SIAM J. SCI. COMPUT.
Vol. 31, No. 2, pp. 1035–1062
© 2008 Society for Industrial and Applied Mathematics

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

IDR(s): A FAMILY OF SIMPLE AND FAST ALGORITHMS FOR SOLVING LARGE NONSYMMETRIC SYSTEMS OF LINEAR EQUATIONS*

PETER SONNEVELD† AND MARTIN B. VAN GIJZEN†

Abstract. We present IDR(s), a new family of efficient, short-recurrence methods for large nonsymmetric systems of linear equations. The new methods are based on the induced dimension reduction (IDR) method proposed by Sonneveld in 1980. IDR(s) generates residuals that are forced to be in a sequence of nested subspaces. Although IDR(s) behaves like an iterative method, in exact arithmetic it computes the true solution using at most $N + N/s$ matrix-vector products, with N the problem size and s the codimension of a fixed subspace. We describe the algorithm and the underlying theory and present numerical experiments to illustrate the theoretical properties of the method and its performance for systems arising from different applications. Our experiments show that IDR(s) is competitive with or superior to most Bi-CG-based methods and outperforms Bi-CGSTAB when $s > 1$.

Key words. iterative methods, induced dimension reduction, Krylov-subspace methods, Bi-CGSTAB, CGS, nonsymmetric linear systems

AMS subject classifications. 65F10, 65F50

DOI. 10.1137/070685804

1. Introduction.

Krylov subspace methods are used extensively for the iterative solution of linear systems of equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

The most popular method for solving large systems with $\boldsymbol{A}$ Hermitian and positive definite of size N is the conjugate gradient (CG) method [8] of Hestenes and Stiefel. The CG method minimizes the $\boldsymbol{A}$-norm of the error over the Krylov subspace

$$\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0) = \text{span}\left(\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \ldots, \boldsymbol{A}^n\boldsymbol{r}_0\right), \tag{1.1}$$

using short recurrences. Here, n is the iteration number, and $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ is the initial residual. Short recurrences imply that only a small number of vectors is needed to carry out the process, so that an extremely simple and efficient method is obtained. Unfortunately, as shown by Faber and Manteuffel [1], it is not possible to derive a method for *general* $\boldsymbol{A}$ that combines an optimal minimization of some error norm over $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0)$ with short recurrences.

The search for efficient Krylov methods for systems with a general matrix $\boldsymbol{A}$ has been dominated by two different approaches, both of which are generalizations of CG. In the first approach, the requirement of short recurrences is removed. The most popular member of this family, GMRES [11], yields iterates that minimize the residual over the Krylov subspace after n iterations, at the expense of having to compute and store a new orthogonal basis vector for the Krylov subspace at every iteration. This

*Received by the editors March 20, 2007; accepted for publication (in revised form) August 8, 2008; published electronically November 26, 2008. Part of this research has been funded by the Dutch BSIK/BRICKS project.
http://www.siam.org/journals/sisc/31-2/68580.html
†Delft University of Technology, Delft Institute of Applied Mathematics, Mekelweg 4, 2628 CD Delft, The Netherlands (P.Sonneveld@tudelft.nl, M.B.vanGijzen@tudelft.nl).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

operation becomes prohibitive, with respect to both memory and computations, if many iterations have to be performed to achieve a desired precision.

The second approach generalizes CG using short recurrences. The archetype of this class is the Bi-CG method of Fletcher [2], which is equivalent to CG in the symmetric case. However, Bi-CG requires two matrix-vector products per iteration, one with $\boldsymbol{A}$ and one with $\boldsymbol{A}^H$, which makes it approximately twice as expensive as CG. Moreover, the method has no optimality property for general $\boldsymbol{A}$.

Since Bi-CG is based on the bi-Lanczos tridiagonalization method [9], the method terminates (in exact arithmetic) in at most N iterations, hence using at most $2N$ matrix-vector products. The search for a faster Bi-CG-type method focused on Sonneveld's idea of making better use of the "wasted" extra matrix-vector multiplication. In his CGS method [14], this is achieved by applying the CG polynomial twice at no extra cost in terms of matrix-vector multiplications. An additional advantage of CGS is that no multiplications with $\boldsymbol{A}^H$ are needed.

For many problems, CGS is considerably faster than Bi-CG, but the convergence behavior is also much more erratic. To overcome this drawback, van der Vorst proposed Bi-CGSTAB [15], which applies the Bi-CG polynomial in combination with a linear minimal residual step at each iteration. This method was generalized first by Gutknecht [6] to BiCGstab2 and later by Sleijpen and Fokkema [12] to the BiCGstab(ℓ) methods that combine Bi-CG with higher-order minimum residual methods. Another approach to stabilize CGS is taken in TFQMR [4]. This method combines the CGS idea with a QMR-type quasi minimization of the residual norm.

Since these developments, many other methods have been proposed that combine the Bi-CG polynomial with another polynomial. For example, Zhang [19] generalized CGS and Bi-CGSTAB in a unified way by a class of product-type methods whose residual polynomials are a product of the Bi-CG polynomial and other polynomials with standard three-term recurrence relations. Another approach to generalize Bi-CGSTAB was proposed by Yeung and Chan, whose ML(k)BiCGSTAB method [18] is a Bi-CGSTAB variant based on multiple left Lanczos starting vectors.

As is clear from the above, research efforts on fast Krylov algorithms based on short recurrences have focused on Bi-CG-type methods. This is probably due to the fact that in the symmetric case Bi-CG is mathematically equivalent to the optimal CG method (albeit at twice the price). However, there is no reason to believe that a different approach cannot yield faster methods. In this paper, we propose an approach that seems to confirm that indeed it is possible to derive competitive or even faster methods for nonsymmetric systems in a way that is not based on Bi-CG or Lanczos.

In order to derive such a method we revisit the *induced dimension reduction* (IDR) algorithm proposed in 1980 by Sonneveld in [17] as an iterative method for solving nonsymmetric systems of equations. The method has several favorable features: it is simple, uses short recurrences, and computes the exact solution in at most $2N$ steps (matrix-vector multiplications) in exact arithmetic.

Analysis of IDR revealed a close relation with Bi-CG. It was shown in [17] that the iteration polynomial constructed by IDR is the product of the Bi-CG polynomial with another, locally minimizing polynomial. Sonneveld's observation that the Bi-CG polynomial could be combined with another polynomial without transpose-matrix-vector multiplications led to the development first of CGS and later of Bi-CGSTAB.

Over the years, CGS and Bi-CGSTAB have completely overshadowed IDR, which is now practically forgotten, except perhaps as the predecessor of CGS. This is unfortunate since, although there is a clear relation between CG-type methods and the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

original IDR method, the underlying ideas are completely different. This suggests that by exploiting the differences new methods may be developed.

Bi-CG, CGS, Bi-CGSTAB, and BiCGstab(ℓ) are essentially based on the computation of two mutually biorthogonal bases for the Krylov subspaces $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0)$ and $\mathcal{K}^n(\boldsymbol{A}^H, \widetilde{\boldsymbol{r}}_0)$. The "S"-part in CGS and the "STAB"-part in Bi-CGSTAB are different ways of making more efficient use of the $\boldsymbol{A}^H$-related information. The finiteness of these methods (in exact arithmetic) comes from the finiteness of any basis for (a subspace of) $\mathbb{C}^N$.

The IDR method, on the other hand, generates residuals that are forced to be in subspaces $\mathcal{G}_j$ of decreasing dimension. These nested subspaces are related by $\mathcal{G}_j = (\boldsymbol{I} - \omega_j \boldsymbol{A})(\mathcal{S} \cap \mathcal{G}_{j-1})$, where $\mathcal{S}$ is a fixed proper subspace of $\mathbb{C}^N$, and the ω_j's are nonzero scalars.

In this paper, we describe IDR(s), a family of new iterative solution algorithms based on the IDR mechanism. We propose a number of improvements and generalizations of the original IDR method as well as new variants that compute the true solution in exact arithmetic using at most $N + N/s$ matrix-vector multiplications.

This paper is organized as follows. In section 2, we present and prove the IDR theorem which provides the theoretical basis for the new algorithms. In section 3, we describe a prototype for the IDR family of algorithms and we analyze the termination and breakdown behavior of the IDR(s) algorithms. In section 4, we discuss what freedom there is in developing IDR-based algorithms and explain among other things how this freedom can be exploited to avoid breakdown of the algorithm. In section 5, we discuss IDR(s) as a polynomial-based algorithm and explain the relationship between IDR(1) and Bi-CGSTAB and between IDR(s) and ML(k)BiCGSTAB. In section 6, we describe the numerical experiments. We present both simple experiments to validate the theoretical properties of IDR(s) and realistic examples to make an evaluative comparison with the best known Bi-CG-type methods: Bi-CGSTAB, Bi-CG, QMR [5], CGS, and BiCGstab(ℓ). We present concluding remarks in section 7.

2. The IDR theorem. The new family of algorithms is based on the IDR theorem. The original IDR theorem was published in [17, p. 550]. Here, we give a generalization of the original result to complex matrices.

THEOREM 2.1 (IDR). *Let $\boldsymbol{A}$ be any matrix in $\mathbb{C}^{N\times N}$, let $\boldsymbol{v}_0$ be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov space $\mathcal{K}^N(\boldsymbol{A}, \boldsymbol{v}_0)$. Let $\mathcal{S}$ denote any (proper) subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of $\boldsymbol{A}$, and define the sequence $\mathcal{G}_j$, $j = 1, 2, \ldots$, as*

$$\mathcal{G}_j = (\boldsymbol{I} - \omega_j \boldsymbol{A})(\mathcal{G}_{j-1} \cap \mathcal{S}),$$

where the ω_j's are nonzero scalars. Then the following hold:

(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1}\ \forall j > 0$.

(ii) $\mathcal{G}_j = \{\boldsymbol{0}\}$ *for some* $j \leq N$.

Proof. We first show by induction that $\mathcal{G}_j \subset \mathcal{G}_{j-1}\ \forall j > 0$. Since $\mathcal{G}_0$ is a full Krylov space, we have

$$\mathcal{G}_1 = (\boldsymbol{I} - \omega_1 \boldsymbol{A})(\mathcal{G}_0 \cap \mathcal{S}) \subset (\boldsymbol{I} - \omega_1 \boldsymbol{A})\mathcal{G}_0 \subset \mathcal{G}_0.$$

Now assume $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ for some $j > 0$, and let $\boldsymbol{x} \in \mathcal{G}_{j+1}$. Then

$$\boldsymbol{x} = (\boldsymbol{I} - \omega_{j+1} \boldsymbol{A})\boldsymbol{y}$$

for some $\boldsymbol{y} \in \mathcal{G}_j \cap \mathcal{S}$. Then $\boldsymbol{y} \in \mathcal{G}_{j-1} \cap \mathcal{S}$ by the induction hypothesis. Hence, $(\boldsymbol{I} - \omega_j \boldsymbol{A})\boldsymbol{y} \in \mathcal{G}_j$. This implies that $\boldsymbol{A}\boldsymbol{y} \in \mathcal{G}_j$, and therefore $(\boldsymbol{I} - \omega_{j+1} \boldsymbol{A})\boldsymbol{y} = \boldsymbol{x} \in \mathcal{G}_j$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

It follows that $\mathcal{G}_{j+1} \subset \mathcal{G}_j$.

We now show that $\mathcal{G}_j = \{\mathbf{0}\}$ for some $j \leq N$. Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, there are two possibilities: either $\mathcal{G}_{j+1}$ is a proper subspace of $\mathcal{G}_j$, or $\mathcal{G}_{j+1} = \mathcal{G}_j$.

In the first case, $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$. The second case can occur only if $\mathcal{G}_j \cap \mathcal{S} = \mathcal{G}_j$. Otherwise, $\dim(\mathcal{G}_j \cap \mathcal{S}) < \dim(\mathcal{G}_j)$, and consequently $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$. So $\mathcal{G}_j \cap \mathcal{S} = \mathcal{G}_j$, and therefore $\mathcal{G}_j \subset \mathcal{S}$. Also $\mathcal{G}_{j+1} = (\boldsymbol{I} - \omega_{j+1}\boldsymbol{A})(\mathcal{G}_j \cap \mathcal{S}) = (\boldsymbol{I} - \omega_{j+1}\boldsymbol{A})\mathcal{G}_j$, which implies that $\mathcal{G}_j$ is an invariant subspace of $\boldsymbol{A}$.

