Parallel Iterative Solution Methods for Linear Systems arising from Discretized PDE's

Henk A. Van der Vorst
Mathematical Institute, University of Utrecht
PO Box 80.010
NL-3508 TA Utrecht, the Netherlands
e-mail: vorst@math.ruu.nl

1 Introduction

In these notes 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, CGS, Bi-CGSTAB, QMR, LSQR and GMRES. We will show how these methods can be derived from simple basic iteration formulas. We will not give convergence proofs, but we will refer for these, as far as available, to litterature.
Iterative methods are often used in combination with so-called preconditioning operators (approximations for the inverses of the operator of the system to be solved). Since these preconditioners are not essential in the derivation of the iterative methods, we will not give much attention to them in these notes. However, in most of the actual iteration schemes, we have included them in order to facilitate the use of these schemes in actual computations.
For the application of the iterative schemes one usually thinks of linear sparse systems, e.g., like those arising in the finite element or finite difference approximations of (systems of) partial differential equations. However, the structure of the operators plays no explicit role in any of these schemes, and these schemes might also successfully be used to solve certain large dense linear systems. Depending on the situation that might be attractive in terms of numbers of floating point operations.

It will turn out that all of the iterative are parallelizable in a straight forward manner. However, especially for computers with a memory hierarchy (i.e., like cache or vector registers), and for distributed memory computers, the performance can often be improved significantly through rescheduling of the operations. We will discuss parallel implementations, and occasionally we will report on experimental findings.

2 Direct versus Iterative

1. Standard Gaussian elimination leads to fill-in, and 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 $\mathcal{O}(nm^4) \sim n^{2\frac{1}{3}}$ [36, 25]. For 2D problems the bandwidth is $\sim n^{\frac{1}{2}}$, so that the number of flops for a direct method then varies like n^2.
If one has to solve many systems with different right-hand sides, then one has to decompose the matrix only once after which the costs for solving each system will vary like $n^{\frac{5}{3}}$ for 3D problems, and like $n^{\frac{3}{2}}$ for 2D problems.

2. For symmetric positive definite systems the error reduction per iteration step of CG is $\sim \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$, with $\kappa = \|A\|_2\|A^{-1}\|_2$ [14, 2, 35].
For discretized second order pde's, over grids with gridsize $\frac{1}{m}$ we typically see $\kappa \sim m^2$. Hence, for 3D problems we have that $\kappa \sim n^{\frac{2}{3}}$, and for 2D problems: $\kappa \sim n$. For an error reduction of ϵ we must have that
$$\left(\frac{1-\frac{1}{\sqrt{\kappa}}}{1+\frac{1}{\sqrt{\kappa}}}\right)^j \approx (1-\frac{2}{\sqrt{\kappa}})^j \approx e^{-\frac{2j}{\sqrt{\kappa}}} < \epsilon.$$
For 3D problems we have that
$$\Rightarrow \quad j \sim -\frac{\log\epsilon}{2}\sqrt{\kappa} \approx -\frac{\log\epsilon}{2}n^{\frac{1}{3}},$$
whereas for 2D problems:
$$j \approx -\frac{\log\epsilon}{2}n^{\frac{1}{2}}.$$
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)
$\Rightarrow$ flops per reduction with ϵ:
$\sim -fn^{\frac{4}{3}}\log\epsilon$ for 3D problems,
and $\sim -fn^{\frac{3}{2}}\log\epsilon$ for 2D problems.

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 $n^{\frac{5}{3}}$, 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 Horst Simon [74]. 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 the 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.

Remarks:

- With suitable preconditioning we may have $\sqrt{\kappa} \sim n^{\frac{1}{6}}$ and the flops count then becomes

$$\sim -fn^{\frac{7}{6}} \log \epsilon,$$

  see, e.g., [37].

- For classes of problems some methods may even be faster: multigrid, fast poisson solvers.

- Storage considerations are also 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.

- Iterative methods can be attractive even when the matrix is dense. Again, in the positive definite symmetric case, if the condition number is $n^{2-2\varepsilon}$ then, since the amount of work per iteration step is $\sim n^2$, and the number of iteration steps $\sim n^{1-\varepsilon}$, the total work estimate is roughly proportional to $n^{3-\varepsilon}$, and this is asymptotically less than the amount of work for Choleski's method, which varies like $\sim n^3$.

The question remains at the moment how well iterative methods can take advantage of modern computer architectures. From Dongarra's linpack benchmark [22] 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 computational speeds reported in [24] for the preconditioned as well as the unpreconditioned conjugate gradient method (ICCG and CG, respectively).

In [24] 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.

3 Basic iteration method

A very basic idea, that leads to many effective iterative solvers, is to to split the matrix of a given linear system in the sum of two matrices, one of which 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:

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

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

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

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

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

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 [90]. We will not discuss this aspect here, since it is not relevant at this stage. Instead of studying the basic iterative methods we will show how other more powerful iteration methods can be constructed 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., $M^{-1}Ax = M^{-1}b$. We will return to this later.

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

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

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

Apparently, the Richardson iteration delivers elements of increasing Krylov subspaces. Including local iteration parameters in the iteration would lead to other elements of the same Krylov subspaces. Let us write such an element still as x_{i+1}. Since $x_{i+1} \in K^{i+1}(A; r_0)$, we have that

$$x_{i+1} = Q_{i+1}(A) r_0,$$

with Q_{i+1} an arbitrary polynomial of degree $i+1$. It follows that

$$\begin{aligned} r_{i+1} &= b - Ax_{i+1} = (I - AQ_{i+1}(A)) r_0 \\ &= \tilde{P}_{i+1}(A) r_0, \qquad (3.0a) \end{aligned}$$

with, just as in the standard Richardson iteration, $\tilde{P}_{i+1}(0) = 1$.
The Richardson iteration can be characterized by the polynomial $P_{i+1}(A) = (I - A)^{i+1}$.

Note that one almost never computes inverses of matrices, like K^{-1}, explicitly. Instead, vectors like $\tilde{r}_i = K^{-1}b - \tilde{A}x_i = K^{-1}(b - Ax_i)$ are usually computed by solving $\tilde{r}_i$ from $K\tilde{r}_i = b - Ax_i$. The matrix K is often sparse, whereas K^{-1} usually is not, so that this procedure is much more efficient both in CPU-time and in computer memory space.

4 Towards optimal iteration methods

The natural question arises whether we can pick up a better x_{i+1} from the Krylov subspace that is generated by the basic iterative method. One would like to see the x_{i+1} for which $\|x_{i+1} - x\|_2$ is minimal.
E.g., $x_1 \in \{r_0\} \Rightarrow x_1 = \alpha_0 r_0$.

$$\|x - x_1\|_2^2 = (x - \alpha_0 r_0, x - \alpha_0 r_0) =$$
$$= (x, x) - 2\alpha_0 (x, r_0) + \alpha^2 (r_0, r_0).$$

Minimizing with respect to α_0 gives

$$\alpha_0 = \frac{(r_0, x)}{(r_0, r_0)},$$

and this is not practical, since x is unknown.

The above expression for α_0 suggests that with a different innerproduct the problem might be solvable: $(x, y)_A \equiv (x, Ay)$.

5 Symmetric matrices

If A is symmetric positive definite then this defines a proper innerproduct:

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

$$(x,x)_A = 0 \Longleftrightarrow x = 0.$$

Now we have that

$$\|x - x_1\|_A^2 = (x - \alpha_0 r_0, x - \alpha_0 r_0)_A$$

$$\Longrightarrow \alpha_0 = \frac{(r_0, x)_A}{(r_0, r_0)_A} = \frac{(r_0, Ax)}{(r_0, Ar_0)}.$$

This looks promising and therefore we will follow that line.

In general we want $\|x_i - x\|_A$ minimal for $x_i \in K^i(A; r_0)$

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

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

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

$$\{r_0, r_1, \ldots, r_i\}$$

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:

$$\text{(5.0a)} \qquad \alpha_{j+1} r_{j+1} = A r_j - \beta_j r_j - \gamma_j r_{j-1}.$$

The proof is as follows.

$r_1 \in K^2(A; r_0) \Rightarrow \alpha_1 r_1 = Ar_0 - \beta_0 r_0$

$r_2 \in K^3(A; r_0) \Rightarrow r_2 \in \{r_0, r_1, A^2 r_0\}$

$\Rightarrow r_2 \in \{r_0, r_1, Ar_1\}$

$$\Rightarrow \quad \alpha_2 r_2 = Ar_1 - \beta_1 r_1 - \gamma_1 r_0$$

Now we use an induction argument.

$$r_{j-1} \in K^j(A; r_0) \;\Rightarrow\; Ar_{j-1} \in K^{j+1}(A; r_0)$$

$$= \{r_0, r_1, \ldots, r_j\}$$

$$\Rightarrow \quad \alpha_j r_j = Ar_{j-1} - \sum_{i=0}^{j-1} \delta_i r_i$$

Because we want the new vector r_j to be orthogonal with respect to all previous ones, the constants δ_i are determined by

$$(Ar_{j-1}, r_k) - \delta_k (r_k, r_k) = 0$$

$$\text{(5.0b)} \qquad (Ar_{j-1}, r_k) = (r_{j-1}, Ar_k)$$

(note that we have used the symmetry of A)

$$= (r_{j-1}, \alpha_{k+1} r_{k+1} + \beta_k r_k + \gamma_k r_{k-1})$$

Here we have used the induction argument for k. Because of the orthogonality it follows that $\delta_k = 0$ for $k = 0, \ldots, j-3$ and hence r_j also satisfies a 3-term recurrence relation.

The values for β_j and γ_j follow from the orthogonality of the residual vectors:

$$\beta_j = (r_j, Ar_j)/(r_j, r_j),$$

and

$$\gamma_j = (r_{j-1}, Ar_j)/(r_{j-1}, r_{j-1}).$$

The value of α_{j+1} determines the proper length of the new residual vector. From the consistency relation (3.0a) we have 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 (5.0a) shows that

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

At the end of this section we will consider the situation where the recurrence relation terminates.

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

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

If we consider the 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

(5.0c) $$AR_i = R_iT_i + \alpha_i r_i e_i^T,$$

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

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)
Further we have for the 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 (5.0c) 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 $\|r_0\|_2^2$ up to $\|r_{i-1}\|_2^2$ 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 [47]. We will exploit the relation between the Lanczos method and the conjugate gradients method for the analysis of the convergence behaviour of the latter method.

Note that for some $j \leq n-1$ the construction of the orthogonal basis must terminate. In that case we have that $AR_{j+1} = R_{j+1}T_{j+1}$. Let y be the solution of the reduced system $T_{j+1}y = e_1$, and $x_{j+1} = R_{j+1}y$. Then it follows that $x_{j+1} = x$, i.e., we have arrived at the exact solution, since $Ax_{j+1} - b = AR_{j+1}y - b = R_{j+1}T_{j+1}y - b = R_{j+1}e_1 - b = 0$ (we have assumed that $x_0 = 0$).

5.1 THE CG-METHOD:

The Conjugate Gradients **CG** method [41] 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 vectors x_i. This can be done cheaper. 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 $y \in I\!R^{i+1}$). 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. Hence

$$x_i = R_i y = R_i T_i^{-1} e_1 = (R_i U_i^{-1})(L_i^{-1} e_1)$$

We concentrate on the factors, placed between parenthesis, separately.

1. $$L_i = \begin{pmatrix} 1 & & & & \\ f_1 & 1 & & & \\ & f_2 & \ddots & & \\ & & \ddots & \ddots & \\ & & & f_{i-1} & 1 \end{pmatrix}$$

   With $q \equiv L_i^{-1} e_1$ we have that q can be solved from $L_i q = e_1 \Rightarrow f_{i-1}q_{i-2} + q_{i-1} = 0 \Rightarrow q_{i-1}$, in recursive manner.

2. Write $B_i \equiv R_i U_i^{-1}$, then we have that

$$R_i = B_i U_i = \begin{pmatrix} & \vdots \\ & \vdots \\ & \vdots \\ & \vdots \end{pmatrix} \times$$

$$\begin{pmatrix} d_0 & g_1 & & & \\ & d_1 & \ddots & & \\ & & \ddots & & \\ & & & \ddots & g_{i-1} \\ & & & & d_{i-1} \end{pmatrix}$$

$$\Rightarrow \quad r_{i-1} = g_{i-1}(B_i)_{i-2} + d_{i-1}(B_i)_{i-1}$$

$$\Rightarrow \quad (B_i)_{i-1}.$$

Glueing these two recurrences together we obtain

$$x_i = \begin{pmatrix} & \vdots \\ & (B_i)_{i-1} \\ & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ \vdots \\ q_{i-1} \end{pmatrix}$$

$$= x_{i-1} + q_{i-1}(B_i)_{i-1}$$

and this is in fact the well-known conjugate gradients method. The name stems from the property that the update vectors $(B_i)_{i-1}$, usually notated as p_{i-1}, are A-orthogonal.
Note that the positive definiteness of A is only exploited as to guarantee the flawless decomposition of the implictly generated tridiagonal matrix T_i. This suggests that the conjugate gradients method may also work for certain non positive definite systems, but then at our own risk [59]. We will later see how other ways of solving the projected system will lead to other well-known methods.

5.1.1 Computational notes

The standard (unpreconditioned) Conjugate Gradient algorithm for the solution of $Ax = b$ can be represented by the following scheme:

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0$;
$\rho_0 = (r_0, r_0)$
for $i = 0, 1, 2, ...$
  $p_i = r_i + \beta_{i-1}p_{i-1}$;
  $q_i = Ap_i$;
  $\alpha_i = \frac{\rho_i}{(p_i, q_i)}$
  $x_{i+1} = x_i + \alpha_i p_i$;
  $r_{i+1} = r_i - \alpha_i q_i$;
  if x_{i+1} accurate enough then quit;
  $\rho_{i+1} = (r_{i+1}, r_{i+1})$;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end;

CG is most often used in combination with a suitable splitting $A = K - R$, and then K^{-1} is called the preconditioner. We will assume that K is also positive definite.
Note first that the CG method can be derived for any choice of the innerproduct. In our derivation we have used the standard innerproduct $(x, y) = \sum x_i y_i$, but we have not used any specific property of that innerproduct. Now we make a different choice:

$$[x, y] \equiv (x, Ky).$$

It is easy to verify that $K^{-1}A$ is symmetric positive definite with respect to [,]:

$$\begin{aligned} [K^{-1}Ax, y] &= (K^{-1}Ax, Ky) = (Ax, y) \\ &= (x, Ay) = [x, K^{-1}Ay]. \end{aligned} \tag{5.1a}$$

Hence, we can follow our CG procedure for solving the preconditioned system $K^{-1}Ax = K^{-1}b$, using the new [,]-innerproduct.
Apparently, we now are minimizing

$$[x_i - x, K^{-1}A(x_i - x)] = (x_i - x, A(x_i - x)),$$

which leads to the remarkable (and known) result that for this preconditioned system we still minimize the error in A-norm, but now over a Krylov subspace generated by $K^{-1}r_0$ and $K^{-1}A$.

In the following computational scheme for preconditioned CG, for the solution of $Ax = b$ with preconditioner K^{-1}, we have replaced the [,]-innerproduct again by the familiar standard innerproduct. E.g., note that with $\tilde{r}_{i+1} = K^{-1}Ax_{i+1} - K^{-1}b$ we have that

$$\rho_{i+1} = [\tilde{r}_{i+1}, \tilde{r}_{i+1}]$$

$$= [K^{-1}r_{i+1}, K^{-1}r_{i+1}] = [r_{i+1}, K^{-2}r_{i+1}]$$

$$= (r_{i+1}, K^{-1}r_{i+1}),$$

and $K^{-1}r_{i+1}$ is the residual corresponding to the preconditioned system $K^{-1}Ax = K^{-1}b$.

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0$;
Solve w_0 from $Kw_0 = r_0$;
$\rho_0 = (r_0, w_0)$
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 $Kw_{i+1} = r_{i+1}$;
  $\rho_{i+1} = (r_{i+1}, w_{i+1})$;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end;

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

The coefficients α_j and β_j, generated by the above scheme, can be used to build the matrix T_i in the following way:

$$T_i = \begin{pmatrix} \ddots & & & \\ \ddots & -\frac{\beta_{j-1}}{\alpha_{j-1}} & & \\ \ddots & \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \ddots & \\ & -\frac{1}{\alpha_j} & \ddots & \\ & & \ddots & \end{pmatrix}. \tag{5.1b}$$

Since $\alpha_j > 0$ and $\beta_j > 0$ we see that the above matrix is similar to the following symmetric tridiagonal matrix:

$$\tilde{T}_i = \begin{pmatrix} \ddots & & \\ \ddots & -\frac{\sqrt{\beta j-1}}{\alpha_{j-1}} & \\ \ddots & \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \ddots \\ & -\frac{\sqrt{\beta_j}}{\alpha_j} & \ddots \\ & & \ddots \end{pmatrix}.$$

The eigenvalues of the leading i^{th} order minor of this matrix are the Ritz values of the preconditioned matrix $K^{-1}A$ with respect to the i-dimensional Krylov subspace spanned by the first i residual vectors. The Ritz values approximate the (extremal) eigenvalues of the preconditioned matrix increasingly well. These approximations can be used to get an impression of the relevant eigenvalues. They can also be used to construct upperbounds for the error in the delivered approximation with respect to the solution [45, 40]. According to the results in [80] the eigenvalue information can also be used in order to understand or explain delays in the convergence behaviour.

5.1.2 The convergence of Conjugate Gradients

The conjugate gradient method (here with $K = I$) constructs in the i^{th} iteration step an x_i, which can be written as

$$x_i - x = P_i(A)(x_0 - x) \quad \text{(cf. (3.0}a\text{))},$$

such that $\|x_i - x\|_A$ is minimal over all polynomials P_i of degree i, with $P_i(0) = 1$.
Let us denote the eigenvalues and the orthonormalized eigenvectors of A by λ_j, z_j. We write $r_0 = \sum_j \gamma_j z_j$. It follows that

$$r_i = P_i(A)r_0 = \sum_j \gamma_j P_i(\lambda_j) z_j$$

and hence

$$\|x_i - x\|_A^2 = \sum_j \frac{\gamma_j^2}{\lambda_j} P_i^2(\lambda_j).$$

Note that only those λ_j play a role in this process for which $\gamma_j = 0$. In particular, if A happens to be semidefinite, i.e., there is a $\lambda = 0$, then this is no problem for the minimization process as long as the corresponding coefficient γ is zero as well. The situation where γ is small, due to rounding errors, is discussed in [45].
Upperbounds on the error (in A-norm) are obtained by observing that

$$\begin{aligned} \|x_i - x\|_A^2 &= \sum_j \frac{\gamma_j^2}{\lambda_j} P_i^2(\lambda_j) \leq \sum_j \frac{\gamma_j^2}{\lambda_j} Q_i^2(\lambda_j) \\ \text{(5.1c)} \qquad &\leq \max_{\lambda_j} Q_i^2(\lambda_j) \sum_j \frac{\gamma_j^2}{\lambda_j}, \end{aligned}$$

for any arbitrary polynomial Q_i of degree i with $Q_i(0) = 1$, where the maximum is taken, of course, only over those λ for which the corresponding $\gamma \neq 0$. When P_i has zeros at all the different λ_j then $r_i = 0$. The conjugate gradients method tries to spread the zeros in such a way that $P_i(\lambda_j)$ is small in a weighted sense, i.e., $\|x_i - x\|_A$ is as small as possible.

