DELFT UNIVERSITY OF TECHNOLOGY

REPORT 07-07

IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems

Peter Sonneveld and Martin B. van Gijzen

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2007

IDR(s): a family of simple and fast algorithms for solving large nonsymmetric systems of linear equations

Peter Sonneveld and Martin B. van Gijzen*

March 19, 2007

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. While state-of-the-art methods such as Bi-CGSTAB require at most $2N$ matrix-vector products to compute an exact solution in exact arithmetic, IDR(s) requires at most $N + N/s$ matrix-vector products, with N the problem size and s the dimension of a pre-chosen 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 $s > 1$.

Keywords. Iterative methods, IDR, Krylov-subspace methods, Bi-CGSTAB, CGS, non-symmetric linear systems.

AMS subject classification. 65F10, 65F50

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 [5] of Hestenes and Stiefel. The CG method minimizes the $\boldsymbol{A}$-norm of the error over the Krylov subspace

$$\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0) = \boldsymbol{r}_0 \oplus \boldsymbol{A}\boldsymbol{r}_0 \oplus \boldsymbol{A}^2\boldsymbol{r}_0 \oplus \cdots \oplus \boldsymbol{A}^n\boldsymbol{r}_0 \ , \tag{1}$$

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.

*Delft University of Technology, Delft Institute of Applied Mathematics, Mekelweg 4, 2628 CD Delft, The Netherlands. E-mail: P. Sonneveld@tudelft.nl, M.B.vanGijzen@tudelft.nl

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 [7], 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 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 [6], 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 [10], 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 [11], which applies the Bi-CG polynomial in combination with a minimal-residual step at each iteration. This method has been generalized by Sleijpen and Fokkema [8] to the BiCGstab(ℓ) method, which combines Bi-CG with a higher-order minimum-residual method.
As mentioned before, 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 closely related to Bi-CG or Lanczos.
In order to derive such a method we revisit the *Induced Dimension Reduction* (IDR) algorithm proposed in 1980 by Sonneveld [13] 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 [13] 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 a pity since, although there is a clear relation between CG-type methods and the 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 bi-orthogonal bases for the Krylov subspaces $\mathcal{K}^n(\boldsymbol{A}, r_0)$ and $\mathcal{K}^n(\boldsymbol{A}^H, \tilde{r}_0)$. 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 $\mathcal{G}_j$'s 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 non-zero 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 exact solution in exact arithmetic using less than $2N$ matrix-vector multiplications. More precisely, IDR(s) requires at most $N + N/s$ matrix-vector multiplications.
The 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. In Section 4 we analyze the termination and break-down behavior of the IDR algorithms. In Section 5, we discuss IDR(s) as a polynomial based algorithm, give alternative formulations and modifications of the algorithm, and explain the relationship between IDR(1) and Bi-CGSTAB. 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 [3], CGS, and BiCGstab(ℓ). We present concluding remarks in Section 7.

2 The Induced Dimension Reduction theorem

The new family of algorithms is based om the *Induced Dimension Reduction* theorem. The original IDR theorem was published in [13, p. 550]. Here, we give a generalization of the original result to complex matrices.

Theorem 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 complete 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
(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ for all $j > 0$.
(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 complete 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, $(\boldsymbol{I} - \omega_{j+1} \boldsymbol{A})\boldsymbol{y} = \boldsymbol{x} \in \mathcal{G}_j$.
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 only occur 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 non-trivial invariant subspace of $\boldsymbol{A}$ is not severe. Because $\mathcal{G}_0$ is a complete Krylov space, all eigenspaces of $\boldsymbol{A}$ in $\mathcal{G}_0$ are one-dimensional. So if, for instance, $\mathcal{S}$ is chosen at random, 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

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 iterates $\boldsymbol{r}_n$ can be written as $\Phi_n(\boldsymbol{A})\boldsymbol{r}_0$, where Φ_n is an n-th degree polynomial: $\Phi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}$.
The next residual $\boldsymbol{r}_{n+1}$ can be generated according to the following general rule

$$\boldsymbol{r}_{n+1} = -\alpha \boldsymbol{A}\boldsymbol{r}_n + \sum_{j=0}^{\widehat{j}} \beta_j \boldsymbol{r}_{n-j} \ . \tag{2}$$

in which the parameters α, β_j are determined by the specific Krylov method. If $\widehat{j} = n$, we have a so-called long recurrence, which implies that the amount of work and the memory requirements grow with n. On the other hand, if $\widehat{j}$ is fixed and small (compared to N), we have a so-called *short recurrence*, which is attractive with respect to computational and memory requirements.
In order to solve a system, we must be able to compute $\boldsymbol{x}_n$ corresponding to $\boldsymbol{r}_n$. This can always be done if the coefficients β_j satisfy $\sum \beta_j = 1$, as can be easily seen from the following analysis. Using the forward difference operator $\Delta \boldsymbol{u}_k = \boldsymbol{u}_{k+1} - \boldsymbol{u}_k$, the recurrence (2) can be written as

$$\boldsymbol{r}_{n+1} = \gamma_0 \boldsymbol{r}_n - \alpha \boldsymbol{A}\boldsymbol{r}_n - \sum_{j=1}^{\widehat{j}} \gamma_j \Delta \boldsymbol{r}_{n-j} \ ,$$

for suitably chosen coefficients γ_j. Now, since $\Delta \boldsymbol{r}_{n-j} = -\boldsymbol{A}\Delta \boldsymbol{x}_{n-j}$, the choice $\gamma_0 = 1$ provides the following update formulae:

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha \boldsymbol{A}\boldsymbol{r}_n - \sum_{j=1}^{\widehat{j}} \gamma_j \Delta \boldsymbol{r}_{n-j} \ ,$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha \boldsymbol{r}_n - \sum_{j=1}^{\widehat{j}} \gamma_j \Delta \boldsymbol{x}_{n-j} \ ,$$

Of course $\gamma_0 = 1$ is equivalent to $\sum \beta_j = 1$. In terms of the polynomials Φ_n, this condition is equivalent to $\Phi_n(0) = 1$.
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 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

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

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

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

The vector $\boldsymbol{v}$ is a combination of the residuals $\boldsymbol{r}_l$ (or $\Delta \boldsymbol{r}_l$) in $\mathcal{G}_j$, and hence can be written as

$$\boldsymbol{v} = \boldsymbol{r}_n - \sum_{j=1}^{\widehat{j}} \gamma_j \Delta \boldsymbol{r}_{n-j} \ . \tag{3}$$