Since $\mathcal{G}_j \subset \mathcal{S}$ and $\mathcal{G}_j \subset \mathcal{G}_0$, and by assumption $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of $\boldsymbol{A}$, it follows that $\mathcal{G}_j = \{\mathbf{0}\}$. Therefore, either the dimension of the $\mathcal{G}_j$ space is reduced at each step, or $\mathcal{G}_j = \{\mathbf{0}\}$. Since $\dim(\mathcal{G}_0) \leq N$, no more than N dimension reduction steps can be performed. Hence there is a $j \leq N$ for which $\mathcal{G}_j = \{\mathbf{0}\}$. ∎

Remark. The restriction that $\mathcal{S}$ and $\mathcal{G}_0$ may not share a nontrivial invariant subspace of $\boldsymbol{A}$ is not severe. Because $\mathcal{G}_0$ is a full Krylov space, all eigenspaces of $\boldsymbol{A}$ in $\mathcal{G}_0$ are one-dimensional. So if, for instance, $\mathcal{S}$ is chosen at random, then the event that one of these eigenspaces is in $\mathcal{S}$ has zero probability.

The above theorem states that it is possible to generate a sequence of nested subspaces of decreasing dimension and that under mild conditions the smallest possible subspace is $\{\mathbf{0}\}$.

3. The IDR(s) algorithm.

3.1. Derivation of the prototype. Let $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ be an $N \times N$ linear system. A Krylov-type solver produces iterates $\boldsymbol{x}_n$ for which the residuals $\boldsymbol{r}_n = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_n$ are in the Krylov spaces $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0)$. Here, $\boldsymbol{x}_0$ is an initial estimate of the solution. As a consequence, the residuals $\boldsymbol{r}_n$ can be written as $\Phi_n(\boldsymbol{A})\boldsymbol{r}_0$, where Φ_n is an nth degree polynomial:[1] $\Phi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}$. Given a recursion for the residuals $\boldsymbol{r}_n$, we must be able to produce a corresponding recursion for $\boldsymbol{x}_n$. Assume this has been possible for the residuals up to step n; then it must be possible to calculate $\boldsymbol{x}_{n+1}$ from the equation

$$\boldsymbol{A}\Delta\boldsymbol{x}_n = -\Delta\boldsymbol{r}_n = [\Phi_n(\boldsymbol{A}) - \Phi_{n+1}(\boldsymbol{A})]\,\boldsymbol{r}_0$$

without actually solving an equation with the matrix $\boldsymbol{A}$. Here, the forward difference operator $\Delta\boldsymbol{u}_k = \boldsymbol{u}_{k+1} - \boldsymbol{u}_k$ is used.

This is always possible if the polynomial difference $\Phi_{n+1}(\tau) - \Phi_n(\tau)$ is divisible by τ, i.e., $\Phi_{n+1}(\tau) = \Phi_n(\tau) + \tau\Psi_n(\tau)$, with $\Psi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}$. With $\Phi_0 \equiv 1$, this implies that $\Phi_n(0) = 1\ \forall n \geq 0$.

Therefore the general Krylov-type solver can be described by recursions of the following form:

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha\boldsymbol{A}\boldsymbol{v}_n - \sum_{l=1}^{\widehat{l}} \gamma_l \Delta\boldsymbol{r}_{n-l}, \tag{3.1}$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha\boldsymbol{v}_n - \sum_{l=1}^{\widehat{l}} \gamma_l \Delta\boldsymbol{x}_{n-l},$$

where $\boldsymbol{v}_n$ is any computable vector in $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0) \setminus \mathcal{K}^{n-1}(\boldsymbol{A}, \boldsymbol{r}_0)$. The integer $\widehat{l}$ is the depth of the recursion. If $\widehat{l} = n$, we have a so-called long recurrence, which implies that

[1] Under very exceptional circumstances, the polynomial may be of lower degree, causing breakdown in most Krylov methods.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

the amount of work and the memory requirements grow with n. On the other hand, if $\widehat{l}$ is fixed and small (compared to N), then we have a so-called *short recurrence*, which is attractive with respect to computational and memory requirements.

The IDR theorem can be applied by generating residuals $\boldsymbol{r}_n$ that are forced to be in the subspaces $\mathcal{G}_j$, where j is nondecreasing with increasing n. Then, under the assumptions of Theorem 2.1, the system will be solved after at most N dimension reduction steps.

The residual $\boldsymbol{r}_{n+1}$ is in $\mathcal{G}_{j+1}$ if

$$\boldsymbol{r}_{n+1} = (\boldsymbol{I} - \omega_{j+1}\boldsymbol{A})\boldsymbol{v}_n \quad \text{with } \boldsymbol{v}_n \in \mathcal{G}_j \cap \mathcal{S}.$$

Now if we choose

$$\boldsymbol{v}_n = \boldsymbol{r}_n - \sum_{l=1}^{\widehat{l}} \gamma_l \Delta \boldsymbol{r}_{n-l}, \tag{3.2}$$

then the expression for $\boldsymbol{r}_{n+1}$ reads

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}_n - \sum_{l=1}^{\widehat{l}} \gamma_l \Delta \boldsymbol{r}_{n-l},$$

which corresponds to (3.1), the general Krylov-solver recursion.

Without loss of generality, we may assume the space $\mathcal{S}$ to be the left nullspace of some $N \times s$ matrix $\boldsymbol{P}$:

$$\boldsymbol{P} = (\boldsymbol{p}_1\ \boldsymbol{p}_2\ \cdots\ \boldsymbol{p}_s), \quad \mathcal{S} = \mathcal{N}(\boldsymbol{P}^H).$$

Since $\boldsymbol{v}_n$ is also in $\mathcal{S} = \mathcal{N}(\boldsymbol{P}^H)$, it additionally satisfies

$$\boldsymbol{P}^H \boldsymbol{v}_n = \boldsymbol{0}. \tag{3.3}$$

Combining (3.2) and (3.3) yields an $s \times \widehat{l}$ linear system for the $\widehat{l}$ coefficients γ_l. Under normal circumstances this system is uniquely solvable if $\widehat{l} = s$. Consequently, computing the first vector in $\mathcal{G}_{j+1}$ requires $s+1$ vectors in $\mathcal{G}_j$, and we may expect $\boldsymbol{r}_n$ to be in $\mathcal{G}_{j+1}$ only for $n \geq (j+1)(s+1)$. We will come back to the exceptional case when the system is not uniquely solvable in the next section.

Define the following matrices:

$$\Delta \boldsymbol{R}_n = (\Delta \boldsymbol{r}_{n-1}\ \Delta \boldsymbol{r}_{n-2}\ \cdots\ \Delta \boldsymbol{r}_{n-s}), \tag{3.4}$$

$$\Delta \boldsymbol{X}_n = (\Delta \boldsymbol{x}_{n-1}\ \Delta \boldsymbol{x}_{n-2}\ \cdots\ \Delta \boldsymbol{x}_{n-s}). \tag{3.5}$$

Then the computation of $\boldsymbol{r}_{n+1} \in \mathcal{G}_{j+1}$ can be implemented by the following algorithm:

Calculate: $\boldsymbol{c} \in \mathbb{C}^s$ from $(\boldsymbol{P}^H \Delta \boldsymbol{R}_n)\boldsymbol{c} = \boldsymbol{P}^H \boldsymbol{r}_n$,
$\boldsymbol{v} = \boldsymbol{r}_n - \Delta \boldsymbol{R}_n \boldsymbol{c}$,
$\boldsymbol{r}_{n+1} = \boldsymbol{v} - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}$.

Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, repeating these calculations will produce new residuals $\boldsymbol{r}_{n+2}, \boldsymbol{r}_{n+3}, \ldots$ in $\mathcal{G}_{j+1}$. Once $s+1$ residuals in $\mathcal{G}_{j+1}$ have been computed, we can expect the next residual to be in $\mathcal{G}_{j+2}$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

In the calculation of the first residual in $\mathcal{G}_{j+1}$, we may choose ω_{j+1} freely, but the same value must be used in the calculations of the subsequent residuals in $\mathcal{G}_{j+1}$. A suitable choice for ω_{j+1} is the value that minimizes the norm of $\boldsymbol{r}_{n+1}$, similarly as is done in, among others, the Bi-CGSTAB algorithm.

Of course, we must update the solution vector $\boldsymbol{x}_{n+1}$ together with the updates for the residual $\boldsymbol{r}_{n+1}$. Furthermore, the process must be initialized; that is, residuals *and* solution updates must be generated by a Krylov-oriented procedure, before we can start the above type of calculations. We present the algorithm in Figure 3.1.

We make the following remarks:

- This prototype is intended not as a practical but as a mathematical algorithm. The implementation of $\Delta \boldsymbol{R}_n = (\Delta \boldsymbol{r}_{n-1} \cdots \Delta \boldsymbol{r}_{n-s})$, etc., as well as the the computation of matrices $\boldsymbol{P}^H \Delta \boldsymbol{R}_n$, can, of course, be done much more efficiently than suggested. We refer to the appendix for a simple but efficient MATLAB code.
- The $s \times s$ system may be (nearly) inconsistent, leading to a (near) breakdown. We will refer to this as *breakdown of type* 1.
  This is similar to what is called *Lanczos breakdown* in Bi-CG-based methods. Working around this problem, however, is far less complicated than in the Bi-CGSTAB algorithm.
- The ω calculation might produce a (nearly) zero ω-value, leading to stagnation of the procedure. This is referred to as *breakdown of type* 2.

Estimates for work and memory requirements are presented in Table 3.1. The operation count for the main operations to perform a full cycle of $s+1$ IDR(s) steps yields $(s+1)$ matrix-vector products, s^2+s+2 inner products, and $2s^2+\frac{7}{2}s+\frac{5}{2}$ vector updates. For this count we refer to the appendix. Note that we have counted scaling of a vector and a simple addition of two vectors as half an update each. Table 3.1 gives an overview of the number of vector operations per matrix-vector multiplication for some IDR(s) variants, and for the most widely used other Krylov methods. This table also gives the memory requirements (excluding storage of the system matrix and of the preconditioner, but including storage for the right-hand side and the solution).

3.2. Performance and exceptions. The original IDR theorem predicts only dimension reduction but does not say by how much. In the original algorithm [17], where $\mathcal{S} = \boldsymbol{p}^{\perp}$ (the $s = 1$ case), the dimension is reduced by one at each step. For that, a step requires two matrix-vector operations, but, in the case of IDR(s), each step requires $(s+1)$ matrix-vector operations, possibly leading to about $(s+1)N$ "matvecs" for the whole finite procedure. Now in practice the method shows a much faster convergence, but, still, we would like to have a reliable prediction for the finite behavior.

The following theorem concerns the rate at which the dimension reduction takes place in the IDR(s) algorithms.

THEOREM 3.1 (extended IDR theorem). *Let $\boldsymbol{A}$ be any matrix in $\mathbb{C}^{N\times N}$, let $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s \in \mathbb{C}^N$ be linearly independent, let $\boldsymbol{P} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s]$, let $\mathcal{G}_0 = \mathcal{K}^N(\boldsymbol{A}, \boldsymbol{r}_0)$ be the full Krylov space corresponding to $\boldsymbol{A}$ and the vector $\boldsymbol{r}_0$, and let the sequence of spaces $\{\mathcal{G}_j,\ j = 1, 2, \ldots\}$ be defined by*

$$\mathcal{G}_j = (\boldsymbol{I} - \omega_j \boldsymbol{A})(\mathcal{G}_{j-1} \cap \mathcal{N}(\boldsymbol{P}^H)),$$

where ω_j are nonzero numbers, such that $\boldsymbol{I} - \omega_j \boldsymbol{A}$ is nonsingular.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Require: $\boldsymbol{A} \in \mathbb{C}^{N\times N}$; $\boldsymbol{x}_0, \boldsymbol{b} \in \mathbb{C}^N$; $\boldsymbol{P} \in \mathbb{C}^{N\times s}$; $TOL \in (0,1)$; $MAXIT > 0$
Ensure: x_n such that $\|\boldsymbol{b} - \boldsymbol{A}x_n\| \leq TOL$
  {*Initialization.*}
  Calculate $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$;

  {*Apply s minimum norm steps, to build enough vectors in $\mathcal{G}_0$*}
  for $n = 0$ to $s-1$ **do**
    $v = \boldsymbol{A}\boldsymbol{r}_n$; $\omega = (\boldsymbol{v}^H\boldsymbol{r}_n)/(\boldsymbol{v}^H\boldsymbol{v})$;
    $\Delta\boldsymbol{x}_n = \omega\boldsymbol{r}_n$; $\Delta\boldsymbol{r}_n = -\omega\boldsymbol{v}$;
    $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \Delta\boldsymbol{r}_n$; $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\boldsymbol{x}_n$;
  end for
  $\Delta\boldsymbol{R}_{n+1} = (\Delta\boldsymbol{r}_n \cdots \Delta\boldsymbol{r}_0)$; $\Delta\boldsymbol{X}_{n+1} = (\Delta\boldsymbol{x}_n \cdots \Delta\boldsymbol{x}_0)$;