We get suitable upperbounds by selecting appropriate polynomials for Q_i. A very well-known upperbound arises by taking for Q_i the i^{th} degree Chebychev polynomial transformed to the interval $[\lambda_{min}, \lambda_{max}]$ and scaled such that its value in 0 is equal to 1.

$$\begin{aligned} \|x_i - x\|_A^2 &\leq \sum \frac{\gamma_j^2}{\lambda_j} T_i^2(\lambda_j) \\ \text{(5.1d)} \qquad &\leq \max_{\lambda_1, \lambda_n} |T_i^2(\lambda_j)| \; \|x_0 - x\|_A^2, \end{aligned}$$

and

$$\text{(5.1e)} \qquad |T_i(\lambda_j)| \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i .$$

The purpose of preconditioning is to reduce the condition number κ.

As we have seen the conjugate gradients algorithm is just an efficient implementation of the Lanczos algorithm. The eigenvalues of the implicitly generated tridiagonal matrix T_i are the Ritz values of A with respect to the current Krylov subspace. It is known from Lanczos theory that these Ritz values converge towards the eigenvalues of A and that in general the extremal eigenvalues of A are first well approximated [46, 58, 63]. Furthermore, the speed of convergence depends on how well these eigenvalues are separated from the others (gap ratio) [63]. This helps us to understand the so-called superlinear convergence behaviour of the conjugate gradient method (as well as other Krylov subspace methods). It can be shown that as soon as one of the extremal eigenvalues is modestly well approximated by a Ritz value, the pocedure converges from then on as a process in which this eigenvalue is absent, i.e., a process with a reduced condition number. Note that superlinear convergence behaviour in this connection is used to indicate linear convergence with a factor that is gradually decreased during the process as more and more of the extremal eigenvalues are sufficiently well approximated (for details on this see [80]).

5.1.3 Further references

A more formal presentation of CG, as well as many theoretical properties, can be found in the textbook by Hackbusch [39]. A shorter presentation is given in [35]. An overview of papers, published in the first 25 years of existence of the method, is given in [34]. Vector processing and parallel computing aspects are discussed in [23] and [57].

5.2 MINRES and SYMMLQ:

When A is not positive definite, but still symmetric, then we can construct an orthogonal basis for the Krylov subspace, as we have seen before. We write the recurrence relations slightly different as

$$AR_i = R_{i+1}\bar{T}_i,$$

with

$$\begin{array}{cc} \quad\longleftarrow \; i \; \longrightarrow & \\ \bar{T}_i = \begin{pmatrix} \ddots & \ddots & & \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots \\ & & & \ddots \end{pmatrix} & \begin{matrix} \uparrow \\ i+1 \\ \downarrow \end{matrix} \end{array}$$

In this case we have the problem that $(\;,\;)_A$ does not define an innerproduct. However we can still try to minimize the residual. We look for an

$$x_i \in \{r_0, Ar_0, \ldots, A^{i-1}r_0\}, \quad x_i = R_i\bar{y}$$

$$\|Ax_i - b\|_2 = \|AR_i\bar{y} - b\|_2$$

$$= \|R_{i+1}\bar{T}_i y - b\|_2$$

Now we exploit the fact that $R_{i+1}D_{i+1}^{-1}$, with $D_{i+1} = diag(\|r_0\|_2, \|r_1\|_2, ..., \|r_i\|_2)$, is an orthonormal transformation with respect to the current Krylov subspace:

$$\|Ax_i - b\|_2 = \|D_{i+1}\bar{T}_i y - \|r_0\|_2 e_1\|_2$$

and this final expression can simply be seen as a minimum norm least squares problem.
The element in the $(i+1, i)$ position of $\bar{T}_i$ can be transformed to zero by a simple Givens rotation and the resulting upper bidiagonal system (the other subdiagonal elements being removed in previous iteration steps) can simply be solved, which leads to the so-called MINRES method [60].
Another possibility is to solve the system $T_i y = \|r_0\|_2 e_1$, as in the CG method (T_i is the upper i by i part of $\bar{T}_i$. Other than in CG we cannot rely on the existence of a Choleski decomposition (since A is not positive definite). An alternative is then to decompose T_i by an LQ-decomposition. This again leads to simple recurrences and the resulting method is known as SYMMLQ [60].

5.3 Parallelism and data locality in preconditioned CG:

For successful application of CG one needs that the matrix A is symmetric positive definite. In other short recurrence methods, other properties of A may be desirable, but we will not exploit these properties explicitly in the discussion on parallel aspects.
Most often, the conjugate gradients method is used in combination with some kind of preconditioning. This means that the matrix A can be thought of to be multiplied with some suitable approximation K^{-1} for A^{-1}. Usually, K is constructed as an approximation of A, such that systems like $Ky = z$ are much more easy to solve as $Ax = b$. Unfortunately, a popular class of preconditioners, based upon incomplete factorization of A, do not lend themselves very much for parallel implementation. We will discuss some approaches to obtain more parallelism in the preconditioner in section 9.1. At the moment we will assume that the preconditioner is chosen such that the parallelism in solving $Ky = z$ is comparable with the parallelism in computing Ap, for given p.
For CG it is also required that the preconditioner K 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 5.1.1. 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 to formulate CG in such a way that synchronization points can be taken together. We will see such approaches further on.
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. Apart from the computation of Ap and the computations with K, we need to load 7 vectors for 10 vector floating point operations. This means that for this part of the computation only 10/7 floating point operation can be carried out per memory reference in average.

Several authors ([11, 52, 53]) have attempted to improve this ratio, and to reduce the number of synchronization points. In our formulation of CG there are two such synchronization points, namely the com-

putation of both innerproducts.
Meurant [52] (see also [68]) has proposed a variant in which there is only one synchronization point, however at the cost of a possibly reduced numerical stability, and one additional innerproduct. In this scheme the ratio between computations and memory references is about 2.
We show here another variant, proposed by Chronopoulos and Gear [11].

x_0= initial guess; $r_0 = b - Ax_0$;
$q_{-1} = p_{-1} = 0; \beta_{-1} = 0$;
Solve w_0 from $Kw_0 = r_0$;
$s_0 = Aw_0$;
$\rho_0 = (r_0, w_0); \mu_0 = (s_0, w_0)$;
$\alpha_0 = \rho_0/\mu_0$;
for $i = 0, 1, 2, ...$
  $p_i = w_i + \beta_{i-1}p_{i-1}$;
  $q_i = s_i + \beta_{i-1}q_{i-1}$;
  $x_{i+1} = x_i + \alpha_i p_i$;
  $r_{i+1} = r_i - \alpha_i q_i$;
  if x_{i+1} accurate enough then quit;
  Solve w_{i+1} from $Kw_{i+1} = r_{i+1}$;
  $s_{i+1} = Aw_{i+1}$;
  $\rho_{i+1} = (r_{i+1}, w_{i+1})$;
  $\mu_{i+1} = (s_{i+1}, w_{i+1})$;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
  $\alpha_{i+1} = \frac{\rho_{i+1}}{\mu_{i+1} - \rho_{i+1}\beta_i/\alpha_i}$;
end i;

In this scheme all vectors need only be loaded once per pass of the loop, which leads to a better exploitation of the data (improved data locality). However, the price is that we need $2n$ flops more per iteration step. Chronopoulos and Gear [11] claim stability, based upon their numerical experiments.
Instead of 2 synchronization points, as in the standard version of CG, we have now only one synchronization point, as the next loop can only be started when the innerproducts at the end of the previous loop have been assembled. Another slight advantage is that these innerproducts can be computed in parallel.
Chronopoulos and Gear [11] propose to further improve the data locality and parallelism in CG by combining s successive steps. Their algorithm is based upon the following property of CG. The residual vectors $r_0, ..., r_i$ form an orthogonal basis (assuming exact arithmetic) for the Krylov subspace spanned by $r_0, Ar_0, ..., A^{i-1}r_0$. When arrived at r_j, the vectors $r_0, r_1, ..., r_j,\ Ar_j, ..., A^{i-j-1}r_j$ also form a basis for this subspace. Hence, we may combine s successive steps by generating $r_j, Ar_j, ..., A^{s-1}r_j$ first, and then do the orthogonalization and the updating of the current solution with this blockwise extended subspace. This approach leads to a slight increase in flops in comparison with s successive steps of the standard CG, and also one additional matrix vector product is required per s steps.
The main drawback in this approach seems to be the potential numerical instability. Depending on the spectral properties of A, the set $r_j, ..., A^{s-1}r_j$ may tend to converge to a vector in the direction of a dominating eigenvector, or, in other words, may tend to dependence for increasing values of s. The authors claim to have seen successful completion of this approach, with no serious stability problems, for small values of s. Nevertheless, it seems that s-step CG, because of these problems, has a bad reputation (see also [69]). However, a similar approach, suggested by Chronopoulos and Kim [12] for other processes such as GMRES, seems to be more promising. Several authors have pursued this research direction, and we will come back to this in section 7.3.

We consider still another variant of CG, in which there is possibility to overlap all of the communication time with useful computations. This variant is just a reorganized 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.
Another advantage over the previous scheme is that no additional operations are required.
It is assumed that the preconditioner K can be written as $K = (LL^T)^{-1}$. Furthermore, it is assumed that the preconditioner has a block structure, corresponding to the gridblocks assigned to the processors, so that communication (if necessary) can be overlapped with computation.

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;

Now we discuss how this scheme may lead to an efficient parallel scheme, and how local memory (vector registers, cache, ...) can be exploited.

1. All computing intensive operations can be carried out in parallel. Only for the operations (2), (3), (7), (8), (9), and (0), communication between processors is required. We have assumed that the communication in (2), (7), and (0) can be largely overlapped with computation.

2. 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).

3. 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).

4. Steps (1), (2), and (3) can be combined: the computation of a segment of p_i can be followed immediately by the computation of a segment of q_i (2), and this can be followed by the computation of a part of the innerproduct in (3). This saves on load operations for segments of p_i and q_i.

5. Depending on the structure of L, the computation of segments of r_{i+1} in (6) can be followed by operations in (7), which can be followed by the computation of parts of the innerproduct in (8), and the computation of the norm of r_{i+1}, required for (9).

6. The computation of β_i can be done as soon as the computation in (8) has been completed. At that moment, the computation for (1) can be started if the requested parts of w_i have been completed in (0).

7. If no preconditioner is used, then $w_i = r_i$, and steps (7) and (0) have to be skipped. Step (8) has to be replaced by $\rho_i = (r_{i+1}, r_{i+1})$. Now we need useful computation in order to overlap the communication for this innerproduct. To this end, one might split the computation in (4) per processor in two parts. The first of these parts are computed in paralell in overlap with (3), while the parallel computation of the other parts is used in order to overlap the communication for the computation of ρ_i.

5.4 Parallel performance of CG:

Some realistic 3D computational fluid dynamics simulation problems, as well as other problems, lead to the necessity to solve linear systems $Ax = b$ with a matrix of very large order, billions of unknowns, say. If not of very special structure, such systems are not likely to be solved by direct elimination methods.
For such very large (sparse) systems we will have to exploit parallelism in combination with suitable solution techniques, like for instance iterative solution methods.
From a parallel point of view CG mimics very well parallel performance properties of a variety of iterative methods such as Bi-CG, CGS, BiCGSTAB, QMR, and others.
In this section we study the performance of CG on parallel distributed memory systems and we report on some supporting experiments on actual existing machines. Guided by our experiments we will discuss the suitability of CG for Massively Parallel Processing systems.

All computational intensive elements in preconditioned CG (updates, innerproducts, and matrix vector operations) are trivially parallelizable for shared memory machines [23], except possibly for the preconditioning step: *Solve* w_{i+1} *from* $Kw_{i+1} = r_{i+1}$. For the latter operation parallelism depends very much on the choice for K. In this section we restrict ourselves to block Jacobi preconditioning, where the blocks have been chosen so that each processor can handle one block independently of the others. For other preconditioners that allow some degree of parallelism see [23].
For a distributed memory machine at least some of the steps require communication between processors: the accumulation of innerproducts and the computation of Ap_i (depending on the non-zero structure of A and the distribution of the non-zero elements over the processors). We consider in some more detail the situation where A is a block-tridiagonal matrix of order N, and we assume that all blocks are of order $\sqrt{N}$:

$$A = \begin{pmatrix} A_1 & D_1 & & \\ D_1 & A_2 & D_2 & \\ & D_2 & \ddots & \ddots \\ & & \ddots & \\ & & & \end{pmatrix},$$

in which the D_i are diagonal matrices, and the A_i are tridiagonal matrices. Such systems occur quite frequently in finite difference approximations in 2 space dimensions. Our discussion can easily be adapted to 3 space dimensions.

5.4.1 Processor configuration and data distribution

For simplicity we will assume that the processors are connected as a 2D grid with $p \times p = P$ processors.
The data have been distributed in a straight forward manner over the processor memories and we have not attempted to fully exploit the underlying grid structure for the given type of matrix in order to reduce communication as much as possible. In fact it will turn out that in our case the communication for the

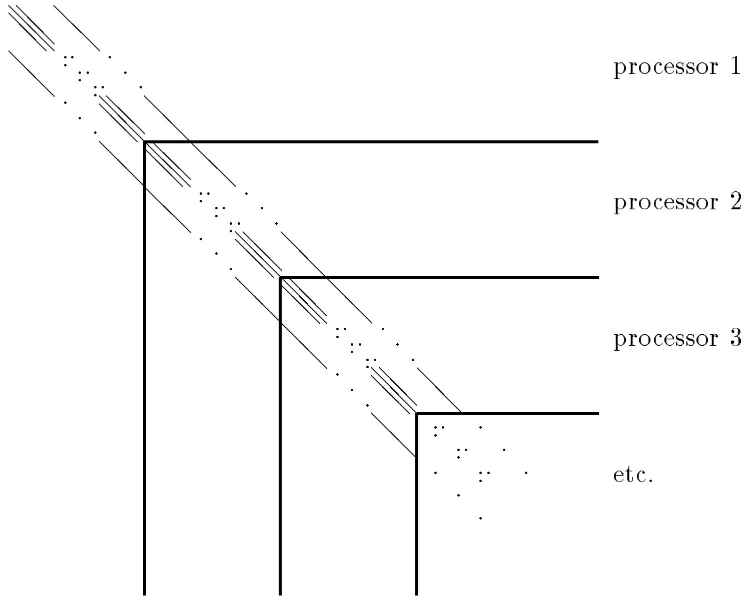

Fig.1: Distribution of A over the processors..

matrix vector product plays only a minor role for matrix systems of large size .
Because of symmetry only the 3 non-zero diagonals in the upper triangular part of A need to be stored, and we have chosen to store successive parts of length N/P of each diagonal in consecutive neighbouring processors. In Figure 1 we see which part of A is represented by the data in the memory of a given processor.

The blocks for block Jacobi are chosen to be the diagonal blocks that are available on each processor, and the various vectors (r_i, p_i, etc.) have been distributed likewise, i.e. each processor holds a section of length N/P of these vectors in its local memory.

5.4.2 Required Communication

matrix vector product It is easily seen for a $2D$ processor grid (as well as for many other configurations, including hypercube and pipeline), that the matrix vector product can be completed with only neighbour-neighbour communication. This means that the communication costs do not increase for increasing values of p. If one follows a domain decomposition way of approach, in which the finite difference discretization grid is subdivided into p by p subgrids (p in x-direction and p in y-direction), then the communication costs are smaller than the computational costs by a factor of $\mathcal{O}(\frac{\sqrt{N}}{p})$.
In [17] much attention is given to this sparse matrix vector product and it is shown that the time for communication can almost completely be overlapped with computational work. Therefore, with adequate coding the matrix vector products do not necessarily lead to serious communication problems, even not for relatively small-sized problems.
On a MEIKO SP1 (located at Utrecht University, this machine has only 4 processors) we have observed, for $N = 90000$, a speed-up by a factor of 1.85 for two processors, and of 1.96 when overlap possibilities are exploited. In both cases we expect, by extrapolating our timing results, a factor of 2 for very large N. According to a naive interpretation of Amdahl's law we might expect a severe degradation in performance for more than two processors. However, if we increase the size of the problem for increasing numbers of processors then the local communication time for the matrix product does not increase so that it does not pose limits on the performance when we increase the value of p.

vector update In our case these operations do not require any communication and we should expect linear speed up when increasing the number of processors P.

inner product For the innerproduct we need global communication for assembly and we need global communication for the distribution of the assembled innerproduct over the processors. For a $p \times p$ processor grid these communication costs are proportional with p. This means that for a constant length of the vectorparts per processor, these communicationcosts will dominate for values of p large enough. This is quite unlike the situation for the matrix vector product and as we will see it may be a severely limiting factor in achieving high speed-ups in a massively parallel environment.

For the MEIKO SP1 we have done some experiments in order to determine the costs of inter proces-

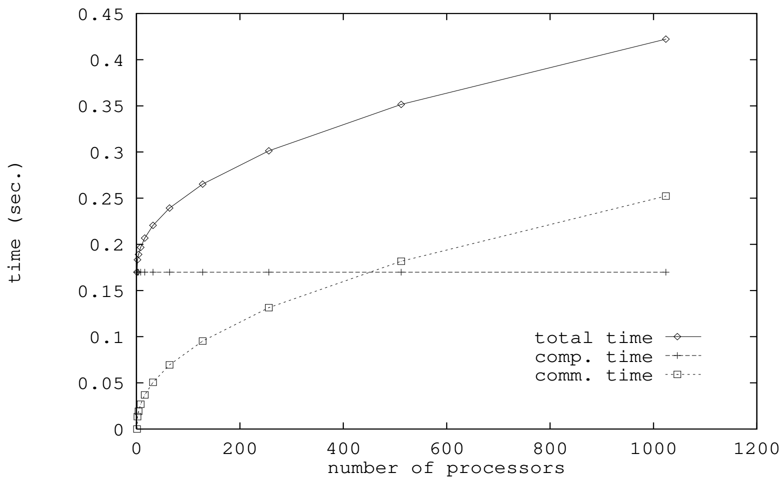

Fig.2: Modelled timings for 1 iteration with CG.

sor communication and for communication. Assuming that the costs for communication (for the innerproducts) grow linearly with the length of the path of communication we have modelled the wall-clock time for 1 iteration with CG, for matrices of order $90000P$, as in Figure 2. Note that we have increased the size of the linear system linearly with the number of processors, which seems realistically since with larger computers one aims to solve larger systems. The value 90000 has been chosen since this is more or less the size of the part of the system that can be kept in the local memory of one processor of our MEIKO machine.

From this Figure we learn that for P slightly larger than 400 the communication costs may be expected to dominate, and eventually they will lead to very low speed-ups (even for systems for which the size is as large as the total memory permits). We also see, that if overlap of communication and computation is possible, then potentially the communication can be hidden for values of P less than 400, but this demands for a reformulation of the CG algorithm. Of course, these expectations are based on a model, but we have also carried out similar experiments on the 512 processor Parsytec GCel-3/512 of the University of Amsterdam [15]. In particular we have observed on that machine that the communication time for the innerproduct increases like $\sqrt{P}$, which just explains the behaviour of our model for the MEIKO-type of architecture.