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

$$\boldsymbol{P}^H \boldsymbol{v} = \boldsymbol{0} \, . \tag{4}$$

Combining (3) and (4) yields an $s \times \widehat{j}$ linear system for the coefficients γ_j. Under normal circumstances this system is uniquely solvable if $\widehat{j} = s$. 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:

$$d\boldsymbol{R}_n = (\Delta \boldsymbol{r}_{n-1} \ \Delta \boldsymbol{r}_{n-2} \ \cdots \ \Delta \boldsymbol{r}_{n-s}) \ , \tag{5}$$

$$d\boldsymbol{X}_n = (\Delta \boldsymbol{x}_{n-1} \ \Delta \boldsymbol{x}_{n-2} \ \cdots \ \Delta \boldsymbol{x}_{n-s}) \ . \tag{6}$$

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

Calculate: $\boldsymbol{c} \in \mathbb{R}^s$ from $(\boldsymbol{P}^H d\boldsymbol{R}_n)\boldsymbol{c} = \boldsymbol{P}^H \boldsymbol{r}_n$,

$\boldsymbol{v} = \boldsymbol{r}_n - d\boldsymbol{R}_n \boldsymbol{c}$,

$\boldsymbol{r}_{n+1} = \boldsymbol{v} - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}$.

Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, repeating these calculations will produce new residuals $\boldsymbol{r}_{n+2}, \boldsymbol{r}_{n+3}, \cdots$, 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 $\mathcal{G}_{j+2}$.
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, amongst 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 1.

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**
    $v = \boldsymbol{A}\boldsymbol{r}_n$; $\omega = (\boldsymbol{v}^H\boldsymbol{r}_n)/(\boldsymbol{v}^H\boldsymbol{v})$;
    $d\boldsymbol{x}_n = \omega\boldsymbol{r}_n$; $d\boldsymbol{r}_n = -\omega\boldsymbol{v}$;
    $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + d\boldsymbol{r}_n$; $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + d\boldsymbol{x}_n$;
  end for
  $d\boldsymbol{R}_{n+1} = (d\boldsymbol{r}_n \cdots d\boldsymbol{r}_0)$; $d\boldsymbol{X}_{n+1} = (d\boldsymbol{x}_n \cdots d\boldsymbol{x}_0)$;

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

Figure 1: The IDR(s) Algorithm.

Remarks.

- This prototype is not intended as a practical but as a mathematical algorithm. The implementation of $d\boldsymbol{R}_n = (d\boldsymbol{r}_{n-1} \cdots d\boldsymbol{r}_{n-s})$ etc, as well as the the computation of matrices $\boldsymbol{P}^H d\boldsymbol{R}_n$ 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*.

  This is exactly the same as what can happen in the Bi-CGSTAB algorithm. In some cases we can use the repair possibilities described in [9]. If the trouble is caused by structural orthogonality between $\boldsymbol{v}$ and $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{v}$, problems vanish by using a complex auxiliary matrix $\boldsymbol{P}$.

Estimates for work and memory requirements are presented in Table 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 appendix A. Note that we have counted scaling of a vector and a simple addition of two vectors as half an update each. Table 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 righthand side and the solution.).

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

Table 1: Vector operations per matrix-vector product and memory requirements

4 Analysis of the algorithm

4.1 Performance and exceptions.

The original IDR theorem only predicts dimension reduction, but does not say by how much. In the original algorithm [13], where $\mathcal{S} = \boldsymbol{p}^{\perp}$ (the $s = 1$ case), the dimension is reduced by one at each step. Here, a step requires 2 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 2 (Extended IDR theorem) *Let $\boldsymbol{A}$ be any matrix in $\mathbb{C}^{N\times N}$, let $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s \in C^N$ 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 complete 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 non-singular.
Let dim$(G_j) = d_j$, then the sequence $\{d_j,\ j = 0, 1, 2, ...\}$ is monotonically non-increasing, and satisfies

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

Proof: Let $\mathcal{U} = \mathcal{G}_{j-1} \cap \mathcal{N}(\boldsymbol{P}^H)$, and let $\boldsymbol{G}_{j-1}$ be a matrix of which the 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 non-singular, 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

$$d_j = \dim(\mathcal{N}(\boldsymbol{P}^H\boldsymbol{G}_{j-1})) = d_{j-1} - \mathrm{rank}(\boldsymbol{P}^H\boldsymbol{G}_{j-1}) \tag{7}$$

On the other hand $\mathrm{rank}(\boldsymbol{P}^H\boldsymbol{G}_{j-1}) = s - \dim(\mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P}))$, 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 $0 \leq d_{j-1} - d_j \leq s$.
Now suppose $\boldsymbol{v} \in \mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P})$, $\boldsymbol{v} \neq \boldsymbol{0}$, then $\boldsymbol{P}\boldsymbol{v} \in \mathcal{N}(\boldsymbol{G}_{j-1}^H)$, hence $\boldsymbol{P}\boldsymbol{v} \perp \mathcal{G}_{j-1}$. Since $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, this implies $\boldsymbol{P}\boldsymbol{v} \perp \mathcal{G}_j$, 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 $\dim(\mathcal{N}(\boldsymbol{G}_{j-1}^H\boldsymbol{P})) \leq \dim(\mathcal{N}(\boldsymbol{G}_j^H\boldsymbol{P}))$. It follows that

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

with $l' = \dim(\mathcal{N}(\boldsymbol{G}_j^H\boldsymbol{P})) \geq l$. So $d_j - d_{j+1} \leq d_{j-1} - d_j$, which proves the theorem. □

Remark: *According to Theorem 2 the dimension reduction per step is between* 0 *and* s*. Zero reduction only occurs if* $\mathcal{G}_j \subset \mathcal{N}(\boldsymbol{P}^H)$*, which is highly improbable, as was remarked after Theorem 1. In practical situations the reduction is* s*, the maximal value. This can be understood by the following observation. According to (7) in the proof of Theorem 2, 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* $\mathrm{rank}(\boldsymbol{P}^H\boldsymbol{G}_{j-1}) < s$*, we must have* $\boldsymbol{p}^H\boldsymbol{G}_{j-1} = \boldsymbol{0}^H$ *for some nonzero* $\boldsymbol{p} = \boldsymbol{P}\boldsymbol{c}$*. Now if* $d_{j-1} > s$*, 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 $d_j = d_{j-1} - s$ *'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, we will speak of the *generic case*, otherwise we have the *non-generic case*. In the non-generic case we call $s - (d_{j-1} - d_j)$ the *deficiency* of the reduction. In Theorem 2 we have proved that the deficiency is non-decreasing during the process.