  {*Building $\mathcal{G}_j$ spaces for $j = 1, 2, 3, \ldots$*}
  $n = s$
  {*Loop over $\mathcal{G}_j$ spaces*}
  while $\|\boldsymbol{r}_n\| > TOL$ **and** $n < MAXIT$ **do**
    {*Loop inside $\mathcal{G}_j$ space*}
    for $k = 0$ to s **do**
      Solve $\boldsymbol{c}$ from $\boldsymbol{P}^H\Delta\boldsymbol{R}_n\boldsymbol{c} = \boldsymbol{P}^H\boldsymbol{r}_n$
      $\boldsymbol{v} = \boldsymbol{r}_n - \Delta\boldsymbol{R}_n\boldsymbol{c}$;
      if $k = 0$ **then**
        {*Entering $\mathcal{G}_{j+1}$*}
        $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{v}$;
        $\omega = (\boldsymbol{t}^H\boldsymbol{v})/(\boldsymbol{t}^H\boldsymbol{t})$;
        $\Delta\boldsymbol{r}_n = -\Delta\boldsymbol{R}_n\boldsymbol{c} - \omega\boldsymbol{t}$;
        $\Delta\boldsymbol{x}_n = -\Delta\boldsymbol{X}_n\boldsymbol{c} + \omega\boldsymbol{v}$;
      else
        {*Subsequent vectors in $\mathcal{G}_{j+1}$*}
        $\Delta\boldsymbol{x}_n = -\Delta\boldsymbol{X}_n\boldsymbol{c} + \omega\boldsymbol{v}$;
        $\Delta\boldsymbol{r}_n = -\boldsymbol{A}\Delta\boldsymbol{x}_n$;
      end if
      $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \Delta\boldsymbol{r}_n$;
      $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\boldsymbol{x}_n$;
      $n = n + 1$;
      $\Delta\boldsymbol{R}_n = (\Delta\boldsymbol{r}_{n-1} \cdots \Delta\boldsymbol{r}_{n-s})$;
      $\Delta\boldsymbol{X}_n = (\Delta\boldsymbol{x}_{n-1} \cdots \Delta\boldsymbol{x}_{n-s})$;
    end for
  end while

FIG. 3.1. *The IDR(s) algorithm.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 3.1
Vector operations per matrix-vector product and memory requirements.

Method	DOT	AXPY	Memory requirements
IDR(1)	2	4	8
IDR(2)	$2\frac{2}{3}$	$5\frac{5}{6}$	11
IDR(4)	$4\frac{2}{5}$	$9\frac{7}{10}$	17
IDR(6)	$6\frac{2}{7}$	$13\frac{9}{14}$	23
GMRES	$\frac{n+1}{2}$	$\frac{n+1}{2}$	$n+3$
Bi-CG	1	$2\frac{1}{2}$	7
QMR	1	4	13
CGS	1	3	8
BiCGSTAB	2	3	7
BiCGstab(2)	$2\frac{1}{4}$	$3\frac{3}{4}$	9
BiCGstab(4)	$2\frac{3}{4}$	$5\frac{1}{4}$	13
BiCGstab(8)	$3\frac{3}{4}$	$8\frac{1}{4}$	21

Let $\dim(\mathcal{G}_j) = d_j$*; then the sequence* $\{d_j,\ j = 0, 1, 2, \dots\}$ *is monotonically nonincreasing and satisfies*

$$0 \le d_j - d_{j+1} \le d_{j-1} - d_j \le s.$$

Proof. Let $\mathcal{U} = \mathcal{G}_{j-1} \cap \mathcal{N}(\boldsymbol{P}^H)$, and let $\boldsymbol{G}_{j-1}$ be a matrix whose columns form a basis for $\mathcal{G}_{j-1}$. Then each $\boldsymbol{x} \in \mathcal{G}_{j-1}$ can be written as $\boldsymbol{x} = \boldsymbol{G}_{j-1}\boldsymbol{c}$ for some $\boldsymbol{c}$. Therefore each $\boldsymbol{x} \in \mathcal{U}$ can be represented as $\boldsymbol{x} = \boldsymbol{G}_{j-1}\boldsymbol{c}$, with $\boldsymbol{c}$ satisfying $\boldsymbol{P}^H\boldsymbol{G}_{j-1}\boldsymbol{c} = \boldsymbol{0}$. Hence $\mathcal{U} = \boldsymbol{G}_{j-1}(\mathcal{N}(\boldsymbol{P}^H\boldsymbol{G}_{j-1}))$, and consequently

$$\mathcal{G}_j = (\boldsymbol{I} - \omega_j\boldsymbol{A})\boldsymbol{G}_{j-1}(\mathcal{N}(\boldsymbol{P}^H\boldsymbol{G}_{j-1})).$$

We assumed $(\boldsymbol{I} - \omega_j\boldsymbol{A})$ to be nonsingular, so

$$d_j = \dim(\mathcal{G}_j) = \dim(\mathcal{U}).$$

Now $\boldsymbol{P}^H\boldsymbol{G}_{j-1}$ is an $s \times d_{j-1}$ matrix; therefore

$$d_j = \dim(\mathcal{N}(\boldsymbol{P}^H\boldsymbol{G}_{j-1})) = d_{j-1} - \operatorname{rank}(\boldsymbol{P}^H\boldsymbol{G}_{j-1}). \tag{3.6}$$

On the other hand, $\operatorname{rank}(\boldsymbol{P}^H\boldsymbol{G}_{j-1}) = s - \dim(\mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P}))$; hence

$$d_j = d_{j-1} - s + l$$

with $l = \dim(\mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P})) \in [0, s]$. This proves that $0 \le d_{j-1} - d_j \le s$.

Now suppose $\boldsymbol{v} \in \mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P})$, $\boldsymbol{v} \ne \boldsymbol{0}$; then $\boldsymbol{P}\boldsymbol{v} \in \mathcal{N}(\boldsymbol{G}_{j-1}^H)$, and hence $\boldsymbol{P}\boldsymbol{v} \perp \mathcal{G}_{j-1}$. Since $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, this implies that $\boldsymbol{P}\boldsymbol{v} \perp \mathcal{G}_j$, and hence $\boldsymbol{v} \in \mathcal{N}(\boldsymbol{G}_j^H\boldsymbol{P})$. So $\mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P}) \subset \mathcal{N}(\boldsymbol{G}_j^H\boldsymbol{P})$, and therefore $\dim(\mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P})) \le \dim(\mathcal{N}(\boldsymbol{G}_j^H\boldsymbol{P}))$. It follows that

$$d_{j+1} = d_j - s + l'$$

with $l' = \dim(\mathcal{N}(\boldsymbol{G}_j^H\boldsymbol{P})) \ge l$. Therefore $d_j - d_{j+1} \le d_{j-1} - d_j$, which proves the theorem. ∎

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Remark. According to Theorem 3.1 the dimension reduction per step is between 0 and s. Zero reduction occurs only if $\mathcal{G}_j \subset \mathcal{N}(\boldsymbol{P}^H)$, which is highly improbable, as was remarked after Theorem 2.1. In practical situations the reduction is s, the maximal value. This can be understood by the following observation. According to (3.6) in the proof of Theorem 3.1, the dimension reduction equals the rank of the $s \times d_{j-1}$ matrix $\boldsymbol{P}^H \boldsymbol{G}_{j-1}$. The columns of $\boldsymbol{G}_{j-1}$ are linearly independent, because they are a basis for $\mathcal{G}_{j-1}$. The columns of $\boldsymbol{P}$ are independent by definition. If $\text{rank}(\boldsymbol{P}^H \boldsymbol{G}_{j-1}) < s$, then we must have $\boldsymbol{p}^H \boldsymbol{G}_{j-1} = \boldsymbol{0}^H$ for some nonzero $\boldsymbol{p} = \boldsymbol{P}\boldsymbol{c}$. Now, if $d_{j-1} > s$, then it is highly improbable that $\boldsymbol{p}$ can be made to satisfy these d_{j-1} relations, having only the s components of $\boldsymbol{c}$ as free parameters.

Unfortunately, this does not prove that $d_j = d_{j-1} - s$ "almost always," since the space $\mathcal{G}_{j-1}$, and therefore the matrix $\boldsymbol{G}_{j-1}$, is not constructed independently from the matrix $\boldsymbol{P}$, which is strongly involved in the construction procedure. It can be shown, however, that for a random choice of $\boldsymbol{P}$, $d_j - d_{j-1} < s$ will happen with zero probability.

If the dimension reduction per step is precisely s throughout the process, then we will speak of the *generic case*; otherwise we have the *nongeneric case*. In the nongeneric case we call $s - (d_{j-1} - d_j)$ the *deficiency* of the reduction. In Theorem 3.1 we have proved that the deficiency is nondecreasing during the process.

COROLLARY 3.2. *In the generic case IDR(s) requires at most* $N + \frac{N}{s}$ *matrix-vector multiplications to compute the exact solution in exact arithmetic.*

Observation of nongenericity. Can nongenericity be recognized during execution of the algorithm? If in some application the IDR(s) algorithm happens to be nongeneric, then for some j_0 we must have $\dim(\mathcal{G}_j \cap \mathcal{S}) < s$ for $j = j_0, j_0 + 1, \ldots$. The only way this can be observed is rank deficiency of the $s \times s$ matrices $\boldsymbol{P}^H \Delta \boldsymbol{R}_n$. However, rank deficiency is not an exclusive property of nongenericity. So it may happen that, after having produced, say, 100 vectors in $\mathcal{G}_j \cap \mathcal{S}$ spanning only a $(s-1)$-dimensional space, the 101th vector happens to be outside this subspace. Therefore a nongeneric case cannot be detected in practice. However, the example mentioned above is also a reason not to worry about nongenericity: rank deficiency is a serious problem anyway, whether we are in the generic case or not. We go into this in the next section.

4. Other IDR-based algorithms. The IDR(s) algorithm that we presented in section 3 is a direct translation of the IDR *theorem* into an actual *algorithm*. There is, however, considerable freedom in this translation. Different ways of using this freedom give *mathematically* different methods. In this section we indicate what freedom there is in developing an IDR-based algorithm and discuss some of the choices that can be made.

There are three elements of choice in the algorithmic translation of the IDR theorem. First, and perhaps the most fundamental, is the choice of the matrix $\boldsymbol{P}$ which defines the subspace $\mathcal{S}$. Second, there are different possible strategies for selecting the factors ω_j. Third, there are different ways to define and calculate the intermediate residuals $\boldsymbol{r}_{j(s+1)+k}$ for $k = 1, 2, \ldots, s$.

4.1. The choice of $\boldsymbol{P}$. Similar to the experiences with the early Lanczos and CG-type methods, algorithms of the IDR(s) family are finite in a structural way but behave like an iterative procedure as well. Only in the case of CG, applied to positive definite Hermitian matrices, do we have a rather complete convergence analysis, on the basis of which we can fine-tune the method to extremely high performance (by

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

preconditioning). The convergence analysis is based on the behavior of the zeros of the CG polynomials (Ritz values), in relation to the (active part of the) spectrum of $\boldsymbol{A}$.

However, if the matrix is not Hermitian, and may have complex eigenvalues, then the convergence analysis collapses.

In the cases of CGS and Bi-CGSTAB, part of the analysis still holds if the matrix is only moderately non-Hermitian and if the (initial) shadow residual, i.e., the starting vector for the left iterates in Bi-CG, is chosen equal to the initial residual. But also if the problem does not satisfy the necessary restrictions for maintaining "theoretical convergence," then the practical convergence often remains satisfactory.

Motivated by this "natural" choice for the shadow residual in the Bi-CG methods, we tried several ways to choose $\boldsymbol{P}$ in relation to the problem. Surprisingly, the IDR(s) algorithms with these "clever" choices of $\boldsymbol{P}$ performed poorly in many test problems. After extensive experimentation, we decided to choose the columns of $\boldsymbol{P}$ as orthogonalization of a set of random vectors. This choice is justified by robustness: in exact arithmetic there is a zero probability that the dimension reduction is less than s. In the comparison with Bi-CGSTAB, however, we use $\boldsymbol{p}_1 = \boldsymbol{r}_0$ in most of our experiments, because this is done in most implementations of Bi-CGSTAB (although originally in Bi-CGSTAB $\boldsymbol{p}$ could be chosen arbitrarily).