Our experiments and our modelling approach clearly show that even a method like CG, which might be anticipated to be highly parallel, may suffer severely from the communication overhead due to the required innerproducts. Our study indicates that if we want reasonable speed-up in a massively parallel environment then the local memories should also be much larger when the number of processors is increased in order to accomodate for systems large enough to compensate for the increased global communication costs.

Another approach would be to modify the CG method such that the innerproducts take relatively less time. Many of such approaches have been studied recently. A quite popular approach is to reformulate CG such that the required innerproducts can be computed simultaneously, so that the communication overhead is reduced (the communication required for 2 simultaneous innerproducts is almost the same as for 1 innerproduct). An extreme form of this approach is to reformulate CG so that a number of basis vectors for the search space are computed without making them orthogonal. The orthogonalization is then carried out afterwards, and in this approach most of the communication can be combined. The numerical stability of these approaches is still a point of concern. For an overview and further references see [6]. For some other iterative methods, such as GMRES, this approach can be quite effective as is shown in [17].

Still another approach is to try to more useful computational work per iteration step, so that the communication for the two innerproducts takes relatively less time. One way to do this is to use polynomial preconditioning, i.e., the preconditioner consists of a number of matrix vector products with the matrix A. This may work well in situations where the matrix vector product requires only little (local) communication. Another way is to apply domain decomposition: the given domain is split into P, say, subdomains with estimated values for the solutions on the interfaces. Then all the subproblems are solved independently and in parallel. This way of approximating the solution may be viewed as a preconditioning step in an iterative method. In this way we do more computational work per communication step. Unfortunately, depending on the problem and on the way of decoupling the subdomains one may need a larger number of iteration steps for larger values of P, which may then, of course, detoriate the overall efficiency of the domain decomposition approach. For more information on this approach we also refer to references given in [6].

If a given architecture permits the overlap of communication with computation, then we may try to reformulate CG in order to create possibilities for overlap. For the (extrapolated) MEIKO this may help for values of P up to about 400. For larger P we will see communication dominating anyhow, but the adverse effects can be lessened. A stable reformulation of CG which has this effect has been described in [20].

6 Unsymmetric 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 $A^TAx = A^Tb$ 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

6.1 Normal Equations:

The first solution seems rather obvious. However, it has 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.

There have been made several proposals to improve the numerical stability of this rather robust approach. The most well-known is by Paige and Saunders [61] 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^TA.

Another approach to improve the numerical stability of this normal equations approach is suggested by Björck and Elfving [8]. They observed that the matrix A^TA is used in the construction of the iteration coefficients through an innerproduct like (p, A^TAp). They simply suggest to replace such an innerproduct by (Ap, Ap).

The use of conjugate gradients in a least squares context, as well as a theoretical comparison with SIRT type methods, is discussed in [81] and [82].

An interesting variant of LSQR is the so-called Craig's method [61]. The easiest way to think of this method is to apply Conjugate Gradients to the system $A^TAx = A^Tb$, with the following choice for the innerproduct

$$[x, y] \equiv (x, (A^TA)^{-1}y),$$

which defines a proper innerproduct if A is of full rank (see section 5.1.1).

First note that the two innnerproducts in CG (as in section 5.1.1 can be computed without inverting A^TA:

$$[p_i, A^TAp_i] = (p_i, p_i),$$

and, assuming that $b \in R(A)$ so that $Ax = b$ has a unique solution x (since A has full rank):

$$\begin{aligned} [r_i, r_i] &= [A^T(Ax_i - b), A^T(Ax_i - b] \\ &= [A^TA(x_i - x), A^T(Ax_i - b)] \\ &= (x_i - x, A^T(Ax_i - b)) \end{aligned}$$

$$= (Ax_i - b, Ax_i - b) \tag{6.1a}$$

Apparently, we are with CG minimizing

$$\begin{aligned} [x_i - x, A^TA(x_i - x)] &= (x_i - x, x_i - x) \\ &= \|x_i - x\|_2^2 \end{aligned} \tag{6.1b}$$

that is, in this approach the Euclidean norm of the error is minimized. Note, however, that the rate of convergence of Craig's method is determined by the condition number of A^TA, so that this method is only attractive if one has a good preconditioner for A^TA.

6.2 FOM and GMRES:

The second approach is to form explicitly an orthonormal basis for the Krylov subspace. Since A is not symmetric we no longer have 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:

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

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

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

$$AV_i = V_i H_i + h_{i+1,i} v_{i+1} e_i^T \,, \tag{6.2a}$$

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

$$AV_i = V_{i+1} \bar{H}_i, \tag{6.2b}$$

after which the projected system, with an $i+1$ by i upper Hessenberg matrix can be solved as a least squares system. In GMRES [72] 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 (6.2a)) is similar to the conjugate gradient approach (or SYMMLQ), the second approach (based upon (6.2b)) is similar 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. This algorithm is referred to as GMRES(m). Below we give a scheme for GMRES(m) which may be suitable to develop a computer code. It solves $Ax = b$, with a given preconditioner K.

```
x_0 is an initial guess;
for j = 1, 2, ....
    Solve r from Kr = b - Ax_0;
    v_1 = r/||r||_2;
    s := ||r||_2 e_1;
    for i = 1, 2, ..., m
        Solve w from Kw = Av_i;
    orthogonalization of w
    against v's, by modified
    Gram-Schmidt process
        for k = 1, ..., i
            h_{k,i} = (w, v_k);
            w = w - h_{k,i} v_k;
        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(x̃, i); quit);
    end i;
    UPDATE(x̃, m);
end j;
```

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

```
Compute y as the solution of Hy = s̃,
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),
s̃ represents the first i components of s;
x̃ = x_0 + y_1 * v_1 + y_2 v_2 + ... + y_i v_i;
s_{i+1} equals ||b - Ax̃||_2;
if this component is not small enough
then x_0 = x̃;
else quit;
```

Another scheme for GMRES, based upon Householder orthogonalization instead of modified Gram-Schmidt has been proposed in [92]. For certain applications it seems attractive to invest in additional computational work in turn for improved numerical properties: the better orthogonality might save iteration steps.

The eigenvalues of H_i are the Ritz values of $K^{-1}A$ with respect to the Krylov subspace spanned by v_1, ..., v_i. They approximate eigenvalues of $K^{-1}A$ increasingly well for increasing dimension i.

There is an interesting and simple relation between the two different Krylov subspace projection approaches (6.2a), the "FOM" approach, and (6.2b), the "GMRES" approach. The projected system matrix $\bar{H}_i$ is transformed by a Givens rotations to an upper triangular matrix (with last row equal to zero). So, in fact, the major difference between FOM and GMRES is that in FOM the last ((i+1)-th row is simply discarded, while in GMRES this row is rotated to a zero vector. Let us characterize the Givens rotation, acting on rows i and i+1, in order to zero the element in position $(i+1, i)$, by the sine s_i and the cosine c_i. Let us further denote the residuals for FOM with an superscript F and those for GMRES with superscript G. Then the above observations lead to the following results for FOM and GMRES (for details see [72] and [9]).

1. The reduction for successive GMRES residuals is given by

   $$\frac{\|r_k^G\|_2}{\|r_{k-1}^G\|_2} = |s_k| \,. \tag{6.2c}$$

   ([72]: p. 862, Proposition 1)

2. If $c_k \neq 0$ then the FOM and the GMRES residuals are related by

   $$\|r_k^G\|_2 = |c_k| \, \|r_k^F\|_2 \tag{6.2d}$$

   ([9]: theorem 5.1)

From these relations we see that when GMRES has a local significant reduction in the norm of the residual (i.e., s_k is small), then FOM gives about the same result as GMRES (since $c_k^2 = 1 - s_k^2$). On the other hand when FOM has a break-down ($c_k = 0$), then the GMRES does not lead to an improvement in the same iteration step.
Because of these relations we can link the convergence behaviour of GMRES with the convergence of Ritz values (the eigenvalues of the "FOM" part of the upper Hessenberg matrix). This has been exploited in [88], for the analysis and explanation of local effects in the convergence behaviour of GMRES.

In order to limit the required amount of memory storage and the amount of flops per iteration step, one often restarts the GMRES method after each m steps. This restarted version is commonly referred to as GMRES(m), while the not-restarted method often is called Full GMRES.

There are various different implementations of FOM and GMRES. Among those equivalent with GMRES are: Orthomin [91], Orthodir [44], Axelsson's method [3] and GENCR [27]. These methods are often more expensive than GMRES per iteration step. Orthomin seems to be still popular, since this variant can be easily truncated (Orthomin(s)), in contrast to GMRES. The truncated or restarted versions of these algorithms are not necessarily mathematically equivalent.
Methods that are mathematically equivalent with FOM are: Orthores [44] and GENCG [13, 93]. In these methods the approximate solutions are constructed such that they lead to orthogonal residuals (which form a basis for the Krylov subspace; analogously to the CG method). A good overview of all these methods and their relations is given in [71].

6.3 Rank-one updates for the Matrix Splitting:

Iterative methods can be derived from a splitting of the matrix, and we have used the very simple splitting $A = I - R$, with $R = I - A$, in order to derive the projection type methods. In [26] it is suggested to update the matrix splitting with information obtained in the iteration process. We will give the flavour of this method here since it turns out that it has an interesting relation with GMRES. This relation is exploited in [89] for the construction of new classes of GMRES-like methods, that can be used as cheap alternatives for the increasingly expensive full GMRES method.

Assume that the matrix splitting in the k-th iteration step is given by $A = H_k^{-1} - R_k$, then we obtain the iteration formula

$$x_k = x_{k-1} + H_k r_{k-1} \quad \text{with} \quad r_k = b - Ax_k.$$

The idea is now to construct H_k by a suitable rank-one update to H_{k-1}:

$$H_k = H_{k-1} + u_{k-1} v_{k-1}^T,$$

which leads to

$$\text{(6.3a)} \quad x_k = x_{k-1} + (H_{k-1} + u_{k-1} v_{k-1}^T) r_{k-1}$$

or

$$\begin{aligned} r_k &= r_{k-1} - A(H_{k-1} + u_{k-1}v_{k-1}^T)r_{k-1} \\ \text{(6.3b)} \quad &= (I - AH_{k-1})r_{k-1} - Au_{k-1}v_{k-1}^T r_{k-1} \\ &= (I - AH_{k-1})r_{k-1} - \mu_{k-1}Au_{k-1}. \end{aligned}$$

The optimal choice for the update would have been to select u_{k-1} such that

$$\mu_{k-1}Au_{k-1} = (I - AH_{k-1})r_{k-1},$$

or

$$\mu_{k-1}u_{k-1} = A^{-1}(I - AH_{k-1})r_{k-1}.$$

However, A^{-1} is unknown and the best approximation we have for it is H_{k-1}. This leads to the choice

$$\text{(6.3c)} \qquad \bar{u}_{k-1} = H_{k-1}(I - AH_{k-1})r_{k-1}.$$

The constant μ_{k-1} is chosen such that $\|r_k\|_2$ is minimal as a function of μ_{k-1}. This leads to

$$\mu_{k-1} = \frac{1}{\|A\bar{u}_{k-1}\|_2^2}(A\bar{u}_{k-1})^T(I - AH_{k-1})r_{k-1}.$$

Since v_{k-1} has to be chosen such that $\mu_{k-1} = v_{k-1}^T r_{k-1}$, we have the following obvious choice for it

$$\text{(6.3d)} \quad \bar{v}_{k-1} = \frac{1}{\|A\bar{u}_{k-1}\|_2^2}(I - AH_{k-1})^T A\bar{u}_{k-1}$$

(note that from the minimization property we have that $r_k \perp A\bar{u}_{k-1}$).

In principle the implementation of the method is quite straight forward, but note that the computation of r_{k-1}, $\bar{u}_{k-1}$ and $\bar{v}_{k-1}$ costs 4 matrix vector multiplications with A (and also some with H_{k-1}). This would make the method too expensive for being of practical interest. Also the updated splitting is most likely a dense matrix if we carry out the updates explicitly.
We will now show, still following the lines set forth in [26], that there are orthogonality properties, following from the minimization step, by which the method can be implemented much more efficiently.
We define

1. $c_k = \frac{1}{\|A\bar{u}_k\|_2} A\bar{u}_k$ (note that $r_{k+1} \perp c_k$)
2. $E_k = I - AH_k$

From (6.3b) we have that $r_k = E_k r_{k-1}$, and from (6.3c):

$$A\bar{u}_k = AH_k E_k r_k = \alpha_k c_k$$

or

$$\text{(6.3e)} \qquad \begin{aligned} c_k &= \frac{1}{\alpha_k}(I - E_k)E_k r_k \\ &= \frac{1}{\alpha_k}E_k(I - E_k)r_k \end{aligned}$$

Furthermore (on behalf of (6.3c)):

$$\begin{aligned} E_k &= I - AH_k \\ &= I - AH_{k-1} - A\bar{u}_{k-1}\bar{v}_{k-1}^T \\ &= I - AH_{k-1} - A\bar{u}_{k-1}(A\bar{u}_{k-1})^T \\ &\quad (I - AH_{k-1})\frac{1}{\|A\bar{u}_k\|_2^2} \\ &= (I - c_{k-1}c_{k-1}^T)E_{k-1} \\ \text{(6.3f)} \qquad &= \prod_{i=0}^{k-1}(I - c_i c_i^T)E_0. \end{aligned}$$

We see that the operator E_k has the following effect on a vector. The vector is multiplied by E_0 and then orthogonalized with respect to c_0, ..., c_{k-1}. Now we have from (6.3e) that

$$c_k = \frac{1}{\alpha_k}E_k y_k,$$

and hence

$$\text{(6.3g)} \qquad c_k \perp c_0, ..., c_{k-1}.$$

A consequence from (6.3g) is that

$$\prod_{j=0}^{k-1}(I - c_j c_j^T) = I - \sum_{j=0}^{k-1} c_j c_j^T = I - P_{k-1}$$

and therefore

$$\text{(6.3h)} \qquad P_k = \sum_{j=0}^{k} c_j c_j^T.$$

The actual implementation is based on the above properties. Given r_k we compute r_{k+1} as follows (and we update x_k in the corresponding way):

$$r_{k+1} = E_{k+1} r_k.$$

With $\xi^{(0)} = E_0 r_k$ we first compute (with the c_j from previous steps):

$$E_k r_k = \xi^{(k)} \equiv (I - \sum_{j=0}^{k-1} c_j c_j^T)\xi^{(0)} = \prod_{j=0}^{k-1}(I - c_j c_j^T)\xi^{(0)}.$$

The expression with $\sum$ leads to a Gram-Schmidt formulation, the expression with $\prod$ leads to the Modified Gram-Schmidt variant.
The computed updates $-c_j^T\xi^{(0)}c_j$ for r_{k+1} correspond to updates

$$c_j^T\xi^{(0)}A^{-1}c_j = c_j^T\xi^{(0)}u_j/\|Au_j\|_2$$

for x_{j+1}. These updates are in the scheme, given below, represented by η.

From (6.3c) we know that

$$\bar{u}_k = H_k E_k r_k = H_k \xi^{(k)}.$$

Now we have to make $A\bar{u}_k \sim c_k$ orthogonal w.r.t. c_0, ..., c_{k-1}, and to update $\bar{u}_k$ accordingly. Once we have done that we can do the final update step to make H_{k+1}, and we can update both x_k and r_k by the corrections following from including c_k. The orthogonalization step can be carried out easily as follows. Define $c_k^{(k)} \equiv \alpha_k c_k = AH_k E_k r_k = (I - E_k)E_k r_k$ (see (6.3e)) $= (I - E_0 + P_{k-1}E_0)\xi^{(k)}$ (see (6.3f)) $= AH_0\xi^{(k)} + P_{k-1}(I - AH_0)\xi^{(k)} = c_k^{(0)} + P_{k-1}\xi^{(k)} - P_{k-1}c_k^{(0)}$. Note that the second term vanishes since $\xi^{(k)} \perp c_0, ..., c_{k-1}$.

The resulting scheme for the k-th iteration step becomes:

1. $\xi^{(0)} = (I - AH_0)r_k$; $\eta^{(0)} = H_0 r_k$;
   for $i = 0, ..., k-1$ do
   $\alpha_i = c_i^T\xi^{(i)}$;
   $\xi^{(i+1)} = \xi^{(i)} - \alpha_i c_i$; $\eta^{(i+1)} = \eta^{(i)} + \alpha_i u_i$;
2. $u_k^{(0)} = H_0\xi^{(k)}$; $c_k^{(0)} = Au_k^{(0)}$;
   for $i = 0, ..., k-1$ do
   $\beta_i = -c_i^T c_k^{(i)}$; $c_k^{(i+1)} = c_k^{(i)} + \beta_i c_i$;
   $u_k^{(i+1)} = u_k^{(i)} + \beta_i u_i$;
   $c_k = c_k^{(k)}/\|c_k^{(k)}\|_2$; $u_k = u_k^{(k)}/\|c_k^{(k)}\|_2$;
3. $x_{k+1} = x_k + \eta^{(k)} + u_k c_k^T\xi^{(k)}$;
   $r_{k+1} = (I - c_k c_k^T)\xi^{(k)}$;

Remarks

1. The above scheme is a Modified Gram Schmidt variant, given in [89], of the original scheme in [26].

2. If we keep H_0 fixed, i.e., $H_0 = I$, then the method is not scaling invariant (the results for $\rho Ax = \rho b$ depend on ρ). In [89] a scaling invariant method is suggested.

3. Note that in the above implementation we have 'only' two matrix vector products per iteration step. In [89] it is shown that in many cases we may also expect comparable converge as for GMRES in half the number of iteration steps.

4. A different choice for $\bar{u}_{k-1}$ does not change the formulas for $\bar{v}_{k-1}$ and E_{k-1}. For each different choice we can derive similar schemes as the one above.

5. From (6.3b) we have

$$r_k = r_{k-1} - AH_{k-1}r_{k-1} - \mu_{k-1}Au_{k-1}.$$

In view of the previous remark we might also make the different choice $\bar{u}_{k-1} = H_{k-1}r_{k-1}$. With this choice, we obtain a variant which is algebraically identical to GMRES (for a proof of this see [89]). This GMRES variant is obtained by the following changes in the previous scheme: Take $H_0 = 0$ (note that in this case we have that $E_{k-1}r_{k-1} = r_{k-1}$, and hence we we may skip part 1 of the above algorithm), and set $\xi^{(k)} = r_k$, $\eta^{(k)} = 0$. In step 2 start with $u_k^{(0)} = \xi^{(k)}$.
The result is a different formulation of GMRES in which we can obtain explicit formulas for the updated preconditioner (i.e., the inverse of A is approximated increasingly well): The update for H_k is $\bar{u}_k c_k^T E_k$ and the sum of these updates gives an approximation for A^{-1}.