Corollary 1 *In the generic case IDR(s) requires at at most* $N + \frac{N}{s}$ *matrix-vector multiplications to compute the exact solution in exact arithmetic.*

For the IDR-algorithms two questions may be of importance, that we will now discuss.

Observation of non-genericy. Can non-genericy be recognized during execution of the algorithm? If in some application the IDR(s) algorithm happens to be non-generic, 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 $\boldsymbol{P}^H d\boldsymbol{R}_n$. However, rank deficiency is not an exclusive property of non-genericy. 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 101-th vector happens to be outside this subspace. Therefore a non-generic case cannot be detected in practice. However, the example mentioned above is also a reason not to worry about non-genericy: rank-deficiency is a serious problem anyhow, no matter whether we are in the generic case or not.

Breakdown of the algorithm. Can the algorithm break down in the generic case, and is that curable? This question is important. Similar to the Bi-CGSTAB method, and other methods related to Bi-CG, there are two distinct ways the algorithm can break down (or will loose digits in practice).
In the generic case, a type 1 breakdown can be cured by *expanding* the previous matrix $d\boldsymbol{R}_{n-1}$ with new residual differences, rather than by *replacing* the least recent residual difference vector by the most recenly computed one. The $s \times s$ system then becomes $s \times l$ with $l > s$, but of course that is not a problem since any solution of this underdetermined system will be in $\mathcal{G}_j \cap \mathcal{S}$. Eventually we will get a better conditioned system for making the first step in $\mathcal{G}_{j+1}$.
This fix can be carried out in different ways, but is always conceptually simpler than the look-ahead / work-around procedures we know for CG-type methods. However, if we actually have a non-generic case, the matrix deficiency will remain, and can not be resolved by the above procedure.

Most type 2 breakdowns can be cured by repair methods as developed in [9]. In some problems, however, the ω calculations fail systematically, for example if $\boldsymbol{A}$ behaves like a skew-symmetric matrix. For these cases Sleijpen and Fokkema proposed BiCGstab(ℓ) [8]. This technique uses higher degree stabilization polynomials, like $p(t) = 1 + b_1 t + b_2 t^2$. 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 vanished completely when we choose the matrix $\boldsymbol{P}$ *complex* rather than real.

4.2 Iterative behavior, choice of $\boldsymbol{P}$.

Similar to the experiences with the early Lanczos and CG type methods, the IDR(s) family is finite in a structural way, but behaves like an iterative procedure as well. Only in the case of CG, applied to positive definite Hermitian matrices, we have a rather complete convergence analysis, on the basis of which we can fine-tune the method to extremely high performance (by 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, the convergence analysis collapses.
In the case of CGS and Bi-CGSTAB, part of the analysis still holds if the matrix is only moderately non-Hermitian, and if the shadow residual is chosen equal to the initial residual. But also if the problem does not satisfy the necessary restrictions for maintaining 'theoretical convergence', the practical convergence often remains satisfactory.
As the numerical examples show, and as all our tests have shown so far, this holds for the IDR(s) algorithms as well. In our experience, the IDR(s) algorithms are certainly not less 'robust' than the other short recursion Krylov methods.
In the beginning of our experimental phase, we expected that it would be wise to make the columns of $\boldsymbol{P}$ somehow related to the problem, just like the shadow vector in Bi-CGSTAB is preferably chosen equal to the initial residual. Surprisingly, the IDR(s) algorithms with these 'clever' choices of $\boldsymbol{P}$ ran terribly 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. Mainly for reasons of comparison with Bi-CGSTAB, however, we use $\boldsymbol{p}_1 = \boldsymbol{r}_0$ in most of our experiments.

5 Polynomial issues.

5.1 Different implementations

The 'basic IDR(s) algorithm' is a prototype. There is considerable freedom in translating the IDR-*theorem* into an actual *algorithm*. Furthermore, for a given algorithm, computational variants exist that are mathematically equivalent, but differ in numerical stability. In this section we will consider some implementations that are *mathematically different*.
We concentrate on the freedom in 'filling' the space $\mathcal{G}_{j+1}$. A new residual in $\mathcal{G}_{j+1}$ can be constructed according to

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

In this code fragment, $\boldsymbol{r}_n$ and the columns of $d\boldsymbol{R}_n$ must be in $\mathcal{G}_j$, in order for $\boldsymbol{r}_{n+1}$ to be in $\mathcal{G}_{j+1}$. In the prototype algorithm, $d\boldsymbol{R}_n$ consists of the most recent residual differences, and when we construct the $\boldsymbol{r}_{n+1}$ 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 $k-1$ degrees of freedom (generally speaking).
In order to analyze the different implementations of dimension reduction, we use the polynomial description. Concentrating on the residuals we have

$$\boldsymbol{r}_n = \Phi_n(\boldsymbol{A})\boldsymbol{r}_0, \quad \Phi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}, \quad \Phi_n(0) = 1 \ . \tag{8}$$

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*.

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

$$\boldsymbol{r}_n = \Omega_j(\boldsymbol{A})\boldsymbol{v} \tag{9}$$

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

$$\Omega_j(t) = (1-\omega_j t)(1-\omega_{j-1}t)\cdots(1-\omega_1 t) \tag{10}$$

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

$$\boldsymbol{P}^H \Omega_l(\boldsymbol{A})\boldsymbol{v} = \boldsymbol{0}, \ l = 0, 1, \ldots, j-1 \tag{11}$$

According to (8) and (9), and since $\boldsymbol{v}$ is in the Krylov space $\mathcal{G}_0$, $\boldsymbol{v}$ can be written as

$$\boldsymbol{v} = \Psi_{n-j}(\boldsymbol{A})\boldsymbol{r}_0 \tag{12}$$

so (11) represents relations between the coefficients of Ψ_{n-j}. Splitting $\boldsymbol{P}$ into columns, the relations (11) read

$$\boldsymbol{p}_k^H \Omega_l(\boldsymbol{A})\Psi_{n-j}(\boldsymbol{A})\boldsymbol{r}_0 = 0, \ k = 1, 2, \ldots, s, \ l = 0, 1, \ldots, j-1 \tag{13}$$

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 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 $\mathcal{G}_j$.
An interesting side effect of this polynomial analysis is the uniqueness of the residuals $\boldsymbol{r}_{(s+1)j}$, that are the very first elements in a new $\mathcal{G}$-space. Independently of the chosen variant for calculation of the other residuals in $\mathcal{G}_j$, the first elements are uniquely determined.
The authors have tested extensively the following choices for calculating the intermediate residuals in $\mathcal{G}_{j+1}$:

1. Use oldest residuals in $\mathcal{G}_j$.
2. Use all known residuals in $\mathcal{G}_j$, and choose the minimum norm solution.
3. Use most recent residuals in $\mathcal{G}_j$.