Although we advocate choosing $\boldsymbol{P}$ as orthogonalization of a set of random vectors on the basis of robustness, other suitable choices for the matrix $\boldsymbol{P}$ can be made on completely different grounds. For example, an interesting alternative is to choose the columns of $\boldsymbol{P}$ blockwise zero or nonzero, where the nonzero parts are nonoverlapping. This choice is inspired by the subdomain deflation technique described in, e.g., [10, 3] and has some clear computational advantages. First, computations like $\boldsymbol{P}^H \boldsymbol{r}_n$ become extremely cheap. The computation of these s inner products requires inner products of vectors of length N/s, hence the equivalence of one inner product of length N. Second, only the nonzero parts of the columns of $\boldsymbol{P}$ need to be stored; hence only space for N numbers is needed to store $\boldsymbol{P}$. Third, in parallel computing, if the vectors $\boldsymbol{p}_k$ are nonzero in only one subdomain, then the communication for the inner product $\boldsymbol{p}_k^H \boldsymbol{r}_i$ involves only the transmission of the local inner product to the other subdomains. No partial inner products need to be summed, since the other subdomains do not contribute to the result.

4.2. The choice of ω. In the previous section we suggested selecting ω such that the norm of $\boldsymbol{r}_{n+1}$ is minimized. This leads to

$$\omega = \frac{\boldsymbol{t}^H \boldsymbol{v}_n}{\boldsymbol{t}^H \boldsymbol{t}}, \quad \text{with } \boldsymbol{t} = \boldsymbol{A}\boldsymbol{v}_n.$$

As was remarked before, this may give $\omega \approx 0$, which yields a (near) breakdown of type 2. Most type 2 breakdowns can be repaired by methods as developed in [13]; i.e., the value of ω is increased if the angle between $\boldsymbol{A}\boldsymbol{v}_n$ and $\boldsymbol{v}_n$ is too small, a technique that we will also use in some numerical experiments.

In some problems, however, the ω calculations fail systematically, for example, if $\boldsymbol{A}$ behaves like a skew-symmetric matrix. For these cases Gutknecht proposed BiCGstab2 [6]. This technique uses quadratic stabilization polynomials. The BiCGstab(ℓ) method [12] of Sleijpen and Fokkema generalizes this idea to stabilization polynomials of degree ℓ. We have not yet found a similar possibility for the IDR(s) algorithms, since this would require a new variant of the IDR theorem. However, in our numerical experiments the problems vanish completely when we choose the matrix $\boldsymbol{P}$ *complex* rather than real.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

In some computational environments, for example in grid computing, the computation of an inner product can be very expensive. In such an environment it may even be an option to choose ω constant, for example, equal to one. Of course, for many problems this will result in a slower rate of convergence.

4.3. The computation of intermediate residuals. In our prototype algorithm presented in Figure 3.1, a new intermediate residual in $\mathcal{G}_{j+1}$ is constructed according to

$$\begin{aligned} &\text{Solve } \boldsymbol{c} \text{ from } (\boldsymbol{P}^H \Delta \boldsymbol{R}_n)\boldsymbol{c} = \boldsymbol{P}^H \boldsymbol{r}_n, \\ &\text{Calculate } \boldsymbol{v} = \boldsymbol{r}_n - \Delta \boldsymbol{R}_n \boldsymbol{c}, \\ &\text{Calculate } \boldsymbol{r}_{n+1} = \boldsymbol{v} - \omega_{j+1} \boldsymbol{A}\boldsymbol{v}. \end{aligned}$$

In this code fragment, $\boldsymbol{r}_n$ and the columns of $\Delta \boldsymbol{R}_n$ must be in $\mathcal{G}_j$, in order for $\boldsymbol{r}_{n+1}$ to be in $\mathcal{G}_{j+1}$. Moreover, the matrix $\boldsymbol{P}^H \Delta \boldsymbol{R}_n$ must be of rank s.

These requirements leave freedom to generate different intermediate residuals and residual differences with desirable properties, as we will explain below.

The choice of $\Delta \boldsymbol{R}_n$*.* In the prototype algorithm, $\Delta \boldsymbol{R}_n$ consists of the s most recent residual differences, and, when we construct the first residual $\boldsymbol{r}_{n+1}$ in $\mathcal{G}_{j+1}$, this is the only possibility (if no breakdown is happening). But in calculating $\boldsymbol{r}_{n+k}$, with $k > 1$, we have $s + k - 1$ vectors $\Delta \boldsymbol{r}_i \in \mathcal{G}_j$ at our disposal to construct a new intermediate residual.

One way to use this freedom is to generate a new matrix $\Delta \boldsymbol{R}_{j+1}$, with all of its columns in $\mathcal{G}_{j+1}$ (hence the change in subscript) using the matrix $\Delta \boldsymbol{R}_j$ with all of its columns in $\mathcal{G}_j$. That is, we reuse the matrices $\Delta \boldsymbol{R}_j$ and $\boldsymbol{P}^H \Delta \boldsymbol{R}_j$ during the steps to generate the intermediate residuals. Clearly, this approach increases the amount of storage, but the advantage is that the conditioning of the matrix $\boldsymbol{P}^H \Delta \boldsymbol{R}_{j+1}$ can be controlled. If we assume that $\boldsymbol{P}^H \Delta \boldsymbol{R}_j$ is well-conditioned, then every new vector $\Delta \boldsymbol{r}_{n+1}$ is computed without the possibility that breakdown of type 1 occurs. If during this process a vector $\Delta \boldsymbol{r}_{n+1}$ is generated such that $\boldsymbol{P}^H \Delta \boldsymbol{R}_{j+1}$ becomes ill-conditioned, then this vector is simply not used as a column in $\Delta \boldsymbol{R}_{j+1}$. This means in practice that a dimension reduction step may be postponed a few iterations, until a set of s columns of $\Delta \boldsymbol{R}_{j+1}$ is generated such that $\boldsymbol{P}^H \Delta \boldsymbol{R}_{j+1}$ is well-conditioned.

However, in the *nongeneric* case the dimension reduction is less than s for the remainder of the iterative process, and this cannot be cured by postponing the dimension reduction step. A work-around for both the generic and the nongeneric case is to reduce the number of vectors in $\boldsymbol{P}$ after a fixed number of unsuccessful iteration steps. This defines a new subspace $\mathcal{S}$, but this subspace includes the original $\mathcal{S}$; hence the dimension reductions from the past are not lost, and only future dimension reductions will correspond to the smaller rank of the reduced matrix $\boldsymbol{P}$.

Orthogonalization of the vectors in $\mathcal{G}_j$*.* In order to be able to update the solution with the residual we use residual difference vectors in the prototype algorithm. This is, however, not the only possibility. If we have a matrix $\boldsymbol{G}_n$ with columns in $\mathcal{G}_j$ and a corresponding matrix $\boldsymbol{U}_n$ such that $\boldsymbol{A}\boldsymbol{U}_n = -\boldsymbol{G}_n$, then it is also possible to update the solution together with the residual. For example, if new vectors in $\mathcal{G}_{j+1}$ are computed using a fixed matrix $\Delta \boldsymbol{R}_j$ as described above, then we could choose to orthogonalize the columns of $\Delta \boldsymbol{R}_j$ for stability reasons. Since all of the columns of $\Delta \boldsymbol{R}_j$ are in $\mathcal{G}_j$, any linear combination of the columns of $\Delta \boldsymbol{R}_j$ is also in $\mathcal{G}_j$. In practice, this orthogonalization should be done while computing a new vector in $\mathcal{G}_{j+1}$. This allows us to monitor if a new vector in $\boldsymbol{G}_{j+1}$ is (nearly) dependent on

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

the previously computed columns, which would result in an ill-conditioned matrix $\boldsymbol{P}^H\boldsymbol{G}_{j+1}$. Moreover, having computed $k < s$ orthogonal columns of $\boldsymbol{G}_{j+1}$, the norm of the kth intermediate residual can be minimized by making it orthogonal to the first k columns of $\boldsymbol{G}_{j+1}$.

We have developed an algorithm with the above properties. The drawback of this algorithm is that it requires considerably more storage and vector operations than the prototype IDR(s) method. It shows, however, the flexibility in developing an IDR-based method.

5. Polynomial issues. Because of the Krylov nature of the IDR(s) algorithms, a polynomial analysis may give some insight into the properties of these algorithms. Concentrating on the residuals we have

$$\boldsymbol{r}_n = \Phi_n(\boldsymbol{A})\boldsymbol{r}_0, \quad \Phi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}, \quad \Phi_n(0) = 1. \tag{5.1}$$

Similar to CGS and Bi-CGSTAB, the algorithm can be interpreted as a construction method for the polynomials Φ_n, and, what is important, the algorithmic requirements can be translated into *relations between polynomials*. This interpretation can be used to answer questions of uniqueness and, of course, for determining possible relationship with other Krylov methods. In particular the relation between IDR(1) and Bi-CGSTAB, but also between IDR(s) and the ML(k)BiCGSTAB method [18], will be investigated.

5.1. Uniqueness of $\boldsymbol{r}_{(s+1)j}$. Let $\boldsymbol{r}_n \in \mathcal{G}_j$ for some $j > 0$; then $\boldsymbol{r}_n = \boldsymbol{r}' - \omega_j\boldsymbol{A}\boldsymbol{r}'$ for some $\boldsymbol{r}' \in \mathcal{G}_{j-1} \cap \mathcal{S}$. Similarly, $\boldsymbol{r}' = \boldsymbol{r}'' - \omega_{j-1}\boldsymbol{A}\boldsymbol{r}''$ for some $\boldsymbol{r}'' \in \mathcal{G}_{j-2} \cap \mathcal{S}$. Going on like this we arrive at

$$\boldsymbol{r}_n = \Omega_j(\boldsymbol{A})\boldsymbol{w}, \tag{5.2}$$

where $\boldsymbol{w} \in \mathcal{G}_0 \cap \mathcal{S}$, and where the polynomial Ω_j is defined by

$$\Omega_j(t) = (1 - \omega_j t)(1 - \omega_{j-1}t)\cdots(1 - \omega_1 t), \quad \Omega_0(t) \equiv 1. \tag{5.3}$$

Obviously, $\Omega_l(\boldsymbol{A})\boldsymbol{w} \in \mathcal{G}_l \cap \mathcal{S}$ for $l = 0, 1, \ldots, j-1$; therefore the following j vectorial relations must be satisfied by $\boldsymbol{w}$:

$$\boldsymbol{P}^H\Omega_l(\boldsymbol{A})\boldsymbol{w} = \boldsymbol{0}, \quad l = 0, 1, \ldots, j-1. \tag{5.4}$$

According to (5.1) and (5.2), and since $\boldsymbol{w}$ is in the Krylov space $\mathcal{G}_0$, $\boldsymbol{w}$ can be written as

$$\boldsymbol{w} = \Psi_{n-j}(\boldsymbol{A})\boldsymbol{r}_0. \tag{5.5}$$

So we have $\Phi_n = \Omega_j\Psi_{n-j}$, and the residuals can be written as

$$\boldsymbol{r}_n = \Omega_j(\boldsymbol{A})\Psi_{n-j}(\boldsymbol{A})\boldsymbol{r}_0. \tag{5.6}$$

Now (5.4) represents relations between the coefficients of Ψ_{n-j}. Splitting $\boldsymbol{P}$ into columns, the relations (5.4) read

$$\boldsymbol{p}_k^H\Omega_l(\boldsymbol{A})\Psi_{n-j}(\boldsymbol{A})\boldsymbol{r}_0 = 0, \quad k = 1, 2, \ldots, s, \quad l = 0, 1, \ldots, j-1. \tag{5.7}$$

Together with the requirements $\Psi_{n-j}(0) = 1$, this represents an inhomogeneous system of $sj + 1$ equations in $n - j + 1$ unknowns. The vectors $\boldsymbol{p}_k$, $k = 1, 2, \ldots, s$, are

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

chosen arbitrarily, not in relation to the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Therefore it is a true exception if these relations can be satisfied if $n - j + 1 < sj + 1$. So, again generally speaking, we must have $n \geq (s+1)j$ for $\boldsymbol{r}_n$ to be in $\mathcal{G}_j$. Therefore we can assume that, independent of the strategy chosen for finding enough vectors in $\mathcal{G}_j$, the first residual in $\mathcal{G}_j$, $\boldsymbol{r}_{(s+1)j} = \Omega_j(\boldsymbol{A})\Psi_{sj}(\boldsymbol{A})\boldsymbol{r}_0$, is uniquely determined by

$$\boldsymbol{p}_k^H \Omega_l(\boldsymbol{A})\Psi_{sj}(\boldsymbol{A})\boldsymbol{r}_0 = 0, \quad k = 1, 2, \ldots, s, \quad l = 0, 1, \ldots, j-1, \tag{5.8}$$
$$\Psi_{sj}(0) = 1.$$

5.2. Relation between IDR(s), Bi-CGSTAB, and ML(k)BiCGSTAB. The relations in (5.7) can be interpreted as formal orthogonality relations for the polynomial Ψ_{n-j}. Define s formal inner products on the space of polynomials as follows:

$$[\varphi, \psi]_k = \boldsymbol{p}_k^H \phi(\boldsymbol{A})\psi(\boldsymbol{A})\boldsymbol{r}_0, \quad k = 1, 2, \ldots, s. \tag{5.9}$$

Then (5.7) can be written as

$$[\Omega_l, \Psi_{n-j}]_k = 0, \quad k = 1, 2, \ldots, s, \quad l = 0, 1, \ldots, j-1, \tag{5.10}$$

and this is equivalent to formal orthogonality of Ψ_{n-j} to all polynomials in $\mathbb{P}^{j-1}$, *with respect to the s inner products* $[\,.\,,\,.\,]_k$.

