Chapter 12:
Optimal Iteration methods
for
large linear systems of equations

G.L.G. Sleijpen and H.A. van der Vorst
Mathematical Institute, University of Utrecht,
P.O.Box 80.010, NL-3508 TA Utrecht, The Netherlands.

12.1 Introduction

When solving PDE's by means of numerical methods one often has to deal with large systems of linear equations, specifically if the PDE is time-independent or if the time-integrator is implicit. For real life problems, these large systems can often only be solved by means of some iterative method. Even if the systems are preconditioned, the basic iterative method often converges slowly or even diverges. We discuss and classify algebraic techniques to accelerate the basic iterative method. Our discussion includes methods like CG, GCR, ORTHODIR, GMRES, CGNR, Bi-CG and their modifications like GMRESR, CG-S, Bi-CGSTAB. We place them in a frame, discuss there convergence behavior and their advantages and drawbacks.

Our aim is to compute acceptable approximations for the solution x of the equation

$$Ax = b, \tag{12.1}$$

where A and b are given, A is a non-singular $n \times n$-matrix, A is sparse, n is large and b an n-vector. We will assume A and b to be real, but our methods are easily generalized to the complex case.

Such equations arise frequently when solving numerically discretized PDE's, for instance, for stationary problems or with implicit time-integration. Solving non-linear equations by means of, for instance, Newton's method may also lead to such systems.

If n is large, direct methods are often too expensive and one has to switch to iterative methods. Usually the basic iterative method (Richardson) does not converge or converges very slowly. By preconditioning (Jacobi, Gauss–Seidel, SSOR, ILU, etc.), one may speed up the convergence, but it may still be too slow or even fail to converge. In this chapter, we discuss algebraic acceleration techniques for these basic preconditioned iterative methods: we concentrate on techniques based on the matrix and discard the origin of the equation (for a discussion of Multigrid techniques, we refer to the next chapter).

These acceleration techniques have resulted in methods like Conjugate Gradients (CG), GMRES, Bi-CG, and others.

For symmetric positive definite problems CG is the optimal acceleration technique. In practice, however, one has often to solve unsymmetric problems; diffusion-advection problems almost always lead to unsymmetric systems. For these problems, there are

many generalizations of CG that accelerate (or aim to accelerate) the basic iterative method. They iteratively produce a sequence (x_k) (either explicitly or implicitly) of approximations x_k of the solution x.

In this chapter we will focus on optimal iterative algorithms for nonsymmetric systems. These algorithms fall in two classes: GMRES and variants (optimal methods with long recurrences), and Bi-CG and variants (quasi-optimal methods with short recurrences).

By mixing methods from these classes one hopes to preserve some good properties and to get rid of, or to weaken, the bad ones. Bi-CGSTAB is such a method. It was developed to stabilize Bi-CG by mixing it with a GMRES type of method.

There is no robust method that solves all problems:
(a) for mathematical reasons,
(b) for reasons of stability and
(c) in the event of only a limited amount of memory.

Sub (a): Consider some methods that are mathematically different (and not for subtle implementation reasons); for instance, GMRES, Bi-CGSTAB and CG applied to the normal equations. Then there are different linear systems $Ax = b$ such that each of the methods is the winner for one of these systems and the worst method for one of the others (see, [21]). Although the examples have to be selected to achieve this, they are certainly not exceptional. One may attempt to classify problems for which certain methods are superior (over others). Unfortunately, it will be hard in practice to determine the class in which a problem fits and, what will be more likely, the problem will often not fit nicely in one class but may have aspects of several classes. These aspects may turn up in different phases of a computation. It may even be the case that by slightly increasing the dimension of the problem, through grid-refinement, advantages of a selected method are lost.
Sub (b): In practice, the computational process will be contaminated by evaluation errors due to finite precision arithmetic. Methods that, for theoretical reasons, should perform the best for certain types of problems may fail to converge at all due these errors. Whether the evaluation errors affect the speed of convergence or not may depend dramatically on the type of problem and its size.
Sub (c): In view of the restricted computational resources, methods may have to be adapted if the size of the matrices involved is too large. Methods that are superior for a certain type of problem may not be adaptable or may loose their superiority when adapted.

In spite of these observations, we have to take the situation as it is. There is no practical alternative and many problems can be solved much faster or even can be solved at all only because of these techniques.

The CG method was introduced in 1954 by Hestenes and Stiefel [15]. But it did not get much attention until in 1971. Then Reid [25] showed how this method can be used to our advantage for certain systems. Since then there has been a growing interest in the CG method mainly for its use as an acceleration technique [2]. Meijerink and van der Vorst [19] showed how efficient the CG method can be if combined with the ILU preconditioner. Pure generalizations of CG to solve non-symmetric equations were introduced in the late seventies and early eighties. GMRES and variants use orthogonal basis. Algorithms of this type were proposed by Vinsome [37], Young and Jea [43], Saad

[26], Elman [8], Axelsson [3] and others. The GMRES algorithm of Saad and Schultz [28] from 1986 seems to be the most popular one of this type. In 1952, Lanczos [17] used bi-orthogonality relations to reduce iteratively a matrix to tri-diagonal form and he suggested how to solve non-symmetric linear systems from the reduced tri-diagonal form. Later, his ideas were adapted for a variety of methods, such as Bi-CG [10], QMR [12] and others. In 1989, Sonneveld [33] introduced a "squared Bi-CG algorithm", the CG-S algorithm: he used the observation that the computational effort in the Bi-CG algorithm to get bi-orthogonality can be used for an additional reduction of the residuals. The "squared-QMR" algorithm [11] is the most robust and efficient algorithm of pure Bi-CG type. The steps in the GMRES-like algorithms are increasingly expensive while the algorithms of Bi-CG type often converge slowly and are sensitive to evaluation errors. Recently, a number of mixed methods were introduced that often suffer less from these drawbacks (see [40], [42], [6], [27], [5], [4], [16], [29] etc.). A lot of research is still focussed on this subject.

In this chapter, we present some of the most powerful methods, discuss the convergence behavior in relation to properties of A, give algorithmical realizations of the methods, show how they are related and discuss briefly their advantages and drawbacks. In our discussions we will sacrifice mathematical rigor for short presentations. In particular, we will not discuss all kind of degenerate cases, and refer the reader to textbooks for a more concise description (e.g., [36, 13]).

12.2 The basic iterative method

12.2.1 The method

Classical iterative methods (Jacobi, Gauss–Seidel, SOR, etc.) are based on a splitting of A:

$$A = K - R \tag{12.2}$$

in which K is a non-singular $n \times n$-matrix. Starting with some suitable approximation x_0 for the solution x the sequence of approximations x_k is produced by iterating

$$x_{k+1} = x_k + K^{-1}(b - Ax_k) \quad \text{for all} \quad k \in \mathbb{N}_0. \tag{12.3}$$

From the expression (12.4) below for the errors $e_k \equiv x - x_k$ and (12.5) for the **residuals** $r_k \equiv b - Ax_k$ we see that the convergence of (x_k) to x is determined by how close $K^{-1}A$ or AK^{-1} is to I .

The matrix K^{-1} is almost never constructed explicitly. It is often more economical to solve $u_k \equiv K^{-1}r_k$ from $Ku_k = r_k$. This means that K has to be chosen so that this system can be solved very efficiently, for instance $K = I$ (Richardson) or $K = \mathrm{diag}(A)$ (Jacobi) are popular choices. Other popular choices for K are the lower triangular part of A (Gauss-Seidel), and the tridiagonal part of A (line-Jacobi). For the slightly more complicated Incomplete LU decompositions see, for instance, [19, 20]. These standard processes converge slowly, but as we will see they can be accelerated by combining information of successive iteration steps.

Note that the method in (12.3) is equivalent to the Richardson iteration applied to the preconditioned system $Bx = c$, with $B = K^{-1}A$ and $c = K^{-1}b$. Hence, there would be no loss of generality in taking the Richardson iteration as our starting point. The subsequent theory to accelerate the basic iteration method in (12.3) actually is a theory for the preconditioned Richardson iteration process.

```
choose x_0,
k = -1,      r_0 = b - A x_0
repeat until ||r_{k+1}|| is small enough:
      { k := k + 1
      { solve u_k from K u_k = r_k,  c_k = A u_k,
      { x_{k+1} = x_k + u_k,    r_{k+1} = r_k - c_k
```

Algorithm 12.1: The basic iterative algorithm.

Since $x = x + K^{-1}(b - Ax)$ we have that

$$e_{k+1} = (I - K^{-1}A)e_k, \tag{12.4}$$

and, since $r_k = Ae_k$,

$$r_0 = b - Ax_0, \quad r_{k+1} = r_k - Au_k \quad \text{with} \quad u_k = K^{-1}r_k. \tag{12.5}$$

The sequence of errors converges to 0 whenever the sequence of residuals does. We would like to stop the iteration if $\|e_k\|$ is small enough, where $\|\cdot\|$ is some inner product norm on the space of n-vectors. Normally, e_k will not be available and stopping criteria have to be based upon the knowledge of r_k (see, for instance, [1]).

Note that

$$x_{k+1} = x_k + u_k. \tag{12.6}$$

The equations (12.5) and (12.6) constitute a practical algorithmical representation of the basic iterative method in (12.3) given in Algorithm 12.1.

In (12.5) we "correct" the k-th residual r_k to obtain a new and hopefully smaller residual r_{k+1} by some vector of the form Au_k, and we know u_k, the update for x_k. This is an important observation since in the accelerated methods often the construction is focussed on the reduction of the residual and the update for the current approximation is obtained more or less as a side product.

12.2.2 Convergence

From (12.5), we have that

$$r_k = (I - AK^{-1})^k r_0. \tag{12.7}$$

Since this is essentially the power method for $I - AK^{-1}$, it is obvious that the convergence behavior will be dictated by the largest eigenvalue in absolute value of that matrix (notated as $\rho(I - AK^{-1})$). For convergence it is necessary that this value be less than 1, and for fast convergence it should be much smaller than 1.

The preconditioned matrix AK^{-1} plays an important role. It determines the convergence also of the other methods to be discussed in this chapter. For ease of presentation we denote this matrix by $\mathbb{A}$, $\mathbb{A} \equiv AK^{-1}$.

By including a parameter, i.e. by working with $\frac{1}{\alpha}K$ instead of K, the residual reduction factor becomes $\rho(I - \alpha\mathbb{A})$, which may be smaller than $\rho(I - \mathbb{A})$ for some $\alpha \neq 1$. Therefore, for some appropriate **relaxation parameter** α one may improve the convergence.

Example 1 Suppose all eigenvalues of $\mathbb{A}$ are in the interval $[\lambda_1, \lambda_n]$ with $\lambda_1 > 0$. With $\alpha \equiv \frac{2}{\lambda_1+\lambda_n}$ we have that

$$\rho(I - \alpha\mathbb{A}) = 1 - \alpha\lambda_1 = \frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1} = \frac{1 - \frac{\lambda_1}{\lambda_n}}{1 + \frac{\lambda_1}{\lambda_n}} \approx 1 - 2\frac{\lambda_1}{\lambda_n}.$$

Hence, in this case

$$\frac{\|r_k\|}{\|r_0\|} \leq \mathcal{C}_E \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^k \approx \mathcal{C}_E \exp\left(-2k\frac{\lambda_1}{\lambda_n}\right), \tag{12.8}$$

where $\mathcal{C}_E$ is some constant that depends on the eigenvectors of $\mathbb{A}$. $\mathcal{C}_E = 1$ if $\mathbb{A}$ is real and symmetric and $\|\cdot\|$ is the Euclidean norm.

We now can estimate how many iteration steps are required to reduce the residual by a factor $\varepsilon > 0$:

$$\frac{\|r_k\|}{\|r_0\|} \leq \varepsilon \quad \text{if} \quad k \geq \frac{1}{2}\frac{\lambda_n}{\lambda_1}\log\frac{\mathcal{C}_E}{\varepsilon}. \tag{12.9}$$

If A is the d-dimensional Laplace operator discretized by symmetric finite differences and K stems from Incomplete LU preconditioning then $\lambda_n/\lambda_1 = \mathcal{O}(h^{-2})$ and $\mathcal{C}_E = \mathcal{O}(1)$ $(h \to 0)$, where h is the mesh size. In this case, the required number of iterations will be proportional to h^{-2}.

In order to make this relaxation procedure practical one should have fairly accurate estimates for the extremal eigenvalues of $\mathbb{A}$ (cf. [36]). Unfortunately, in general, this information is not readily available. However, since one is free to select a different relaxation parameter α_k in each iteration step, one can easily obtain relaxation parameters that help to reduce the residual. For instance, by choosing α_k such that $\|r_{k+1}\| = \|r_k - \alpha_k\mathbb{A}r_k\|$ is minimal, or equivalently, $r_{k+1} - \mathbb{A}r_k$, we obtain the so-called **LMR** (**local minimum residual**) method (related to the steepest descent method). Although this is certainly an improvement over the standard iteration approach, it can still not compete with the methods to come. However, as we will see, this LMR approach gives the clue for the global minimum residual methods.

12.3 Krylov subspace methods

Note that, for the residual in the LMR method, we have that

$$r_k = (I - \alpha_k\mathbb{A})(I - \alpha_{k-1}\mathbb{A})\cdots(I - \alpha_1\mathbb{A})r_0. \tag{12.10}$$

Obviously r_k is a special combination of the vectors r_0, $\mathbb{A}r_0$, ..., $\mathbb{A}^k r_0$. These vectors form a basis[1] for the **Krylov subspace** $\mathcal{K}_{k+1}(\mathbb{A}; r_0)$ of dimension $k+1$, and the question arises whether we may be able to select a better vector from this subspace.