The variants 1 and 2 both require many more vectors to keep in memory, and, even if carefully coded, variant 1 is of poor numerical quality in some cases, and variant 2 offers only a modest stability improvement, if any. Therefore we present only the third variant of the IDR(s) algorithm.

5.2 Relation between IDR and Bi-CGSTAB

The relations in (13) 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:

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

Then (13) can be written as

$$[\Omega_l, \Psi_{n-j}]_k = 0, \;\; k = 1, 2, \ldots, s, \;\; l = 0, 1, \ldots, j-1 \tag{14}$$

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* $[\,.\,,\,.\,]_k$.
The idea of polynomials that are simultaneously orthogonal to lower degree polynomials with respect to more than one type of inner product is quite new, and we did not yet attempt to develop theory for it. In the case $s = 1$, however, we have classic theory. We have $sj = j$, and

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

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 being unity in the origin. *This is exactly the Bi-CG-polynomial.*
The remark in [11] about the mathematical equivalence between the old IDR and Bi-CGSTAB is only true for the 'even iterates' $\boldsymbol{r}_{2j}$. The odd steps of every variant of IDR(1), including the old method described in [13], are always different from the odd steps in Bi-CGSTAB. This is because Bi-CGSTAB must calculate the Bi-CG coefficients α_j and β_j, which is completely determined by the classical organization of the algorithm with *residuals and search directions*. The Bi-CGSTAB choice for the odd-numbered residuals must have an IDR(1) interpretation, but it is not clear what this is.
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 $\widetilde{\boldsymbol{p}}_n(\boldsymbol{A})\boldsymbol{r}_0$, 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 [1]. Furthermore, the vectors $\boldsymbol{r}_n$ and $\boldsymbol{A}\boldsymbol{v}_n$ are made orthogonal to a 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 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 only decide to stop after each $s+1$ 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 Matlab's, 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

[1] The numbering differs from that in [11].

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 [2].

6.1 A 1D Convection-Diffusion Problem.

The first example we discuss is a 1D convection-diffusion problem. With this academic example we illustrate the termination behaviour of IDR(s) as predicted by Theorem 4.1. Moreover, this example also illustrates the correspondence in the convergence behaviour of Bi-CGSTAB and IDR(1).
The test problem is the finite difference discretisation of the following differential equation:

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

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 initial guess the nullvector was chosen. For the columns of $\boldsymbol{P}$ we took the orthogonalisation 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 4.1 the IDR(s) methods should terminate (in exact arithmetic) at the exact solution within 120, 90, 75, and 70 matrix-vector products ('matvec's), respectively. Figure 2 displays for the four methods the norm of the true residual (scaled by the norm of the right-hand side vector) as 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.
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 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 method. In the next examples IDR(s) will be used as an iterative method, i.e. we want to compute a sufficiently accurate approximation to the solution in far less iterations than needed to reach the point where termination at the exact solution should occur.

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

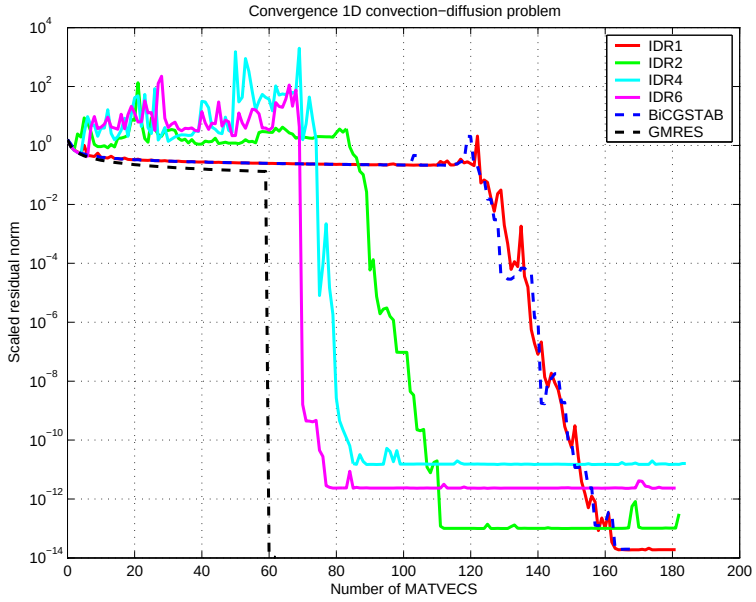

Figure 2: Breakdown of IDR(s), Bi-CGSTAB and GMRES

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 problems 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 problems.
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:

$$\mathbf{F} = \frac{\boldsymbol{\tau}}{\rho H}\,. \tag{15}$$

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:

$$\psi = C_k \quad \text{on } \Gamma_k, \quad k = 1, \cdots, K, \tag{16}$$

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.

$$\oint_{\Gamma_k} r\,\frac{\partial\psi}{\partial n}\,ds = -\oint_{\Gamma_k} \mathbf{F}\cdot\mathbf{s}\,ds. \tag{17}$$

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 [12], are listed below.

- Wind stress $\boldsymbol{\tau}$: Long term averaged data for January [4],
- 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 [12]. The solution is plotted in Figure 3.