Comparing to Bi-CGSTAB. Since Bi-CGSTAB is historically related to the original IDR method, we first investigate the case $s = 1$. In this case only classic theory on orthogonal polynomials plays a role. We have $sj = j$, and

$$[\Omega_l, \Psi_j] = 0, \quad l = 0, 1, \ldots, j-1,$$

with $[\phi, \psi] = \boldsymbol{p}^H \phi(\boldsymbol{A})\psi(\boldsymbol{A})\boldsymbol{r}_0$. So, independent of the choices for ω_l in the algorithm, the "Ψ-part" of the polynomial will be the unique orthogonal polynomial of degree j, with respect to this formal inner product, and be unity in the origin. *This is exactly the Bi-CG-polynomial.*

The remark in [15] about the mathematical equivalence between the old IDR and Bi-CGSTAB is true only for the "even IDR residuals" $\boldsymbol{r}_{2j}$. Bi-CGSTAB as described in [15] does not compute intermediate (i.e., "odd") residuals. This is because Bi-CGSTAB must calculate the Bi-CG coefficients α_j and β_j, and this calculation is completely dictated by the classical organization of the algorithm with *residuals and search directions.*

We can also think the other way round. The Bi-CG method produces residuals $\widetilde{\boldsymbol{r}}_n = \varphi_n(\boldsymbol{A})\boldsymbol{r}_0$, and search directions $\widetilde{\boldsymbol{p}}_n = \psi_n(\boldsymbol{A})\boldsymbol{r}_0$, linked together by beautiful formulas, in which the well-known coefficients α and β play an essential role. For our comparison, one relation is of importance:

$$\widetilde{\boldsymbol{r}}_{n+1} = \widetilde{\boldsymbol{r}}_n - \alpha_n \boldsymbol{A}\widetilde{\boldsymbol{p}}_n.$$

In Bi-CGSTAB, the vectors $\boldsymbol{r}_n = \Omega_n(\boldsymbol{A})\widetilde{\boldsymbol{r}}_n$ and $\boldsymbol{v}_n = \Omega_n(\boldsymbol{A})\widetilde{\boldsymbol{p}}_n$ play a role.[2] Furthermore, the vectors $\boldsymbol{r}_n$ and $\boldsymbol{A}\boldsymbol{v}_n$ are made orthogonal to a fixed "shadow residual." With respect to the IDR philosophy, this implies that both $\boldsymbol{r}_n$ and $\boldsymbol{A}\boldsymbol{v}_n$ are in $\mathcal{G}_n$.

Indeed, instead of producing $s + 1$ residuals in $\mathcal{G}_j$, we can also produce only one residual, and s, "search directions" in $\mathcal{G}_j$, and we get a genuinely different variant of

[2] The numbering differs from that in [15].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

IDR(s). The authors have implemented this variant, and tested it, and the outcome was a nearly as stable algorithm. There is, however, a drawback. It is slightly more expensive in vector operations, and it does not produce intermediate residuals. So we can decide to stop only after every $s+1$ steps. Trying to retrieve intermediate residual information is rather expensive and complicated.

Bi-CGSTAB is the first example of a "search-direction" variant of IDR(s). Some implementations, for instance those of MATLAB, produce intermediate residuals. These vectors, however, are reliable purely by accident: some part of the update is qualified as intermediate residual, but it could as well have been dropped.

Comparing to ML(k)BiCGSTAB. The ML(k)BICGSTAB algorithm is a transpose free method, based on a Lanczos-type method with multiple left starting vectors, which has drawn relatively little attention. Like IDR(s), it is theoretically a finite termination method, which terminates after at most $N + \frac{N}{k}$ matrix-vector multiplications.

A complete comparison with ML(k)BICGSTAB is much more difficult than with Bi-CGSTAB, since the derivations and implementations of both methods differ very much. But to a certain extent a relation can be established. According to its derivation in [18], the residuals in the ML(k)BiCGSTAB algorithm, with k replaced by s, and j replaced by $j-1$, satisfy

$$\widehat{\boldsymbol{r}}_{(j-1)s+i} = \Omega_j(\boldsymbol{A})\widetilde{\Psi}_{(j-1)s+i}(\boldsymbol{A})\boldsymbol{r}_0, \quad j = 1, 2, \ldots, \quad i = 1, 2, \ldots, s, \tag{5.11}$$

with $\widetilde{\Psi}_{(j-1)s+i} \in \mathbb{P}^{(j-1)s+i}$, satisfying

$$\widehat{\boldsymbol{p}}_n^H \widetilde{\Psi}_{(j-1)s+i}(\boldsymbol{A})\boldsymbol{r}_0 = 0, \quad n = 1, 2, \ldots, (j-1)s+i, \tag{5.12}$$

in which the vectors $\widehat{\boldsymbol{p}}_n$ are defined by

$$\widehat{\boldsymbol{p}}_{j's+i'} = \left(\boldsymbol{A}^H\right)^{j'} \boldsymbol{p}_{i'}, \quad j' = 0, 1, \ldots, \quad i' = 1, 2, \ldots, s. \tag{5.13}$$

The vectors $\widetilde{\boldsymbol{r}}_n = \widetilde{\Psi}_n(\boldsymbol{A})\boldsymbol{r}_0$ are the so-called *ML(s)BiCG* residuals, similar to ordinary BiCG residuals but constructed using multiple left starting vectors.

If we choose $i = s$ in formula (5.11), then we get

$$\widehat{\boldsymbol{r}}_{js} = \Omega_j(\boldsymbol{A})\widetilde{\Psi}_{js}(\boldsymbol{A})\boldsymbol{r}_0$$

with $\widetilde{\Psi}_j$ determined by (5.12) and (5.13). Now since both $\{\Omega_j\}_{j=0}^l$ and $\{t^j\}_{j=0}^l$ are bases for the space $\mathbb{P}^l$, (5.12) *and* (5.13) *are equivalent to the relations in* (5.8).

Therefore, if $\boldsymbol{r}_0$, $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s$, and the numbers ω_l are chosen the same in both methods, we must have

$$\boldsymbol{r}_{js+j} = \widehat{\boldsymbol{r}}_{js}$$

for $j = 0, 1, \ldots$.

Yet there are essential differences.

1. ML(k)BiCGSTAB is considerably more complicated to implement and more costly in storage and vector operations. The operation count for the main operations to perform a full cycle of $s+1$ IDR(s) steps yields $(s+1)$ matrix-vector products, $s^2 + s + 2$ inner products, and $2s^2 + \frac{7}{2}s + \frac{5}{2}$ vector updates. In comparison, the operations for a full cycle of s ML(k)BiCGSTAB iterations are [18] $(s+1)$ matrix-vector products, $s^2 + 2s$ inner products, and

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

$\frac{5}{2}s^2 + \frac{11}{2}s + 1$ vector updates. Also the (vector) storage requirements are higher for ML(k)BiCGSTAB: IDR(s) needs space for $5 + 3s$ vectors, whereas ML(k)BiCGSTAB needs space for $4 + 4s$ vectors.

2. The IDR(s) residual $\boldsymbol{r}_{js+j}$ is the first one in $\mathcal{G}_j$. The value ω_j is determined just before the calculation of this residual. Then this ω_j is used for the following s steps.
   In ML(s)BiCGSTAB, on the other hand, the residual $\widehat{\boldsymbol{r}}_{js} = \Omega_j(\boldsymbol{A})\widetilde{\Psi}_{js}(\boldsymbol{A})\boldsymbol{r}_0$ is *the last one with the factor* Ω_j *involved*. The next s residuals $\widehat{\boldsymbol{r}}_{js+i}$ carry the factor Ω_{j+1}, according to (5.11). The algorithmic step in which ω_j was calculated in ML(s)BiCGSTAB is performed far before it is done in IDR(s). So it is impossible to arrive at the same ω_j values, unless they are chosen constant or according to some other predefined rule.
3. The third difference is in the other residuals. In ML(k)BiCGSTAB, the requirements for the residuals are very severe: the polynomials $\widetilde{\Psi}_n$ must all represent the ML(k)BiCG residuals, implying that a lot of specific inner products are zero. In the IDR(s) algorithm only the first residual in a new $\mathcal{G}_{j+1}$ space happens to be related to these polynomials, whereas the intermediate residuals (for "filling" the space) can be chosen in many different ways, as long as the residuals appear in $\mathcal{G}_{j+1}$.
4. The most crucial difference between both methods is in the different approaches for deriving them. The IDR approach offers freedom for algorithmic variants that are not possible in a (multi-)Lanczos approach. This is illustrated by the relatively simple way in which type-1 breakdowns can be repaired, as is described in section 4.3. In IDR(s) we can continue producing extra vectors in $\mathcal{G}_j$ until a sufficiently well-conditioned subset is obtained to produce an element of $\mathcal{G}_{j+1}$. The formal relationship between both methods then vanishes completely, since the polynomial analysis in section 5.1 is no longer valid.

As far as we can now see, neither algorithm can be considered as a variant of the other.

6. Numerical examples. In this section we consider four different examples. The first example is one-dimensional and is included to confirm the theoretical properties of the algorithm. The other three are more realistic and are typical for three different problem classes.

We have performed the experiments with MATLAB 6.5 and have used the standard MATLAB implementation of Bi-CGSTAB, Bi-CG, CGS, and QMR. The tests with BiCGstab(ℓ) have been performed with the MATLAB code of Sleijpen.[3]

6.1. A one-dimensional convection-diffusion problem. The first example we discuss is a one-dimensional convection-diffusion problem. With this academic example we illustrate the termination behavior of IDR(s) as predicted by Theorem 3.1. Moreover, this example also illustrates the correspondence in the convergence behavior of Bi-CGSTAB and IDR(1).

The test problem is the finite difference discretization of the following differential equation:

$$-\frac{d^2u}{dx^2} + w\frac{du}{dx} = 0, \quad x \in (0,1),$$

[3] http://www.math.uu.nl/people/sleijpen/.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

with boundary conditions $u(0) = u(1) = 1$. The convection parameter w is chosen such that $\frac{wh}{2} = 0.5$, in which h is the mesh size. We have taken a total of 60 grid points, excluding the boundary nodes, which yields for the grid size $h = \frac{1}{61}$. Central differences are used for both the convection and the diffusion term.

We have solved the system with four (unpreconditioned) variants of IDR(s) using for s the values 1, 2, 4, and 6. As the initial guess the nullvector was chosen. For the columns of $\boldsymbol{P}$ we took the orthogonalization of $s-1$ random vectors, complemented with the initial residual. To investigate the stagnation level of the different methods, each iterative process is continued until no further reduction of the true residual norm is achieved.

The system consists of 60 equations; hence according to Theorem 3.1 the IDR(s) methods should terminate (in exact arithmetic) at the exact solution within 120, 90, 75, and 70 matrix-vector products (matvecs), respectively. Figure 6.1 displays for the four methods the norm of the true residual (scaled by the norm of the right-hand side vector) as a function of the number of matvecs. The figure also shows the convergence curves for full GMRES and Bi-CGSTAB. Note that in exact arithmetic GMRES should terminate within 60 matvecs and Bi-CGSTAB within 120 matvecs.