6. Also in this GMRES-variant we are still free to select u_k a little bit different. Remember that the leading factor H_{k-1} in (6.3c) was introduced as an approximation for the actually desired A^{-1}. With $\bar{u}_{k-1} = A^{-1}r_{k-1}$, we would have that $r_k = E_{k-1}r_{k-1} - \mu_{k-1}r_{k-1} = 0$ for the minimizing μ_{k-1}. We could take other approximations for the inverse (with respect to the given residual r_{k-1}), e.g., the result vector y obtained by a few steps GMRES applied to $Ay = r_{k-1}$. This leads to the so-called GMRESR family of nested methods (for details see [89]). See also section 6.4. A similar algorithm, named FGMRES, has been derived independently by Saad [70]. In FGMRES the search directions are preconditioned, whereas in GMRESR the residuals are preconditioned. This gives GMRESR direct control over the reduction in norm of the residual. As a result GMRESR can be made robust while FGMRES may suffer from break-down. A further disadvantage of the FGMRES formulation is that this method cannot be truncated, or selectively orthogonalized, as GMRESR can.
In [4] a generalized conjugate gradient method is proposed, a variant of which produces in exact arithmetic identical results as the proper variant of GMRESR, though at higher computational costs and with a classical Gram–Schmidt orthogonalization process instead of the modified process as in GMRESR.

6.4 GMRESR and GMRES$\star$:

By Van der Vorst and Vuik [89] it has been shown how the GMRES-method 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[89].
Similar schemes have been proposed recently. In FGMRES[70] 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.
In exact arithmetic GMRES$\star$ is very close to the Generalized Conjugate Gradient method[4]; GMRES$\star$, however, leads to a more efficient computational scheme.

The GMRES$\star$ algorithm can be described by the following computational scheme:

x_0 is an initial guess; $r_0 = b - Ax_0$;
for $i = 0, 1, 2, 3, \ldots$
  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
    iteration method)
  for $k = 0, \ldots, i-1$
    $\alpha = (c_k, c)$
    $c = c - \alpha c_k$
    $z^{(m)} = z^{(m)} - \alpha u_k$
  $c_i = c/\|c\|_2$; $u_i = z^{(m)}/\|c\|_2$
  $x_{i+1} = x_i + (c_i, r_i)u_i$
  $r_{i+1} = r_i - (c_i, r_i)c_i$
  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. If stagnation occurs, i.e. no progress at all is made in the inner iteration, then it is suggested by Van der Vorst and Vuik[89] to do one (or more) steps of the LSQR method, which guarantees a reduction (but this reduction is often only small).

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. The 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 Bi-CGSTAB. As has been shown by Van der Vorst[87] there are a number of different situations where we may expect stagnation or slow convergence for GMRES(m). In such cases it does not seem wise to use this method.

On the other hand it may also seem questionable whether a method like Bi-CGSTAB 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 Bi-CG 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. It has been shown for the related Conjugate Gradients method that the orthogonality relations are determined largely by the distribution of the weights at the lower end of the spectrum and on the isolated eigenvalues at the upper end of the spectrum[82]. 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 Bi-CGSTAB as the inner iteration process for GMRES$\star$.

De Sturler and Fokkema[19] propose to prevent the outer search directions explicitly from being reinvestigated again in the inner process. This is done by keeping the Krylov subspace that is build 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

(6.4a) $$A_i = (I - C_i C_i^T)A.$$

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 inner products 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. De Sturler and Fokkema[19] suggestions are made for such strategies. Of course, these strategies in cases where we see too little residual reducing effect in the inner iteration process in comparison with the outer iterations of GMRES$\star$.

6.5 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 (5.0a):

(6.5a) $$\alpha_{j+1} r_{j+1} = A r_j - \beta_j r_j - \gamma_j r_{j-1}.$$

As we have seen in the proof for the orthogonality of such a set of vectors (see section 4), we needed the symmetry of the matrix A. In the nonsymmetric case we need instead of (5.0b) that

$$(A r_{j-1}, r_k) = (r_{j-1}, A^T r_k) = 0 \quad \text{for} \quad k < j-2.$$

By similar arguments as in the proof for (5.0a) we conclude that (6.5a) can be used to generate a basis 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,

$$r_j \perp K^{j-1}(A^T; s_0),$$

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 s_j for $K^i(A^T; s_0)$ satisfy the same recurrence relation as the vectors r_j, i.e., with identical recurrence coefficients, then we see that

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

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

(6.5b) $$A R_i = R_i T_i + \alpha_i r_i e_i^T,$$

but now we use the matrix $S_i = [s_0, s_1, ..., s_{i-1}]$ for the projection of the system

$$S_i^T (A x_i - b) = 0,$$

or

$$S_i^T A R_i y - S_i^T b = 0.$$

Using (6.5b) we find that y_i satisfies

$$S_i^T R_i T_i y = (r_0, s_0) e_1.$$

Since $S_i^T R_i$ is a diagonal matrix with diagonal elements (r_j, s_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 [47]. We see that we are in problems when a diagonal element of $S_i^T R_i$ becomes (near) zero, this is referred to in litterature as a serious (near) breakdown. The way to get around this difficulty is the so-called Look-ahead strategy, which comes down to taking a number of successive basis vectors for the Krylov subspace together and to make them blockwise bi-orthogonal. This has been worked out in detail in [62] and [30], [31], [32].
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 has been shown for Conjugate Gradients, the LU decomposition of the tridiagonal system can be updated from iteration to iteration and this leads to a recursive update of the solution vector. This avoids to save all intermediate r and s vectors. This variant of Bi-Lanczos is usually called Bi-Conjugate Gradients, or shortly Bi-CG [28].
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 breakdown of the Bi-CG algorithm. Note that this breakdown can be avoided in the Bi-Lanczos formulation of this iterative solution scheme. It is also avoided in the QMR approach (see Section 5.4.2).
Note that for symmetric matrices Bi-Lanczos generates the same solution as Lanczos, provided that $s_0 = r_0$, 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 cheap 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 in coming subsections.

The following scheme may be used for a computer implementation of the Bi-CG method. In the scheme the equation $Ax = b$ is solved with a suitable preconditioner K.

> x_0 is an initial guess; $r_0 = b - Ax_0$;
> solve w_0 from $Kw_0 = r_0$;
> $\tilde{r}_0$ is an arbitrary vector such that $(w_0, \tilde{r}_0) \neq 0$,
> usually one chooses $\tilde{r}_0 = r_0$ or $\tilde{r}_0 = w_0$;
> solve $\tilde{w}_0$ from $K^T \tilde{w}_0 = \tilde{r}_0$;
> $p_{-1} = \tilde{p}_{-1} = 0; \beta_{-1} = 0; \rho_0 = (w_0, \tilde{r}_0)$;
> for $i = 0, 1, 2, \ldots.$
> $p_i = w_i + \beta_{i-1} p_{i-1}$;
> $\tilde{p}_i = \tilde{w}_i + \beta_{i-1} \tilde{p}_{i-1}$;
> $z_i = Ap_i$;
> $\alpha_i = \frac{\rho_i}{(\tilde{p}_i, z_i)}$;
> $r_{i+1} = r_i - \alpha_i z_i$;
> $\tilde{r}_{i+1} = \tilde{r}_i - \alpha_i A^T \tilde{p}_i$;
>   solve w_{i+1} from $Kw_{i+1} = r_{i+1}$;
>   solve $\tilde{w}_{i+1}$ from $K^T \tilde{w}_{i+1} = \tilde{r}_{i+1}$;
> $\rho_{i+1} = (\tilde{r}_{i+1}, w_{i+1})$;
> $x_{i+1} = x_i + \alpha_i p_i$;
>   if x_{i+1} is accurate enough then quit;
> $\beta_i = \frac{\rho_{i+1}}{\rho_i}$
> end

As with conjugate gradients, the coefficients α_j and β_j, $j = 0, \ldots, i-1$ build the matrix T_i, as given in formula (5.1b). This matrix is, for BiCG, in general not similar to a symmetric matrix. Its eigenvalues can be viewed as Petrov-Galerkin approximations, with respect to the spaces $\{\tilde{r}_j\}$ and $\{r_j\}$, of eigenvalues of A. For increasing values of i they tend to converge to eigenvalues of A. The convergence patterns, however, may be much more complicated and irregular as in the symmetric case.

6.5.1 Another derivation of Bi-CG

An alternative way to derive Bi-CG comes from considering the following symmetric linear system:

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} \hat{x} \\ x \end{pmatrix} = \begin{pmatrix} b \\ \hat{b} \end{pmatrix}, \quad \text{or} \quad B\tilde{x} = \tilde{b},$$

for some suitable vector $\hat{b}$.
If we select $\hat{b} = 0$ and apply the CG-scheme to this system, then we obtain LSQR again. However, if we select $\hat{b} \neq 0$ and apply the CG scheme with the preconditioner

$$\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix},$$

in the way as is shown in section 4.4.1, then we obtain right away the unpreconditioned Bi-CG scheme for the system $Ax = b$. Note that the CG-scheme can be applied since $K^{-1}B$ is symmetric (but not positive definite) with respect to the bilinear form

$$[p, q] \equiv (p, Kq),$$

which is not a proper innerproduct. Hence, this formulation clearly reveals the two principal weaknesses of Bi-CG (i.e., the causes for break-down situations). Note that if we restrict ourselves to vectors

$$p = \begin{pmatrix} p_1 \\ p_1 \end{pmatrix},$$

then $[p, q]$ defines a proper innerproduct. This situation arises for the Krylov subspace that is created for B and $\tilde{b}$ if $A = A^T$ and $\hat{b} = b$. If, in addition, A is positive definite then $K^{-1}B$ is positive definite symmetric with respect to the generated Krylov subspace, and we obtain the CG-scheme (as expected).
More generally, the choice

$$K = \begin{pmatrix} 0 & K_1 \\ K_1^T & 0 \end{pmatrix},$$

where K_1 is a suitable preconditioner for A, leads to the preconditioned version of the Bi-CG scheme, as given in section 5.4.

The above presentation of Bi-CG was inspired by a closely related presentation of BI-CG in [42]. The latter paper gives a rather untractable reference for the choice of the system $B\tilde{x} = \tilde{b}$ and the preconditioner

$$\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}$$

to [43].

6.5.2 QMR

The QMR method [32] relates to Bi-CG in a similar way as MINRES relates to CG. For stability reasons the basis vectors r_j and $\tilde{r}_j$ are normalized (as is usual in the underlying Bi-Lanczos algorithm, see [94]), which leads to other coefficients in the 3-term recursion formulas.
If we group the residual vectors r_j, for $j = 0, ..., i-1$ in a matrix R_i, then we can write the recurrence relations as

$$AR_i = R_{i+1}\bar{T}_i,$$

with

$$\bar{T}_i = \begin{array}{cc} \longleftarrow \;\; i \;\; \longrightarrow & \\ \begin{pmatrix} \ddots & \ddots & & \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots \\ & & & \ddots \end{pmatrix} & \begin{array}{c} \uparrow \\ i+1 \\ \downarrow \end{array} \end{array}$$

Similar as for MINRES we would like to construct the x_i, with

$$x_i \in \{r_0, Ar_0, \ldots, A^{i-1}r_0\}, \quad x_i = R_i\bar{y}$$

for which

$$\begin{aligned} \|Ax_i - b\|_2 &= \|AR_i\bar{y} - b\|_2 \\ &= \|R_{i+1}\bar{T}_i y - b\|_2 \\ &= \|R_{i+1}D_{i+1}^{-1}\{D_{i+1}\bar{T}_i y - \|r_0\|_2 e_1\}\|_2 \end{aligned}$$

is minimal. However, in this case that would be quite an amount of work since the columns of R_{i+1} are not necessarily orthogonal. Freund and Nachtigal [32] suggest to solve the miniminum norm least squares problem

$$\text{(6.5c)} \qquad \min_{y \in R^i} \|D_{i+1}\bar{T}_i y - \|r_0\|_2 e_1\|_2.$$

This leads to the simplest form of the QMR method. A more general form arises if the least squares problem (6.5c) is replaced by a weighted least squares problem. No strategies are yet known for optimal weights, however.
In [32] the QMR method is carried out on top of a look-ahead variant of the bi-orthogonal Lanczos method, which makes the method more robust. Experiments suggest that QMR has a much smoother convergence behaviour than Bi-CG, but it is not essentially faster than Bi-CG.

6.5.3 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 the above. Sonneveld 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 [79] 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. These three methods have been compared in many studies (see, e.g., [67, 10, 65, 55]).
However, CGS usually shows a very irregular convergence behaviour. This behaviour can even lead to cancellation and a spoiled solution [86]. See also section 6.5.4.

The following scheme carries out the CGS process for the solution of $Ax = b$, with a given preconditioner K:

x_0 is an initial guess; $r_0 = b - Ax_0$;
$\tilde{r}_0$ is an arbitrary vector, such that
$(r_0, \tilde{r}_0) \neq 0$,
e.g., $\tilde{r}_0 = r_0; \rho_0 = (r_0, \tilde{r}_0)$;
$\beta_{-1} = \rho_0; p_{-1} = q_0 = 0$;
for $i = 0, 1, 2, ...$
  $u_i = r_i + \beta_{i-1} q_i$;
  $p_i = u_i + \beta_{i-1}(q_i + \beta_{i-1} p_{i-1})$;
  solve $\hat{p}$ from $K\hat{p} = p_i$;
  $\hat{v} = A\hat{p}$;
  $\alpha_i = \frac{\rho_i}{(\tilde{r}_0, \hat{v})}$;
  $q_{i+1} = u_i - \alpha_i \hat{v}$;
  solve $\hat{u}$ from $K\hat{u} = u_i + q_{i+1}$
  $x_{i+1} = x_i + \alpha_i \hat{u}$;
  if x_{i+1} is accurate enough then quit;
  $r_{i+1} = r_i - \alpha_i A\hat{u}$;
  $\rho_{i+1} = (\tilde{r}_0, r_{i+1})$;
  if $\rho_{i+1} = 0$ then method fails to converge !;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end

In exact arithmetic, the α_j and β_j are the same constants as those generated by BiCG. Therefore, they can be used to compute the Petrov-Galerkin approximations for eigenvalues of A.

6.5.4 Effects of irregular convergence

By very irregular convergence we refer to the situation where successive residual vectors in the iterative process differ in orders of magnitude in norm, and some of these residuals may be even much bigger in norm than the starting residual. We will try to give an impression why this is a point of concern, even if eventually the (updated) residual satisfies a given tolerance. For more details we refer to Sleijpen et al[75, 77].

We will say that an algorithm is *accurate* for a certain problem if the *updated residual* r_j and the *true residual* $b - Ax_j$ are of comparable size for the j's of interest.

The best we can hope for is that for each j the error in the residual is only the result of applying A to the update w_{j+1} for x_j in finite precision arithmetic:

$$r_{j+1} = r_j - Aw_{j+1} - \Delta_A w_{j+1} \tag{6.5d}$$

if

$$x_{j+1} = x_j + w_{j+1}, \tag{6.5e}$$

for each j, where Δ_A is an $n \times n$ matrix for which $|\Delta_A| \preceq n_A\, \overline{\xi}\, |A|$: n_A is the maximum number of nonzero matrix entries per row of A, $|B| \equiv (|b_{ij}|)$ if $B = (b_{ij})$, $\overline{\xi}$ is the relative machine precision, the inequality $\preceq$ refers to element-wise $\leq$. In the Bi-CG type methods that we consider, we compute explicitly the update Aw_j for the residual r_j from the update w_j for the approximation x_j by matrix multiplication: for this part, (6.5d) describes well the local deviations caused by evaluation errors.

In the "ideal" case (i.e. situation (6.5d) whenever we update the approximation) we have that

$$\begin{aligned} r_k - (b - Ax_k) &= \sum_{j=1}^{k} \Delta_A w_j \\ &= \sum_{j=1}^{k} \Delta_A (e_{j-1} - e_j), \end{aligned} \tag{6.5f}$$

where the perturbation matrix Δ_A may depend on j and e_j is the approximation error in the jth approximation: $e_j \equiv x - x_j$. Hence,

$$\begin{gathered} |\|r_k\| - \|b - Ax_k\|| \leq \\ 2k\, n_A\, \overline{\xi}\, \||A|\| \sum_{j=0}^{k} \|e_j\| \leq \\ 2\,\Gamma\,\overline{\xi} \sum_{j=0}^{k} \|r_j\|, \\ \text{where} \quad \Gamma \equiv n_A\, \||A|\|\, \|A^{-1}\|. \end{gathered} \tag{6.5g}$$

Except for the factor Γ, the last upper–bound appears to be rather sharp. We see that approximations with large approximation errors may ultimately lead to an inaccurate result. Such large local approximation errors are typical for CGS, and Van der Vorst[86] describes an example of the resulting numerical inaccuracy is given. If there are a number of approximations with comparable large approximation errors, then their multiplicity may replace the factor k, otherwise it will be only the largest approximation error that makes up virtually the bound for the deviation.

Example. Figure 3 illustrates nicely the loss of accuracy as described above; for other examples, cf. [86]. The convergence history of the updated residuals (the 'circles': ∘∘) and the true residuals (the solid curve: ——) of CGS is given for the matrix SHERMAN4 from the Harwell-Boeing set of test matrices. Here, as in other figures, the norm of the residuals, on log-scale, is plotted (along the vertical axis) against the number of matrix-vector multiplications (along the horizontal axis). The dotted curve $(\cdots\cdot)$ represents the estimated inaccuracy: $2\,\overline{\xi} \sum_{j \leq i} \|r_j\|$ (here with $\Gamma = 1$; cf. (6.5g)).

We will discuss two approaches that lead to a smoother convergence.
— Approaches to obtain the smoothing effect by adding a few lines to existing codes leave the speed of convergence essentially unchanged. One of these approaches leads to optimal accurate approximations

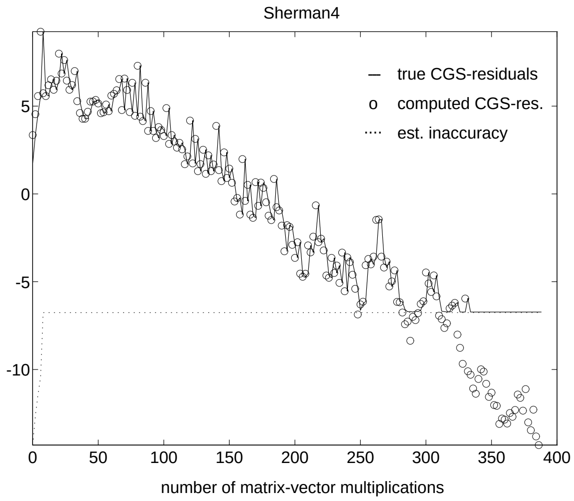

Fig.3: Convergence plot CGS for the true residuals and the updated residuals.

[76] and will be discussed in Section 7. For other ones, we refer to the literature (e.g., [95]).
— In the next section, we concentrate on techniques that really change the convergence: they smooth down and speed up the convergence, and lead to more accurate approximations, all at the same time.