With $q_k(t) = \prod_{j=1}^{k}(1 - \alpha_j t)$, we can rewrite (12.10) as

$$r_k = q_k(\mathbb{A})r_0. \tag{12.11}$$

We call any method that produces a sequence (x_k) a **Krylov subspace method**, or **polynomial method**, if the k-th residual $r_k = b - Ax_k$ can be written as in (12.11) for

[1] Here, as elsewhere in this chapter, in order to avoid technical subtilities, we assume that these vectors are linearly independent, that is, we assume that any Krylov space $\mathcal{K}_{k+1}(\mathbb{A}; v_0)$ that we discuss has dimension $k+1$. This assumption limits the size of k.

some polynomial q_k of degree k with $q_k(0) = 1$.
Since $q_k(0) = 1$, we may write $q_k(t) = 1 - t\tilde{q}_{k-1}(t)$, in which $\tilde{q}_{k-1}$ is a $(k-1)$-st degree polynomial, it follows that $x_k = x_0 + K^{-1}\tilde{q}_{k-1}(\mathbb{A})r_0$, so for any polynomial method r_k is the residual of a computable approximation x_k. Obviously, the basic iterative process is polynomial as well (with $q_k(t) = (1-t)^k$).
For some of the methods to be discussed it will be helpful to exploit the polynomial point of view (specially CG-S and Bi-CGSTAB), for other methods it is easier to take the Krylov subspace point of view. A well-known approach is to select beforehand a fixed set of polynomials (e.g., Chebyshev polynomials) and to construct an algorithm from these polynomials (e.g., the Chebyshev iteration method [36]). This requires explicit knowledge of the spectrum of the operator A; we will not further discuss these **stationary** methods here.

12.4 Optimal Krylov subspace methods

For (non-stationary) polynomial methods, one would like to have residuals r_k as in (12.11) that are as small as possible. In the numerical literature, this is interpreted in two different ways. One way is to determine a polynomial q_k of degree k, with $q_k(0) = 1$, so that $\|r_k\| = \|q_k(\mathbb{A})r_0\|$ is minimal, the other way is to determine r_k in such a way that it is orthogonal to the space spanned by r_0, $\mathbb{A}r_0$, ..., $\mathbb{A}^{k-1}r_0$, i.e., the new residual should be free of components in the space that has already been explored.

Though the first approach, the **MR** (global **Minimum Residual**) approach, seems more obvious, it leads to slightly more complicated algorithms, whereas the second approach, the **OR** (**Orthogonal Residual**) approach, has led to more popular algorithms. For instance, if the preconditioned matrix $\mathbb{A}$ is symmetric, then the orthogonality approach leads to the famous Conjugate Gradients method. Successful application of this method is restricted to symmetric *positive definite* matrices, while for other symmetric matrices it is better to rely on the MINRES [23] algorithm, which is based on the MR approach.

Likewise, in the unsymmetric case we obtain GMRES [28] by following the minimization approach, and FOM [26], if we follow the orthogonality approach. Depending on the particular construction of the minimal residual, or the orthogonal residual, we obtain differently named methods, but if they do not break down, then in exact arithmetic they produce the same results as either GMRES or FOM. For instance, methods as GCR, ORTHOMIN, GENCR, and ORTHODIR, are equivalent with GMRES, and besides disadvantages they may also have advantages over GMRES. For instance, in GMRES the approximated solution is only available at the very end of the iteration process, while in the less-robust GCR algorithm one has the current approximation available in each iteration step.

In [41] the convergence behavior of the MR approach and the OR approach is studied, in relation to spectral properties of the matrix $\mathbb{A}$. From a practical point of view there is not much difference in the speed of convergence between both processes: if MR converges well, that is there is significant progress in one iteration step, then OR does about as well in the same iteration step. However, when MR stagnates temporarily, then OR may break down. Therefore, we will present only algorithms of the MR-type.

Although GMRES is more popular and more efficient than GCR, we concentrate on GCR: GCR allows adaptations — algorithms that do not represent the MR approach — that are often more efficient with respect to both use of memory and computational cost. GMRES is less adjustable to these situations. Moreover, there is a variant of GCR

that is competitive to GMRES. We refer the reader to literature for more details on the other algorithms of MR-type and on the algorithms of OR-type.

12.4.1 Convergence

We consider, for the discussion of convergence, some sequence of residuals r_k that satisfy the minimization property. The reduction of the residuals can be estimated by

$$\frac{\|r_k\|}{\|r_0\|} \leq \min_{q \in \mathcal{P}_k^1} \frac{\|q(\mathbb{A})r_0\|}{\|r_0\|} \leq \min_{q \in \mathcal{P}_k^1} \|q(\mathbb{A})\|, \tag{12.12}$$

where $\mathcal{P}_k^1$ is the space of all polynomials q of degree k with $q(0) = 1$.

Apparently, of all Krylov subspace methods, this MR approach gives the best reduction of the residual and it does not require any à priori knowledge of the eigenvalues of $\mathbb{A}$.

The expression at the right-hand side of (12.12) can be bounded in terms of quantities that depend on the eigensystem of $\mathbb{A}$. The spectrum $\sigma(\mathbb{A})$ is involved and, if $\mathbb{A}$ has a basis of eigenvectors, the condition number $\mathcal{C}_E$ of this basis; the spectrum is the collection of all eigenvalues of and the condition number of a basis is the condition number of the matrix of which the columns are the basis vectors. For instance[2],

$$\min_{q \in \mathcal{P}_k^1} \|q(\mathbb{A})\| \leq \mathcal{C}_E \varepsilon_k(\sigma(\mathbb{A})), \tag{12.13}$$

where, for any subset $\mathbf{E}$ of $\mathbb{C}$, ε_k is defined by

$$\varepsilon_k(\mathbf{E}) \equiv \min_{q \in \mathcal{P}_k^1} \max \left\{ |q(\lambda)| \,\middle|\, \lambda \in \mathbf{E} \right\}. \tag{12.14}$$

If one has a subset $\mathbf{E}$ of $\mathbb{C}$ that contains $\sigma(\mathbb{A})$ then one can give an à priori upper-bound on the size of r_k, because: $\varepsilon_k(\sigma(\mathbb{A})) \leq \varepsilon_k(\mathbf{E})$. If, for instance, $\mathbf{E}$ is an ellipse then $\varepsilon_k(\mathbf{E})$ can be bounded using Chebyshev polynomials (see [9]). The following example shows an upper-bound in case $\mathbf{E} = [\lambda_1, \lambda_n]$ and $\lambda_1 > 0$.

Example 1 (continued 1) If $\mathbb{A}$ has a basis of eigenvectors then

$$\frac{\|r_k\|}{\|r_0\|} \leq \mathcal{C}_E \, \varepsilon_k([\lambda_1, \lambda_n]) \lesssim \mathcal{C}_E \left(\frac{1 - \sqrt{\frac{\lambda_1}{\lambda_n}}}{1 + \sqrt{\frac{\lambda_1}{\lambda_n}}} \right)^k \approx \mathcal{C}_E \, \exp\left(-2k \sqrt{\frac{\lambda_1}{\lambda_n}} \right). \tag{12.15}$$

Compared to the basic iterative method with optimal relaxation parameter, the number of steps, we need[3] to reduce the initial residual, by a factor ε, is then less by a multiplicative factor $\sqrt{\frac{\lambda_n}{\lambda_1}}$:

$$\frac{\|r_k\|}{\|r_0\|} \leq \varepsilon \quad \text{if} \quad k \geq \frac{1}{2} \sqrt{\frac{\lambda_n}{\lambda_1}} \log \frac{\mathcal{C}_E}{\varepsilon}. \tag{12.16}$$

In case of the discretized Laplace operator, the number of steps required is proportional to h^{-1} (compare this to the $\mathcal{O}(h^{-2})$ for the unaccelerated standard method) .

[2] If the eigenvectors do not form a basis or if $\mathcal{C}_E$ is too large one can obtain better upper-bounds for $\min_{q \in \mathcal{P}_k^1} \|q(\mathbb{A})\|$ by using the so-called ε-pseudospectrum of $\mathbb{A}$; cf. [21] and [35].

[3] Both the estimates in (12.8) and (12.15) are asymptotically sharp in some generic sense.

The MR approach has a number of advantages over any stationary polynomial method:

(i) it does not need any information in advance on the spectrum of $\mathbb{A}$;

(ii) it attains the fastest speed of convergence possible by any Krylov subspace method;

(iii) the convergence accelerates in the course of the computation.

In view of (12.12), (i) and (ii) need no additional comment. However, we wish to point out that, although the convergence properties do not depend on the knowledge of the spectrum, they do essentially depend on the spectrum. Given a matrix $\mathbb{A}$, we can not change its spectrum and thus influence the speed of convergence. However, only A is given and the preconditioner K has been added precisely for the purpose of improving the convergence. Therefore, it is important, for the construction of a good preconditioner, to know how the speed of convergence and spectral properties are related (see, for instance, [38]). Also with this observation in mind, we will briefly comment on (iii).

Eigenvalues of the orthogonal "projection" of $\mathbb{A}$ onto a Krylov space (of dimension k) are called **Ritz values** (of order k). They tend to converge to the eigenvalues of $\mathbb{A}$, where isolated eigenvalues and extremal eigenvalues tend to be approximated sooner [31]. If, in the k-th step, a set E of eigenvalues of $\mathbb{A}$ has been "found" by Ritz values (i.e. the eigenvalues in E have been approximated accurately), then, from this step on, the set E does not seem to "affect" the speed of convergence anymore: an upper-bound of $\|r_{k+l}\|/\|r_k\|$ decreases like $\varepsilon_l(\sigma(\mathbb{A})\backslash E)$ and the convergence accelerates [41]. This phenomenon is explained by the fact that the finding of an eigenvalue corresponds to the elimination of the component of the residual in the direction of the associated eigenvector. Whether the acceleration is of significance depends on the size of the found eigenvalues and on how well they are separated from the ones not found yet, the "hidden" ones. If Ritz values find a small eigenvalue λ (that is, with a small absolute value) that is relatively well separated from the hidden eigenvalues (relative with respect to $|\lambda|$) then significant acceleration may be expected. Especially the extremal eigenvalues that are fairly well separated from the "inner" eigenvalues seem to be found quickly [32, 31]. Therefore, if the smaller eigenvalues of the preconditioned matrix $\mathbb{A}$ are fairly extremal and relatively well separated the convergence of the MR approach may expected to exhibit considerable accelerations occurring in an early stage of the process.

Except for a factor like $\mathcal{C}_E$, the size of the errors, residuals and the speed of convergence (measured in some norm) are independent of the basis that has been chosen to represent the problem. If $\mathbb{A}$ has a basis of eigenvectors, these quantities only depend on:

(i) the distribution of the eigenvalues of $\mathbb{A}$;

(ii) the sizes $|\mu_1|^2, \ldots, |\mu_n|^2$, where $r_0 = \sum_{j=1}^n \mu_j v_j$ and $v_1, \ldots, v_n$ are the eigenvectors of $\mathbb{A}$;

(iii) $\mathcal{C}_E$.

As observed above, the distribution of the eigenvalues determines the convergence behavior: speed and acceleration strongly depend on it. The sizes $|\mu_j|$ of the eigenvector components of the initial residual do not seem to be of much influence: moderate variations on these sizes seem to yield only minor changes in convergence behavior without essentially changing its character [32, 31]. These observations are only correct in case $\mathcal{C}_E$ is not too large. If $\mathcal{C}_E$ is large anything may happen (example 5 can be viewed as an extreme illustration for this observation).

12.4.2 Algorithms

The standard approach for the construction of the r_k with the orthogonality or minimality property is to construct an orthogonal basis for the Krylov subspace $\mathcal{K}_k(\mathbb{A};\tilde{c}_0)$ where either $\tilde{c}_0 = r_0$ as in GMRES or $\tilde{c} = \mathbb{A}r_0$ as in GCR (to be discussed below). This can be done in various ways, one more stable than an other, but the most often followed approach is to construct this basis by the Modified Gram-Schmidt method. We discuss this method first.

Modified Gram-Schmidt. Assume we have some **basis $\tilde{c}_0, \tilde{c}_1, \ldots$ of the Krylov space of** $\mathbb{A}$ generated by $\tilde{c}_0$, that is, for each k, $\mathcal{K}_k(\mathbb{A};\tilde{c}_0) = \mathrm{span}(\tilde{c}_0, \ldots, \tilde{c}_{k-1})$. From this basis we construct inductively an orthogonal basis $c_0, c_1, \ldots$ of the same Krylov space.

Take $c_0 = \tilde{c}_0$.

To explain how to construct c_k if $c_0, \ldots, c_{k-1}$ have been constructed, consider the map P_k given by

$$P_k \equiv \sum_{j=0}^{k-1} \frac{c_j c_j^T}{c_j^T c_j}. \tag{12.17}$$

P_k is the orthogonal projection on $\mathcal{K}_k(\mathbb{A};\tilde{c}_0)$. Hence, $I - P_k$ projects on the orthogonal complement of this space. Now, let

$$c_k \equiv \tilde{c}_k - P_k\tilde{c}_k = \tilde{c}_k - \sum_{j=0}^{k-1} \tau_{jk} c_j \quad \text{with} \quad \tau_{jk} \equiv \frac{(\tilde{c}_k, c_j)}{(c_j, c_j)}. \tag{12.18}$$

Then c_k is orthogonal to $c_0, \ldots, c_{k-1}$ and, clearly, $c_0, \ldots, c_k$ span $\mathcal{K}_{k+1}(\mathbb{A};\tilde{c}_0)$: c_k is our next orthogonal vector. Here, we orthogonalize by the well-known Gram-Schmidt process.

