

Golub, G.H. and Van Loan, C.F., 1989. "Matrix Computations," The Johns Hopkins University Press, Baltimore.

Gutknecht, M.H., 1993. "Variants of BICGSTAB for matrices with complex spectrum," *SIAM J. Sci. Comput.*, **14**, p. 1020.

M. R. Hestenes, M.R. and Stiefel, E., 1954. "Methods of conjugate gradients for solving linear systems," *J. Res. Natl. Bur. Stand.*, **49**, p. 409.

Lanczos, C., 1952. "Solution of systems of linear equations by minimized iterations," *J. Res. Natl. Bur. Stand*, **49**, p. 33.

Ortega, J.M., 1988. "Introduction to Parallel and Vector Solution of Linear Systems," Plenum Press, New York and London.

Paige, C.C. and Saunders, M.A., 1975. "Solution of sparse indefinite systems of linear equations," *SIAM J. Numer. Anal.*, **12**, p. 617.

Paige, C.C. and Saunders, M.A., 1982. "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Soft.*, **8**, p. 43.

Parlett, B.N., Taylor, D.R., and Liu, Z.A., 1985. "A look-ahead Lanczos algorithm for unsymmetric matrices," *Math. Comp.*, **44**, p. 105.

Pommerell, C., 1992. "Solution of large unsymmetric systems of linear equations," PhD thesis, Swiss Federal Institute of Technology, Zürich.

Saad, Y, 1993. "A flexible inner-outer preconditioned GMRES algorithm," *SIAM J. Sci. Comput.*, **14**, p. 461.

Saad, Y. and Schultz, M.H., 1986. "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, **7**, p. 856.

Simon, H.D., 1989. "Direct sparse matrix methods," In: Almond, J.C. and Young, D.M., editors, *Modern Numerical Algorithms for Supercomputers*, pages 325–444, Austin, 1989. The University of Texas at Austin, Center for High Performance Computing.

Sleijpen, G.L.G., Van der Vorst, H.A., and Fokkema, D.R., 1994. "Bi-CGSTAB(ℓ) and other hybrid bi-cg methods," to appear in *Numerical Algorithms.*

Sleijpen, G.L.G. and Fokkema, D.R., 1993. "BICGSTAB(ℓ) for linear equations involving unsymmetric matrices with complex spectrum," *ETNA*, **1**, p. 11.

Sonneveld, P. 1989. "CGS: a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, **10**, p. 36.

Van der Vorst, H.A., 1992. "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems," *SIAM J. Sci. Statist. Comput.*, **13**, p. 631.

Van der Vorst, H.A. and Vuik, C., 1994. "GMRESR: A family of nested GMRES methods," to appear in *Num. Lin. Alg. with Appl.*.

Varga, R.S., 1962. "Matrix Iterative Analysis," Prentice-Hall, Englewood Cliffs NJ.