6.5.5 Bi-CGSTAB

Bi-CGSTAB [86] is based on the following observation. Instead of squaring the Bi-CG polynomial, we can construct other iteration methods, by which x_i are generated so that $r_i = \tilde{P}_i(A)P_i(A)r_0$ with other i^{th} degree polynomials $\tilde{P}$. An obvious possibility is to take for $\tilde{P}_j$ a polynomial of the form

(6.5h) $Q_i(x) = (1-\omega_1 x)(1-\omega_2 x)...(1-\omega_i x),$

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 $r_j = Q_j(A)P_j(A)r_0$.
The preconditioned Bi-CGSTAB algorithm for solving the linear system $Ax = b$, with preconditioning K reads as follows:

> x_0 is an initial guess; $r_0 = b - Ax_0$;
> $\bar{r}_0$ is an arbitrary vector, such that
> $(\bar{r}_0, r_0) \neq 0$, e.g., $\bar{r}_0 = r_0$;
> $\rho_{-1} = \alpha_{-1} = \omega_{-1} = 1$;
> $v_{-1} = p_{-1} = 0$;
> for $i = 0, 1, 2, ...$
> $\rho_i = (\bar{r}_0, r_i)$; $\beta_{i-1} = (\rho_i/\rho_{i-1})(\alpha_{i-1}/\omega_{i-1})$;
> $p_i = r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}v_{i-1})$;
> Solve $\hat{p}$ from $K\hat{p} = p_i$;
> $v_i = A\hat{p}$;
> $\alpha_i = \rho_i/(\bar{r}_0, v_i)$;
> $s = r_i - \alpha_i v_i$;
> if $||s||$ small enough then
> $x_{i+1} = x_i + \alpha_i\hat{p}$; quit;
> Solve z from $Kz = s$;
> $t = Az$;
> $\omega_i = (t, s)/(t, t)$;
> $x_{i+1} = x_i + \alpha_i\hat{p} + \omega_i z$;
> if x_{i+1} is accurate enough then quit;
> $r_{i+1} = s - \omega_i t$;
> end

The matrix K in this scheme represents the preconditioning matrix and the way of preconditioning [86]. The above scheme in fact carries out the Bi-CGSTAB procedure for the explicitly postconditioned linear system

$$AK^{-1}y = b,$$

but the vectors y_i and the residual have been back-transformed to the vectors x_i and r_i corresponding to the original system $Ax = b$. Compared to CGS two extra innerproducts need to be calculated.
In exact arithmetic, the α_j and β_j have the same values as those generated by Bi-CG and CGS. Hence, they can be used to extract eigenvalue approximations for the eigenvalues of A (see Bi-CG).
Bi-CGSTAB can be viewed as the product of Bi-CG and GMRES(1). Of course, other product methods can be formulated as well. Gutknecht [38] has proposed BiCGSTAB2, which is constructed as the product of Bi-CG and GMRES(2).

6.5.6 Derivation of Bi-CGSTAB

The polynomial P_i and related polynomials are implicitly defined by the Bi-CG scheme.

Bi-CG:

> x_0 is an initial guess; $r_0 = b - Ax_0$;
> $\hat{r}_0$ is an arbitrary vector, such that
> $(\hat{r}_0, r_0) \neq 0$, e.g., $\hat{r}_0 = r_0$;
> $\rho_0 = 1$;
> $\hat{p}_0 = p_0 = 0$;
> for $i = 1, 2, 3, ...$
> $\rho_i = (\hat{r}_{i-1}, r_{i-1})$; $\beta_i = (\rho_i/\rho_{i-1})$;
> $p_i = r_{i-1} + \beta_i p_{i-1}$;
> $\hat{p}_i = \hat{r}_{i-1} + \beta_i\hat{p}_{i-1}$;
> $v_i = Ap_i$;
> $\alpha_i = \rho_i/(\hat{p}_i, v_i)$;
> $x_i = x_{i-1} + \alpha_i p_i$;
> if x_i is accurate enough then quit;
> $r_i = r_{i-1} - \alpha_i v_i$;
> $\hat{r}_i = \hat{r}_{i-1} - \alpha_i A^T\hat{p}_i$;
> end

From this scheme it is straight forward to show that $r_i = P_i(A)r_0$ and $p_{i+1} = T_i(A)r_0$, in which $P_i(A)$ and $T_i(A)$ are i-th degree polynomials in A. The Bi-CG scheme then defines the relations between these polynomials:

$$T_i(A)r_0 = (P_i(A) + \beta_{i+1}T_{i-1}(A))r_0,$$

and

$$P_i(A)r_0 = (P_{i-1}(A) - \alpha_i AT_{i-1}(A))r_0.$$

In the Bi-CGSTAB scheme we wish to have recurrence relations for

$$r_i = Q_i(A)P_i(A)r_0.$$

With Q_i as in (6.5h) and the Bi-CG relation for the factor P_i and T_i, it then follows that

$$\begin{aligned} Q_i(A)P_i(A)r_0 &= \\ (1-\omega_i A)Q_{i-1}(A)&(P_{i-1}(A) - \alpha_i AT_{i-1}(A))r_0 \\ = \{Q_{i-1}(A)P_{i-1}(A) &- \alpha_i AQ_{i-1}(A)T_{i-1}(A)\}r_0 \\ -\omega_i A\{(Q_{i-1}(A)P_{i-1}(A) &- \alpha_i AQ_{i-1}(A)T_{i-1}(A))\}r_0. \end{aligned}$$

Clearly, we also need a relation for the product $Q_i(A)T_i(A)r_0$. This can also be obtained from the Bi-CG relations:

$$\begin{aligned} Q_i(A)T_i(A)r_0 &= Q_i(A)(P_i(A) + \beta_{i+1}T_{i-1}(A))r_0 \\ &= Q_i(A)P_i(A)r_0 + \beta_{i+1}(1-\omega_i A)Q_{i-1}(A)T_{i-1}(A)r_0 \\ &= Q_i(A)P_i(A)r_0 + \beta_{i+1}Q_{i-1}(A)T_{i-1}(A)r_0 \\ &\quad -\beta_{i+1}\omega_i AQ_{i-1}(A)T_{i-1}(A)r_0. \end{aligned}$$

Finally we have to recover the Bi-CG constants ρ_i, β_i, and α_i by innerproducts in terms of the new vectors that we now have generated.
E.g., β_i can be computed as follows. First we compute

$$\tilde{\rho}_{i+1} = (\hat{r}_0, Q_i(A)P_i(A)r_0) = (Q_i(A^T)\hat{r}_0, P_i(A)r_0).$$

By construction $P_i(A)r_0$ is orthogonal with respect to all vectors $U_{i-1}(A^T)\hat{r}_0$, where U_{i-1} is an arbitrary polynomial of degree $i-1$ at most. This means that we have to consider only the highest order term of $Q_i(A^T)$ when computing $\tilde{\rho}_{i+1}$. This term is given by $(-1)^i\omega_1\omega_2\cdots\omega_i(A^T)^i$. We actually wish to compute

$$\rho_{i+1} = (P_i(A^T)\hat{r}_0, P_i(A)r_0),$$

and since the highest order term of $P_i(A^T)$ is given by $(-1)^i\alpha_1\alpha_2\cdots\alpha_i(A^T)^i$, it follows that

$$\beta_i = (\tilde{\rho}_i/\tilde{\rho}_{i-1})(\alpha_{i-1}/\omega_{i-1}).$$

The other constants can be derived similarly.

Note that in our discussion we have focussed on the recurrence relations for the vectors r_i and p_i, while in fact our main goal is to determine x_i. As in all CG-type methods, x_i itself is not required for continuing the iteration, but it can easily be determined as a "sideproduct" by realizing that an update of the form $r_i = r_{i-1} - \gamma Ay$ corresponds to an update $x_i = x_{i-1} + \gamma y$ for the current approximated solution.

By writing r_i for $Q_i(A)P_i(A)r_0$ and p_i for $Q_{i-1}(A)T_{i-1}(A)r_0$, we obtain the following scheme for Bi-CGSTAB (we trust that, with the foregoing observations, the reader will now be able to verify the relations in Bi-CGSTAB). In this scheme we have computed the ω_i so that $r_i = Q_i(A)P_i(A)r_0$ is minimized in 2-norm as a function of ω_i.

6.5.7 Bi-CGSTAB2 and variants

The residual $r_k = b - Ax_k$ in the Bi-Conjugate Gradient method, when applied to $Ax = b$ with start x_0 can be written formally as $P_k(A)r_0$, where P_k is a k-degree polynomial. These residuals are constructed with one operation with A and one with A^T per iteration step. It was pointed out in [79] that with about the same amount of computational effort one can construct residuals of the form $\tilde{r}_k = P_k^2(A)r_0$, which is the basis for the CGS method. This can be achieved without any operation with A^T. The idea behind the improved efficiency of CGS is that if $P_k(A)$ is viewed as a reduction operator in BiCG, then one may hope that the square of this operator will be a twice as powerful reduction operator. Although this is not always observed in practice, one typically has that CGS converges faster than BiCG. This, together with the absence of operations with A^T, explains the success of the CGS method. A drawback of CGS is that its convergence behavior can look quite erratic, that is the norms of the resdiduals converge quite irregularly, and it may easily happen that $\|r_{k+1}\|_2$ is much larger than $\|r_k\|_2$ for certain k (for an explanation of this see [84]).
In [86] it was shown that by a similar approach as for CGS, one can construct methods for which r_k can be interpreted as $r_k = P_k(A)Q_k(A)r_0$, in which P_k is the polynomial associated with BiCG and Q_k can be selected free under the condition that $Q_k(0) = 1$. In [86] it was suggested to construct Q_k as the product of k linear factors $1 - \omega_j A$, where ω_j was taken to minimize locally a residual. This approach leads to the BiCGSTAB method. Because of the local minimization, BiCGSTAB displays a much smoother convergence behavior than CGS, and more surprisingly it often also converges (slightly) faster. One weak point in BiCGSTAB is that we get break-down if an ω_j is equal to zero. One may equally expect negative effects when ω_j is small. In fact, BiCGSTAB can be viewed as the combined effect of BiCG and GCR(1), or GMRES(1), steps. As soon as the GCR(1) part of the algorithm (nearly) stagnates, then the BiCG part in the next iteration step cannot (or only poorly) be constructed.

Another dubious aspect of BiCGSTAB 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 GCR(1) stagnates forever, whereas a method like GCR(2) (or 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 [38] for the construction of the BiCGSTAB2 method. In the odd-numbered iteration steps the Q-polynomial is expanded by a linear factor, as in BiCGSTAB, 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 [64]).
However, our presentation suggests a possible weakness in the construction of BiCGSTAB2, namely in the odd-numbered steps the same problems may occur as in BiCGSTAB. Since the even-numbered steps rely on the results of the odd-numbered steps, this may equally lead to unnecessary break-downs or poor convergence. In [78] another, and even simpler approach was taken to arrive at the desired even-numbered steps, without the necessity of the construction of the intermediate BiCGSTAB-type step in the odd-numbered steps. Hence, in this approach the polynomial Q is constructed straight-away as a product of quadratic factors, without ever constructing a 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 is shown in [78] 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 BiCG 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 BICG 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 [77], 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 BiCGSTAB 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.

Bi-CGSTAB(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 BiCG 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 BiCG 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 more general BiCGSTAB(ℓ) schemes see [78, 77].
Another advantage of BiCGSTAB(2) over BiCGSTAB2 is in its efficiency. The BiCGSTAB(2) algorithm requires 14 vector updates, 9 innerproducts and 4 matrix vector products per full cycle. This has to be compared with a combined odd-numbered and even-numbered step in BiCGSTAB2, which requires 22 vector updates, 11 innerproducts, and 4 matrix vector products, and with two steps of BiCGSTAB which require 4 matrix vector products, 8

innerproducts and 12 vector updates. The numbers for BiCGSTAB2 are based on an implementation described in [64].
Also with respect to memory requirements, BiCGSTAB(2) takes an intermediate position: it requires 2 n-vectors more than BiCGSTAB and 2 n-vectors less than BiCGSTAB2.

For distributed memory machines the innerproducts may cause communication overhead problems (see, e.g., [16]). We note that the BiCG steps are very similar to conjugate gradient iteration steps, so that we may consider all kind of tricks that have been suggested to reduce the number of synchronization points caused by the 4 innerproducts in the BiCG parts. For an overview of these approaches see [6]. If on a specific computer it is possible to overlap communication with communication, then the BiCG parts can be rescheduled as to create overlap possibillities: 1. the computation of ρ_1 in the even BiCG 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 BiCG step.
3. The computation of ρ_1 for the odd BiCG step can be done just before the update for x at the end of the even BiCG step.
4. The computation of γ in the odd BiCG 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 the method BiCGSTAB(2) a (slight) advantage over Bi-CGSTAB. Furthermore we note that the updates in the GCR(2) may lead to more efficient code than for BiCGSTAB, since some of them can be combined.

Our next numerical example illustrates quite nicely the difference in convergence behavior of some of the methods that we have discussed.

Example. We consider an advection dominated 2nd order PDE, with Dirichlet boundary conditions, on the unit cube (this equation was taken from [50]):

$$(6.5\text{i}) \qquad -u_{xx} - u_{yy} - u_{zz} + 1000\, u_x = f.$$

The function f is defined by the solution

$$u(x, y, z) = xyz(1-x)(1-y)(1-z).$$

This equation was discretized using $22 \times 22 \times 22$ volumes, resulting in a seven-diagonal linear system of order 10648. In order to make differences between iterative methods more visible, we have here and in our other examples not use any form of preconditioning.

In Figure 4 we see a plot of the convergence history. Bi-CGSTAB almost stagnates, as might be anticipated from the fact that this linear system has eigenvalues with relatively large imaginary parts. Surprisingly, Bi-CGSTAB does even worse than Bi-CG. For

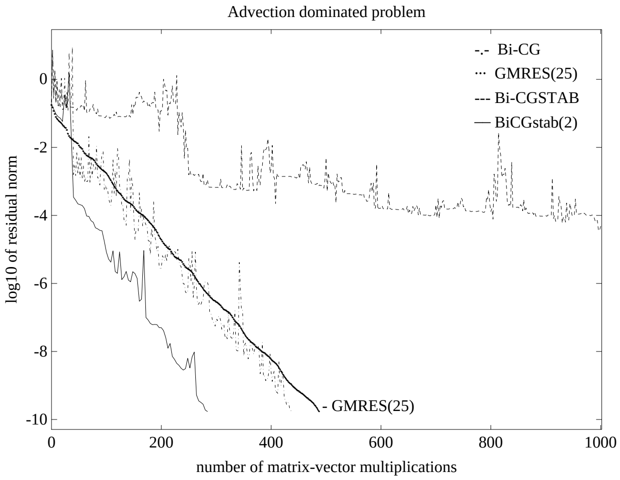

Fig.4: Convergence plot.

this type of matrices this behavior of Bi-CGSTAB is not uncommon and, as we will see in the next subsection, this can be explained by the poor recovery of the Bi-CG iteration coefficients α_k and β_k. Bi-CGstab(2) converges quite nicely and almost twice as fast as Bi-CG. GMRES(25) is about as fast as Bi--CG. Since the GMRES steps are much more expensive, BiCGstab(2) is the most efficient method here.

6.6 Maintaining Convergence:

The BiCGstab methods are designed for smooth convergence, with the purpose to avoid loss of local bi-orthogonality in the underlying Bi-CG process. This is important, since then the convergence of the Bi--CG part is exploited as much as possible. However, local bi-orthogonality may also be disturbed by, for instance, inaccuracies in the Bi-CG coefficients α and β. They are the quotients of scalars $\rho \equiv (r_i, \widehat{r}_0)$ and $\gamma \equiv (Ap, \widehat{r}_0)$ (see the algorithms for BiCGSTAB and BiCGSTAB(2)) and they will be inaccurate if ρ or γ is relatively small (see (6.6b)). The question is, when does this occur and how can it be avoided? Here, we will concentrate on ρ only, but similar arguments apply to γ as well.

As in the introduction of this section, r_i is the residual $r_i = \widetilde{P}_i(A)P_i(A)r_0$ where $\widetilde{P}_i$ is an appropriate polynomial of degree i with $\widetilde{P}_i(0) = 1$. Now, ρ is given by

$$(6.6\text{a}) \qquad \rho \equiv \rho_i \equiv (\widetilde{P}_i(A)P_i(A)r_0, \widehat{r}_0).$$

The scalar ρ_i can be small if the underlying Bi-Lanczos process nearly breaks down (i.e. $(\widetilde{P}_i(A)P_i(A)r_0, \widehat{r}_0) \approx 0$ relatively, for any polynomial $\widetilde{P}_i$ of degree i). Also an 'unlucky' choice of $\widetilde{P}_i$ may lead to a small ρ_i (which occurs in Bi-CGSTAB if the GCR(1) part stagnates). Here, we will concentrate on typical Bi-CGSTAB situations. Therefore, we assume that the Bi-Lanczos process itself (and the LU decomposition) does not (nearly) break down.

The relative rounding error ϵ_i in ρ_i can relatively and sharply be bounded by

$$(6.6b) \qquad |\epsilon_i| \leq n\,\overline{\xi}\,\frac{(|r_i|, |\widehat{r}_0|)}{|(r_i, \widehat{r}_0)|} \leq n\,\overline{\xi}\,\frac{\|r_i\|\,\|\widehat{r}_0\|}{|(r_i, \widehat{r}_0)|}.$$

For a small relative error we want to have the expression at the right-hand side as small as possible.

Since the Bi-CG residual $P_i(A)r_0$, here to be denoted by s_i, is orthogonal to $\mathcal{K}_i(A^T; \widehat{r}_0)$ it follows that

$$(r_i, \widehat{r}_0) = \theta_i (A^i s_i, \widehat{r}_0)$$

if

$$\widetilde{P}_i(A) = \theta_i A^i + \theta^{(i)}_{i-1} A^{i-1} + \ldots.$$

Therefore, since $\|\widehat{r}_0\|/|(A^i s_i, \widehat{r}_0)|$ does not depend on $\widetilde{P}_i$, minimizing the right-hand side of (6.6b) is equivalent to minimizing

$$(6.6c) \qquad \frac{\|\widetilde{P}_i(A)s_i\|}{|\theta_i|}$$

with respect to all polynomials $\widetilde{P}_i$ of exact degree i with $\widetilde{P}_i(0) = 1$. This minimization problem is solved by the FOM polynomial P_i^{F}, here associated with the initial residual s_i: P_i^{F} is the i^{th} degree polynomial for which $r_i^{\mathrm{F}} = P_i^{\mathrm{F}}(A)s_i$ (cf. Section 6.2). This polynomial is characterized by:

$$P_i^{\mathrm{F}}(A)s_i \perp \mathcal{K}_i(A; s_i) \text{ and } P_i^{\mathrm{F}}(0) = 1.$$

For optimally accurate coefficients, we should select FOM polynomials for our polynomials $\widetilde{P}_i$. However, since the hybrid Bi-CG methods are designed to avoid all the work for the construction of an orthogonal basis, the selection of complete FOM polynomials is out of the question.