Note that

$$I - P_k = I - \sum_{j=0}^{k-1} \frac{c_j c_j^T}{c_j^T c_j} = (I - \frac{c_{k-1} c_{k-1}^T}{c_{k-1}^T c_{k-1}}) \cdot \ldots \cdot (I - \frac{c_0 c_0^T}{c_0^T c_0}). \tag{12.19}$$

Hence, the orthogonalization can be performed recursively. The stability of this recursive way of computing an orthogonal system of vectors is satisfactory: evaluation errors arising in the course of the computation are also projected and subtracted and thus suppressed. So, c_k can be computed by the **mod.G-S (modified Gram-Schmidt)** process:

$$c_k^{(0)} = \tilde{c}_k, \quad c_k^{(j+1)} = c_k^{(j)} - \tau_{jk} c_j \ \ (j = 0, \ldots, k-1), \quad c_k = c_k^{(k)}, \tag{12.20}$$

where, in exact arithmetic, the $\tau_{jk} = (c_k^{(j)}, c_j)/(c_j, c_j)$ coincide with the ones in (12.18).

Generalized Conjugate Residuals and related algorithms. Suppose $\tilde{c}_0, \tilde{c}_1, \ldots$ is a basis of the Krylov space of $\mathbb{A}$ generated by $\tilde{c}_0 \equiv \mathbb{A}r_0$. If $c_0, c_1, \ldots$ is the sequence of orthogonal vectors produced by mod.G-S then $P_k(r_0)$ is the best approximation of r_0 in $\mathbb{A}\mathcal{K}_k(\mathbb{A};r_0) = \mathcal{K}_k(\mathbb{A};\tilde{c}_0)$ and $r_k = (I - P_k)r_0$ is the k-th residual in the MR approach. The correction $P_k r_0$ can easily be computed by inner products and vector updates.

In order to update the approximation, the corrections of the residuals should be of the form Au where we have to know u, the update of the approximation. Suppose we

choose x_0,
$k = -1, \quad r_0 = b - Ax_0$,
repeat until $\|r_{k+1}\|$ is small enough:
$k := k + 1$,
solve $\tilde{u}$ from $K\tilde{u} = r_k$, $\tilde{c} := A\tilde{u}$,
compute by mod.G-S $u_k = \tilde{u} - Q_k\tilde{c}$, $c_k = \tilde{c} - P_k\tilde{c}$,
$\gamma_k = (c_k, c_k)$, $\alpha_k = (r_k, c_k)/\gamma_k$,
$x_{k+1} = x_k + \alpha_k u_k$, $r_{k+1} = r_k - \alpha_k c_k$

Algorithm 12.2: The GCR algorithm.

have chosen the updates $\tilde{u}_j$, $\tilde{u}_0 = K^{-1}r_0$, and the $\tilde{c}_j$, with $\tilde{c}_j \equiv A\tilde{u}_j$, form a basis of the Krylov space generated by $\mathbb{A}$. Then the u_j for which $c_j = Au_j$ can easily be computed: because, if

$$c_k = \tilde{c}_k - \sum_{j=0}^{k-1} \tau_{jk} c_j, \quad u_k \equiv \tilde{u}_k - \sum_{j=0}^{k-1} \tau_{jk} u_j \quad \text{with} \quad \tau_{jk} = (\tilde{c}_k, c_j)/(c_j, c_j). \tag{12.21}$$

and if $Au_j = c_j$ for $j < k$ then obviously $Au_k = c_k$. The computation of c_k and of u_k involves the same inner products. The c_k and the τ_{jk} can be computed by mod.G-S (see (12.20)). Since (in exact arithmetic) the "modified" τ_{ij} and the original ones coincide, the u_k can also be computed by an analog of (12.20).

For ease of reference, we recall the definition of P_k and we introduce the map Q_k.

$$P_k = \sum_{j=0}^{k-1} \frac{c_j c_j^T}{c_j^T c_j} \quad \text{and} \quad Q_k \equiv \sum_{j=0}^{k-1} \frac{u_j c_j^T}{c_j^T c_j}. \tag{12.22}$$

Note that

$$P_k = AQ_k, \quad c_k = \tilde{c}_k - P_k\tilde{c}_k \quad \text{and} \quad u_k = \tilde{u}_k - Q_k\tilde{c}_k. \tag{12.23}$$

Now, $x_k = x_0 + Q_k r_0$. By (12.19), we actually have that

$$x_{k+1} = x_k + \alpha_k u_k \quad \text{and} \quad r_{k+1} = r_k - \alpha_k c_k \quad \text{with} \quad \alpha_k = (r_k, c_k)/(c_k, c_k). \tag{12.24}$$

Remains to explain how to choose the $\tilde{u}_k$ such that the $\tilde{c}_k \equiv A\tilde{u}_k$ form a basis of the Krylov space of $\mathbb{A}$ generated by $\mathbb{A}r_0$: for any k, we have to choose a $\tilde{u}_k$ in $K^{-1}\mathcal{K}_{k+1}(\mathbb{A}; r_0)$ that does not belong to $K^{-1}\mathcal{K}_k(\mathbb{A}; r_0)$. In literature, one can find two choices for $\tilde{u}_k$: GCR and ORTHODIR. We will present GCR.

Recall that $r_k = r_0 + \zeta_1 \mathbb{A}r_0 + \ldots + \zeta_k \mathbb{A}^k r_0$ for some real ζ_i. We may choose $\tilde{u}_k = K^{-1}r_k$: if $\zeta_k \neq 0$ then $\tilde{u}_k$ has the desired property. This choice and the equations (12.23) (or (12.21)) and (12.24) lead to **GCR** (Generalized Conjugate Residuals) [7, 8] or **ORTHOMIN** [37] given in Algorithm 12.2.

In order to facilitate the presentation of (i) a cheap variant of GCR, and (ii) a way to compute the spectrum of $\mathbb{A}$ (see 12.4.3), we represent the relations in GCR in terms of matrices.

A matrix representation of GCR. We define the $n \times k$ matrix R_k as the matrix with columns $r_0, r_1, \ldots, r_{k-1}$ and the $n \times k$ matrix C_k as the matrix with the orthonormal vectors $\frac{1}{\sqrt{\gamma_0}} c_0$, $\ldots$, $\frac{1}{\sqrt{\gamma_{k-1}}} c_{k-1}$ as its columns. Now, the GCR relation $\tilde{c} = A\tilde{u} = \mathbb{A}r_k =$

$c_k + P_k\tilde{c}$ leads to $\mathbb{A}R_k = C_kT_k$, where T_k is the $k \times k$ upper-triangular matrix with the scaled projection-coefficients $\tau_{ij}\sqrt{\gamma_i}$ (cf. (12.21) and Algorithm 12.2) in the strict upper-triangle and $\sqrt{\gamma_i}$ on its diagonal. Since the columns of C_k form an orthonormal system, we have that $C_k^TC_k$ is the $k \times k$ identity matrix.

Let D_k the $k \times k$ diagonal matrix with diagonal coefficients $\alpha_0\sqrt{\gamma_0}, \ldots, \alpha_{k-1}\sqrt{\gamma_{k-1}}$. Since $r_{k+1} = r_k - \alpha_k c_k$ we have that $R_{k+1}S_k = C_kD_k$, where S_k is the $(k+1) \times k$ matrix given by $S_ke_j = e_{j+1} - e_j \quad (j = 1, \ldots, k)$.

Summarizing,

$$\mathbb{A}R_k = C_kT_k \quad \text{with} \quad C_k^TC_k = I \quad \text{and} \quad R_{k+1}S_k = C_kD_k. \tag{12.25}$$

Break down. GCR may **break down** before we have an accurate approximation. This happens if $\alpha_k = 0$ for some $k \leq n$. Then $r_{k+1} \in \mathcal{K}_k(\mathbb{A}; r_0)$ and $c_{k+1} = 0$. Consequently, the c_j's are not independent (now, also $\gamma_{k+1} = 0$ and division by γ_{k+1} is impossible).

Example 3 Consider the equation with $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $K^{-1} = I$ and $b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. The choice $x_0 = 0$ yields $r_0 = b$ and $\alpha_0 = (r_0, Ar_0)/(c_0, c_0) = 0$.

GCR does not break down if, for instance, $A^T + A$ is positive definite, or, equivalently, if $\mathrm{Re}(Ay, y) > 0$ for all y.

One can avoid breakdown by replacing, in the GCR algorithm, "`solve` $\tilde{u}$ `from` $K\tilde{u} = r_k$" by "`solve` $\tilde{u}$ `from` $K\tilde{u} = c_{k-1}$". The resulting algorithm is known as ORTHODIR [43]. Except for some multiplicative scalar factor, the c_k in GCR and ORTHODIR coincide if GCR does not break down.

Costs. We do not need to store the α_k. Moreover, x_{k-1} and r_{k-1} are only needed in the $(k-1)$-th step and can then be overwritten by x_k and r_k, respectively (we have labeled the x_k, r_k and α_k with an index k, for ease of discussion only). The computational cost in the $(k-1)$-th step and the memory requirement between the $(k-1)$-th and k-th step can be summarized by:

$$\begin{array}{ll} \text{computational effort:} & 1 \text{ Mv}, \quad k \text{ dot}, \quad 2k \text{ axpy} \\ \text{memory requirement:} & k \times \text{“}c_j\text{”}, \quad k \times \text{“}u_j\text{”}, \quad x_k, \quad r_k \end{array} \tag{12.26}$$

where 1 **Mv** is the time needed to solve an equation $\tilde{u} = K^{-1}r$ and to evaluate the matrix-vector multiplication $A\tilde{u}$, 1 **dot** the time needed for one inner product and 1 **axpy** for one vector update (of the form $\alpha x + y$).

As an alternative, one can compute c_k by evaluating Au_k. This requires an additional matrix-vector multiplication, while evaluating $\tilde{c} - P_k\tilde{c}$ requires k axpy. For expensive Mv the alternative may be unattractive.

A cheap variant of GCR. We will show that there is no need to compute the u_j nor r_j nor to update x_j in each step. The computation of the approximate solution x_k can be postponed until the norm of the residual is small enough. As will turn out, as far as it concerns large n-vectors, this saves on memory ($\approx 50\%$) and on computational cost ($\approx 33\%$): in the k-th step we only have to store x_0, r_0, r_k and $k \times$ “c_j” and instead of $2k$ axpy, we only need k axpy.

We employ the matrix representation of GCR in (12.25).
If GCR does not break down then $\mathcal{K}_k(\mathbb{A};r_0) = \{R_k\vec{\gamma}\,\big|\,\vec{\gamma}\in\mathbb{R}^k\}$.
Therefore, if $\vec{\gamma} = (\gamma_0,\ldots,\gamma_{k-1})^T$ solves the minimization problem

$$\|r_k\| = \min\Big\{\|r_0 - \mathbb{A}u\|\,\Big|\,u\in\mathcal{K}_k(\mathbb{A};r_0)\Big\} = \min_{\vec{\gamma}\in\mathbb{R}^k}\|r_0 - \mathbb{A}R_k\vec{\gamma}\| \tag{12.27}$$

then $x_k = x_0 + K^{-1}R_k\vec{\gamma}$. Since $\mathbb{A}R_k = C_kT_k$ and $C_k^TC_k = I$ (cf. (12.25)), the solution of this least-square problem (12.27) is given by $\vec{\gamma} = T_k^{-1}C_k^Tr_0$. In practice, we are interested in the case where $k \ll n$. Hence, since T_k is upper-triangular, $\vec{\gamma}$ can be computed cheaply from r_0 and $c_0,\ldots,c_{k-1}$. Since we wish to save on memory, we don't want to store all the r_j's. Fortunately, the r_j, needed in the computation of x_k from $R_k\vec{\gamma}$ can easily be reconstructed from r_0, the c_j's and the α_j's, because: $r_j = r_{j-1} - \alpha_{j-1}c_{j-1} = \ldots = r_0 - \alpha_0c_0 - \ldots - \alpha_{j-1}c_{j-1}$. Therefore, if W_k is the $k\times k$ upper-triangular matrix of which all upper-triangular entries are 1 we have that

$$K(x_k - x_0) = (\gamma_0 + \ldots + \gamma_{k-1})r_0 - C_{k-1}D_{k-1}W_{k-1}(\gamma_1,\ldots,\gamma_{k-1})^T. \tag{12.28}$$

This variant of GCR is comparable to the well-known GMRES algorithm [28]: both are algorithmical representations of the MR approach requiring the same amount of memory and the same number of large vector operations in each step. The algorithms are based on the same principle: they both produce only *one* sequence of orthogonal vectors (the sequence of the GCR variant spans $\mathcal{K}_k(\mathbb{A};\mathbb{A}r_0)$, while the one of GMRES spans $\mathcal{K}_k(\mathbb{A};r_0)$) and they do not assemble x_k as long as the norm of the residual is not small enough. In contrast to GCR and its variant, GMRES does not break down. However, one can easily formulate a cheap variant of ORTHODIR [34] that is as robust and as cheap as GMRES. As a side-product, GCR computes all approximations x_k. If one wishes to track the progress of the computations by monitoring the approximations one can not employ the cheap variant of GCR (nor GMRES).

Discussion. The obvious disadvantage of GCR as well as the other methods like GMRES, FOM, ORTHODIR, etc., is that the amount of storage grows (all c_j's have to be stored), and the work per iteration step grows also (all the orthogonalization work increases per step). Therefore, the price for the optimality of GCR (it delivers the minimum residual over all Krylov subspace methods) may be too high in practical situations. An obvious way out is to restart after each m iteration steps. The method is then referred to as GCR(m). GMRES(m) is restarted GMRES, etc.. Restarting is the most popular strategy in practice, but, unfortunately, the convergence behavior may depend strongly on the choice of m. An unlucky choice may result in slow convergence or even in no convergence at all: by a new start the information that accelerates the speed of convergence (see 12.4.1) is lost and has to be collected again. In contrast to GMRES (or to the cheap variant of GCR), GCR allows to follow other strategies, for instance to maintain only the last m vectors u_j (and c_j; truncation) or some appropriate linear combination of u_j's (assemblage) for the search space. Although such a strategy leads to a better convergence behavior, also then convergence is not guaranteed. In general one may say that the selection of a proper value for m and an good assemblage strategy requires experience and skill. GMRESR, to be discussed in 12.6, and other adaptations of GCR (as in [30, 34]) use even more information from previous steps, thus retaining better the convergence properties of the MR approach.