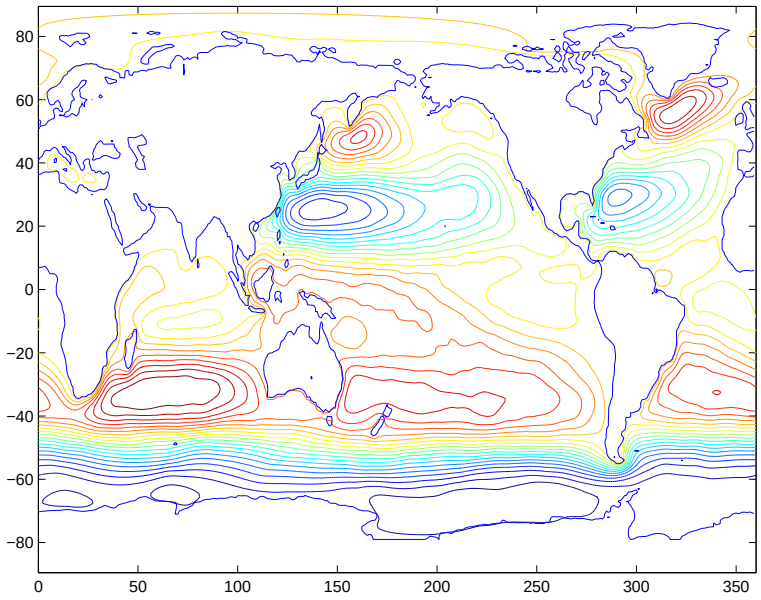

Figure 3: Solution of the ocean problem.

The resulting system consists of 42248 equation. 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 preconditioner we use ILU(0), which we apply symmetrically.
The resulting system is solved with the IDR variants IDR(1), IDR(2), IDR(4), 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 4. Although in this example the methods are used as iterative techniques,

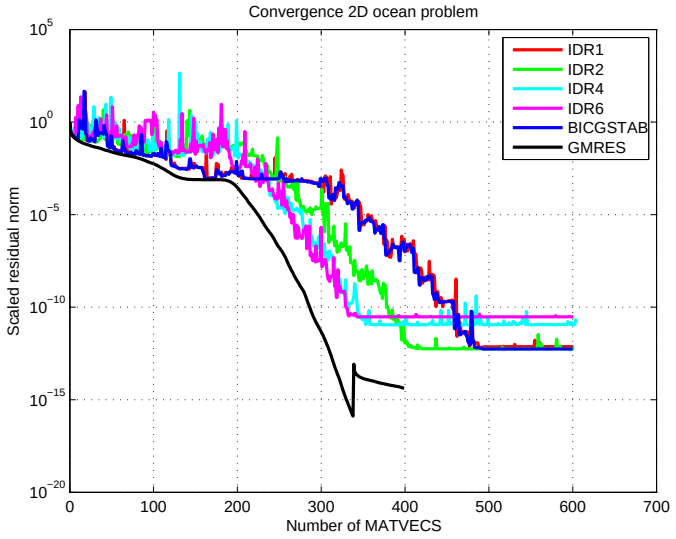

Figure 4: Convergence for the ocean problem of IDR(s), Bi-CGSTAB and GMRES

rather then as direct methods as in the previous example, there is considerable qualitative agreement in behaviour 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 othonormal set of basisvectors has to be computed and stored. This is for most application prohibitivily expensive, and the gain in precision is for most practical applications unimportant.

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 2. This table also includes the results for 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

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

Table 2: Number of matrix-vector multiplications to solve the system such that the (true) norm of the scaled residual is less than 10^{-8}

.

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

6.3 A 3D Convection-Dominated Problem

The next test problem is rather academic and is taken from [8]. 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 problems 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 discretisation 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 discretised using central differences for both the convection and diffusion terms. We take 52 gridpoints in each directions (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 orthogonalisation of the initial residual complemented with $s-1$ real random vectors. No preconditioner is applied. The iterative process is terminated once the residual norm, devided 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 5 shows the convergence behaviour of the different methods. The figure shows the poor convergence behaviour of

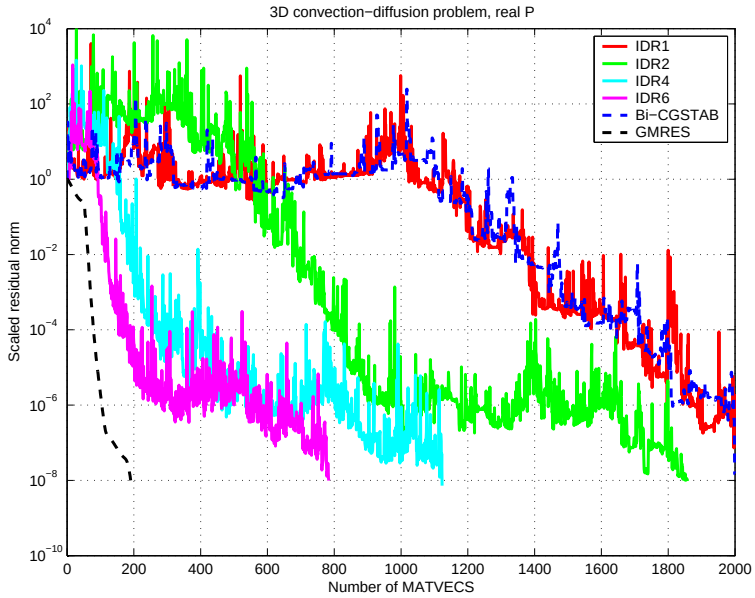

Figure 5: Convergence for IDR(s) (with real $\boldsymbol{P}$), Bi-CGSTAB and GMRES.

Bi-CGSTAB for this problem, and as can be expected also for IDR(1). No convergence is achieved for both methods within 2000 matvecs. Increasing s significantly improves the convergence behaviour of IDR(s). However, compared with the optimal convergence of GMRES, the rate of convergence is still rather poor. We have tabulated in Table 3 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.

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

Table 3: Number of matrix-vector multiplications to solve the system such that the (true) norm of the scaled residual is less than 10^{-8}.