For efficiency reasons, we have used products of first degree polynomials in Bi-CGSTAB and products of degree ℓ polynomials in BiCGstab(ℓ). Of course, our arguments can also be applied to such low degree factors. Therefore, suppose that $s = Q_{i-\ell}(A)P_i(A)r_0$ (as BiCGstab(ℓ)) has been computed and that the vectors s, As, $\ldots$, $A^\ell s$ are available. The suggestion for BiCGstab(ℓ) to minimize the residual over this particular Krylov subspace is equivalent to selecting a polynomial factor q_i ($Q_i = q_i Q_{i-\ell}$) of exact degree ℓ with $q_i(0) = 1$ such that

$$(6.6d) \qquad \|q_i(A)s\|$$

is minimal, while in this situation, for optimal accurate coefficients, we rather would like to minimize

$$(6.6e) \qquad \pmb{|}\, q_i(A)s \,\pmb{|}$$

where, with θ_i such that $q_i(A) = \theta_i A^\ell + \ldots$,

$$(6.6f) \qquad \pmb{|}\, q_i(A)s \,\pmb{|} \equiv \frac{\|q_i(A)s\|}{|\theta_i|}.$$

The GMRES polynomial q_i^{G} of degree ℓ solves (6.6d), the FOM polynomial q_i^{F} solves (6.6e). For small residuals, the FOM polynomial is not optimal:

$$\|q_i^{\mathrm{G}}(A)s\| = |c_i|\,\|q_i^{\mathrm{F}}(A)s\|$$

with c_i as in (6.2d). Similarly, for accurate coefficients, the GMRES polynomial is not optimal [75]:

$$\pmb{|}\, q_i^{\mathrm{F}}(A)s \,\pmb{|} = |c_i|\, \pmb{|}\, q_i^{\mathrm{G}}(A)s \,\pmb{|}$$

with the same scalar c_i. For degree 1 factors, as in Bi-CGSTAB, (assuming no preconditioning)

$$(6.6g) \qquad c_i = \frac{(s, As)}{\|s\|\,\|As\|},$$

and c_i is the cosine of the angle between s and As (in the BiCGSTAB algorithm, t represents As).

Clearly, for extremely small $|c_i|$, say $|c_i| \leq \sqrt{\xi}$ (in the $\ell = 1$ case, this means that s and As are almost orthogonal), taking GMRES polynomials for the degree ℓ factors will lead to inaccurate coefficients ρ_i, α and β, while FOM polynomials on the other hand will lead to large residuals. In both situations, the speed of convergence will seriously be deteriorated. The same phenomena can be observed when in a consecutive number of sweeps $|c_i|$ is small, but not necessarily extremely small (say, it takes k sweeps before $|c_{i-k}c_{i-k+1}\cdots c_i| \leq \sqrt{\xi}$). In other words, the inaccuracies seem to accumulate. This seems to occur quite often in practise. E.g., for linear equation stemming from PDEs with large advection terms, Bi-CGSTAB often stagnates, although all c_i may be larger than, say .1, and none of the ω_i can considered to be relatively small ($\omega_i = c_i\|s\|/\|As\|$).

Both Bi-CGSTAB and BiCGstab(ℓ) are built on top of the same Bi-CG process. At roughly the same computational costs, one sweep of BiCGstab(ℓ) covers the same Bi-CG track as ℓ sweeps of Bi-CGSTAB. In one sweep of BiCGstab(ℓ), GMRES(ℓ) is applied once, in ℓ sweeps of BiCGSTAB, GMRES(1) is applied ℓ times. For two reasons it pays off to use GMRES(ℓ) instead of $\ell\times$GMRES(1):

1. Due the super-linear convergence, one sweep of GMRES(ℓ) may be expected to give a better residual reduction than ℓ times GMRES(1).
2. In ℓ steps of GMRES(1), ℓ small c_i's may contribute to inaccuracies in the coefficients α and β, where GMRES(ℓ) contributes ato this only once.

BiCGstab(ℓ) profits from GMRES(ℓ) by a better residual reduction in the GMRES part and by the faster convergence of a better recovered Bi-CG due to the more stable computations. However, we do not recommend to take ℓ large; $\ell = 2$ or $\ell = 4$ will usually lead already to almost optimal speed of convergence. The computational costs increase slightly

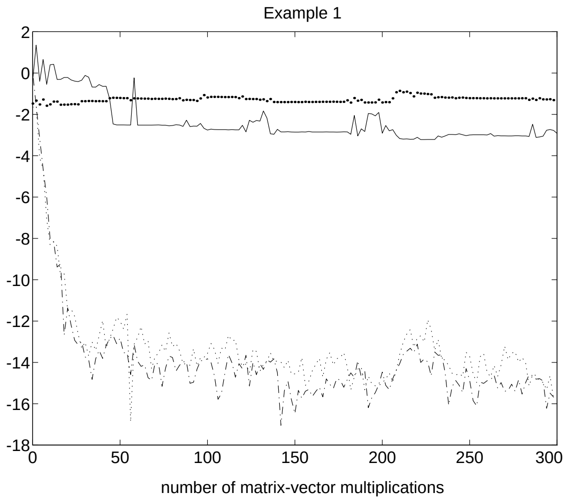

Fig.5: Convergence Bi-CGSTAB.

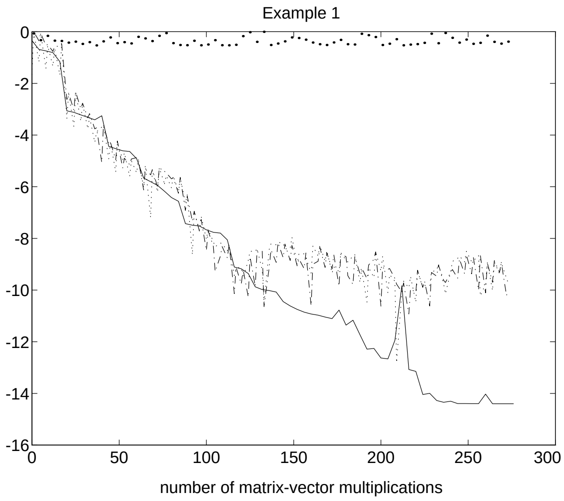

Fig.7: Convergence BiCGstab(2).

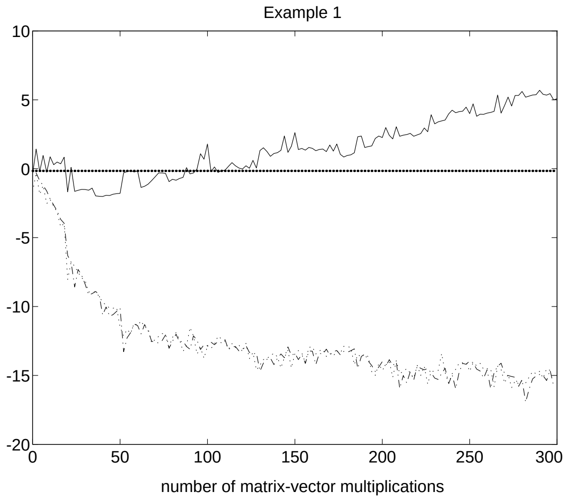

Fig.6: Convergence stabilized Bi-CGSTAB.

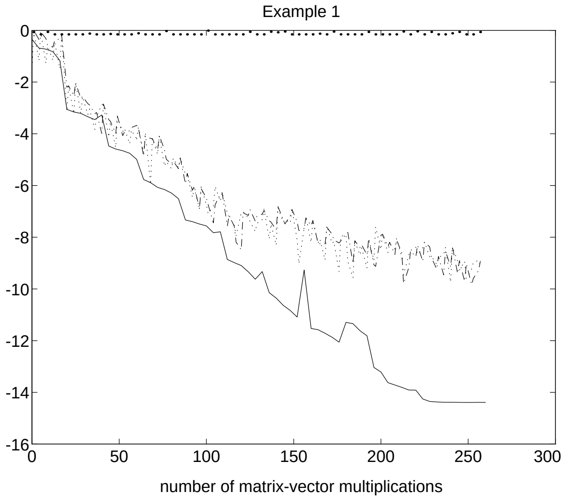

Fig.8: Convergence stab. BiCGstab(2).

by increasing ℓ (i.e. $2\ell+10$ vector updates and $\ell+7$ inner products per 4 matrix multiplications), and more vectors have to be stored ($2\ell+5$ vectors). Moreover, the method is less accurate for larger ℓ due to the fact that intermediate residuals (as r and $r-\omega_1 s$ in the Bi-CGSTAB(2) algorithm) can be large, with similar negative effects as in Section 6.5.4.

For Bi-CGSTAB there is a simple strategy that relaxes the danger of error amplification in consecutive sweeps with small $|c_i|$: replace in the Bi-CGSTAB algorithm the line

'$\omega = (s,t)/(t,t)$'

by the piece of code in Algorithm 1. In this way we limit the size of $|c|$. The constant .7 is rather arbitrarily and may be replaced by any other fixed non-small constant less than 1. Since GMRES(1) reduces well only if $|c_i| \approx 1$ (see (6.2c)), this strategy still profits from a possible good reduction by GMRES(1). A similar strategy that is equally cheap and easy to implement can be applied to BiCGstab(ℓ); see [75] for details.

We give a few numerical examples that demonstrate the cumulative effects of small $|c|$'s and that illustrate the effects of limiting its sizes.

Examples. The figures for the examples display, all on log-scale, the values for each iteration step of
– the residual-norms $\|r\|$, by solid curves (——);
– the scaled ρ, $\widehat{\rho} \equiv |(r,\widehat{r}_0)|/(\|r\|\,\|\widehat{r}_0\|)$ (cf. (6.6b)), by dashed-dotted curves (- · - ·);
– the scaled γ: $\widehat{\gamma} \equiv |(Ap,\widehat{r}_0)|/(\|Ap\|\,\|\widehat{r}_0\|)$, by dotted curves (·····);
– $|c|$, resp. $\max(|c|, 0.7)$, by bullets (●●●).

Before describing the examples, we will discuss part of the results.
In the figures 5–16, we see that the scaled ρ and the scaled γ behave similarly (the dashed-dotted – and dotted curves coincide more or less). Further, none of the $|c|$ is extremely small even not in cases where the $\widehat{\rho}$ and $\widehat{\gamma}$ are. The decrease of $\widehat{\rho}$ for values of $\widehat{\rho}$ not in the range of the machine precision ($\geq 10^{-12}$)

$$
\begin{aligned}
&\quad\vdots\\
&c = (s,t)/(\|s\|\,\|t\|);\\
&\omega = \mathrm{sign}(c)\,\max(|c|, 0.7)\,\|s\|/\|t\|;\\
&\quad\vdots
\end{aligned}
$$

Alg.1: Limiting the size of $|c|$.

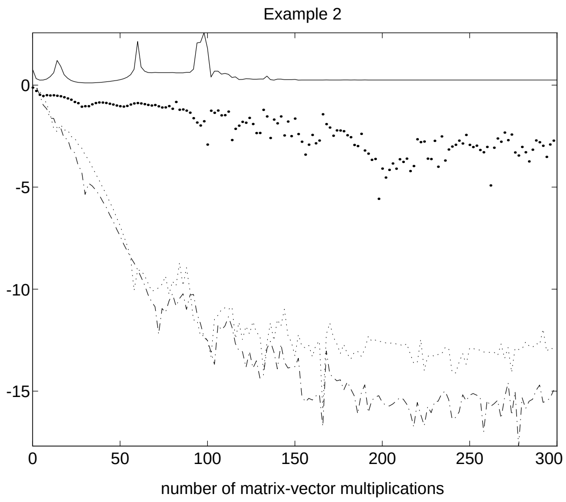

Fig.9: Convergence Bi-CGSTAB.

seems to be proportional to the product of previous $|c|$'s. In all the examples, the method stagnates if $\widehat{\rho}$'s or $\widehat{\gamma}$'s become extremely small, say less than 10^{-12}. In these cases, almost all significance of the Bi-CG coefficients α and β will be lost. Limiting the size of $|c|$ (Algorithm 1) slows down the decrease of $\widehat{\rho}$ and $\widehat{\gamma}$. In the caption of the figures, we used the adjective 'stabilized' to indicate that we used the limiting strategy. Often 'stabilizing' is enough to overcome the stagnation phase, and to lead to a converging process.

Example 1 (Figures 5–8). BiCGstab(2) converges. Although stabilizing Bi-CGSTAB leads to more accurate Bi-CG coefficients in the initial phase of the process, this is apparently not enough to restore full convergence.

Example 2 (Figures 9–12). Increasing ℓ to $\ell = 2$ leads to a slowly converging BiCGstab(2) process (many more than 300 matrix vector multiplications are needed; not shown in the graph). Our simple stabilizing strategy works well here.

Example 3 (Figures 13–16). The combined improvements, stabilizing and increasing ℓ to $\ell = 2$, are necessary for convergence.

For the first example, we have taken the PDE of (6.5i). The righ-hand side f is defined by the solution

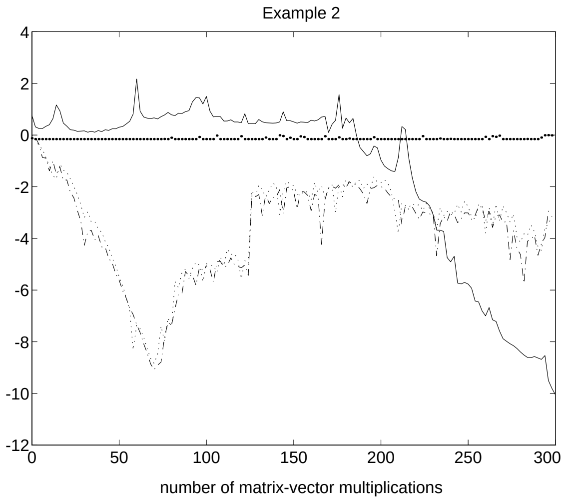

Fig.10: Convergence stabilized Bi-CGSTAB.

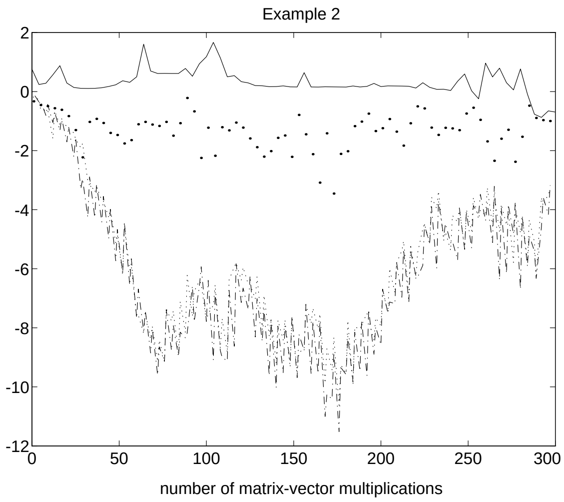

Fig.11: Convergence BiCGstab(2).

$$u(x,y,z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z).$$

The discretization is with $10 \times 10 \times 10$ finite volumes (no preconditioner has been used).

In the second and third example [33, 70], we have discretized

$$-u_{xx} - u_{yy} + a(xu_x + yu_y) + bu = f$$

on the unit-square with Dirichlet boundary conditions, with 63×63 finite volumes, taking $a = 100$ and $b = -200$, respectively 66×66, $a = 1000$ and $b = 10$ (no preconditioner has been used). The function f is such that the discrete solution is constant 1 (on the grid).

6.7 Generalized CGS:

We have now discussed in some detail the family of BiCGstab(ℓ) methods, but one should not deduce from this that these methods are to be preferred over CGS in all circumstances. We have had very good experiences with CGS in the context of solving nonlinear problems with Newton's method. It turns out

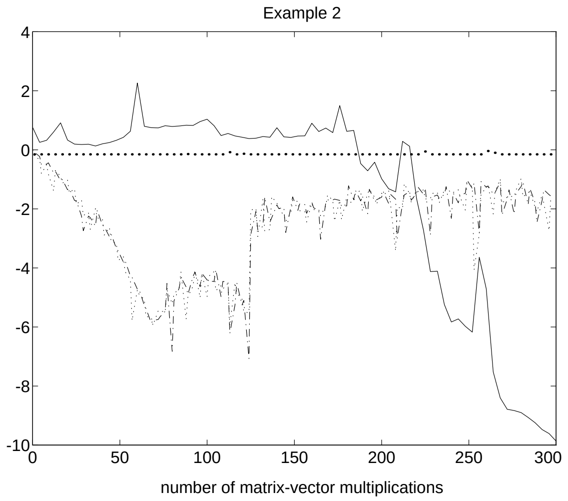

Fig.12: Convergence stab. BiCGstab(2).

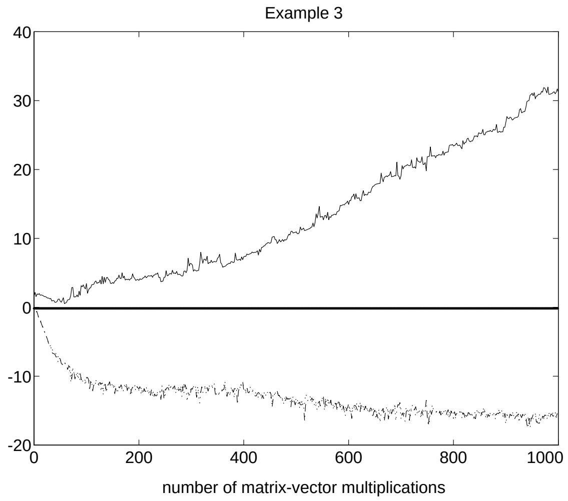

Fig.14: Convergence stabilized Bi-CGSTAB.

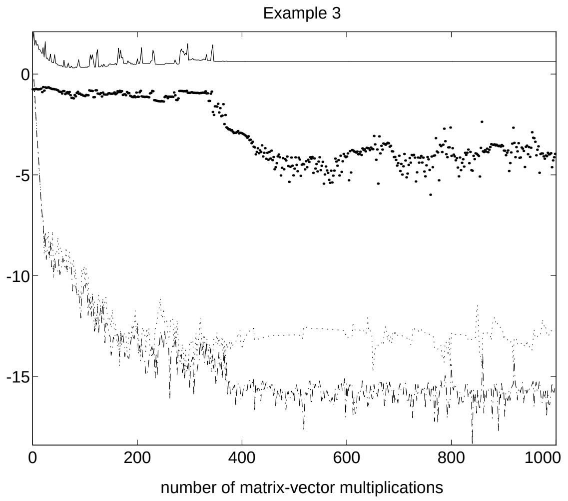

Fig.13: Convergence Bi-CGSTAB.

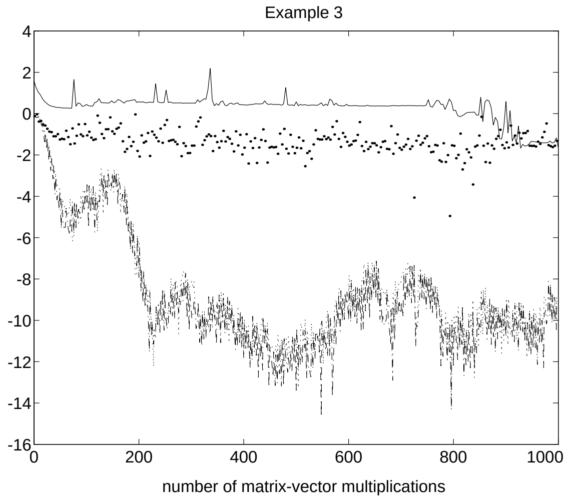

Fig.15: Convergence BiCGstab(2).

that we can exploit some of the presented ideas also to improve on CGS itself.

