DELFT UNIVERSITY OF TECHNOLOGY

REPORT 10-16

An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties

Martin B. van Gijzen and Peter Sonneveld

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2010

An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties

Martin B. van Gijzen and Peter Sonneveld*

Abstract

The IDR(s) method that is proposed in [18] is a very efficient limited memory method for solving large nonsymmetric systems of linear equations. IDR(s) is based on the induced dimension reduction theorem, that provides a way to construct subsequent residuals that lie in a sequence of shrinking subspaces. The IDR(s) algorithm that is given in [18] is a direct translation of the theorem into an algorithm. This translation is not unique. This paper derives a new IDR(s) variant, that imposes (one-sided) bi-orthogonalization conditions on the iteration vectors. The resulting method has lower overhead in vector operations than the original IDR(s) algorithms. In exact arithmetic, both algorithms give the same residual at every $s+1$-st step, but the intermediate residuals, and also the numerical properties differ. We show through numerical experiments that the new variant is more stable and more accurate than the original IDR(s) algorithm, and that it outperforms other state-of-the-art techniques for realistic test problems.

Keywords. Iterative methods, IDR, IDR(s), Krylov-subspace methods, large sparse non-symmetric linear systems.

AMS subject classification. 65F10, 65F50

1 Introduction

We consider the linear system

$$\boldsymbol{Ax} = \boldsymbol{b}$$

with $\boldsymbol{A} \in \mathbb{C}^{N\times N}$ a large, sparse and non-symmetric matrix. In 1980, Sonneveld proposed the iterative method IDR [21] for solving such systems. The IDR method 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 non-zero scalars.
Recently, it was recognized that this IDR approach is quite general and can be used as a framework for deriving iterative methods. This observation has led to the development of IDR(s) [18]. IDR(s) has attracted considerable attention, e.g., see [7, 8, 9, 10, 12]. The examples that are described in [18] show that IDR(s), with $s > 1$ and not too big, outperforms the well-known Bi-CGSTAB method [19] for important classes of problems. Although the working principle of IDR(s) differs from that of Bi-CGSTAB, it turns out that both methods are mathematically closely related. Specifically, IDR(1) is mathematically

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

equivalent with Bi-CGSTAB, and IDR(s) with $s > 1$ is closely related (but not mathematically equivalent) to the Bi-CGSTAB generalisation ML(k)BiCGSTAB[22] of Yeung and Chan. We refer to [18] for the details. In [14], Bi-CGSTAB is considered as an IDR method, and that paper explains how IDR ideas can be incorporated into Bi-CGSTAB.
The prototype IDR(s) algorithm that is described in [18] is only one of many possible variants. One of the possibilities to make alternative IDR methods is a different computation of the *intermediate* residuals. In IDR(s), the residual is uniquely defined in every $s+1$-st step, [18], Section 5.1. This step corresponds to the calculation of the first residual in $\mathcal{G}_j$. In order to advance to $\mathcal{G}_{j+1}$, s additional residuals in $\mathcal{G}_j$ need to be computed. These intermediate residuals are not uniquely defined and their computation leaves freedom to derive algorithmic variants. In exact arithmetic, the residuals at every $s+1$-st step do not depend on the way the intermediate residuals are computed. The numerical stability and efficiency of the specific IDR algorithm, however, do depend on the computation of the intermediate residuals.
In this paper we will derive an elegant, efficient and in our experience numerically very stable IDR-based method that imposes and exploits as much as possible (one-sided) bi-orthogonality conditions between the intermediate residuals and the pre-chosen vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_s$, that define the subspace $\mathcal{S}$. We will denote this new IDR variant by *IDR(s)-biortho*[1] to distinguish it from *IDR(s)-proto*, the prototype algorithm in [18]. IDR(s)-biortho uses less vector operations per iteration than IDR(s)-proto, and has better stability properties, in particular for large values of s.
This paper is organised as follows:
The next section describes a general framework for deriving an IDR-based method. It starts with reviewing the IDR theorem. Then it explains how the theorem can be used to compute the first residual in $\mathcal{G}_{j+1}$ and the corresponding approximation for the solution, given sufficient vectors in $\mathcal{G}_j$. Furthermore it explains how sufficient intermediate residuals and vectors in $\mathcal{G}_{j+1}$ can be computed in order to advance to the next lower dimensional subspace, and what freedom there is in generating these intermediate vectors.
Section 3 derives the IDR(s)-biortho variant by filling in the freedom in generating the intermediate residuals by imposing bi-orthogonality conditions between the intermediate residuals and the vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_s$.
Section 4 presents numerical experiments that compare IDR(s)-biortho and IDR(s)-proto. It also shows experiments to illustrate the excellent performance of IDR(s) in comparison with state-of-the-art methods like Bi-CGSTAB, GMRES [11], BiCGstab(ℓ) [13].
We make concluding remarks in Section 5.