Finally, note that GCR(1) is precisely the LMR algorithm. So, any GCR(m) can be viewed as an improvement of the LMR algorithm.

12.4.3 Techniques to compute eigenvalues

Consider the matrix representation of GCR in (12.25).
By (12.25), we have that $\mathbb{A}C_k = \mathbb{A}R_{k+1}S_kD_k^{-1} = C_{k+1}\tilde{B}_k$, where $\tilde{B}_k = T_{k+1}S_kD_k^{-1}$ is $(k+1)\times k$ Hessenberg. If B_k is the upper $k\times k$ block of $\tilde{B}_k$ then B_k is Hessenberg as well and, with respect to the orthonormal basis of normalized c_j, B_k is the matrix of $\mathbb{A}$ projected on the Krylov subspace $\mathcal{K}_k(\mathbb{A};\mathbb{A}r_0)$.

Clearly, the Hessenberg matrix B_k can easily be computed from the projection coefficients and the scalars α_j and γ_j in GCR. Since B_k is Hessenberg, its eigenvalues can cheaply be computed for instance by the QR-algorithm (see [13]) if $k \ll n$. These eigenvalues are k-th order Ritz values of $\mathbb{A}$ (cf. 12.4.1). Hence, as a side product, GCR (and other methods) allows a cheap computation of Ritz values and this provides information about the spectrum of $\mathbb{A}$.

This information is useful if one wishes to assess the effect of the preconditioner. The insight gained in this way may press for another preconditioner or may help to select the most appropriate algorithm. It also may help to construct a better preconditioner (see, for instance, [5] and [16]). Moreover, in a context of discretized PDE's, it also may give information on the stability of the discretization method.

12.5 Conjugate Gradients for Normal Equations

Although in the context of advection-diffusion equations, A hardly ever is symmetric, we can easily find a preconditioner for which the preconditioned matrix $\mathbb{A}$ is symmetric: $K^{-1} = A^T$. The symmetry can be used to improve greatly the efficiency of the separate steps of CGR. The resulting algorithm CRNR and the related CGNR and LSQR [24] are robust: they do not break down. Although, the choice $K^{-1} = A^T$ leads to cheap iteration steps it may slow down the speed of convergence dramatically. For this reason, in spite of the cheap steps, these algorithms are not attractive. We include them in our discussion since they are useful to improve the robustness of other algorithms by mixing them with the others.

CR and CG. Consider GCR and assume for the moment that $\mathbb{A}$ is symmetric. Since

$$\begin{aligned} r_k - \text{span}\,(c_0,\ldots,c_{k-1}) &= \mathbb{A}\,\mathcal{K}_k(\mathbb{A};r_0) \supset \mathbb{A}\,(\mathbb{A}\mathcal{K}_{k-1}(\mathbb{A};r_0)) \\ &= \mathbb{A}\,\text{span}\,(c_0,\ldots,c_{k-2}), \end{aligned}$$

we see that $(r_k, \mathbb{A}c_j) = 0$ if $j < k-1$. Hence, by symmetry, $(\mathbb{A}r_k, c_j) = 0$ if $j < k-1$. Therefore, $u_k = K^{-1}r_k - \beta_{k-1}u_{k-1}$ and $c_k = \mathbb{A}r_k - \beta_{k-1}c_{k-1}$, where $\beta_k = (\mathbb{A}r_k, c_{k-1})/(c_{k-1}, c_{k-1})$ (cf. (12.20)). Apparently, in the symmetric case, in the GCR algorithm,

$$\texttt{compute by mod.G-S}\ \ u_k = \tilde{u} - Q_k\tilde{c},\ \ c_k = \tilde{c} - P_k\tilde{c},$$

may be replaced by

$$u_k = \tilde{u} - \beta_k u_{k-1}, \quad c_k = \tilde{c} - \beta_k c_{k-1} \quad \text{with} \quad \beta_k \equiv (\tilde{c}, c_{k-1})/\gamma_{k-1}. \tag{12.29}$$

Moreover, using the symmetry and orthogonality relations, one can show that

$$\beta_k = -\frac{\rho_k}{\rho_{k-1}} \ \text{ and } \ \alpha_k = \frac{\rho_k}{\gamma_k} \ \text{ where } \ \rho_k = (\tilde{c}, r_k), \tag{12.30}$$

choose x_0,
$k = -1, \quad r_0 = b - Ax_0, \quad u_{-1} = 0, \quad \rho_{-1} = 1,$
repeat until $\|r_{k+1}\|$ is small enough:

$$\left\{ \begin{array}{l} k := k+1, \\ \tilde{u} := A^T r_k, \\ \rho_k = \|r_k\|^2, \ \ \beta_k = -\rho_k/\rho_{k-1}, \\ u_k = \tilde{u} - \beta_k u_{k-1}, \ \ c_k = Au_k, \\ \gamma_k = \|u_k\|^2, \ \ \alpha_k = \rho_k/\gamma_k, \\ x_{k+1} = x_k + \alpha_k u_k, \ \ r_{k+1} = r_k - \alpha_k c_k \end{array} \right.$$

Algorithm 12.3: The CGNR algorithm.

thus saving another dot. This results in the **CR** (**Conjugate Residuals**) algorithm.

CR is cheap in memory and computational costs: each step requires 1 Mv, 2 dot and 4 axpy. However, this efficient form of GCR for symmetric problems is sensitive to errors due to finite precision arithmetic. In the GCR algorithm these errors also are projected orthogonally to the previously constructed set of vectors c_j while CR projects orthogonally only on one vector. Consequently, the orthogonality of the sequence $c_0, \ldots, c_k$ may be (and will be) lost if k increases. As a consequence, in actual computation, CR may show no accelerations (cf. 12.4.1). In spite of this instability and possible lack of acceleration the speed of convergence is relatively high. One can show (using Paige's theory in [22]) that, in spite of evaluation errors, for the residuals r_k computed by the CG algorithm we have

$$\frac{\|r_k\|}{\|r_0\|} \lesssim \left(\frac{1 - \sqrt{\frac{\lambda_1}{\lambda_n}}}{1 + \sqrt{\frac{\lambda_1}{\lambda_n}}} \right)^k \approx \exp\left(-2k\sqrt{\frac{\lambda_1}{\lambda_n}} \right) \quad \text{if} \quad \sigma(\mathbb{A}) \subset [\lambda_1, \lambda_n]. \tag{12.31}$$

If $\mathbb{A}$ is symmetric and positive definite then CR does not break down and $[x, y] := (\mathbb{A}^{-1}x, y)$ defines an inner product. If we substitute the definition of the new inner product in the CR algorithm formulated with respect to the new inner product we find one form of the well-known **CG** (**Conjugate Gradients**) algorithm [15]. Now $\rho_k = (\mathbb{A}^{-1}\tilde{c}, r_k) = \|r_k\|^2$ and tracking the size of the residuals does not require an additional dot. Moreover, $\gamma_k = [c_k, c_k] = [\tilde{c}, c_k] = (r_k, c_k)$. If we compute $c_k = Au_k$ then we do not need $\tilde{c}$ and we can save an additional axpy.

CGNR. If CG is applied to the problem with preconditioner K given by $K^{-1} = A^T$ then $\mathbb{A} = AA^T$ is symmetric positive definite and the method is called **CGNR** (**Conjugate Gradients for Normal Relations**; see Algorithm 12.3). Similarly, CRNR is preconditioned CR with preconditioner $K = (A^T)^{-1}$. CGNR is (equivalent to) unpreconditioned CG applied to the **normal equations** $A^T Ax = A^T b$.

If (r_k) is the sequence of residuals from the CGNR then one can show that[4]

$$\frac{\|r_k\|}{\|r_0\|} \leq \varepsilon_k \left(\sigma(A^T A) \right) \lesssim \left(\frac{1 - \frac{1}{\mathcal{C}(A)}}{1 + \frac{1}{\mathcal{C}(A)}} \right)^k \approx \exp\left(-2k \frac{1}{\mathcal{C}(A)} \right), \tag{12.32}$$

[4]The first estimate is sharp in some generic sense (see [14]).

where $\mathcal{C}(A)$ is the condition number of A. The upper bound in (12.32) does not involve the condition number of the eigenvector basis of A (cf. 12.4.1). One can show that the convergence of CGNR depends only on the singular values of A (that is, on the square root of the eigenvalues of A^TA) and *not* on the conditioning of the eigenvector basis of A.

Example 1 (continued 3) If $K = I$ and A is symmetric then the condition number of A is the ratio of the largest eigenvalue λ_n and the smallest eigenvalue λ_1: λ_n/λ_1. In CGNR the condition number, instead of its square root, determines the speed of convergence (cf. (12.8) and (12.32)).

In view of the above example, we expect that GCR converges much faster than CGNR specifically when $\mathcal{C}(\mathbb{A})$ is large, $\mathcal{C}_E$ is not too large, and $\mathbb{A}$ is not too far from being symmetric. A poor performance of CGNR should not be credited to the CG process. Usually a given large symmetric problem can best be solved by CG. However, our initial problem was not symmetric. We made it symmetric by multiplying by A^T, thus, risking the introduction of a matrix with an very large condition number. Still, there are situations, as the following example shows, for which CGNR performs much better than GCR.

Example 5 If A is the $n \times n$-circular matrix (i.e. $Ae_j = e_{j-1}$ for $j = 2, \ldots, n$ and $Ae_1 = e_n$) then GCR with $K = I$ needs n steps to solve the equation $Ax = e_1$ with $x_0 = 0$ (exactly, or with an accuracy $< \frac{1}{n}$) while CRNR only needs 1 step.

If A is the $2n \times 2n$-block diagonal matrix with consecutive diagonal blocks $\begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$ then GCR needs 2 steps to solve the equation $Ax = e_1$ with $x_0 = 0$ while CGNR needs $\approx n$ steps.

12.6 Nested Krylov subspace methods

The computational cost per step and the amount of storage grows in any of the algorithms of optimal Krylov subspace type. For GCR, we considered strategies to limit the costs (see the discussion in 12.4.2). In this section, we place this approach in a more general setting to find more flexible and robust algorithms. We took the ideas from [42] (see also [4, 27]).

As with GCR, suppose we have an orthogonal system $c_0, \ldots, c_{k-1}$ and for any of the c_j we also have the u_j for which $c_j = Au_j$. With P_k and Q_k as in (12.19) and (12.22) the vector r_k of the form $r_k = r_0 - \sum_{j=0}^{k-1} \gamma_j c_j$ that has minimal norm is given by $r_k = r_0 - P_k r_0$. This minimal vector is the residual of the approximation $x_k = x_0 + Q_k r_0$. By means of the stabilized recursion as in (12.18), x_k and its residual r_k can be computed efficiently. The recursion is particularly efficient if we extend the orthogonal system of c_j's recursively: then the consecutive x_k and r_k can be computed as in (12.24).

So, here as well as in the derivation of GCR, we have to answer the question how to compute the next u_k for which $c_k = Au_k$ is orthogonal to any of the c_j with $j < k$. The next c_k should also reduce the norm of the residual $r_{k+1} = r_k - \alpha_k c_k$ as much as possible. Clearly, if u solves the equation $Au = r_k$ then $c_k = Au$ gives the best reduction. Unfortunately, we can not obtain this solution u cheaply (otherwise we also would be able to solve the original equation $Ax = b$), but any good approximation $\tilde{u}$ of u may be

choose x_0,
$k = -1, \quad r_0 = b - Ax_0$,
repeat until $\|r_{k+1}\|$ is small enough:
$$\left\{ \begin{array}{l} k := k+1, \\ \text{find an approximate solution } \tilde{u} \text{ of } Au = r_k, \quad \tilde{c} := A\tilde{u}, \\ \text{compute by mod.G-S} \quad u_k = \tilde{u} - Q_k\tilde{c}, \quad c_k = \tilde{c} - P_k\tilde{c}, \\ \gamma_k = (c_k, c_k), \;\; \alpha_k = (r_k, c_k)/\gamma_k, \\ x_{k+1} = x_k + \alpha_k u_k, \;\; r_{k+1} = r_k - \alpha_k c_k \end{array} \right.$$

Algorithm 12.4: The GMRESR algorithm.

worth considering. If $u_k \equiv \tilde{u} - Q_k A\tilde{u}$ then $c_k \equiv Au_k = (I - P_k)A\tilde{u}$ is orthogonal to the previous c_j and, since $r_k = (I - P_k)r_k$ we also have

$$\|r_k - \alpha c_k\| = \|r_k - \alpha A u_k\| = \|(I - P_k)(r_k - \alpha A\tilde{u})\| \leq \|r_k - \alpha A\tilde{u}\|.$$

Hence, if $\tilde{u}$ (or $\alpha\tilde{u}$) is a good approximate solution (i.e. $\|r_k - \alpha A\tilde{u}\|$ is "small"), then αu_k is not worse and c_k has the required orthogonality property.

So, we propose Algorithm 12.4 in which we have to specify how to

"find an approximate $\tilde{u}$ to the solution u of $Au = r_k$".