As was remarked before, the disappointing convergence behaviour 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 [8] have shown that this problem can be overcome by combining Bi-CG with higher order minimal residual polynomials, thus creating a

polynomial Q_l that admits complex roots. This idea has given rise to the elegant Bi-CGstab(ℓ) method. In Figure 6 we show the convergence of this method. The improvement

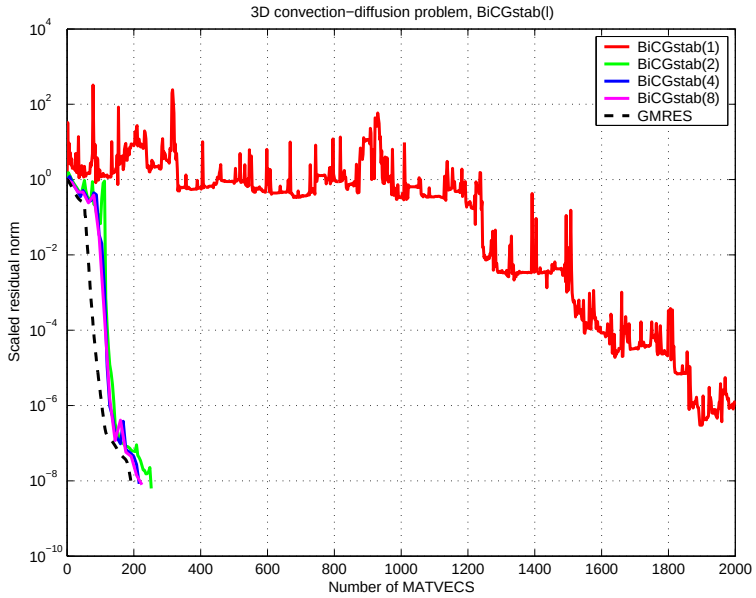

Figure 6: Convergence for BiCGstab(ℓ).

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 4.1, 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 7 Clearly, choosing $\boldsymbol{P}$ complex also solves the convergence problem: the number of IDR(6) iterations is 242, only slightly more than for Bi-CGstab(8). This is, however, at the price of turning a real computation into a complex computation.

6.4 A 3D 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 harmonic, then the acoustic pressure field has the factored form

$$p(\boldsymbol{x}, t) = \widehat{p}(\boldsymbol{x}) e^{2\pi i f t}. \tag{18}$$

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

$$\frac{-(2\pi f)^2}{c^2}\widehat{p} - \Delta\widehat{p} = \delta(\boldsymbol{x} - \boldsymbol{x}_s) \quad \text{in } \Omega. \tag{19}$$

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 is modelled by the boundary condition

$$\frac{\partial \widehat{p}}{\partial n} = 0 \ , \tag{20}$$

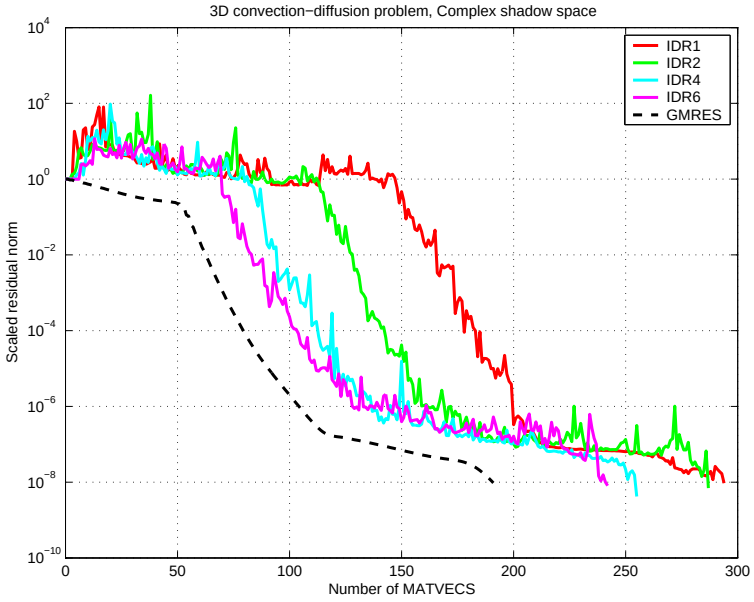

Figure 7: Convergence of IDR(s) with complex $\boldsymbol{P}$.

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

$$\frac{\partial \widehat{p}}{\partial n} = -\frac{2\pi i f}{c}\widehat{p}\,. \tag{21}$$

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

$$[-(2\pi f)^2\boldsymbol{M} + 2\pi i f\boldsymbol{C} + \boldsymbol{K}]\boldsymbol{p} = \boldsymbol{b} \tag{22}$$

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 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 8 shows the convergence of IDR(s), for s equal to 1, 2, 4 and 6, and for BiCGstab(ℓ), for ℓ equal to 1, 2, 4, and 8.

Table 4 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

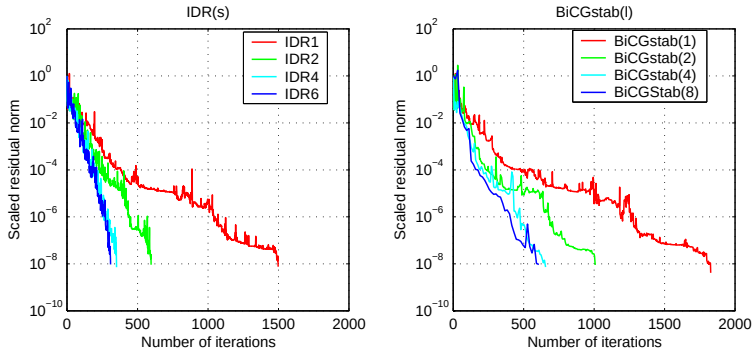

Figure 8: Convergence of IDR(s) and of BiCGstab(ℓ).

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

Table 4: Number of matrix-vector multiplications and elapsed time to solve the Helmholtz problem.

yield a final (true) residual of the same magnitude, which indicates that the achieved accuracy is the same for all methods.

In [9], 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 to this they propose to use not 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:

$\omega = (\boldsymbol{t}^H \boldsymbol{v})/(\boldsymbol{t}^H \boldsymbol{t})$
$\rho = (\boldsymbol{t}^H \boldsymbol{v})/(\|\boldsymbol{t}\|\|\boldsymbol{v}\|)$
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 this value we used in our experiments, for both BiCGstab(ℓ) and IDR(s).
Figure 9 shows the convergence of IDR(s), for s equal to 1, 2, 4 and 6, and for 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.