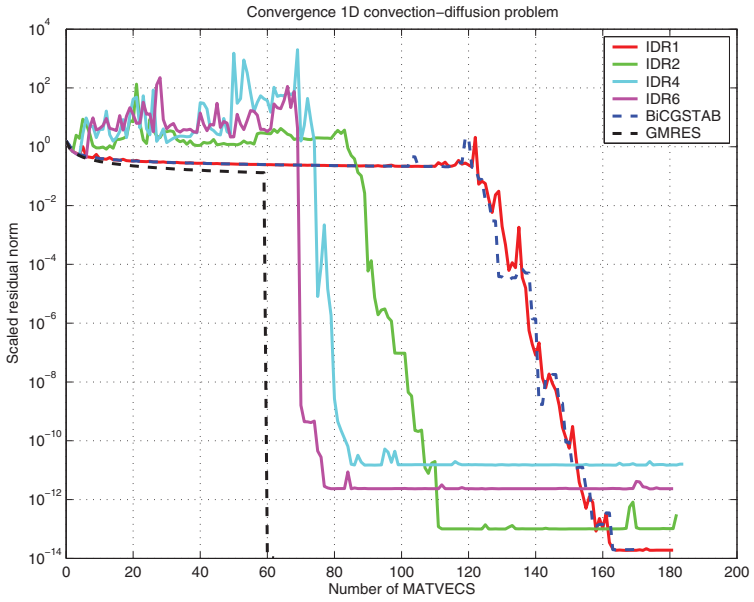

FIG. 6.1. *Finite termination of IDR(s), Bi-CGSTAB, and GMRES.*

The figure clearly shows for all the methods a sharp drop of the residual norm around the point where termination of the algorithm should occur. Also the convergence curves of IDR(1) and BiCGSTAB are essentially the same, which confirms the fact that in exact arithmetic the residual norms of the two methods should be the same at the even steps. The norms of the true residuals of all of the methods stagnate at a level close to machine precision, although IDR(4) and IDR(6) stagnate at a slightly higher level than the other methods. This difference can be attributed to the peaks in the residual norms in the initial iterations.

In this example, we investigated the property that IDR(s) is a finite termination method. In the next examples IDR(s) will be used as an iterative method; i.e., we

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

want to compute a sufficiently accurate approximation to the solution in far fewer iterations than needed to reach the point where termination at the exact solution should occur.

6.2. An example from oceanography. The second example that we discuss is a convection-diffusion problem from oceanography. This realistic example is typical for a wide class of problems encountered in CFD. The system matrices of this kind of problem are real and nonsymmetric, with eigenvalues that have a positive real part and a small (or zero) imaginary part. Bi-CGSTAB is often quite efficient for this type of problem.

Steady barotropic flow in a homogeneous ocean with constant depth and in near equilibrium can be described by the following partial differential equations:

$$-r\,\Delta\psi - \beta\,\frac{\partial\psi}{\partial x} - = (\nabla \times \mathbf{F})_z \quad \text{in } \Omega.$$

Here, Δ is the Laplace operator, ψ is the stream function, and $\mathbf{F}$ is the external force field caused by the wind stress $\boldsymbol{\tau}$ divided by the average depth of the ocean H times the water density ρ:

$$\mathbf{F} = \frac{\boldsymbol{\tau}}{\rho H}\,. \tag{6.1}$$

The other parameters are the bottom friction coefficient r and the Coriolis parameter β. The zero normal velocity boundary condition implies that the stream function is constant on continent boundaries:

$$\psi = C_k \quad \text{on } \Gamma_k, \quad k = 1, \ldots, K, \tag{6.2}$$

where K is the number of continents. The values of the constants C_k are a priori unknown. In order to determine them one has to impose integral conditions, stating that the water level is continuous around each island or continent:

$$\oint_{\Gamma_k} r\,\frac{\partial\psi}{\partial n}\,ds = -\oint_{\Gamma_k} \mathbf{F}\cdot\mathbf{s}\,ds. \tag{6.3}$$

The equations are commonly expressed in spherical coordinates to map the physical domain onto a rectangular domain. The coordinate transformation causes singularities on the poles. The singularity at the South Pole gives no problem since the South Pole is land. The singularity at the North Pole is solved by imposing the Dirichlet condition $\psi = 0$ on the North Pole.

The values for the physical parameters, which are taken from [16], are listed below:

- wind stress $\boldsymbol{\tau}$: long term averaged data for January [7],
- average depth $H = 500\,\mathrm{m}$,
- water density $\rho = 1000\,\mathrm{kg/m^3}$,
- earth radius $R = 6.4 \cdot 10^6\,\mathrm{m}$,
- coriolis parameter $\beta = 2.3 \cdot 10^{-11}\cos\theta\,(\mathrm{ms})^{-1}$,
- bottom friction coefficient $r = 5 \cdot 10^{-6}\,\mathrm{s}^{-1}$.

The above problem has been discretized with the technique described in [16]. The solution is plotted in Figure 6.2.

The resulting system consists of 42,248 equations. The matrix is nonsymmetric, but has a positive definite symmetric part, meaning that all eigenvalues are in the right-half plane. The number of nonzeros in the matrix is almost 300,000. As the preconditioner we use ILU(0), which we apply symmetrically.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

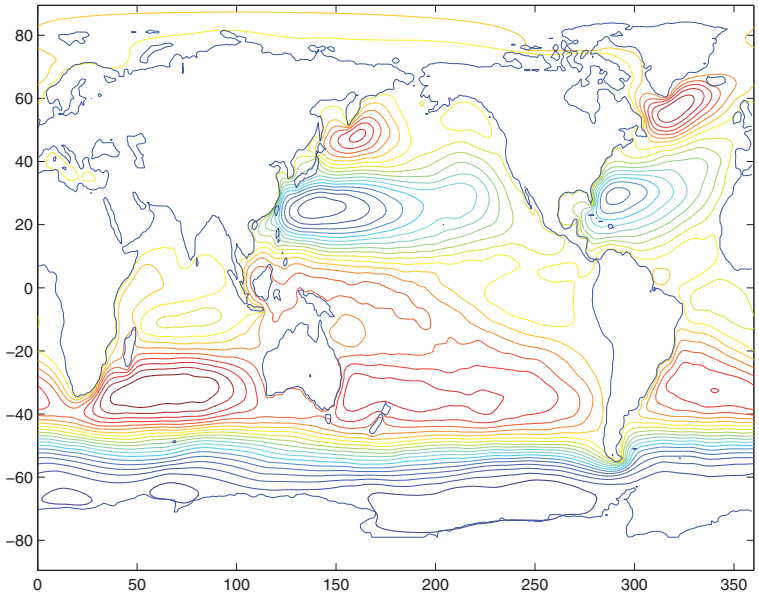

FIG. 6.2. *Solution of the ocean problem.*

The resulting system is solved with the IDR variants IDR(1), IDR(2), IDR(4), and IDR(6). For comparison we have also solved the system with Bi-CGSTAB and with full GMRES. We stress that full GMRES is not a limited memory method. In each iteration a new orthonormal basis vector for the Krylov subspace is computed and stored, which makes the method quite expensive, both with respect to memory and with respect to computations. GMRES is included only because it is optimal with respect to the number of matrix-vector multiplications. Since GMRES is optimal in this sense, none of the other methods can converge faster with respect to the number of matvecs. It is therefore quite interesting to determine how close the convergence curves of the other (limited memory) methods are to the optimal convergence curve of GMRES.

In order to assess the numerical accuracy of the methods we compute in each iteration the true residual of the (preconditioned) system, and we continue the iterative process until the stagnation level has been reached. The convergence curves of the different methods are plotted in Figure 6.3. Although in this example the methods are used as iterative techniques, rather than as finite termination methods as in the previous example, there is considerable qualitative agreement in the behavior of the methods for the two examples. We make the following observations:

- The required number of matrix-vector multiplications decreases if s is increased. The convergence curves of IDR(4) and IDR(6) are close to the optimal convergence curve of GMRES.
- The convergence curves of IDR(1) and Bi-CGSTAB agree well. The other variants of IDR(s) converge significantly faster than Bi-CGSTAB.
- The (scaled) norm of the true residual of all methods except GMRES stagnates at a level between 10^{-10} and 10^{-12}. GMRES stagnates near machine precision, but to achieve this extra accuracy an orthonormal set of basis vectors has to be computed and stored. This is for most applications prohibitively expensive, and the gain in precision is for most practical applications unim-

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

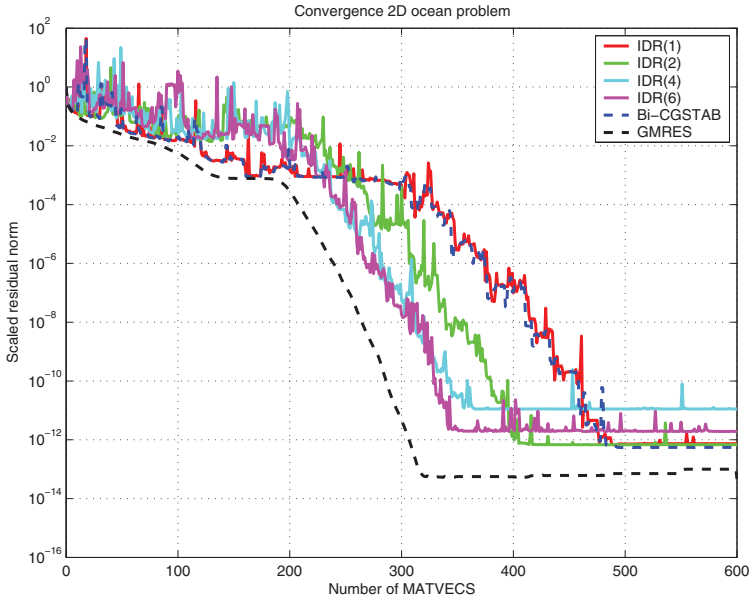

FIG. 6.3. *Convergence for the ocean problem of IDR(s), Bi-CGSTAB, and GMRES.*

TABLE 6.1
Number of matrix-vector multiplications to solve the ocean problem such that the (true) norm of the scaled residual is less than 10^{-8}.

Method	Number of matvecs
GMRES	265
Bi-CG	638
QMR	624
CGS	stagnates
Bi-CGSTAB	411
IDR(1)	420
IDR(2)	339
IDR(4)	315
IDR(6)	307
BiCGstab(1)	420
BiCGstab(2)	424
BiCGstab(4)	424
BiCGstab(8)	432

portant.

In order to make a more quantitative comparison, we have checked for each of the methods after how many matvecs the norm of the scaled residual drops below 10^{-8}. The results are tabulated in Table 6.1. This table also includes the results for BiCGstab(ℓ), Bi-CG, QMR, and CGS. The results in this table clearly show that IDR(s) outperforms the other limited memory methods with respect to the number of matvecs, in particular for higher values of s. The IDR(6) variant is close to optimal with respect to the number of matvecs. The difference with full GMRES is 42, which is only about 15% more than the minimum possible. For comparison, Bi-CGSTAB takes 411 matvecs, or 50% more than the minimum.

We did not tabulate the computing times for this example since the standard

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

MATLAB routines are not as optimized with respect to efficiency as our own IDR(s) routine.

6.3. A three-dimensional convection-dominated problem. The next test problem is rather academic and is taken from [12]. This problem was proposed as an example for which Bi-CGSTAB does not work well, due to the strong nonsymmetry of the system matrix. Specifically, the problem is caused by the fact that the matrix has eigenvalues with large imaginary parts. We recall that Bi-CGSTAB is a combination of linear minimal residual steps and Bi-CG steps. Bi-CGSTAB does not work well for this type of problem because the linear minimal residual steps produce a polynomial Ω_l (cf. section 5.2) that is a product of real linear factors. Consequently, Ω_l has real roots, and hence is unsuited as a residual minimizing polynomial, which should have roots close to the eigenvalues. Analogously, IDR(s) is a combination of linear minimal residual steps and IDR reduction. The linear minimal residual steps generate the same type of polynomial Ω_l as Bi-CGSTAB. It is therefore interesting to see if IDR(s) also performs poorly for this problem and, if so, to examine possible remedies.

The test problem is the finite difference discretization of the following partial differential equations on the unit cube $[0\,,\,1] \times [0\,,\,1] \times [0\,,\,1]$ with Dirichlet boundary conditions:

$$u_{xx} + u_{yy} + u_{zz} + 1000u_x = F.$$

The vector F is defined by the solution $u(x, y, z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$. The partial differential equation is discretized using central differences for both the convection and the diffusion terms. We take 52 grid points in each direction (including boundary points), which yields a system of 125,000 equations.