Since K^{-1} approximates A^{-1} in some sense, the choice $\tilde{u} = K^{-1}r_k$ yields a reasonable approximation of the solution u of the equation $Au = r_k$. However, this choice gives GCR.

The problem $Au = r_k$ can be solved approximately also by some iterative method. The algorithm that uses l steps of GMRES for this purpose is referred to as **GMRESR(l)** [42]. The vectors c_j can be used to speed-up the "local" iterative method [34]. Obviously, one can also restart methods as GMRESR(l) after m GCR iterations steps [42] or one can maintain only m GCR vectors c_j or some linear combinations of c_j's [34].

Numerical experiments show that GMRES(lm) (or GCR(lm)) and the restarted version GMRESR(m, l) of GMRESR(l) often require the same number of Mv's to obtain an equally good approximation (i.e. with a residual of comparable size; [42]). On average per Mv, GMRESR(m, l) is cheaper than GMRES(lm) (and GCR(lm)) by a factor $\frac{2}{l} + \frac{1}{m}$ ($\frac{1}{l} + \frac{2}{m}$, respectively) with respect to the other computational costs (axpys and dots). We have a same factor in favor of GMRESR(m, l) for the amount of storage that is needed. Therefore, depending on the complexity of a matrix multiplication (on the sparseness of A and the choice for the preconditioner) GMRESR(m, l) can be very competitive. Since the optimal choice for l and m depends on the complexity of one Mv but also on the speed of convergence it is hard to make a good à priori choice.

From example 5 we learned that GCR may stagnate where CGNR performs extremely well. This experience may suggest a possibility to avoid stagnation in GCR: use Algorithm 12.4 and (once in a while) a few steps CGNR to solve approximately $Au = r_k$.

12.7 Quasi-optimal Krylov subspace methods

For a number of linear equations, the methods discussed so far may still be unattractive because of growing costs per iteration step (as GMRES) or slow convergence (as GCR(4) and CGNR). We now will discuss methods that use only a few vectors from previous steps and that still are the best in some adapted sense.

If $c_0, c_1, \ldots$ is a basis of the Krylov space of $\mathbb{A}$ generated by c_0 then matrix B of $\mathbb{A}$ with respect to this basis is upper Hessenberg and $\mathbb{A}C = CB$, where C is the $n \times n$-matrix with $c_0, c_1, \ldots$ in its consecutive columns. If the c_j form an orthogonal system then we are back in the situation as discussed before. Now, we hope to find a (non-orthogonal) basis of c_j's by which we can project efficiently on the Krylov subspace $\mathcal{K}_k(\mathbb{A}; \mathbb{A}r_0)$ (as in GCR or on $\mathcal{K}_k(\mathbb{A}; r_0)$ as in GMRES): our projections should involve short recursions as in (12.29). In methods as GCR, the projection coefficients appear in the Hessenberg matrix (see 12.4.3). Only a few of these coefficients are non-zero in each step if and only if the Hessenberg matrix B has only a few non-zero coefficients on each row. Therefore, we may hope to find short recursions if, for instance, B has only a few non-zero diagonals. For clarity of arguments, we consider the case where B is tri-diagonal, or equivalently, where B^T is upper Hessenberg as well. Since $\mathbb{A}^T(C^{-1})^T = (C^{-1})^T B^T$ we see that B^T is upper Hessenberg if and only if the columns of $(C^{-1})^T$ form a basis of the Krylov space of $\mathbb{A}^T$ generated by the first column of $(C^{-1})^T$.

Since $C^{-1}C = I$, the columns $\tilde{c}_0, \tilde{c}_1, \ldots$ of C^{-1} are characterized by bi-orthonormality with respect to the sequence $c_0, c_1, \ldots$, i.e.: $(c_j, \tilde{c}_i) = 0$ (bi-orthogonal) and $(c_j, \tilde{c}_j) = 1$ (bi-normal) $(i \neq j)$. Therefore, in order to obtain a basis for the Krylov space of $\mathbb{A}$ generated by c_0 for which the matrix B of $\mathbb{A}$ is tri-diagonal, we have to construct a pair of bi-orthonormal sequences that form a basis of the Krylov space of $\mathbb{A}$ and of $\mathbb{A}^T$, respectively. We will perform this construction by an analog of Gram-Schmidt, by the **Lanczos bi-orthogonalization** process starting with c_0 and some vector $\tilde{c}_0$. The construction may break down if, for the choice of the generator $\tilde{c}_0$, the Hessenberg matrix can not be tri-diagonal. In such a situation one can start again with another $\tilde{c}_0$ or one can try to cure the breakdown by aiming at, for instance, a locally five-diagonal Hessenberg matrix (by **looking ahead**, [12]): here, for simplicitly, we will assume that the processes do not break down (in finite precision arithmetic this is a rare event, indeed).

12.7.1 Convergence

Bi-orthogonal basis of Krylov spaces give (non-orthogonal) projections P_k onto the Krylov subspace $\mathcal{K}_k(\mathbb{A}; c_0)$ and lead to residuals r_k given by $r_k = (I - P_k)r_0$. These residuals are optimal in the sense that $\|C^{-1}r_k\| \leq \|C^{-1}q_k(\mathbb{A})r_0\|$ for each polynomial $q_k \in \mathcal{P}_k^1$ and (cf. (12.12))

$$\frac{\|r_k\|}{\|r_0\|} \leq \mathcal{C}(C) \min_{q \in \mathcal{P}_k^1} \|q(\mathbb{A})\|. \tag{12.33}$$

If $c_0 = r_0$ (as in QMR) then $\mathcal{C}(C)$ may be replaced by the condition number of the matrix with columns $c_0, \ldots, c_k$ which is less than $k \max_{j \leq k} \|c_j\| \max_{j \leq k} \|\tilde{c}_j\|$. One can cheaply keep track of this quantity.

In exact arithmetic, these methods will exhibit accelerations as is the case for the optimal Krylov subspace methods (cf. 12.4.1). Although, here also, the accelerations depend on spectral properties of $\mathbb{A}$, the situation is obscured by the non-orthogonal basis transformation C.

When employing Lanczos bi-orthogonalization, C, and therefore the speed of convergence, depends on the choice of $\tilde{c}_0$ ($\tilde{r}_0$ in the algorithms below). Unfortunately, there is no good strategy for selecting $\tilde{c}_0$. One often chooses $\tilde{c}_0 = c_0$ and restarts with a new choice in the rare case of breakdown. Furthermore, the bi-orthogonality (cf. 12.5) will get lost in the course of the computation due to evaluation errors. This slows down the speed of convergence and may even lead to divergence. Nevertheless, for a large class

of problems, these methods appear to do well. Compared to GCR, say, each step is so cheap that this often compensates less optimal convergence.

12.7.2 Lanczos bi-orthogonalization

If $c'_0, c'_1, \ldots$ and $\tilde{c}'_0, \tilde{c}'_1, \ldots$ are basis of the Krylov space of $\mathbb{A}$ and of $\mathbb{A}^T$, respectively, and $c_0, \ldots, c_{k-1}$ and $\tilde{c}_0, \ldots, \tilde{c}_{k-1}$ are constructed from them by bi-orthogonalization, then we construct our next c_k and $\tilde{c}_k$ by means of the map $\tilde{P}_k$ given by

$$\tilde{P}_k \equiv \sum_{j=0}^{k-1} \frac{c_j \tilde{c}_j^T}{\tilde{c}_j^T c_j}.$$

$\tilde{P}_k$ is a projection (non-orthogonal, unless $c_j = \tilde{c}_j$ for all j) onto the span of $c_0, \ldots, c_{k-1}$, hence onto $\mathcal{K}_k(\mathbb{A}; c_0)$. Moreover, since $(I - \tilde{P}_k)c_j = 0 \quad (j < k)$, we see that $(I - \tilde{P}_k^T)y - c_j$ for any y and $j < k$. Similarly, $(I - \tilde{P}_k)y - \tilde{c}_j$. Hence, $c_k \equiv (I - P_k)c'_k$ and $\tilde{c}_k \equiv (I - P_k^T)\tilde{c}'_k$ are our next bi-orthogonal vectors.

The sequences $c_0, c_1, \ldots$ and $\tilde{c}_0/\tilde{c}_0^T c_0, \tilde{c}_1/\tilde{c}_1^T c_1, \ldots$ are bi-orthonormal.

If, for all j, $c'_j = \mathbb{A}c_{j-1}$ and $\tilde{c}'_j = \mathbb{A}^T \tilde{c}_{j-1}$ then the coefficients in the projection of the c'_j constitute the matrix B of $\mathbb{A}$ with respect to $c_0, c_1, \ldots$. Since this **Lanczos matrix** is tri-diagonal, the projections are given by

$$c_k = \mathbb{A}c_{k-1} - \sum_{j=1}^{2} \tau_{k-j\,k} c_{k-j}, \quad \text{and} \quad \tilde{c}_k = \mathbb{A}^T \tilde{c}_{k-1} - \sum_{j=1}^{2} \tilde{\tau}_{k-j\,k} \tilde{c}_{k-j} \tag{12.34}$$

for some scalars τ_{jk} and $\tilde{\tau}_{jk}$ $(\tau_{jk} = (\mathbb{A}c_{k-1}, \tilde{c}_j)/(c_j, \tilde{c}_j), \ldots)$.

Lanczos [17] used this bi-orthogonalization procedure for finding eigenvalues of $\mathbb{A}$. After k steps, we have the $k \times k$ upper block of B. The eigenvalues of this block can easily be computed for the k's of interest ($k \ll n$). They also approximate the eigenvalues of $\mathbb{A}$, the extremal eigenvalues first, and provide useful information (cf. 12.4.3 and 12.4.1).

We apply these observations in a GCR type of algorithm finding Bi-CG. In a GMRES type of algorithm they lead to QMR [12]. For QMR, we refer to the literature.

12.7.3 Bi-conjugate gradient

We choose some x_0 and $\tilde{s}_0$ and we take $r_0 \equiv b - Ax_0$, $c_0 \equiv \mathbb{A}r_0$ and $\tilde{u}_0 \equiv \mathbb{A}^T \tilde{s}_0$. Suppose $u_0, \ldots, u_{k-1}$ are such that $c_0, \ldots, c_{k-1}$, with $c_j = Au_j$, form a basis of $\mathcal{K}_k(\mathbb{A}; \mathbb{A}r_0)$ that is bi-orthogonal with respect to a basis $\tilde{u}_0, \ldots, \tilde{u}_{k-1}$ of $\mathcal{K}_k(\mathbb{A}^T; \tilde{u}_0)$. Consider the maps $\tilde{P}_k$ and $\tilde{Q}_k$ given by

$$\tilde{P}_k(y) \equiv \sum_{j=0}^{k-1} \frac{c_j \tilde{u}_j^T}{\tilde{u}_j^T c_j} \quad \text{and} \quad \tilde{Q}_k(y) \equiv \sum_{j=0}^{k-1} \frac{u_j \tilde{u}_j^T}{\tilde{u}_j^T c_j}. \tag{12.35}$$

$\tilde{P}_k$ is a projection that bi-orthogonalizes and $\tilde{Q}_k$ is the original of $\tilde{P}_k$ under A: $\tilde{P}_k = A\tilde{Q}_k$. Obviously, we can now copy the construction of the GCR algorithm by choosing $c'_k = \mathbb{A}r_k$ and $\tilde{u}'_k = \mathbb{A}^T \tilde{s}_k$, where $\tilde{s}_k = (I - \tilde{P}_k^T)\tilde{s}_0$. This leads to an algorithm that could be called Bi-CR (Bi-Conjugate Residuals). As in GCR, $r_k = r_{k-1} - \alpha_{k-1} c_{k-1}$ and $\tilde{s}_k = \tilde{s}_{k-1} - \tilde{\alpha}_{k-1} \tilde{u}_{k-1}$ for some appropriate scalars α_{k-1} and $\tilde{\alpha}_{k-1}$. As expected, we also have short recursions for c_k and $\tilde{u}_k$. We even have

$$c_k = \mathbb{A}r_k - \beta_k c_{k-1}, \quad \tilde{u}_k = \mathbb{A}^T \tilde{s}_k - \tilde{\beta}_k \tilde{u}_{k-1} \quad \text{with} \quad \tilde{\beta}_k = \beta_k \equiv \frac{(\mathbb{A}r_k, \tilde{u}_{k-1})}{(c_{k-1}, \tilde{u}_{k-1})},$$

choose x_0 and $\tilde{r}_0$,
$k = -1, \quad r_0 = b - Ax_0, \quad u_{-1} = 0, \quad \rho_{-1} = 1,$
repeat until $\|r_{k+1}\|$ is small enough:

$$\left\{ \begin{array}{ll} k := k+1, & \\ \text{solve } u' \text{ from } Ku' := r_k, & \tilde{c}' = \mathbb{A}^T \tilde{r}_k \\ \rho_k = (r_k, \tilde{r}_k), \ \beta_k = -\rho_k / \rho_{k-1}, & \\ u_k = u' - \beta_k u_{k-1}, \quad c_k = Au_k, & \tilde{c}_k = \tilde{c}' - \beta_k \tilde{c}_{k-1} \\ \gamma_k = (c_k, \tilde{r}_k), \ \alpha_k = \rho_k / \gamma_k, & \\ x_{k+1} = x_k + \alpha_k u_k, \quad r_{k+1} = r_k - \alpha_k c_k, & \tilde{r}_{k+1} = \tilde{r}_k - \alpha_k \tilde{c}_k \end{array} \right.$$

Algorithm 12.5: The Bi-CG algorithm.

$$r_{k+1} = r_k - \alpha_k c_k, \quad \tilde{s}_{k+1} = \tilde{s}_k - \tilde{\alpha}_k \tilde{u}_k \quad \text{with} \quad \tilde{\alpha}_k = \alpha_k \equiv \frac{(r_k, \tilde{u}_k)}{(c_k, \tilde{u}_k)}.$$

To compute the scalars β_k and α_k, neither the $\tilde{s}_j$, nor the $\tilde{u}_j$, nor the $c'_j = \mathbb{A} r_j$ are needed: since, we can compute c_k from $c_k = Au_k$ and, for instance,

$$(\mathbb{A} r_k, \tilde{u}_{k-1}) = (r_k, \mathbb{A}^T \tilde{u}_{k-1}), \qquad (c_{k-1}, \tilde{u}_{k-1}) = (r_{k-1}, \mathbb{A}^T \tilde{u}_{k-1}).$$

We can compute $\mathbb{A}^T \tilde{u}_j$ from $\mathbb{A}^T \tilde{u}_k = \mathbb{A}^T \mathbb{A}^T \tilde{s}_k - \tilde{\beta}_k \mathbb{A}^T \tilde{u}_{k-1}$ and $\mathbb{A}^T \tilde{s}_{k+1} = \mathbb{A}^T \tilde{s}_k - \tilde{\alpha}_k \mathbb{A}^T \tilde{u}_k$. This saves one axpy as compared to Bi-CR.