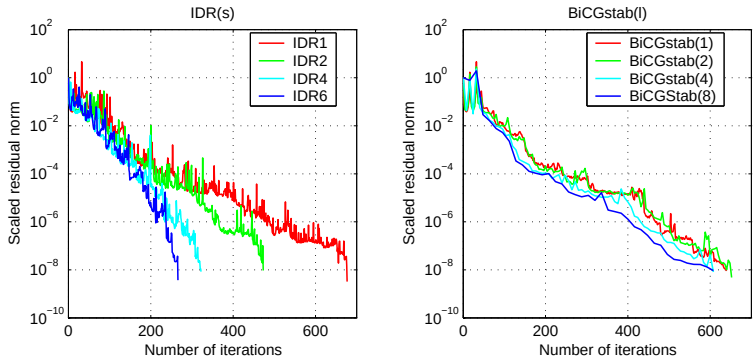

Figure 9: Convergence of IDR(s) and of BiCGstab(ℓ), new choice for ω.

Table 5 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

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

Table 5: Number of matrix-vector multiplications and elapsed time for the Helmholtz problem with improved computation of ω.

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 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 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 proved that in exact arithmetic the maximum number of

matrix-vector products for IDR(s), with $s > 1$, to reach the exact solution is lower than $2N$, the maximum number for Bi-CG-based methods like Bi-CGSTAB, and we have made it plausible that this number is $N + N/s$. Our numerical experiments confirm these two facts.

We have presented a simple and, according to extensive numerical testing, numerically stable implementation of the algorithm. This algorithm is an almost direct translation of the IDR theorem. There is, however, much freedom in how to implement the algorithm. Many variants and extensions are possible, and we have indicated some of them. For example, it is easy to extend the algorithm with a 'look-ahead-like' mechanism to avoid a break down.
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 $d\boldsymbol{r}_n$ and $d\boldsymbol{x}_n$ requires some prundence: statements like $d\boldsymbol{r} = -\boldsymbol{A}d\boldsymbol{x}$ should be used whenever possible. We have also observed in several tests that choosing *all* 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 our experiments, increasing s also yields a significant decrease in the number of iterations.
We have performed and presented numerous experiments. In all 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 3D 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 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.

Acknowledgements: The first author wish 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. The second author wishes to thank IMM at the Technical University of Denmark in Lyngby for the hospitality he received during several visits.
Part of this research has been funded by the Dutch BSIK/BRICKS project.