We have solved this problem with IDR(1), IDR(2), IDR(4), IDR(6), Bi-CGSTAB, and GMRES. For the the columns of $\boldsymbol{P}$ space we take our standard choice, i.e., the orthogonalization of the initial residual complemented with $s-1$ real random vectors. No preconditioner is applied. The iterative process is terminated once the residual norm, divided by the norm of the right-hand side vector, drops below 10^{-8}. The convergence test is performed on the recursively computed residual, as would be the case in practice. At the end of the process a check is performed if the norm of the true residual matches that of the recursively updated residual, which was the case for all tests we present here. Figure 6.4 shows the convergence behavior of the different methods. The figure shows the poor convergence behavior of Bi-CGSTAB for this problem, as can also be expected for IDR(1). No convergence is achieved for both methods within 2000 matvecs. Increasing s significantly improves the convergence behavior of IDR(s). However, compared with the optimal convergence of GMRES, the rate of convergence is still rather poor. We have tabulated in Table 6.2 for each method the required number of matvecs to achieve the desired accuracy. This table also includes the results for CGS, Bi-CG, and QMR. We note that the results of Bi-CG and QMR are quite satisfactory. These methods do not use linear minimal residual steps. CGS does not converge due to the well-known lack of robustness of this method.

As was remarked before, the disappointing convergence behavior of both Bi-CGSTAB and IDR(s) can be attributed to the poor performance of the minimal residual step in the algorithms. Sleijpen and Fokkema [12] overcome this problem by combining Bi-CG with higher-order minimal residual polynomials, thus creating a polynomial Q_l that admits complex roots. This has given rise to the elegant Bi-CGstab(ℓ) method. Figure 6.5 shows the convergence of this method, and Table 6.2

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

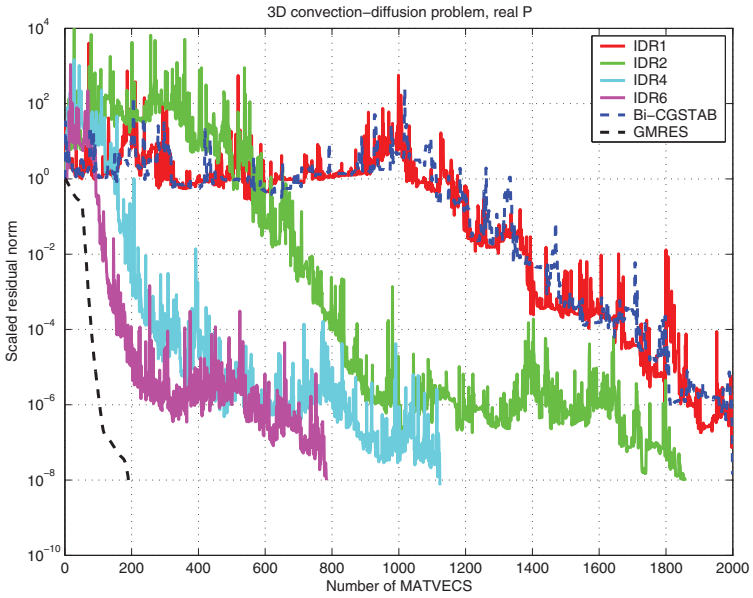

FIG. 6.4. *Convergence for IDR(s) (with real* $\boldsymbol{P}$*), Bi-CGSTAB, and GMRES.*

TABLE 6.2

Number of matrix-vector multiplications to solve the three-dimensional convection-diffusion problem such that the (true) norm of the scaled residual is less than 10^{-8}.

Method	Number of matvecs
GMRES	191
BiCG	454
QMR	450
CGS	n.c.
Bi-CGSTAB	n.c.
IDR(1)	n.c.
IDR(2)	1858
IDR(4)	1125
IDR(6)	784
BiCGstab(1)	n.c.
BiCGstab(2)	252
BiCGstab(4)	216
BiCGstab(8)	224

gives the required number of matrix-vector multiplications. The improvement in the convergence is quite spectacular; the required number of matvecs for BiCGstab(2) drops to 252, and for BiCGstab(4) and BiCGstab(8) to 216 and 224, respectively, which is very close to 191, the number of matvecs for GMRES.

As was remarked in section 4, it is not obvious how to derive an IDR variant that uses higher-order minimal residual steps. There is, however, another solution to this problem: by choosing $\boldsymbol{P}$ *complex*, instead of real, the polynomial Q_l can have complex roots. Following this idea we have rerun the example using the orthogonalization of randomly chosen complex vectors for the columns of $\boldsymbol{P}$. The convergence of the IDR(s) methods is shown in Figure 6.6. Clearly, choosing $\boldsymbol{P}$ complex also solves the convergence problem: the number of IDR(6) iterations is 242, only slightly more than

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

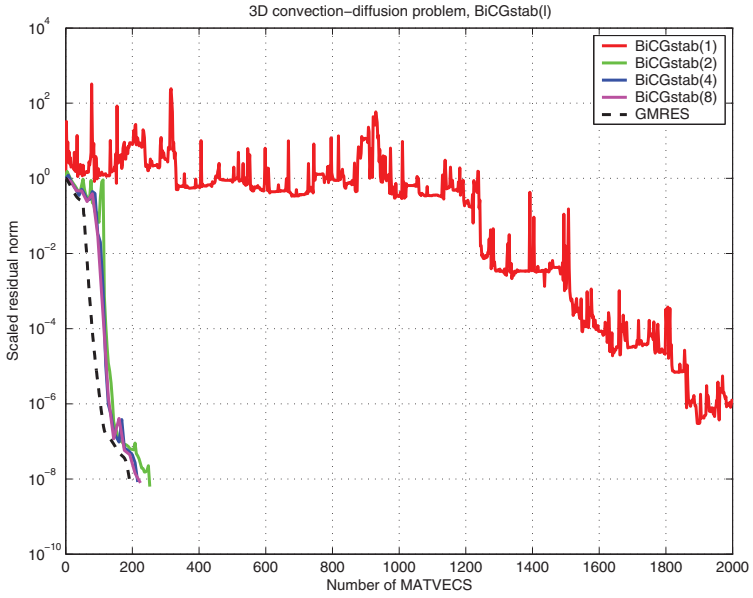

FIG. 6.5. *Convergence for BiCGstab(ℓ).*

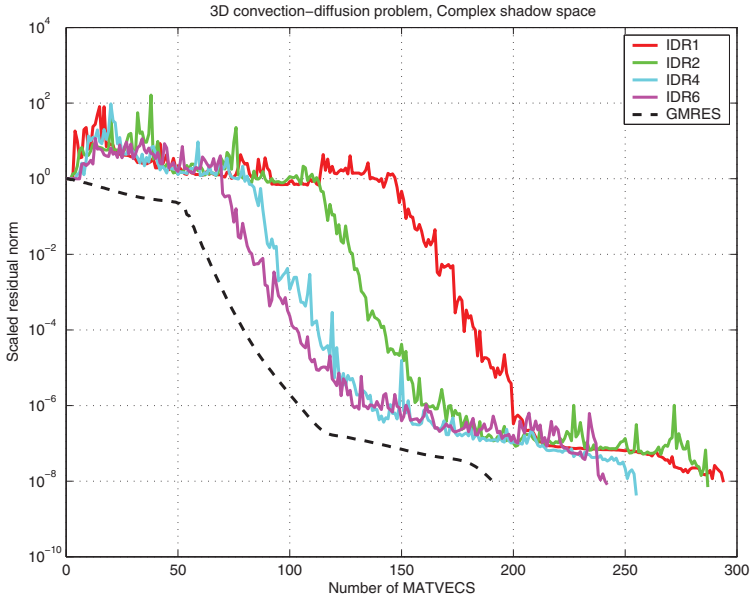

FIG. 6.6. *Convergence of IDR(s) with complex* $\boldsymbol{P}$.

for Bi-CGstab(8). This is, however, at the price of turning a real computation into a complex computation.

6.4. A three-dimensional Helmholtz problem. As our last example we consider sound propagation in a room of dimension $4 \times 4 \times 4m^3$. If the sound source is

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

harmonic, then the acoustic pressure field has the factored form

$$p(\boldsymbol{x},t) = \widehat{p}(\boldsymbol{x})e^{2\pi i f t}. \tag{6.4}$$

The pressure function $\widehat{p}$ can be determined from the so-called Helmholtz equation, which is given by

$$\frac{-(2\pi f)^2}{c^2}\widehat{p} - \Delta\widehat{p} = \delta(\boldsymbol{x}-\boldsymbol{x}_s) \quad \text{in } \Omega. \tag{6.5}$$

Here, Δ is the Laplace operator, c is the sound speed (approximately 340 m/s in air), and $\delta(\boldsymbol{x}-\boldsymbol{x}_s)$ represents a harmonic point source that is located at $\boldsymbol{x}_s$, which is in the center of the room. Five of the walls are reflecting, which are modelled by the boundary condition

$$\frac{\partial\widehat{p}}{\partial n} = 0, \tag{6.6}$$

whereas the remaining wall is sound absorbing, which is modelled by

$$\frac{\partial\widehat{p}}{\partial n} = -\frac{2\pi i f}{c}\widehat{p}. \tag{6.7}$$

The above problem is discretized with the finite element method using linear tetrahedral elements on a grid with grid size $h = 8$ cm. The resulting system is given by

$$[-(2\pi f)^2\boldsymbol{M} + 2\pi i f\boldsymbol{C} + \boldsymbol{K}]\boldsymbol{p} = \boldsymbol{b}. \tag{6.8}$$

The size of this system is 132651, and the number of nonzero diagonals in the matrix is 19. The system matrix is complex, symmetric, and indefinite. The frequency we use in the experiments is 100 Hz.

In the experiments we focus on the comparison between IDR(s) and BiCGstab(ℓ). We use standard ILU(0) as the preconditioner. For reasons of comparison with BiCGstab(ℓ) we take for the columns of $\boldsymbol{P}$ the orthonormal basis vectors for the space spanned by the initial residual, complemented with $s-1$ randomly generated vectors. Figure 6.7 shows the convergence of IDR(s) for s equal to 1, 2, 4, and 6 and of BiCGstab(ℓ) for ℓ equal to 1, 2, 4, and 8.

Table 6.3 gives the comparison between the different methods in terms of numbers of matvecs and the measured CPU time that is needed to reduce the norm of the initial residual by a factor of 10^8. Note that a preconditioned matrix-vector multiplication is equivalent to approximately 38 vector operations. The BiCGstab(ℓ) code and the IDR(s) code are both optimized with respect to computing time, and for this reason we have included the elapsed times in the table. For both classes of methods the elapsed times are almost proportional to the number of matrix-vector multiplications, which indicates that this number gives a good measure for the performance of the methods. As is clear from the results in the table, IDR(4), and in particular IDR(6), are superior to BiCGstab(ℓ) in the above experiments; they outperform BiCGstab(ℓ) with about a factor of two. We mention that all methods yield a final (true) residual of the same magnitude, which indicates that the accuracy achieved is the same for all methods.

In [13], Sleijpen and van der Vorst explain that a small value for the minimal residual parameter ω can have a negative effect on the accuracy of the Bi-CG parameters and, as a consequence, on the convergence of Bi-CGSTAB. As a possible cure

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

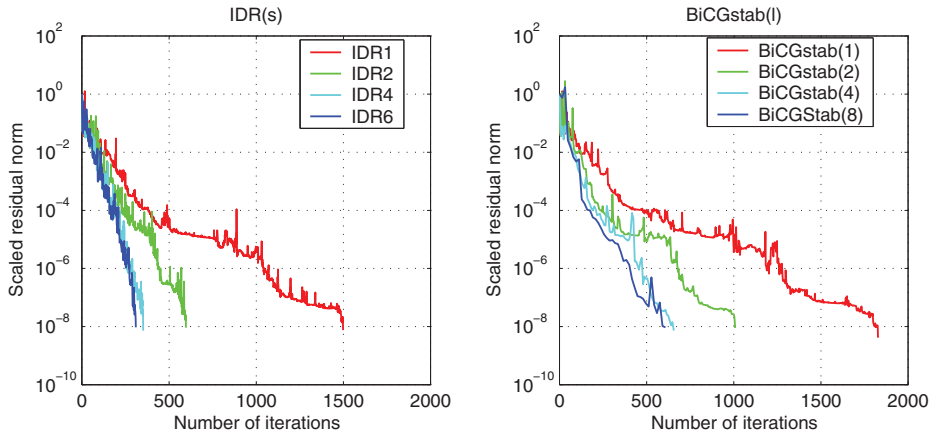

FIG. 6.7. *Convergence of IDR(s) and of BiCGstab(ℓ).*

TABLE 6.3
Number of matrix-vector multiplications and elapsed time to solve the Helmholtz problem.