We write $\tilde{c}_j \equiv \mathbb{A}^T \tilde{u}_j$ and $\tilde{r}_j \equiv \mathbb{A}^T \tilde{s}_j$. In particular, we have that $\tilde{u}_0 = \tilde{r}_0$. Since we were free to select any initial $\tilde{r}_0$ we as may well select $\tilde{r}_0$ instead of $\tilde{s}_0$.

Now, note that

$$c_k, \, r_k = (I - \tilde{P}_k) r_0 - \mathcal{K}_k(\mathbb{A}^T; \tilde{r}_0) = \text{span}\,(\tilde{u}_0, \ldots, \tilde{u}_{k-1}) \ni \tilde{r}_j \quad (j < k). \tag{12.36}$$

Therefore, for the scalar β_k we have that

$$\beta_k = \frac{(r_k, \tilde{c}_{k-1})}{(r_{k-1}, \tilde{c}_{k-1})} = \frac{(r_k, \tilde{r}_{k-1} - \tilde{r}_k)}{(r_{k-1}, \tilde{r}_{k-1} - \tilde{r}_k)} = -\frac{(r_k, \tilde{r}_k)}{(r_{k-1}, \tilde{r}_{k-1})}.$$

For the scalar α_k the following relations are useful. They follow from applying (12.36).

$$(r_k, \tilde{u}_k) = (r_k, \tilde{r}_k) \quad \text{and} \quad (r_k, \tilde{c}_k) = (c_k, \tilde{r}_k),$$

and this leads to **Bi-CG** (**Bi-Conjugate Gradients** [17, 10]) in Algorithm 12.5.

Each Bi-CG step is cheap. Estimate (12.33) may give some theoretical information on the speed of convergence. However, we don't have à priori information on $\mathcal{C}(C)$ and since we wish to keep k relatively small, we will never have this information.

The relation between Bi-CG and the Lanczos method in 12.7.2 may be exploited (as in GCR, 12.4.3) to recover the Lanczos tri-diagonal matrix from the Bi-CG coefficients α_k and β_k and to find approximations for the extremal eigenvalues of $\mathbb{A}$ (see 12.7.2).

12.7.4 Hybrid Bi-CG methods

Although the computational costs of Bi-CG is low in terms of dots and axpys, each step requires two matrix multiplications, which is double the costs of GCR. Moreover, one Mv involves the multiplication by the transpose of $\mathbb{A}$ which is not always easily available.

We will discuss modifications of Bi-CG that improve the speed of convergence and that use multiplications by $\mathbb{A}$ only; two multiplications per step.

We are not really interested in the Bi-CG vectors $\tilde{c}_k$ and $\tilde{r}_k$. Actually, we only need $\tilde{r}_k$ to compute β_k and α_k through the scalars ρ'_k and γ_k. However, we can compute these by means of any vector $\tilde{s}_k$ of the form $\tilde{s}_k = q_k(\mathbb{A}^T)\tilde{r}_0$ where q_k is some polynomial in $\mathcal{P}_k$ of which the leading coefficient is non-trivial and known.
To prove this we consider a $q_k \in \mathcal{P}_k$ with non-zero leading coefficient σ_k, that is $q_k(t) - \sigma_k t^k$ is a polynomial of degree $k-1$. $\tilde{r}_k = p_k(\mathbb{A}^T)\tilde{r}_0$ for some $p_k \in \mathcal{P}_k^1$ and $p_k(t) - \tau_k t^k$ is of degree $k-1$ for $\tau_k = (-\alpha_{k-1})(-\alpha_{k-1})\ldots(-\alpha_0)$. Hence, both the vectors $\tilde{r}_k - \tau_k(\mathbb{A}^T)^k\tilde{r}_0$ and $\tilde{s}_k - \sigma_k(\mathbb{A}^T)^k\tilde{r}_0$ belong to $\mathcal{K}_{k-1}(\mathbb{A}^T;\tilde{r}_0)$. Since both the vectors r_k and c_k are orthogonal to this space (see (12.36)), we see that

$$\rho_k \equiv (r_k, \tilde{r}_k) = \frac{\tau_k}{\sigma_k}\rho'_k \quad \text{with} \quad \rho'_k \equiv (r_k, \tilde{s}_k)$$
$$\gamma_k \equiv (c_k, \tilde{r}_k) = \frac{\tau_k}{\sigma_k}\gamma'_k \quad \text{with} \quad \gamma'_k \equiv (c_k, \tilde{s}_k).$$

Hence

$$\beta_k = -\frac{\tau_k\sigma_{k-1}}{\sigma_k\tau_{k-1}}\frac{\rho'_k}{\rho'_{k-1}} \quad \text{and} \quad \alpha_k = \frac{\rho'_k}{\gamma'_k}. \tag{12.37}$$

This observation does not improve the algorithm significantly: in order to compute $\tilde{s}_k$ recursively, in each step, we will still have to multiply by $\mathbb{A}^T$. We might save two vector updates, by taking $\tilde{s}_k = \mathbb{A}^T\tilde{s}_{k-1}$, but this approach may affect the stability seriously. However, one might try to invest the effort of computing $\tilde{s}_k$ in an attempt to force an additional reduction of r_k. For this purpose, the following observation is important

$$\rho'_k = (r_k, q_k(\mathbb{A}^T)\tilde{r}_0) = (q_k(\mathbb{A})r_k, \tilde{r}_0) \quad \text{and} \quad \gamma'_k = (c_k, q_k(\mathbb{A}^T)\tilde{r}_0) = (q_k(\mathbb{A})c_k, \tilde{r}_0). \tag{12.38}$$

In the ideal case the operator $p_k(\mathbb{A})$ reduces r_0. One may try to select q_k such that $\mathbb{Q}_k \equiv q_k(\mathbb{A})$ again reduces r_k as much as possible. In such a case it would be an advantage to avoid computation of r_k and u_k, and one might try to compute immediately $\mathbf{r}_k \equiv \mathbb{Q}_k r_k$, the associated search direction $\mathbf{u}_k$ and approximation $\mathbf{x}_k$ by appropriate recursions. As (12.37) and (12.38) show, one can use these vectors to compute the Bi-CG scalars β_k and α_k. If the polynomials q_k are related by simple recursions, these recursions can be used to compute efficiently the iterates $\mathbf{r}_k$, $\mathbf{u}_k$ and $\mathbf{x}_k$ without computing r_k and u_k explicitly. Therefore, the computational effort of the Bi-CG algorithm to build the shadow Krylov subspace can also be used to obtain an additional reduction (by $\mathbb{Q}_k$) for the Bi-CG residual r_k. Since r_{2k} would have been constructed from the (quasi-) best polynomial in $\mathcal{P}_{2k}^1$, we may not expect that $\|\mathbf{r}_k\| \ll \|r_{2k}\|$ (in exact arithmetic): which says that a method based on $\mathbb{Q}_k$ can only converge twice as fast (in terms of costs). Since a Bi-CG step involves two Mv's it only makes sense to compute $\mathbf{r}_k$ instead of r_k if we can obtain $\mathbf{r}_k$ from $\mathbf{r}_{k-1}$ by 4 Mv at most and a few vector updates and if we can update simultaneously the corresponding approximation $\mathbf{x}_k$, where $\mathbf{r}_k = b - A\mathbf{x}_k$. This has been realized in CG-S, Bi-CGSTAB and Bi-CGstab(l) (these algorithms require 2 Mv's per step).

We give details on Bi-CGSTAB: it is a very attractive algorithm that can easily be presented now.
For a sequence of scalars ω_k we consider the polynomials q_k given by ($q_0 \equiv 1$)

$$q_k(t) = (1 - \omega_{k-1}t)q_{k-1}(t) \quad (t \in \mathbb{R}) \quad \text{for all} \quad k. \tag{12.39}$$

choose $\mathbf{x}_0$ and $\tilde{r}_0$,
$k = -1$, $\mathbf{r}_0 = b - A\mathbf{x}_0$, $\mathbf{u}_{-1} = \mathbf{c}_{-1} = 0$, $\alpha_{-1} = 0$, $\omega_{-1} = \rho'_{-1} = 1$,
repeat until $\|\mathbf{r}_{k+1}\|$ is small enough:

$$\begin{cases} k := k+1, \\ \rho'_k = (\mathbf{r}_k, \tilde{r}_0), \quad \beta_k = -(\alpha_{k-1}/\omega_{k-1})(\rho'_k/\rho'_{k-1}), \\ \mathbf{u}_k = -\beta_k \mathbf{u}_{k-1} + K^{-1}(\mathbf{r}_k + \beta_k \omega_{k-1} \mathbf{c}_{k-1}), \quad \mathbf{c}_k = A\mathbf{u}_k, \\ \gamma'_k = (\mathbf{c}_k, \tilde{r}_0), \quad \alpha_k = \rho'_k/\gamma'_k, \\ \mathbf{r}_{k+\frac{1}{2}} = \mathbf{r}_k - \alpha_k \mathbf{c}_k, \quad \mathbf{u}_{k+\frac{1}{2}} = K^{-1}\mathbf{r}_{k+\frac{1}{2}}, \quad \mathbf{c}_{k+\frac{1}{2}} = A\mathbf{u}_{k+\frac{1}{2}}, \\ \omega_k = (\mathbf{r}_{k+\frac{1}{2}}, \mathbf{c}_{k+\frac{1}{2}})/(\mathbf{c}_{k+\frac{1}{2}}, \mathbf{c}_{k+\frac{1}{2}}), \\ \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{u}_k + \omega_k \mathbf{u}_{k+\frac{1}{2}}, \quad \mathbf{r}_{k+1} = \mathbf{r}_{k+\frac{1}{2}} - \omega_k \mathbf{c}_{k+\frac{1}{2}} \end{cases}$$

Algorithm 12.6: The Bi-CGSTAB algorithm.

Since $\mathbb{A}\mathbb{Q}_k K u_k = \mathbb{Q}_k c_k$, the definition $\mathbf{u}_k \equiv K^{-1}\mathbb{Q}_k K u_k$ yields $A\mathbf{u}_k = \mathbf{c}_k$. A combination of the relation $\mathbb{Q}_k = \mathbb{Q}_{k-1} - \omega_{k-1}\mathbb{A}\mathbb{Q}_{k-1}$ (for k and $k+1$; from (12.39)) with the Bi-CG relations $Ku_k = r_k - \beta_k K u_{k-1}$ and $r_{k+1} = r_k - \alpha_k c_k$ gives

$$\begin{aligned} K\mathbf{u}_k = \mathbb{Q}_k K u_k &= \mathbb{Q}_k r_k - \beta_k \mathbb{Q}_{k-1} K u_{k-1} + \beta_k \omega_{k-1} \mathbb{A}\mathbb{Q}_{k-1} K u_{k-1} \\ &= \mathbf{r}_k - \beta_k K \mathbf{u}_{k-1} + \beta_k \omega_{k-1} \mathbf{c}_{k-1} \\ \mathbf{r}_{k+1} = Q_{k+1} r_{k+1} &= \mathbb{Q}_k r_k - \alpha_k \mathbb{Q}_k c_k - \omega_k \mathbb{A}(\mathbb{Q}_k r_k - \alpha_k \mathbb{Q}_k c_k) \\ &= \mathbf{r}_k - \alpha_k \mathbf{c}_k - \omega_k \mathbb{A}(\mathbf{r}_k - \alpha_k \mathbf{c}_k). \end{aligned}$$

Since the updates $\mathbf{c}_k$ and $\mathbb{A}(\mathbf{r}_k - \alpha_k \mathbf{c}_k)$ of $\mathbf{r}_k$ have an original $\mathbf{u}_k$ and $K^{-1}(\mathbf{r}_k - \alpha_k \mathbf{c}_k)$, respectively, under A, it is clear how to update $\mathbf{x}_k$. Leaves us to specify how to choose the ω_k. If we minimize $\|\mathbf{r}_{k+1}\|$ as in LMR then we have **Bi-CGSTAB** (the stabilized version of CG-S) [40] in Algorithm 12.6.

Each step of the algorithm requires two multiplications by $\mathbb{A}$ and no multiplication by $\mathbb{A}^T$. Apart from these Mv's, each step is only slightly more expensive than a Bi-CG step. Moreover, Bi-CGSTAB produces a residual $\mathbf{r}_k$ in a Krylov subspace of double dimension and it often finds a very small residual in that larger space: as we learned from experiments, $\|\mathbf{r}_k\| \ll \|r_k\|$ for many problems.