In the Newton method one has to solve a Jacobian system for the correction. This can be done by any method of choice, e.g., CGS or BiCGstab(ℓ). Often fewer Newton steps are required to solve a non-linear problem accurately when using CGS. Although the BiCGstab methods tend to solve each of the linear systems (defined by the Jacobi matrices) faster, the computational gain in these inner loops does not always compensate for the loss in the outer loop because of more Newton steps.

This phenomenon can be understood as follows. For eigenvalues λ that are extremal in the convex hull of the set of all eigenvalues of A (the Jacobian matrix), the values $P_i(\lambda)$ of the Bi-CG polynomials P_i tend to converge more rapidly towards zero than for eigenvalues λ in the interior. Since CGS squares the Bi-CG polynomials, CGS may be expected to reduce extremely well the components of the initial residual r_0 in the direction of the eigenvectors associated with extremal eigenvalues λ: with reduction factor $P_i(\lambda)^2$. Of course, the value $P_i(\lambda)$ can also be large, specifically for interior eigenvalues and in an initial stage of the process. CGS amplifies the associated components, (which also explains the typical irregular convergence behavior of CGS). The BiCGstab polynomial Q_i does not have this tendency of favoring the extremal eigenvalues. Therefore, the BiCGstab methods tend to reduce all eigenvector components equally well: on average, the "interior components" of a BiCGstab residual r_i are smaller than the corresponding components of a CGS residual $\widetilde{r}_i$, while, with respect to the exterior components the situation is the other way around. However, the non-linearity of a non-linear problem seems often to be represented rather well by the space spanned by the "extremal eigenvectors". With respect to this space, and hence with respect to the complete space, Newtons scheme with CGS behaves like an exact Newton scheme.

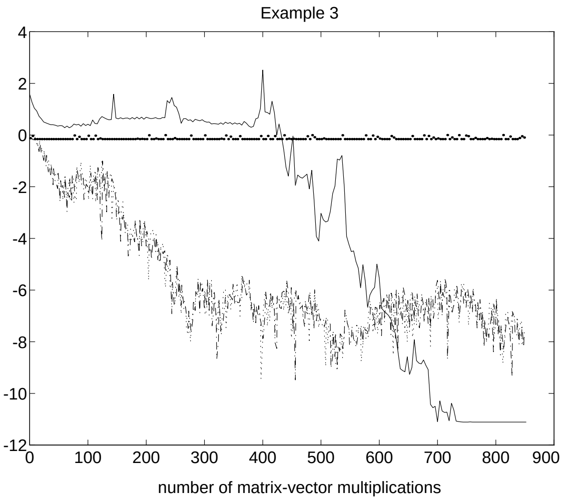

Fig.16: Convergence stab. BiCGstab(2).

We would like to preserve this property when constructing iterative schemes for Newton iterations. Fokkema et al [29] suggest polynomials $\widetilde{P}_i$ that lead to efficient algorithms (small modifications of the CGS algorithm) with a convergence that is slightly smoother, faster, and more accurate, than for CGS, but that still has the property of reducing extremal components quadratically. As a linear solver for isolated linear problems these "generalized CGS" schemes do not seem to have much advantage over BiCGstab(ℓ), but as a linear solver in a Newton scheme for non-linear problems, they often do rather well.

7 RELIABLE UPDATING

In all the Bi-CG related methods we see that the approximation for x and the residual vector r are updated by different vectors, and that the value for x does not influence the further iteration process, whereas the value for r does. In exact arithmetic the updated r is equal to the true residual $b - Ax$, but in rounded arithmetic it is unavoidable that differences between r and $b - Ax$ arise. This means that we may be misled for our stopping criteria, which are usually based upon knowledge of the updated r (and that we may have iterated too far in vain).

In this section we will discuss some techniques that have been proposed recently for the improvement of the updating steps. It turns out that this can be settled by relatively easy means.

Although the techniques in the previous section led to smoother and faster convergence and more accurate approximations the approximation may still not as accurate as possible. Here, we strive for optimal accuracy, i.e the updated r_i should be very close to the values of $b - Ax_i$, while leaving the convergence of the updated r intact.

First, we observe that even if x_m is the exact solution then the residual, computed in rounded arithmetic as $b - Ax_m$, may not be expected to be zero: using the notation of Section 6.5.4,

$$\begin{aligned} \|b - Ax_m\| &\leq \overline{\xi}\,(\|b\| + n_A\,\||A|\|\,\|x_m\|) \\ \text{(7.1a)} \qquad &\leq 2\,\Gamma\,\overline{\xi}\,\|b\|. \end{aligned}$$

Therefore, the best we can strive for is an approximation x_m for which the true residual and the updated one differ in order of magnitude by the initial residual times the relative machine precision ($\mathcal{O}(\overline{\xi}\,\|r_0\|)$; recall that we assumed $x_0 = 0$, and hence $r_0 = b$).

Now it becomes also obvious why it is a bad idea to replace the updated residual in each step by the true one. Except from the fact that this would cost an additional matrix vector multiplication in each step, it also introduces errors in the recursions for the residuals. Although these errors may be expected to be small relatively to r_0, they will be large relatively to r_i if $\|r_i\| \ll \|r_0\|$. This perturbs the local bi-orthogonality of the underlying Bi-CG process and it may significantly slow down the speed of convergence. This observation suggests to replace the updated residual by the true one only if the updated residual has the same order of magnitude as the initial residual. However, meanwhile x_i and r_i may have drifted apart, and replacing r_i by $b - Ax_i$ brings in the "error of x_i" in the recursion (bounded as in (6.5g)), and again the speed of convergence may be affected. Although it is a good idea to use true residuals at strategic places, the approximation x_i should first be 'tied' more closely to the updated residual r_i. We can achieve this by updating x_i cumulatively: if $x_i = x_0 + w_1 + \ldots + w_i$ (cf. (6.5d)) then we actually compute x_i in groups as

$$\text{(7.1b)} \qquad x_i = x_0 + x'_1 + x'_2 + \ldots$$

where, for some decreasing sequence of indices $\pi(1) = 1$, $\pi(2)$, ..., x'_j represents the sum of a group;

$$x'_j = w_{\pi(j)} + w_{\pi(j)+1} + \ldots + w_{\pi(j+1)-1}, \text{ etc..}$$

Simultaneously, we compute r_i as

$$\text{(7.1c)} \qquad r_i = r_0 - Ax'_1 - Ax'_2 - \ldots.$$

In this way we can control the size of the updates for x_i and r_i, and we avoid large errors (cf. (6.5g)): for a proper choice of the $\pi(j)$, the x'_j will be small even if some of the w_j are large.

In the modification of the algorithms that we will propose in Algorithm 2, we kept in mind that we only may allow errors which
(a) are small with respect to the initial residual r_0 (otherwise accuracy will be disturbed) and
(b) are small with respect to the present updated residual r_i (otherwise local bi-orthogonality may be jeopardized).

In Section 3, we have explained that it is no restriction to take $x_0 = 0$, arguing that this situation can be forced simply by a shift: shift x by x_0, and b by Ax_0. This shift can be made explicit in the hybrid Bi-CG algorithms by making three changes:
(i) adding as a last line to the initialization phase

$$x = x_0; \quad x' = 0; \quad b' = r_0;$$

(ii) adding as a last line in the algorithms (just after 'end')

$$x = x + x';$$

(iii) replacing all x_i (and x) by x' (skipping the index i).
Even in rounded arithmetic, this modification will not change the value of any of the vectors and scalars in the computational scheme, except for the x's. Since $x + x'$ is the approximation that we are interested in, one also may want to change the termination criterion. We propose to replace the line

if x is accurate enough then quit;

by

if $\|r_{i+1}\|$ is small enough then quit;

To allow for a more accurate way of updating of the residual and the approximation, we suggest to add another few lines just before 'end' in the algorithm, as is shown in Algorithm 2. We suggest to replace the

```
    ⋮
x = x_0;   x' = 0;   b' = r_0;
for i = 0, 1, 2, ...
        ⋮
        ⋮  Replace all x_i and x by x'.
        ⋮
    if r_{i+1} is small enough then quit;
    set 'compute_res' and 'update_app';
    if 'compute_res' is true
        r_{i+1} = b' - Ax';
        if 'update_app' is true
            x = x + x'; x' = 0; b' = r_{i+1};
        endif
    endif
endfor
x = x + x';
```

Alg.2: For accurate approximations.

updated residual by the true one on strategically chosen steps (we have to explain when the value of the boolean functions *'compute_res'* is true). However, we also suggest to shift the problem once in a while (when the boolean function *'update_app'* is true) in order to let the right-hand decrease (cf. (7.1a)). Here we use the fact that, in exact arithmetic, also these intermediate shifts do not change the iteration parameters and vectors (except for the vectors x). Observe that the updated residual r_{i+1} is replaced by the true residual $b' - Ax'$ of the shifted problem if *'compute_res'* is true.

For this we propose the following strategy.
Update x and b' only if the residual is significantly smaller than the initial residual, while an intermediate residual was larger (cf. (7.1b), (7.1c) and reminder (a)):

$$\text{(7.1d)} \quad \begin{array}{l} \textit{'update\_app'} = \text{true} \\ \text{if} \quad \|r_{i+1}\| \le \|b\|/100 \;\;\&\;\; \|b\| \le \mu \\ \text{else } \textit{'update\_app'} = \text{ false}, \end{array}$$

where $\mu \equiv \max \|r_i\|$ and the maximum is taken over all residuals since the previous update of x and b' (since the previous *'update_app'* is true).
The bound in (7.1a) suggests that the norm $\|b\|$ of the initial residual should be used as criterion for shifting the problem (*'update_app'* is true if $\|r_{i+1}\| \le \|b\|$ & $\|r_i\| \ge \|b\|$). However, if the process converges irregularly this would lead to many shifts. The relaxed version in (7.1d) turns out to work equally well at less costs.
Compute a true residual whenever *'compute_res'* is true and if a previous residual is larger than the initial residual and significantly larger than the present updated residual:

$$\text{(7.1e)} \quad \begin{array}{l} \textit{'compute\_res'} = \text{true} \\ \text{if} \quad \|r_{i+1}\| \le M/100 \;\;\&\;\; \|b\| \le M \\ \qquad \text{or } \textit{'update\_app'} \text{ is true} \\ \text{else } \textit{'compute\_res'} = \text{false}, \end{array}$$

where $M \equiv \max \|r_i\|$ and the maximum is taken over all residuals since the last computation of the true residual.
Replacing the updated residual by the true one perturbs the recursion for the residuals. If the residual decreases too much since the previous replacement, the perturbation may become large relatively to the present residual (reminder (b)). Therefore, *'compute_res'* may be true more often than *'update_app'*.

We suggest to add the above strategy to an existing code. That means that an additional matrix-vector multiplication has to be performed whenever a true residual has to be computed. The conditions (7.1d) and (7.1e) are chosen as to minimize the number of these additional computations. One also may try to skip a matrix-vector multiplication in one of the preceding lines of the algorithm, which requires some additional care for BiCGstab(ℓ), but which easily can be accomplished for CGS.

If CGS is modified as suggested, then the new lines do not require additional matrix vector multiplications, and there is no need to restrict the number of

computations of true local residuals. For this CG
variant, Neumaier [56] suggested places where the
and b' can be updated for accurate approximations
update x and b' whenever the residual decreases with
respect to the previous 'best residual',

$$\text{(7.1f)}\qquad \begin{array}{l} \textit{`update\_app'} = \mathsf{true} \\ \mathsf{if} \quad \|r_{i+1}\| \leq \|b'\| \\ \mathsf{else}\ \textit{`compute\_res'} = \mathsf{false}. \end{array}$$

The modifications according to Neumaier's approac
are given in Algorithm 3. Observe that the norm o

$\vdots$
$x = x_0;\ x' = 0;\ b' = r_0;\ \mu' = \|b'\|;$
for $i = 0, 1, 2, \ldots$
  $\vdots$ *Replace all x_i and x by x'.*
  $\vdots$
Skip the CGS update for r
together with the MV involved
in this update. Compute instead
  $r_{i+1} = b' - Ax';\ \mu = \|r_{i+1}\|;$
  if μ is small enough then quit;
  if $\mu \leq \mu'$
    $x = x + x';\ x' = 0;$
    $b' = r_{i+1};\quad \mu' = \mu;$
  endif
endfor
$x = x + x';$

Alg.3: Neumaier's strategy for CGS.

the b' (the residuals with respect to the x) strictly decrease: the Neumaier trick also smoothes convergence (without improving its speed!).

Below, we discuss the effects of our strategies in practise. We illustrate our observations by a simple numerical example.

Example. Figure 17 shows the convergence history of the *true residuals* as produced by standard CGS, and by the modified versions of CGS as suggested above, applied to the SHERMAN4 matrix of the Harwell-Boeing collection (as in the example of Section 6.5.4). The dotted curve ($\cdots$) represents the results for standard CGS. We also applied modified CGS as in Algorithm 2, using the update criterions (7.1d) and (7.1e). The solid curve (—–) represents the results for this *simple strategy*, while the dashed-dotted curve (- $\cdot$ - $\cdot$) represents the results for Neumaier's strategy in Algorithm 3. On log-scale, the norm of the true residuals $\|b - Ax_i\|$, $\|b - A(x + x')\|$, respectively, is plotted against the number of matrix-vector multiplications. Neumaier's strategy as well our's lead to approximations that are accurate (cf. (7.1a)): comparing $\|r_0\|$

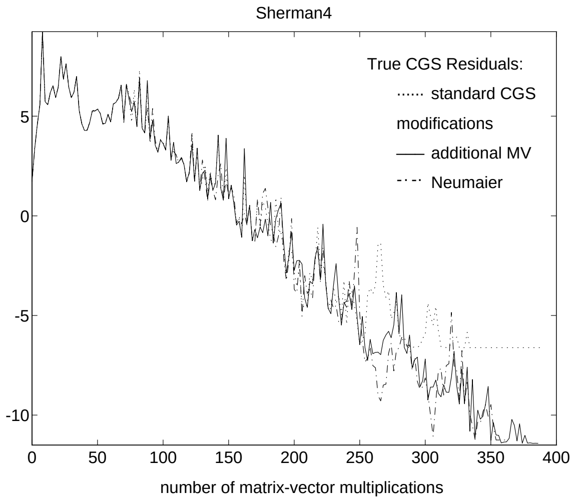

Fig.17: Reliable updates.

with the norm of the smallest true residual, we see that a reduction is obtained by a factor $\approx 10^{-14}$ ($\overline{\xi} = 2.2\,10^{-16}$). Standard CGS does not produce true residuals smaller than $\approx 10^{-9}\|r_0\|$, which is approximately $\overline{\xi} \cdot \max \|r_i\| \approx 2.2\,10^{-16} \cdot 10^{7}\|r_0\|$; cf. (6.5g). Observe that, though the convergence histories do not coincide for residuals less than $\approx 10^{1}$, the speed of convergence is not affected: the modified versions exhibit a rate of convergence that is very similar to the one of the updated residuals in standard CGS as shown in Figure 3.

Experiments for other examples and with other iterative schemes, as Bi-CGSTAB and BiCGstab(ℓ), led to similar conclusions. Although, two observations should be made.

— Quite often the improvements are much more spectacular than for this SHERMAN4 example: CGS may produce intermediate residuals as large as $\|r_0\|/\overline{\xi}$ and none of the digits in the finial approximation of standard CGS will be correct.

— There are some differences between CGS and the BiCGstab methods: (i) as observed above, Neumaier's strategy only works well for CGS, while the simple strategy of Algorithm 2 can always be applied. (ii) Especially for the BiCGstab methods, the simple strategy of Algorithm 2 with update criterions (7.1d) and (7.1e) does not lead to much additional work. The additional computation of a true residual takes place after the process encounters residuals that are (much) larger than the initial residual. Since BiCGstab(ℓ) tends to show much smoother convergence behavior than CGS, for small ℓ, the additional work in these methods is usually much less than for CGS. In the SHERMAN4 example, our strategy for CGS requires 7 additional matrix-vector multiplications (*'compute_res'* is true 7 times) and one special update of the approximation (*'update_app'* is true only once). For BiCGstab(ℓ), $\ell \leq 6$, only 1 additional matrix-vector multiplication was needed. Neumaier's strategy for CGS does not require additional matrix-vector multiplications (but 364 additional updates for the approximation were needed).

8 Termination Criteria

An important point, when using iterative processes, is to decide when to terminate the process. Popular stopping criteria are based on the norm of the current residual, or on the norm of the update to the current approximation to the solution (or a combination of these norms). More sophisticated criteria have been discussed in litterature.
In [45] a practical termination criterion for the conjugate gradient method is considered. Suppose we want an approximation x_i for the solution x for which

$$\|x_i - x\|_2/\|x\|_2 \leq \varepsilon,$$

where ε is a tolerance set by the user.
It is shown in [45] that such an approximation is obtained by CG as soon as

$$\|r_i\|_2 \leq \mu_1\|x_i\|_2\varepsilon/(1+\varepsilon),$$

where μ_1 stands for the smallest eigenvalue of the positive definite symmetric (preconditioned) matrix A. Of course, in most applications the value for μ_1 will be unknown, but with the iteration coefficients of CG we can build the tridiagonal matrix T_i, and compute the smallest eigenvalue (Ritz value) $\mu_1^{(i)}$ of T_i, which is an approximation for μ_1. In [45] a simple algorithm for the computation of $\mu_1^{(i)}$, along with the CG algorithm, is described, and it is shown that a rather robust stopping criterion is formed by

$$\|r_i\|_2 \leq \mu_1^{(i)}\|x_i\|_2\varepsilon/(1+\varepsilon).$$

A similar criterion has also been suggested earlier in [40].

A quite different, but much more generally applicable approach has been suggested in [1]. In this approach the approximate solution of an iterative process is regarded as the exact solution of some (nearby) linear system, and computable bounds for the perturbations with respect to the given system are given. A nice overview of termination criteria has been presented in [6]: Section 4.2.

9 Implementation Aspects

For effective use of the given iteration schemes, it is necessary that they can be implemented such that high computing speeds are achievable. It is most likely that high computing speeds will be realized only by parallel architectures and therefore we must see how well iterative methods fit to such computers.

The iterative methods only need a handful of basic operations per iteration step

- Vector updates: in each iteration step the current approximation to the solution is updated by a correction vector. Often the corresponding residual vector is also obtained by a simple update, and we have update formulas as well for the correction vector (or search direction).

- Innerproducts: In many methods the speed of convergence is influenced by carefully constructed iteration coefficients. These coefficients are sometimes known analytically, but more often they are computed by innerproducts, involving residual vectors and search directions, as in the methods discussed in the previous sections.

- Matrix vector products: In each step at least one matrix vector product has to be computed with the matrix of the given linear system. Sometimes also matrix vector products with the transpose of the given matrix are required (e.g., BiCG). Note that it is not necessary to have the matrix explicitly, it suffices to be able to generate the result of the matrix vector product.

- Preconditioning: It is common practice to precondition the given linear system by some preconditioning operator. Again it is not necessary to have this operator in explicit form, it is enough to generate the result of the operator aplied to some given vector. The preconditioner is applied as often as the matrix vector multiply in each iteration step.