Method	Number of matvecs	Elapsed time [s]
IDR(1)	1500	3322
IDR(2)	598	1329
IDR(4)	353	783
IDR(6)	310	698
BiCGstab(1)	1828	3712
BiCGstab(2)	1008	2045
BiCGstab(4)	656	1362
BiCGstab(8)	608	1337

to this they propose not to use a pure minimal residual step but to increase the value of ω if this value is too small. A similar approach can be applied to the IDR(s) algorithm. In the setting of this algorithm the computation of ω according to the strategy of Sleijpen and van der Vorst becomes the following:

$\omega = (\boldsymbol{t}^H\boldsymbol{v})/(\boldsymbol{t}^H\boldsymbol{t})$
$\rho = (\boldsymbol{t}^H\boldsymbol{v})/(\|\boldsymbol{t}\|\|\boldsymbol{v}\|)$
if $|\rho| < \kappa$ **then**
    $\omega = \omega\kappa/|\rho|$
end if

The value κ is user-defined. Sleijpen and van der Vorst recommend 0.7 as a suitable value for κ, and we used this value in our experiments for both BiCGstab(ℓ) and IDR(s).

Figure 6.8 shows the convergence of IDR(s) for s equal to 1, 2, 4, and 6 and of BiCGstab(ℓ) for ℓ equal to 1, 2, 4, and 8 with this new choice for ω. Clearly, the lower-order members of both families of methods show a greatly improved rate of convergence.

Table 6.4 tabulates for all methods the numbers of matvecs that are needed to reduce the norm of the initial residual by a factor of 10^8. With the technique of Sleijpen and van der Vorst to compute ω we achieve a further reduction of computing time, which makes the comparison between IDR(s) and BiCGstab(ℓ) even more favorable

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

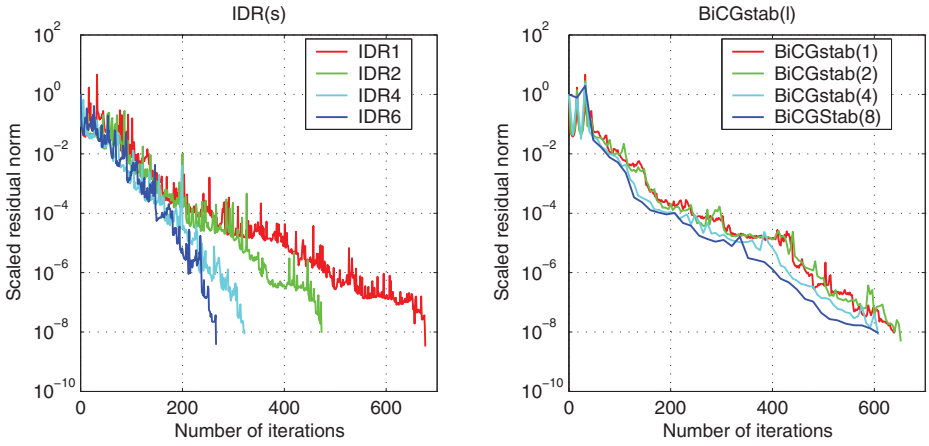

FIG. 6.8. *Convergence of IDR(s) and of BiCGstab(ℓ) with the new choice for ω.*

TABLE 6.4
Number of matrix-vector multiplications and elapsed time for the Helmholtz problem with improved computation of ω.

Method	Number of matvecs	Elapsed time [s]
IDR(1)	678	1483
IDR(2)	474	1051
IDR(4)	323	716
IDR(6)	267	601
BiCGstab(1)	640	1300
BiCGstab(2)	652	1323
BiCGstab(4)	608	1263
BiCGstab(8)	608	1337

for the former than when the standard choice for computing ω is used.

We mention that we have also tried this technique for the other examples that we have discussed in this paper, but for these examples we did not observe such a significant improvement in the rate of convergence of either IDR(s) or BiCGstab(ℓ).

7. Concluding remarks. We have presented a new approach for solving nonsymmetric systems of linear equations. Our approach is based on the IDR theorem. The resulting family of solution algorithms, which we call IDR(s), uses short recurrences and hence a limited amount of memory. This is in contrast to methods like GMRES. We have shown that IDR(1) is mathematically equivalent to Bi-CGSTAB, in the sense that the two algorithms produce the same residuals at even steps. We have also explained the mathematical relation with ML(k)BiCGSTAB and have proved that in exact arithmetic the maximum number of matrix-vector products for IDR(s), with $s > 1$, to reach the exact solution is $N + N/s$, a property that IDR(s) shares with ML(k)BiCGSTAB, if k $= s$.

We have presented a simple and, according to extensive numerical testing, numerically stable implementation of the method. This algorithm is an almost direct translation of the IDR theorem. There is, however, much freedom in how to implement an IDR-based method. Many variants and extensions are possible, and we have

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

indicated some of them. For example, it is easy to extend the algorithm with a "look-ahead-like" mechanism to avoid a breakdown.

The implementation of IDR(s) algorithms has to be done with great care. Like other short-recurrence Krylov methods, IDR(s) algorithms are quite sensitive to round-off errors. Especially, the consistency of $\Delta \boldsymbol{r}_n$ and $\Delta \boldsymbol{x}_n$ requires some prudence: statements like $\Delta \boldsymbol{r} = -\boldsymbol{A}\Delta \boldsymbol{x}$ should be used whenever possible. We have also observed in several tests that choosing *all* of the columns of $\boldsymbol{P}$ randomly improved the stability of the method, and we believe that this randomness is essential for the robustness.

The most basic variant of our algorithm, IDR(1), is about as expensive as Bi-CGSTAB, in terms of computations and memory requirements, and in our experience is just as stable. Increasing s makes the algorithm slightly more expensive per iteration, but, in all of our experiments, increasing s also yields a significant decrease in the number of iterations.

We have performed and presented numerous experiments. In all of our examples, IDR(s), with $s > 1$, is superior to Bi-CGSTAB. Increasing s always sped up the convergence, for most problems to a level close to the optimal convergence (in terms of matvecs) of full GMRES. Even for known difficult problems, such as those with a highly nonsymmetric or with an indefinite matrix, IDR(s) was among the most efficient methods. For instance, for a three-dimensional Helmholtz-type problem, IDR(6) outperformed Bi-CGstab(8) by a factor of more than two in terms of CPU time and the original Bi-CGSTAB by a factor of six.

We feel that this paper has advanced the theory and practice of iterative solution methods for large nonsymmetric linear systems in two major aspects:

1. The IDR theorem offers a new approach for the development of iterative solution algorithms, different from the classical Bi-CG or GMRES-based approaches.
2. The IDR(s) algorithm presented in this paper is quite promising and seems to outperform the state-of-the-art Bi-CG-type methods for important classes of problems.

Appendix. Prototype for IDR(s) algorithms for MATLAB. We present a frame for the algorithms as an M-file, for use with, for instance, MATLAB or Octave.

```
function [x,resvec]=idrs(A,b,s,tol,maxit,x0);
%
%--------------- Creating start residual: ----------
N = length(b);
x = x0;
r = b - A*x;
normr = norm(r);

tolr = tol * norm(b);              % tol: relative tolerance
resvec=[normr];

if (normr <= tolr)                 % Initial guess is a good enough solution
   iter=0;
   return;
end;

%----------------- Shadow space: --------------------

   rand('state', 0);                %for reproducibility reasons.
```

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

```
   P = rand(N,s);
   P(:,1) = r;                      % Only for comparison with Bi-CGSTAB
   P = orth(P)';                    % transpose for efficiency reasons.

%---------------- Produce start vectors: ------------

dR = zeros(N,s); dX = zeros(N,s);
for k = 1:s
   v = A*r;
   om = dot(v,r)/dot(v,v);
   dX(:,k) = om*r; dR(:,k) = -om*v;
   x = x + dX(:,k); r = r + dR(:,k);
   normr = norm(r);
   resvec = [resvec;normr];
   M(:,k) = P*dR(:,k);
end

%----------------- Main iteration loop, build G-spaces: ----------------

iter = s;
oldest = 1;
m = P*r;

while ( normr > tolr ) &   ( iter < maxit )
   for k = 0:s
      c = M\m;
      q = -dR*c;                         % s-1 updates + 1 scaling
      v = r + q;                         % simple addition
      if ( k == 0 )  % 1 time:
         t = A*v;                        % 1 matmul
         om = dot(t,v)/dot(t,t);         % 2 inner products
         dR(:,oldest) = q - om*t;        % 1 update
         dX(:,oldest) = -dX*c + om*v;    % s updates + 1 scaling
      else                               %
         dX(:,oldest) = -dX*c + om*v;    % s updates + 1 scaling
         dR(:,oldest) = -A*dX(:,oldest); % 1 matmul
      end

      r = r + dR(:,oldest);              % simple addition
      x = x + dX(:,oldest);              % simple addition
      iter = iter + 1;

      normr=norm(r);                     % 1 inner product (not counted)
      resvec = [resvec;normr];

      dm = P*dR(:,oldest);               % s inner products
      M(:,oldest) = dm;
      m = m + dm;

      % cycling s+1 times through matrices with s columns:
      oldest = oldest + 1;
      if ( oldest > s )
         oldest = 1;
      end
```

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

```
   end; % k = 0:s
end; %while
return
```

Acknowledgments. The first author wishes to thank Jens-Peter M. Zemke for bringing back the IDR idea into his mind, by simply asking: What happened? He also thanks his colleague Jos van Kan for many stimulating discussions. Finally, he is grateful to the Delft Institute of Applied Mathematics for offering him a desk after his retirement. Both authors are grateful to Piet Wesseling and Marielba Rojas for their careful reading of the manuscript, and to the referees for their insightful comments. The second author wishes to thank IMM at the Technical University of Denmark in Lyngby for the hospitality he received during several visits.

REFERENCES

[1] V. Faber and T. Manteuffel, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Numer. Anal., 21 (1984), pp. 352–362.
[2] R. Fletcher, *Conjugate gradient methods for indefinite systems*, in Numerical Analysis, Lecture Notes in Math. 506, Springer-Verlag, Berlin, Heidelberg, New York, 1976, pp. 73–89.
[3] J. Frank and C. Vuik, *On the construction of deflation-based preconditioners*, SIAM J. Sci. Comput., 23 (2001), pp. 442–462.
[4] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.
[5] R. W. Freund and N. M. Nachtigal, *QMR: A quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.
[6] M. H. Gutknecht, *Variants of BiCGstab for matrices with complex spectrum*, SIAM J. Sci. Comput., 14 (1993), pp. 1020–1033.
[7] S. Hellerman and M. Rosenstein, *Normal monthly wind stress over the world ocean with error estimates*, J. Phys. Oceanography, 13 (1983), pp. 1093–1104.
[8] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.
[9] C. Lanczos, *Solution of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.
[10] R. A. Nicolaides, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 355–365.
[11] Y. Saad and M. H. Schultz, *GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[12] G. L. G. Sleijpen and D. R. Fokkema, *BiCGstab(ℓ) for linear equations involving matrices with complex spectrum*, Electron. Trans. Numer. Anal., 1 (1994), pp. 11–32.
[13] G. L. G. Sleijpen and H. A. van der Vorst, *Maintaining convergence properties of BiCGstab methods in finite precision arithmetic*, Numer. Algorithms, 10 (1995), pp. 203–223.
[14] P. Sonneveld, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.
[15] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.
[16] M. B. van Gijzen, C. B. Vreugdenhil, and H. Oksuzoglu, *A finite element discretization for stream-function problems on multiply connected domains*, J. Comput. Phys., 140 (1998), pp. 30–46.
[17] P. Wesseling and P. Sonneveld, *Numerical experiments with a multiple grid and a preconditioned Lanczos type method*, in Approximation Methods for Navier-Stokes Problems, Lecture Notes in Math. 771, Springer-Verlag, Berlin, Heidelberg, New York, 1980, pp. 543–562.
[18] M.-C. Yeung and T. F. Chan, *ML(K)BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors*, SIAM J. Sci. Comput., 21 (1999), pp. 1263–1290.
[19] S.-L. Zhang, *GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems*, SIAM J. Sci. Comput., 18 (1997), pp. 537–551.

Downloaded 12/30/12 to 152.3.102.242. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.