Similarly, one can deduce algorithms for other choices of the polynomials q_k. If p_k is the Bi-CG polynomial then the choice $q_k = p_k$ yields **CG-S** (**Conjugate Gradient-Squared**) [33]. Bi-CG gives simple recurrence relations for the p_k leading to simple recurrence relations for CG-S. $p_k(\mathbb{A})$ is the (quasi-)best reductor for r_0 of the given form. In cases where $p_k(\mathbb{A})$ also reduces r_k CG-S is an attractive method. But, since it is not constructed for r_k, we may not expect that it reduces r_k equally well. As experiments show there are situations in which $p_k(\mathbb{A})$ even amplifies r_k. This causes irregular convergence or even divergence and makes the method more sensitive to evaluation error [39] (if $\|\mathbf{r}_{k+1}\|$ is small while $\|\mathbf{r}_k\|$ is large then $\mathbf{r}_{k+1}$ must have been the difference of large vectors. Such a situation is a source of large evaluation errors). Bi-CGSTAB converges more smoothly. Consequently, it is less sensitive to evaluation errors and converges faster than Bi-CG and CG-S. Actually, Bi-CGSTAB was designed to "smooth" the erratic convergence behavior of CG-S.

BiCGstab(l) [29] takes for q_k products of MR polynomials of degree l, thus generalizing Bi-CGSTAB that uses degree one MR polynomials. For $l > 1$ (say $l = 2$ or $l = 4$) this is a competitive algorithm, especially for problems involving matrices with complex

spectrum as arise from discretized advection-diffusion equations [29]. For these problems, Bi-CGSTAB often performs poorly [18]: eigenvalues with relatively large imaginary part slow down the convergence of Bi-CGSTAB or may lead to stagnation due the fact that the ω_k are real [29] (even in complex arithmetic).

These hybrid Bi-CG methods compute explicitly the Bi-CG coefficients α_k and β_k and, as in the Bi-CG case, we can cheaply find approximations for the extremal eigenvalues of $\mathbb{A}$ (see 12.7.3 and 12.7.2).

The QMR algorithm also has a "transpose free" (i.e. the algorithm does not use multiplications by A^T) squared version (see [11]); we will not discuss this algorithm here.

12.8 Numerical examples

So far we did not comment on the choice for the initial approximation x_0.
One often takes $x_0 = 0$. However, when solving time-dependent PDE's the solution from the previous time level may be a more educated guess. If the PDE is autonomous then the orthogonal basis vectors for a Krylov subspace (and the associated search directions) on one time level can be used to improve the initial guess for the next time level (in case GCR was used, take $x_0 + Q_k r_0$ instead of x_0, where the u_j and c_j that define Q_k stem from the previous time level and x_0 and r_0 are the present initial guess and associated residual). Actually, this comes down to an update of the preconditioner $(K^{-1} + Q_k(I - AK^{-1})$ instead of K^{-1}) that can also be used if the PDE is non-autonomous.

In this section we will discuss some numerical experiments. These experiments are intended to show some characteristic behavior of some of the methods discussed in this chapter. We do not pretend that the problems are solved in the best possible way. For instance, in some experiments we used a preconditioner, whereas in others we did not. A suitable preconditioner can improve the speed of convergence of any of the methods, but this is not the point we would like to make.

All partial differential equations were discretized with finite volumes discretization. With exception of the first and the third example, we took $x_0 \equiv 0$ as an initial guess. However, in our experiments the convergence history was mainly determined by the preconditioned matrix $\mathbb{A}$ (in line with our observations in 12.4.1). The choice of x_0 (or, equivalently, of the solution or inhomogeneous term) did not have much influence. Other choices postponed phenomena by only a few iteration steps, but they gave approximately the same picture. The experiments were done in double precision on a SUN SPARC 670 MP. Since we are interested in the performance of the iterative solution methods we did not try to find the most optimal stopping criterion. For a discussion about more appropriate stopping criteria we refer to [1]. We stopped when the residual reduction was less than 10^{-9}, i.e. $\|r_k\|_2 \leq 10^{-9}\|r_0\|_2$, or when the number of matrix multiplications exceeded 1000: the size of the error in the final approximation may be much smaller than the size of the discretization error. The figures show the convergence behavior of the iterative methods. Horizontally the number of matrix multiplications is counted. At the end of this section we give in Table 12.1 an overview of the required CPU-time for the norm of the *true* residual $b - Ax_k$ for several iterative methods. The numbers between brackets () are the log of the norm of the final *true* residuals: $^{10}\log(\|b - Ax_k\|_2)$. The log of the norm of the computed updated residuals can be seen from the figures. A '*' in Table 12.1 indicates that the method did not meet the required tolerance

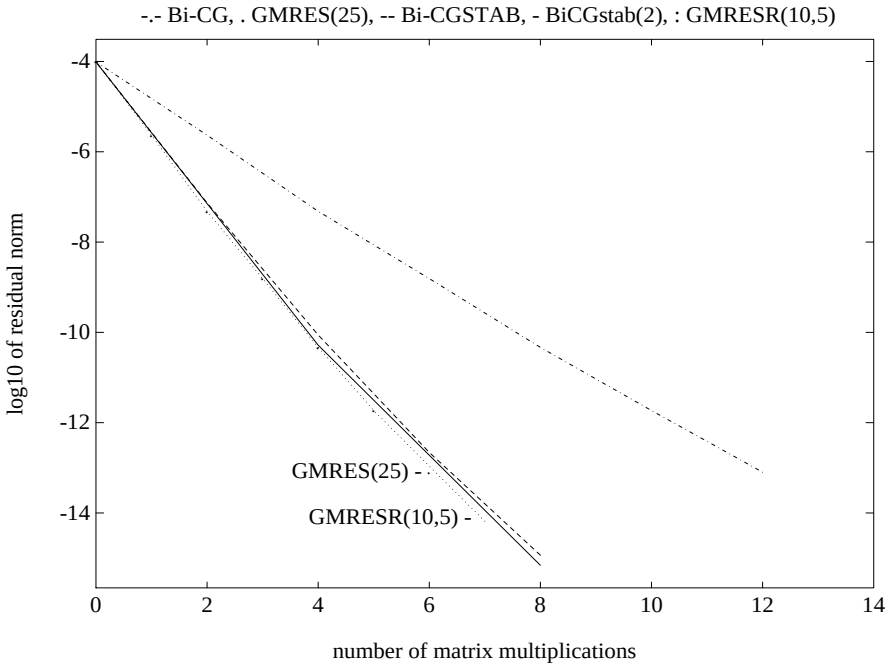

Figure 12.1: Convergence plot of numerical example 12.8.1.

before 1000 multiplications with the matrix A. We did our experiments for GMRES(25), GMRESR(10,5), Bi-CG, Bi-CGSTAB and BiCGstab(2). The first two algorithms have the same memory requirements.

12.8.1 Numerical example 1; the Molenkamp test problem

First we consider the Molenkamp problem.
We discretized the spatial derivatives by finite volumes using (81×81) volumes. For time integration, we used the trapezoidal rule with $\Delta t = 1/500$ (leading to the Crank-Nicolson scheme, see chapter 2, section 2.4).

Figure 12.1 shows the convergence histories to compute an accurate approximation on time level $t_1 = \Delta t$. For x_0 we took the vector given by the initial condition. Other time levels show comparable pictures if we take the solution of the preceding time level as initial guess.

GMRES(25) meets the required accuracy in 6 iteration steps. Hence it does not restart and its residuals are the best possible, i.e. they are the ones with minimal norm in the Krylov subspaces. From Figure 12.1 we see that the other methods find equally good approximations in the same Krylov subspaces (recall that Bi-CG needs two Mv's to increase the dimension of the Krylov space by one). Since Bi-CGSTAB is on average per Mv the cheapest method, it appears to be the best method for this example. However, the differences are small since the additional costs (additional to the Mv's) can hardly grow in six iteration steps.

On the subsequential time levels, we did not try to use the basis of the Krylov subspace and the Hessenberg matrix as computed by GMRES(25), although this would make GMRES(25) very cheap.

Surprisingly as it may look, it often occurs that the Krylov subspace methods perform equally well in the first few iteration steps: the fact that Bi-CG uses skew projections and the BiCGstab methods combine such skew projections with local minimalization

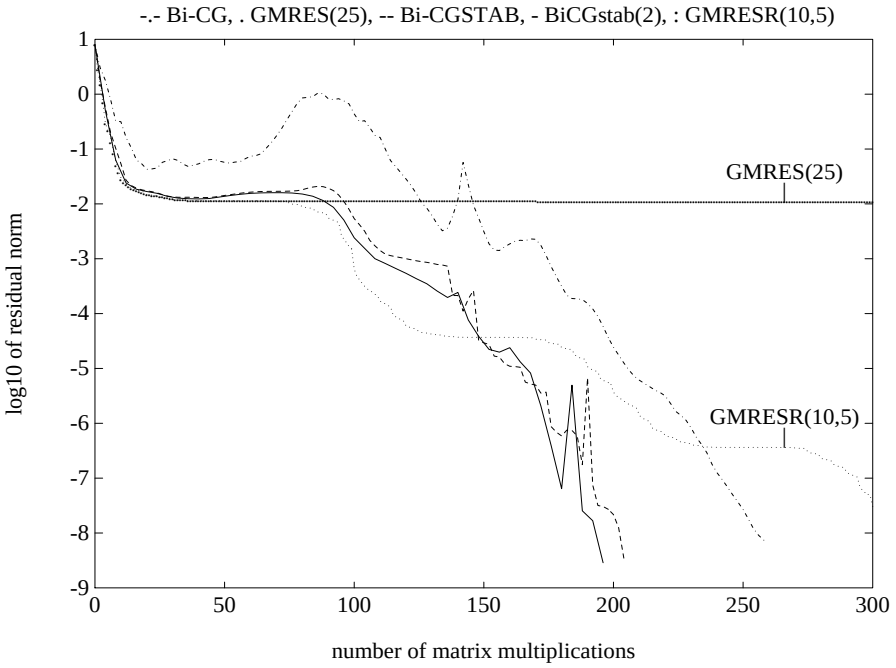

Figure 12.2: Convergence plot for numerical example 12.8.2.

leads to deviation from the optimal approach only after a few iteration steps. But here, we only needed a few steps to find an accurate approximation. In the next examples, the methods converge slower, due to a less favorable eigenvalue distribution, need more steps, and show completely different convergence behaviors already after a few steps.

12.8.2 Numerical example 2

Next we give an example where Bi-CG and the BiCGstab methods perform better than the methods based on the MR approach.

The symmetric positive definite linear system stems from an (81×81) discretization of

$$-\frac{\partial}{\partial x} D \frac{\partial c}{\partial x} - \frac{\partial}{\partial y} D \frac{\partial c}{\partial y} = 1, \tag{12.40}$$

over the unit square, with Dirichlet boundary conditions along $y = 0$ and Neumann conditions along the other parts of the boundary. The function D is defined as

$$D = 1000 \quad \text{for} \quad 0.1 \leq x, y \leq 0.9 \quad \text{and} \quad D = 1 \quad \text{elsewhere.}$$

ILU [19] was used as preconditioner. This example was taken from [40]. A convergence plot is given in Figure 12.2.

In the first few iteration steps all the methods perform equally well. None of the methods makes much progress in the steps 30–75. For GMRES(25) this is fatal: due to the restart it does not get beyond the point of stagnation. In view of the convergence history of GMRES(10,5), we expect GMRES(50) to do much better, although we did not try. In general, it is hard to select the proper m: for instance, GMRES(40) may stagnate while GMRES(41) may converge nicely. The selection of l and m for the best GMRESR(m, l) is equally difficult. Here, we see that GMRESR(10,5) performs well. The fact that the method is restarted at each 50-th matrix multiplication is reflected in the periodic stagnation phase.

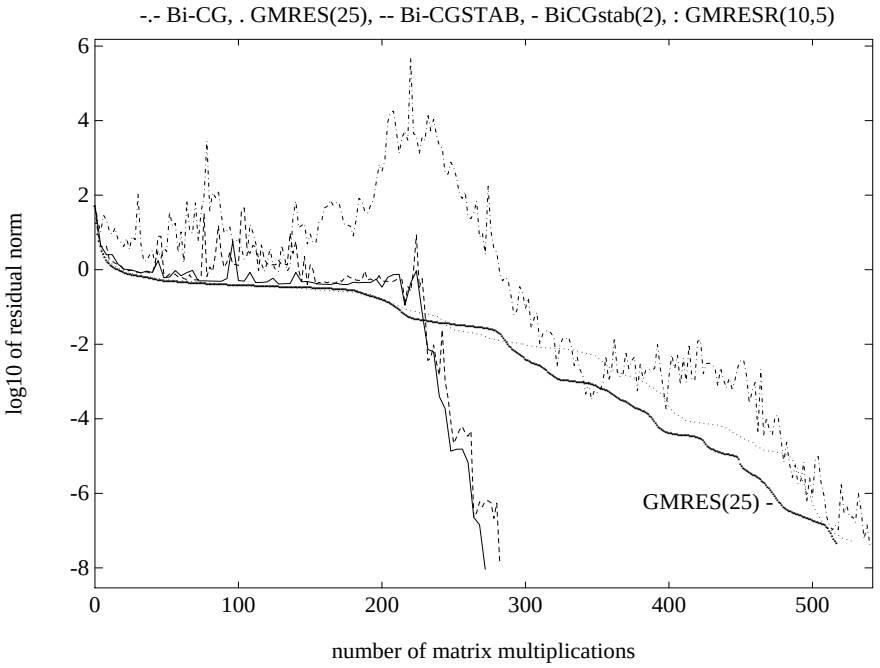

Figure 12.3: Convergence plot of numerical example 12.8.3.