For problem sizes large enough the innerproducts, vectorupdates and matrix vector product are easily parallelized and vectorized. The more successful preconditionings, i.e, based upon incomplete LU decomposition, are not easily parallelizable. For that reason one is often satisfied with the use of only diagonal scaling as a preconditioner on highly parallel computers, such as the CM2 [7].

On distributed memory computers we need large grained parallelism in order to reduce synchronization overhead. This can be achieved by combining the work required for a successive number of iteration steps. The idea is to construct first in parallel a straight forward Krylov basis for the search subspace in which an update for the current solution will be determined. Once this basis has been computed, the vectors are orthogonalized, as is done in Krylov subspace methods. The construction as well as the orthogonalization can be done with large grained parallelism, and has sufficient degree of parallelism in it. This approach has been suggested for CG in [11] and for GMRES in [12], [5] and [18]. One of the disadvantages in this approach is that a straight forward basis, of the form $y, Ay, A^2y, ..., A^iy$ is usually very ill-conditioned. This is in sharp contrast to the optimal condition of the orthogonal basis set constructed by most of the projection type methods and it puts severe limits on the number of steps that can be combined. However, in [5] and [18] ways to improve the condition of a parallel generated basis are suggested and it seems possible to take larger numbers of steps, say 25, together. In [18] the effects of this approach on the communication overhead are studied and compared with experiments done on moderately massive parallel transputer systems.

9.1 Parallelism in the preconditioner:

In this section we consider a number of possibilities to obtain parallelism in the standard Incomplete Choleski preconditioner [51]. The linear systems are supposed to arise from standard finite difference discretisations of second order pde's over rectangular grids in two or three dimensional space.

9.1.1 Overlapping Local Preconditioners

Radicati di Brozolo and Robert [66] suggest to partition the given matrix A in (slightly) overlapping blocks along the main diagonal. Note that a given non-zero entry of A is not necessarily contained in one of these blocks. But experience suggests that this approach is more successful if these blocks cover all the non-zero entries of A. The idea is to compute in parallel local preconditioners for all of the blocks, e.g.,

$$A_n = L_n {D_n}^{-1} U_n - R_n. \tag{9.1a}$$

Then, when solving $Kw = r$ in the preconditioning step, we partition r in (overlapping) parts r_n, according to A_n, and we solve the systems $L_n D_n^{-1} U_n w_n = r_n$ in parallel. Finally we define the elements of w to be equal to corresponding elements of the w_n's in the nonoverlapping parts and to the average of them in the overlapped parts.

Radicati di Brozolo and Robert [66] report on timing results obtained on an IBM3090-600E/VF for GMRES preconditioned by overlapped incomplete LU decomposition for a 2D system of order 32400 with a bandwidth of 360. For p processors ($1 \leq p \leq 6$) they subdivide A in p overlapping parts, the overlap being so large that thses blocks cover all the nonzero entries of A. They found experimentally an overlap of about 360 elements to be optimal for their problem. This approach led to a speedup of roughly p. In some cases a speedup even slightly larger than p was observed, apparantly due to the fact that the parallel preconditioner was slightly more effective than the standard one in those cases.

9.1.2 Repeated Twisted Factorization

Meurant [54] reports on timing results obtained with a CRAY Y-MP/832, using an incomplete repeated twisted block factorization for 2D problems. In his experiments the L of the incomplete factorization has a block structure, i.e., L has alternatingly a block below the diagonal, one above, one below, and it ends with one above the diagonal. For this approach Meurant reports a speedup, for preconditioned CG, close to 6 on the 8-processor CRAY Y-MP. This speedup has been measured relative to the same repeated twisted factorization process executed on one single processor. Meurant also reports an increase in the number of iteration steps, due to this repeated twisting. This implies that the effective speedup with respect to the nonparallel code is only about 4.

9.1.3 Twisted and Nested Twisted Factorization

For 3D problems we have used the blockwise twisted approach [23] in the z- direction, i.e. the (x, y)-planes in the grid were treated in parallel from bottom and top inwards. Over each plane we used the diagonalwise ordering, in order to achieve high vector speeds on each processor.
On a dedicated CRAY X-MP/2 this led, for preconditioned CG, to a reduction by a factor of close to 2 in wall clock time with respect to the CPU time for the nonparallel code on one single processor. For the microtasked code the wall clock time on the 2-processor system was measured for a dedicated system, whereas for the nonparallel code the CPU time was measured on a moderately loaded system. In some situations the speedup was even slightly larger than 2, due to better convergence properties of the twisted incomplete preconditioner.
The effects of these and other orderings on the convergence of preconditioned methods and on the amount of parallelism have been studied in [21].

We can also apply the twisted incomplete factorization in a nested way [83]. For 3D problems this can be exploited by twisting also the blocks corresponding to (x, y) planes in the y-direction. Over the resulting blocks, corresponding to half (x, y) planes, we may apply diagonal ordering in order to fully vectorize the four parallel parts.
By this approach we have been able to reduce the wall clock time by a factor of 3.3, for preconditioned CG, on the 4-processor CONVEX C-240. In this case the total CPU time, used by all of the processors, is roughly equal to the CPU time required for single processor execution [85]. Other then for the experiments on the CRAY X-MP/2, as reported before, we have relied on the autotasking capabilities of the Fortran compiler for the C-240, for all of the code, except for the preconditioning part. Since some statements in the code lead to rather short vector lengths, this may explain partially why the factor 3.3 for the CONVEX C-240 stays well behind the theoretically

expected factor of about 3.9. Another reason might be that we were not completely sure whether our testing machine was executing constantly in stand alone mode during the time of our timing experiments. Even the system itself needs some CPU-time from time to time.

9.1.4 Hyperplane Ordering

For a CYBER 205 it has been reported how to obtain long vectorlengths for certain 3D situations ([23], [73]), and, of course, this approach can also be followed in order to obtain parallelism. This has been done by Berryman et.al. [7] for parallelizing standard ICCG on a Connection Machine CM-2. For a 4K processor machine they report a computational speed of 52.6 Mflops for the (sparse) matrix vector product, while 13.1 Mflops has been realized for the preconditioner, using the hyperplane approach.
This reduction in speed by a factor of 4 makes it attractive to use only diagonal scaling as a preconditioner in some situations, for massively parallel machines like the CM-2. The latter approach has been followed by Mathur and Johnsson [48] for finite element problems.

We have used the hyperplane ordering for preconditioned CG on an ALLIANT FX/4, for 3D systems with dimensions $n_x = 40, n_y = 39$ and $n_z = 30$. For 4 processors this led to a speedup of 2.61, to be compared with a speedup of 2.54 for the CG-process with only diagonally scaling as a preconditioner. The fact that both speedups are quite far below the optimal value of 4, must be attributed to cache effects [85]. These cache effects can be largely removed, when using the reduced system approach suggested by Meier and Sameh [49]. However, for the 3D systems that we have tested sofar, the reduced system approach led, in average, to about the same CPU times as for the hyperplane approach, on Alliant FX/8 and FX/80 computers.

10 *

References

[1] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13:138–144, 1992.

[2] O. Axelsson. Solution of linear systems of equations: iterative methods. In V. A. Barker, editor, *Sparse Matrix Techniques*, Berlin, 1977. Copenhagen 1976, Springer Verlag.

[3] O. Axelsson. Conjugate gradient type methods for unsymmetric and inconsistent systems of equations. *Lin. Alg. and its Appl.*, 29:1–16, 1980.

[4] O. Axelsson and P. S. Vassilevski. A black box generalized conjugate gradient solver with inner iterations and variable-step preconditioning. *SIAM J. Matrix Anal. Appl.*, 12(4):625–644, 1991.

[5] Zhaojun Bai, Dan Hu, and Lothar Reichel. A Newton basis GMRES implementation. Technical Report 91-03, University of Kentucky, 1991.

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

[7] H. Berryman, J. Saltz, W. Gropp, and R. Mirchandaney. Krylov methods preconditioned with incompletely factored matrices on the CM-2. Technical Report 89-54, NASA Langley Research Center, ICASE, Hampton, VA, 1989.

[8] A. Björck and T. Elfving. Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations. *BIT*, 19:145–163, 1979.

[9] P. N. Brown. A theoretical comparison of the Arnoldi and GMRES algorithms. *SIAM J. Sci. Statist. Comput.*, 12:58–78, 1991.

[10] G. Brussino and V. Sonnad. A comparison of direct and preconditioned iterative techniques for sparse unsymmetric systems of linear equations. *Int. J. for Num. Methods in Eng.*, 28:801–815, 1989.

[11] A. T. Chronopoulos and C. W. Gear. s-Step iterative methods for symmetric linear systems. *J. on Comp. and Appl. Math.*, 25:153–168, 1989.

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

[13] P. Concus and G. H. Golub. A generalized Conjugate Gradient method for nonsymmetric systems of linear equations. Technical Report STAN-CS-76-535, Stanford University, Stanford, CA, 1976.

[14] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computations*. Academic Press, New York, 1976.

[15] G. C. (Lianne) Crone. The conjugate gradient method on the Parsytec GCel-3/512. *to appear in FGCS.*

[16] L. Crone and H. van der Vorst. Communication aspects of the conjugate gradient method on distributed-memory machines. *Supercomputer*, X(6):4–9, 1993.

[17] E. de Sturler. A parallel restructured version of GMRES(m). Technical Report 91-85, Delft University of Technology, Delft, 1991.

[18] E. de Sturler. A parallel variant of GMRES(m). In R. Miller, editor, *Proc. of the fifth Int.Symp. on Numer. Methods in Eng.*, 1991.

[19] E. De Sturler and D. R. Fokkema. Nested Krylov methods and preserving the orthogonality. In N. Duane Melson, T.A. Manteuffel, and S.F. McCormick, editors, *Sixth Copper Mountain Conference on Multigrid Methods*, volume Part 1 of *NASA Conference Publication 3324*, pages 111–126. NASA, 1993.

[20] J. Demmel, M. Heath, and H. van der Vorst. Parallel linear algebra. In *Acta Numerica 1993.* Cambridge University Press, Cambridge, 1993.

[21] Shun Doi. On parallelism and convergence of incomplete LU factorizations. *Appl. Num. Math.*, 7:417–436, 1991.

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

[23] 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, Philadelphia, PA, 1991.

[24] Jack J. Dongarra and Henk A. van der Vorst. Performance of various computers using standard sparse linear equations solving techniques. *Supercomputer*, 9(5):17–29, 1992.

[25] I. S. Duff, A. M. Erisman, and J.K.Reid. *Direct methods for sparse matrices.* Oxford University Press, London, 1986.

[26] T. Eirola and O. Nevanlinna. Accelerating with rank-one updates. *Lin. Alg. and its Appl.*, 121:511–520, 1989.

[27] H. C. Elman. *Iterative methods for large sparse nonsymmetric systems of linear equations.* PhD thesis, Yale University, New Haven, CT, 1982.

[28] R. Fletcher. *Conjugate gradient methods for indefinite systems*, volume 506 of *Lecture Notes Math.*, pages 73–89. Springer-Verlag, Berlin–Heidelberg–New York, 1976.

[29] D. R. Fokkema, G.L.G. Sleijpen and H.A. Van der Vorst. Generalized Conjugate Gradient Squared. *Preprint 851*, Dept. Math., University Utrecht, 1994.

[30] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993.

[31] R. W. Freund and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, part 2. Technical Report 90.46, RIACS, NASA Ames Research Center, 1990.

[32] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Num. Math.*, 60:315–339, 1991.

[33] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[34] G. H. Golub and D.P. O'Leary. Some history of the conjugate gradient and lanczos algorithms: 1948-1976. *SIAM Review*, 31:50–102, 1989.

[35] G. H. Golub and C. F. van Loan. *Matrix Computations.* North Oxford Academic, Oxford, 1983.

[36] G. H. Golub and C. F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, 1989.

[37] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[38] M. H. Gutknecht. Variants of BICGSTAB for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14:1020–1033, 1993.

[39] W. Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Teubner, Stuttgart, 1991.

[40] L. A. Hageman and D. M. Young. *Applied Iterative Methods.* Academic Press, New York, 1981.

[41] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.

[42] C. P. Jackson and P. C. Robinson. A numerical study of various algorithms related to the preconditioned Conjugate Gradient method. *Int. J. for Num. Meth. in Eng.*, 21:1315–1338, 1985.

[43] D. A. H. Jacobs. Preconditioned Conjugate Gradient methods for solving systems of algebraic equations. Technical Report RD/L/N 193/80, Central Electricity Research Laboratories, 1981.

[44] K. C. Jea and D. M. Young. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Lin. Algebra Appl.*, 34:159–194, 1980.

[45] E. F. Kaasschieter. A practical termination criterion for the Conjugate Gradient method. *BIT*, 28:308–322, 1988.

[46] C. Lanczos. An iteration method for the solution of the egenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand*, 45:225–280, 1950.

[47] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand*, 49:33–53, 1952.

[48] K. K. Mathur and S. L. Johnsson. The finite element method on a data parallel computing system. Technical Report CS 89-2, Thinking Machines Corporation, 1989. to appear in International Journal of High-Speed Computing.

[49] U. Meier and A. Sameh. The behavior of conjugate gradient algorithms on a multivector processor with a hierarchical memory. Technical Report CSRD 758, University of Illinois, Urbana, IL, 1988.

[50] U. Meier Yang. Preconditioned Conjugate Gradient-Like Methods for Nonsymmetric Linear Systems. *Preprint*, Center for Research and Development, University of Illinois at Urbana-Champaign, 1992.

[51] 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.Comp.*, 31:148–162, 1977.

[52] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24:623–633, 1984.

[53] G. Meurant. Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2. Technical Report LBL-18023, University of California, Berkeley, CA, 1984.

[54] G. Meurant. The conjugate gradient method on vector and parallel supercomputers. Technical Report CTAC-89, University of Brisbane, July 1989.

[55] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13:778–795, 1992.

[56] A. Neumaier. Oral presentation at the Oberwolfach meeting: Numerical Linear Algebra, Oberwolfach, 1994.

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

[58] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Math. Appl.*, 10:373–381, 1972.

[59] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Num. Lin. Alg. with Appl.*, 2:115-134, 1995.

[60] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[61] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Soft.*, 8:43–71, 1982.

[62] B. N. Parlett, D. R. Taylor, and Z. A. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp.*, 44:105–124, 1985.

[63] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, N.J., 1980.

[64] Claude Pommerell. *Solution of large unsymmetric systems of linear equations.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 1992.

[65] Claude Pommerell and Wolfgang Fichtner. PILS: An iterative linear solver package for ill-conditioned systems. Technical Report 91/5, ETH Zürich, 1991.

[66] G. Radicati di Brozolo and Y. Robert. Vector and parallel CG-like algorithms for sparse non-symmetric systems. Technical Report 681-M, IMAG/TIM3, Grenoble, 1987.

[67] G. Radicati di Brozolo and Y. Robert. Parallel conjugate gradient-like algorithms for solving sparse non-symmetric systems on a vector multiprocessor. *Parallel Computing*, 11:223–239, 1989.

[68] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6:865–881, 1985.

[69] Y. Saad. Krylov subspace methods on supercomputers. Technical report, RIACS, Moffett Field, CA, September 1988.

[70] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14:461–469, 1993.

[71] Y. Saad and M. H. Schultz. Conjugate Gradient-like algorithms for solving nonsymmetric linear systems. *Math. of Comp.*, 44:417–424, 1985.

[72] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[73] J. J. F. M. Schlichting and H. A. van der Vorst. Solving 3D block bidiagonal linear systems on vector computers. *Journal of Comp. and Appl. Math.*, 27:323–330, 1989.

[74] Horst D. Simon. Direct sparse matrix methods. In James C. Almond and David M. Young, editors, *Modern Numerical Algorithms for Supercomputers*, pages 325–444, Austin, 1989. The University of Texas at Austin, Center for High Performance Computing.

[75] G. L. G. Sleijpen and H.A. Van der Vorst. Maintaining convergence properties of BICGSTAB methods in finite precision arithmetic. Technical report, University Utrecht, Department of Mathematics, 1994.

[76] G. L. G. Sleijpen and H.A. Van der Vorst. Reliable updated residuals in hybrid Bi-CG methods. *Preprint Nr. 886*, Dept. Math., University Utrecht, 1994.

[77] G. L. G. Sleijpen, H.A. Van der Vorst, and D. R. Fokkema. Bi-CGSTAB(ℓ) and other hybrid bi-cg methods. *Numerical Algorithms*, 7:75–109, 1994.

[78] G. L. G. Sleijpen and D. R. Fokkema. BICGSTAB(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *ETNA*, 1:11–32, 1993.

[79] P. Sonneveld. CGS: a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10:36–52, 1989.

[80] A. van der Sluis and H. A. van der Vorst. The rate of convergence of conjugate gradients. *Numer. Math.*, 48:543–560, 1986.

[81] A. van der Sluis and H. A. van der Vorst. Numerical solution of large sparse linear algebraic systems arising from tomographic problems. In G. Nolet, editor, *Seismic Tomography*, chapter 3, pages 49–83. Reidel Pub. Comp., Dordrecht, 1987.

[82] A. van der Sluis and H. A. van der Vorst. SIRT- and CG-type methods for the iterative solution of sparse linear least-squares problems. *Lin. Alg. and its Appl.*, 130:257–302, 1990.

[83] H. A. van der Vorst. The convergence behavior of some iterative solution methods. In R. Gruber, J. Periaux, and R. P. Shaw, editors, *Proc. of the fifth Int.Symp. on Numer. Methods in Eng.*, 1989. vol 1.

[84] 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. Yu. Kolotilina, editors, *Preconditioned Conjugate Gradient Methods*, Berlin, 1990. Nijmegen 1989, Springer Verlag. Lecture Notes in Mathematics 1457.

[85] H. A. van der Vorst. Experiences with parallel vector computers for sparse linear systems. *Supercomputer*, 37:28–35, 1990.

[86] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.

[87] H. A. van der Vorst. Conjugate gradient type methods for nonsymmetric linear systems. In R. Beauwens and P. de Groen, editors, *Iterative Methods in Linear Algebra*, Amsterdam, 1992. IMACS Int. Symp., Brussels, Belgium, 2-4 April, 1991, North-Holland.

[88] H. A. van der Vorst and C. Vuik. The superlinear convergence behaviour of GMRES. *JCAM*, 48:327–341, 1993.

[89] H. A. van der Vorst and C. Vuik. GMRESR: A family of nested GMRES methods. *Num. Lin. Alg. with Appl.*, 1:369–386, 1994.

[90] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs N.J., 1962.

[91] P. K. W. Vinsome. ORTOMIN: an iterative method for solving sparse sets of simultaneous linear equations. In *Proc.Fourth Symposium on Reservoir Simulation*, pages 149–159. Society of Petroleum Engineers of AIME, 1976.

[92] H. F. Walker. Implementation of the GMRES method using Householder transforma- tions. *SIAM J. Sci. Stat. Comp.*, 9:152–163, 1988.

[93] O. Widlund. A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.

[94] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.

[95] L. Zhou and H. F. Walker. Residual smoothing techniques for iterative methods. *SIAM J. Sci. Comput.*, 15:297–312, 1994.