A 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 reproducability reasons.
   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
   end; % k = 0:s
end; %while
return
```

References

[1] V. Faber and T. Manteuffel: *Necessary and sufficient conditions for the existence of a conjugate gradient method.;* SIAM J. Numer. Anal. **21**: pp. 352-362, (1984)

[2] R. Fletcher: *Conjugate gradient methods for indefinite systems;* Lecture notes in Mathematics 506, Springer-Verlag, Berlin, Heidelberg, New York, pp. 73-89, (1976)

[3] R.W. Freund and N.M. Nachtigal: *QMR: A quasi-minimal residual method for non-Hermitian linear systems;* Numerische Mathematik **60**: pp. 315-339, (1991)

[4] S. Hellerman and M. Rosenstein: *Normal monthly wind stress over the world ocean with error estimates;* Journal of Physical Oceanography **13**: pp. 1093-1104, (1983)

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

[6] C. Lanczos: *Solution of linear equations by minimized iterations;* Journal of Research of the National Bureau of Standards **49**: pp. 33-53, (1952)

[7] Y. Saad and M.H. Schultz: *GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems;* SIAM J. Sci. Statist. Comput. **7**: pp. 856-869, (1986)

[8] G.L.G. Sleijpen and D.R. Fokkema: *BiCGstab(ℓ) for linear equations involving matrices with complex spectrum;* ETNA **1**: pp. 11-32, (1994)

[9] G.L.G. Sleijpen and H.A. van der Vorst: *Maintaining convergence properties of BiCGstab methods in finite precision arithmetic;* Numerical Algorithms **10**: pp. 203-223, (1995)

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

[11] 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. Comp. **13**: pp. 631-644, (1992) .

[12] 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**: pp. 30-46, (1998)

[13] P. Wesseling and P. Sonneveld: *Numerical Experiments with a Multiple Grid- and a Preconditioned Lanczos Type Method;* Lecture Notes in Mathematics 771, Springer-Verlag, Berlin, Heidelberg, New York, pp. 543-562, (1980)

B Probability of exceptions for random $\boldsymbol{P}$.

In this Appendix we will study the question how likely it is that a breakdown of type 1 occurs. We will refer to the behavior when no (near-) breakdown of type 1 occurs as *regular behavior*, and a case in which the behavior is regular, we call a *regular case*.

The behaviour of IDR(s) algorithms is dependent on the choice of $\boldsymbol{P}$. From the analysis of the algorithm can easily be derived that replacing $\boldsymbol{P}$ by $\boldsymbol{PC}$, where $\boldsymbol{C}$ is a nonsingular $s \times s$ matrix, will only influence the round-off behaviour of the algorithm. Normally, we choose s random vectors in $\mathbb{C}^N$, which we orthonormalize for stabilty reasons. Mathematically this modification of $\boldsymbol{P}$ does not make any difference, and we still call this a random $\boldsymbol{P}$.

As was mentioned in section 4, 'clever' choices of $\boldsymbol{P}$ are mostly not successful. We will now prove that a random choice for $\boldsymbol{P}$ garantuees the IDR(s) algorithms 'almost always' to behave regularly, that is $\boldsymbol{r}_{j(s+1)}$ is the first residual in $\mathcal{G}_j$, as long as the process is not in its final stage. For simplicity, we assume the matrix $\boldsymbol{A}$ to be non-defective.

Theorem 3 (Almost always regularity) *Let $\boldsymbol{A} \in \mathbb{C}^{N\times N}$ be non-defective, let $\boldsymbol{r}_0 \in \mathbb{C}^N$, let $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s$ be randomly chosen vectors in $\mathbb{C}^N$, and let the $N \times s$ matrix $\boldsymbol{P}$ be defined by $\boldsymbol{P} = (\boldsymbol{p}_1\ \boldsymbol{p}_2\ \ldots\ \boldsymbol{p}_s)$. Let $\widetilde{N} = dim(\mathcal{G}_0)$ be the Krylov dimension of the problem.*

Then if $js \leq \widetilde{N}$, $\boldsymbol{r}_{j(s+1)}$ is the unique residual in $\mathcal{K}^{(j(s+1)} \bigcap \mathcal{G}_j$, for almost every choice of $\boldsymbol{P}$.

Proof: We proceed similarly as in 5.1. According to (9), and (12), the residuals $\boldsymbol{r}_n$ in $\mathcal{G}_j$ can be represented as

$$\boldsymbol{r}_n = \Omega_j(\boldsymbol{A})\Psi_{n-j}(\boldsymbol{A})\boldsymbol{r}_0, \quad \text{with } \Psi(t) = 1 + \sum_{m=1}^{n-j} c_m t^l$$

From (13) follows that the coefficients c_m satisfy

$$\sum_{m=1}^{n-j} \boldsymbol{p}_k^H \boldsymbol{A}^{i+m}\boldsymbol{r}_0\, c_m \;=\; -\boldsymbol{p}_k^H \boldsymbol{A}^i \boldsymbol{r}_0,\ k = 1, 2, \ldots, s,\ i = 0, 1, \ldots, j-1$$

We consider the case $n - j = js$, in which this linear system is square. We have to prove that the corresponding solution exists and is unique, for almost every choice of $\boldsymbol{P}$. So we must inspect the regularity of the square matrix

$$\boldsymbol{B} = (b_{l,m}),\ \text{with } b_{i*s+k,m} = \boldsymbol{p}_k^H \boldsymbol{A}^{i+m}\boldsymbol{r}_0 \tag{23}$$

We first rule out the possibility that $\boldsymbol{B}$ is singular for all choices of $\boldsymbol{P}$. Let $\boldsymbol{U} = (\boldsymbol{u}_1\, \boldsymbol{u}_2\, \ldots\, \boldsymbol{u}_{\widetilde{N}})$ be an $N \times \widetilde{N}$ matrix of eigenvectors of $\boldsymbol{A}$ in $\mathcal{G}_0$, and let $\boldsymbol{r}_0 = \boldsymbol{U}\widetilde{\boldsymbol{r}}_0$, then since $\boldsymbol{A}$ is not defective we have $\boldsymbol{AU} = \boldsymbol{UD}$, where $\boldsymbol{D}$ is a diagonal matrix, consisting of eigenvalues of $\boldsymbol{A}$. Since $\mathcal{G}_0$ is a complete Krylov space, the entries $d_1, d_2, \ldots, d_{\widetilde{N}}$ of $\boldsymbol{D}$ are all different from each others, and since the Krylov space is generated by $\boldsymbol{r}_0$, all entries of $\widetilde{\boldsymbol{r}}_0$ are nonzero.

Let $\widetilde{\boldsymbol{p}}_k = \boldsymbol{U}^H \boldsymbol{p}_k$, then the entries of $\boldsymbol{B}$ can be written as

$$b_{si+k,m} = \widetilde{\boldsymbol{p}}^H \boldsymbol{D}^{i+m} \widetilde{\boldsymbol{r}}_0 = \sum_{l=1}^{\widetilde{N}} w_{k,l} d_l^{i+m}$$

with $w_{k,l} = \overline{\widetilde{p}_{k,l}} \widetilde{r}_l$.

Now we choose $\boldsymbol{P}$ in a very special way, by choosing the weigths $w_{k,l}$ as follows

$$w_{k,l} = \begin{cases} 1, & k = 1, 2, \ldots, s,\ l = (k-1)s + r,\ r = 1, 2, \ldots, j \\ 0, & \text{for all other } k \text{ and } l \text{ pairs} \end{cases}$$

Then for this choice, the entries of $\boldsymbol{B}$ simply read

$$b_{si+k,m} = \sum_{l=1}^{j} d_{(k-1)s+l}^{i+m}$$

Let $\Pi(t) = \prod_{l=1}^{sj} (t - d_l)$ and consider the sj lagrange polynomials on the mesh $d_1, d_2, \ldots, d_{sj}$:

$$L_r(t) = \frac{\Pi(t)}{\Pi'(d_r)(t - d_r)} = \sum_{m=1}^{sj} L_{m,r} t^{m-1}$$

The $sj \times sj$ matrix $\boldsymbol{L} = (L_{m,r})$ is non-singular, and the matrix $\widetilde{\boldsymbol{B}} = \boldsymbol{B}\boldsymbol{L}$ is singular if and only if $\boldsymbol{B}$ is. Now the entries of $\widetilde{\boldsymbol{B}}$ read

$$\widetilde{b}_{si+k,r} = \sum_{m=1}^{sj} b_{si+k,m} L_{m,r} = \sum_{l=1}^{j} d_{(k-1)s+l}^{i+1} L_r(d_{(k-1)s+l}) = \begin{cases} d_r^{i+1} & (k-1)s < r \leq (k-1)s + j \\ 0 & \text{for other values of } r \end{cases}$$

Now consider a linear combination $\boldsymbol{y}^H \widetilde{\boldsymbol{B}}$ of rows of $\widetilde{\boldsymbol{B}}$:

$$\sum_{k=1}^{s} \sum_{i=0}^{j-1} y_{k,i} \widetilde{b}_{si+k,r} = \begin{cases} d_r Y_k(d_r) & (k-1)s < r \leq (k-1)s + j \\ 0 & \text{for other values of } r \end{cases}$$

where $Y_k(t) = \sum_{i=0}^{j-1} y_{k,i} t^i$, are polynomials of degree less than j.

Now $\boldsymbol{y}^H \widetilde{\boldsymbol{B}} = \boldsymbol{0}^H$ implies

$$Y_k(d_r) = 0, r = (k-1)s + l,\ l = 1, 2, \ldots, j$$

A polynomial of degree less than j cannot have j distinct zeros, unless it vanishes identically. Therefore $y_{k,i} = 0$ for all appropriate k and i, or equivalently: $\boldsymbol{y} = \boldsymbol{0}$.

Hence $\widetilde{\boldsymbol{B}}$, and $\boldsymbol{B}$ are non-singular.

So there is a choice for $\boldsymbol{P}$ such that $\boldsymbol{B}$ is non-singular. Now consider the determinant of $\boldsymbol{B}$ as a function of $\boldsymbol{P}$:

$$F(\boldsymbol{P}) = \det(\boldsymbol{B})$$

Then $F(\boldsymbol{P})$ is a homogeneous polynomial in of degree sj in the sN entries of $\boldsymbol{P}$, which is not identically zero. The zeros of a non identically vanishing polynomial of degree sj are in the union of at most sj distinct $sN-1$-dimensional manifolds. The measure of this set is zero, and therefore the probability for $\boldsymbol{P}$ to be in this set is zero. $\square$