The Bi-CG methods do not restart and show some acceleration. The BiCGstab methods are faster than Bi-CG (although not twice as fast). BiCGstab(2) is slightly faster than Bi-CGSTAB, but not fast enough to compensate for the more expensive steps (cf. Table 12.1). Bi-CGSTAB is the best method here, but BiCGstab(2) is comparable.

12.8.3 Numerical example 3

In our third example we consider an advection dominated 2-nd order partial differential equation, with Dirichlet boundary conditions, on the unit square:

$$-\frac{\partial^2 c}{\partial x^2} - \frac{\partial^2 c}{\partial y^2} + a\frac{\partial c}{\partial x} + 100\frac{\partial c}{\partial y} = F, \tag{12.41}$$

where $a = 100$ for $x \in [0, \frac{1}{4}] \cup [\frac{1}{2}, \frac{3}{4}]$ and $a = -100$ otherwise. The function F is defined by the solution $c(x, y) = \sin(\pi x)\sin(\pi y)$. This equation was discretized using (81×81) volumes, resulting in a five-diagonal linear system of order 6600. We took a random initial guess x_0 and we did not use a preconditioner.

The convergence histories for this example show similarity to the ones in the previous example (though Bi-CG converges more irregularly). One often sees that Krylov subspace methods stagnate in the first phase of the iteration process when applied to problems arising from advection-diffusion PDE's with strongly varying diffusion coefficients or with large advection terms. Often GMRES(m) requires a large m to overcome the stagnation phase.

12.8.4 Numerical example 4

In our fourth example we consider an advection dominated 2-nd order partial differential equation, with Dirichlet boundary conditions, on the unit cube (this equation was taken

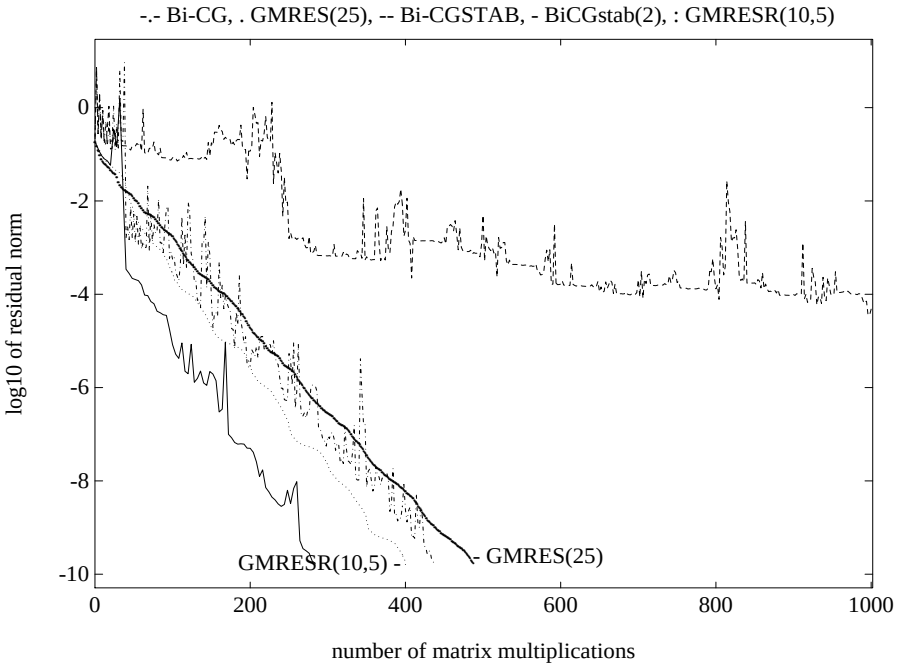

Figure 12.4: Convergence plot of numerical example 12.8.4.

from [18]):

$$\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} + \frac{\partial^2 c}{\partial z^2} + 1000\frac{\partial c}{\partial x} = F. \tag{12.42}$$

The function F is defined by the solution $c(x, y, z) = xyz(1-x)(1-y)(1-z)$. This equation was discretized using $(22 \times 22 \times 22)$ volumes, resulting in a seven-diagonal linear system of order 11000. No preconditioning was used.

In Figure 12.4 we see a plot of the convergence history. Bi-CGSTAB stagnates as might be anticipated from the fact that this linear system has large complex eigenpairs. Surprisingly, Bi-CGSTAB does even worse than Bi-CG. For this type of matrices this behavior of Bi-CGSTAB is not uncommon and might be explained by the poor first degree minimal residual reductions. In that case the Bi-CG iteration coefficients α_k and β_k are not recovered very well. BiCGstab(2) converges quite nicely and almost twice as fast as Bi-CG. Bi-CG looses bi-orthogonality among the residuals in a very early phase. This is also the case for the Bi-CG method that underlies BiCGstab(2). This explains why these methods do not accelerate (in contrast to what might be expected). But apparently BiCGstab(2) has less problems than Bi-CG.

GMRES(25) and GMRESR(10,5) converged nicely but not quite as fast as BiCGstab(2). Since their steps are also more expensive, BiCGstab(2) is the best method here (cf. Table 12.1).

12.8.5 Numerical example 5

Our last example shows that there is no guarantee that success is not ensured. For none of the five test methods we found an accurate approximation: Bi-CGSTAB broke down while the other four methods stagnated on a residual reduction level 10^{-1}.

The nonsymmetric linear system comes from a $(22 \times 22 \times 22)$ finite volume discretiza-

Table 12.1: CPU-time and log10 of the *true* residual norm (see the introduction of 12.8).

METHOD	Example 1	Example 2	Example 3	Example 4
GMRES(25)	1.07 (-13.0)	stagnation	91.4 (-7.3)	78.9 (-9.8)
GMRESR(10,5)	1.11 (-14.2)	41.9 (-8.2)	53.2 (-7.3)	59.4 (-9.8)
Bi-CG	1.15 (-13.0)	22.1 (-8.1)	29.1 (-7.4)	34.8 (-9.7)
Bi-CGSTAB	0.82 (-14.9)	17.8 (-8.5)	15.6 (-7.8)	87.7 (-4.3) *
BiCGstab(2)	1.00 (-15.2)	19.4 (-8.5)	17.3 (-8.0)	28.8 (-9.8)

tion of

$$\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} + \frac{\partial^2 c}{\partial z^2} + 10^5 x^2 (\frac{\partial c}{\partial x} + \frac{\partial c}{\partial y} + \frac{\partial c}{\partial z}) = F, \tag{12.43}$$

on the unit cube with Dirichlet boundary conditions (this equation was taken from [18]). The function F is defined by the solution $c(x, y, z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$. We did not use any preconditioning.

We also tried other methods, other parameter values and a number of standard preconditioning strategies. None of our attempts was successful, but we did not try to find a preconditioner adjusted to the stream lines.

12.9 Conclusions

There is no method that solves (quickly and) accurately any linear equation $Ax = b$. However, the class of Krylov subspace methods and its hybrids contain very attractive algorithms. As a side product, they can provide useful information about the spectrum of $\mathbb{A}$. If a problem requires a large m to get GMRES(m) to converge and Bi-CG and CG-S stagnate, methods as restarted GMRESR(m, l) (with $l = 20$, $m = 10$?) or BiCGstab(l) (with $l = 1, 2, 4$) are often very attractive alternatives. Especially, BiCGstab(2) (and BiCGstab(4)) seems to be a very promising competitive algorithm to solve non-symmetric linear systems as arising from discretized advection diffusion PDE's.

Acknowledgement. We are grateful to Diederik Fokkema for providing the numerical material.

References

[1] ARIOLI M., I. DUFF AND D. RUIZ: Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, **13** (1992), pp. 138–144.

[2] AXELSSON O.: On preconditioning and convergence acceleration in sparse matrix problems. *Rep. CERN 74-10*, (Geneve, 1974).

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

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

[5] Bai Z., D. Hu and L. Reichel: A Newton basis GMRES implementation. *Tech. Report 91–03* (Unviversity of Kentucky, 1991).

[6] Eirola T. and O. Nevanlinna: Accelerating with rank-one updates. *Linear Algebra Appl.*, **121** (1989), pp. 511–520.

[7] Eisenstat S.C., H.C. Elman and M.H. Schultz: Varational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, **20** (1983), pp. 245–357.

[8] Elman H.C.: Iterative methods for large, sparse, nonsymmetric systems of linear equations. *PhD thesis and Res. Rep. # 229* (Dept. of Comp. Sci., Yale U., 1982).

[9] Fischer B. and R.W. Freund: On the constrained Chebyshev approximation problem on ellipses. *J. Approx. Theory*, **62** (1990), pp. 297–315.

[10] Fletcher R.: Conjugate gradient methods for indefinite systems. In G. Watson,. ed., *Proc. of the Dundee Biennial Conference on Numerical Analysis* (Springer-Verlag, New York, 1975).

[11] Freund R.W.: A Transpose-Free Quasi-Minimal Residual Algorithm for non-Hermitian Linear Systems. *SIAM J. Sci. Stat. Comput.*, to appear.

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

[13] Golub G.H. and C.F. Van Loan: *Matrix Computations* (Second edition, The John Hopkins University Press, Baltimore and London, 1989).

[14] Greenbaum A.: Comparison of splittings used with the conjugate gradient algorithm. *Numer. Math.*, **33** (1979), pp. 181–194.

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

[16] Kharchenko S.A. and A.Yu. Yeremin: Eigenvalue translation based preconditioners for the GMRES(k) method. *Research Report EM-RR 2/92* (Elegant Math. Inc., 1992).

[17] Lanczos C.: Solution of systems of linear equations by minimized iteration. *J. Res. Nat. Bur. Stand.*, **49** (1952), pp. 33–53.

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

[19] Meijerink J.A. and H.A. van der Vorst: An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *ACCU Report*, (Utrecht, 1974), *Math. Comput.*, **31** (1977), pp. 148–162.

[20] Meijerink J.A. and H.A. van der Vorst: Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. Comp. Phys.*, **44** (1981), pp. 134–155.

[21] Nachtigal N.M., S.C. Reddy and L.N. Trefethen: How fast are nonsymmetric matrix iterations? *SIAM J. Sci. Stat. Comput.*, to appear.

[22] PAIGE C.C.: Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra Appl.*, **33** (1980), pp. 235–258, 1980.

[23] PAIGE C.C. AND M. A. SAUNDERS: Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, **12** (1975), pp. 617–629.

[24] PAIGE C.C. AND M. A. SAUNDERS: LSQR, an algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, **8** (1982), pp. 43–71.

[25] REID J.K.: On the method of conjugate gradients for the solution of large sparse systems of linear equations. *Proc. Conf. on large sparse sets of linear equations*, (Academic Press, New York, 1971).

[26] SAAD Y.: Krylov subspace method for solving large unsymmetric linear systems. *Math. Comput.*, **37** (1981), pp. 105–126.

[27] SAAD Y.: A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Stat. Comput.*, **14** (1993).

[28] SAAD Y. AND M.H. SCHULTZ: GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, **7** (1986), pp. 856–869.

[29] SLEIJPEN G.L.G. AND D.R. FOKKEMA: BiCGstab(l) for linear equations involving unsymmetric matrices with complex spectrum. *Preprint 772* (Dep. Math., University of Utrecht, 1993), *ETNA*, to appear.

[30] SLEIJPEN G.L.G. AND H.A. VAN DER VORST: Krylov subspace methods for large linear systems of equations. *Preprint 803* (Dep. Math., University of Utrecht, 1993).

[31] SLEIJPEN G.L.G. AND A. VAN DER SLUIS: Further results on the convergence behaviour of CG and Ritz values. *Preprint 677* (Dep. Math., University of Utrecht, 1991).

[32] VAN DER SLUIS A.: The convergence behaviour of conjugate gradients and Ritz values in various circumstances. in *Iterative methods in linear algebra*, R. Beauwens and P. de Groen, eds. (North Holland, Amsterdam, London, New York, Tokyo, 1992).

[33] SONNEVELD P.: CGS, a fast Lanczos-type solver for nonsymetric linear systems. *SIAM J. Sci. Stat. Comput.*, **10** (1989), pp. 36–52.

[34] DE STURLER E. AND D.R. FOKKEMA: Nested Krylov methods and preserving the orthogonality. *Preprint 796* (Dept. Math., University of Utrecht, 1993).

[35] TREFETHEN L.N.: *Non-normal matrices and pseudo-eigenvalues in numerical analysis.* Book in preparation

[36] VARGA R.S.: *Matrix Iterative Analysis* (Prentice-Hall, Englewood Cliffs N.J., 1962).

[37] VINSOME P.K.W.: ORTHOMIN, an iterative method for solving sparse sets of simultaneous linear equations. Paper SPE 5729, 4th *Symposium of Numerical Simulation of Reservoir Performance of the Society of Petrolium Engineers of the AIME*, (Los Angeles, 1976).

[38] VAN DER VORST H.A. AND G.L.G. SLEIJPEN: The effect of incomplete decomposition preconditioning on the convergence of conjugate gradients. *Incomplete Decompositions, Proceedings of the Eigth GAMM Seminar* (Vieweg Verlag, Braunschweig, 1992).

[39] VAN DER VORST H.A.: The convergence behavior of preconditioned CG and CG-S in the presence of rounding errors, *Lecture Notes in Math.*, **1457** (Springer, Berlin, 1990), 126–136.

[40] VAN DER VORST H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, **13** (1992), pp. 631–644.

[41] VAN DER VORST H.A. AND C. VUIK: The rate of convergence of GMRES. *Preprint* (Dep. Math., RUU, 1990).

[42] VAN DER VORST H.A. AND C. VUIK: GMRESR: A family of nested GMRES methods. *Tech. Report 91–80* (Delft University of Technology, Faculty of Tech. Math., Delft, 1991).

[43] YOUNG D.M. AND K.C. JEA: Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, **34** (1980), pp. 159–194.