2 Designing an IDR-based algorithm

2.1 The IDR theorem

At the basis of every IDR algorithm is the IDR Theorem [18], which is given below.

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 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*

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

[1]Other authors have used different names for IDR(s)-biortho: the method is called IDR(s)BiO in [7], IDRBiO in [8] and Bi-IDR(s) in [10].

where the ω_j's are nonzero scalars. Then
(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ *for all* $\mathcal{G}_{j-1} \neq \{\mathbf{0}\},\ j > 0$.
(ii) $\mathcal{G}_j = \{\mathbf{0}\}$ *for some* $j \leq N$.

For the proof we refer to [18].
Without loss of generality, we may assume the space $\mathcal{S}$ to be the left null space of some (full rank) $N \times s$ matrix $\boldsymbol{P}$:

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

2.2 General recursions

Let $\boldsymbol{Ax} = \boldsymbol{b}$ be an $N \times N$ linear system. A Krylov-type solver produces iterates $\boldsymbol{x}_n$ for which the residuals $\boldsymbol{r}_n = \boldsymbol{b} - \boldsymbol{Ax}_n$ are in the Krylov spaces $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0) = \operatorname{span}\{\boldsymbol{r}_0, \boldsymbol{Ar}_0, \ldots, \boldsymbol{A}^n\boldsymbol{r}_0\}$. Here, $\boldsymbol{x}_0$ is an initial estimate of the solution.
An IDR-based method can be made by using recursions of the following form

$$\begin{aligned} \boldsymbol{r}_{n+1} &= \boldsymbol{r}_n - \alpha\boldsymbol{Av}_n - \sum_{i=1}^{\widehat{i}} \gamma_i \boldsymbol{g}_{n-i} \qquad (1) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \alpha\boldsymbol{v}_n + \sum_{i=1}^{\widehat{i}} \gamma_i \boldsymbol{u}_{n-i} \end{aligned}$$

in which $\boldsymbol{v}_n$ is any computable vector in $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0) \setminus \mathcal{K}^{n-1}(\boldsymbol{A}, \boldsymbol{r}_0)$, $\boldsymbol{g}_{n-i} \in \mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0)$, and $\boldsymbol{u}_{n-i}$ such that

$$\boldsymbol{g}_{n-i} = \boldsymbol{Au}_{n-i}\ . \qquad (2)$$

These recursions are quite general and hold for many Krylov subspace methods.
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, according to Theorem 1, $\boldsymbol{r}_n \in \{\mathbf{0}\}$ for some n.

2.3 A dimension-reduction step: computing the first residual in $\mathcal{G}_{j+1}$

According to Theorem 1, the residual $\boldsymbol{r}_{n+1}$ is in $\mathcal{G}_{j+1}$ if

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

If we choose

$$\boldsymbol{v}_n = \boldsymbol{r}_n - \sum_{i=1}^{\widehat{i}} \gamma_i \boldsymbol{g}_{n-i} \qquad (3)$$

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

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \omega_{j+1}\boldsymbol{Av}_n - \sum_{i=1}^{\widehat{i}} \gamma_i \boldsymbol{g}_{n-i}\ , \qquad (4)$$

which corresponds to (1), with $\alpha = \omega_{j+1}$.
Now suppose that $\boldsymbol{r}_n, \boldsymbol{g}_{n-i} \in \mathcal{G}_j,\ i = 1, \ldots, \widehat{i}$. This implies that $\boldsymbol{v}_n \in \mathcal{G}_j$. If we choose $\gamma_i, i = 1, \ldots, \widehat{i}$ such that in addition $\boldsymbol{v}_n \in \mathcal{S}$, then by Theorem 1 we have $\boldsymbol{r}_{n+1} \in \mathcal{G}_{j+1}$.
If $\boldsymbol{v}_n \in \mathcal{S} = \mathcal{N}(\boldsymbol{P}^H)$, it satisfies

$$\boldsymbol{P}^H\boldsymbol{v}_n = \mathbf{0}\,. \qquad (5)$$

Combining (3) and (5) yields an $s \times \widehat{i}$ linear system for the coefficients γ_i. Except for special circumstances, this system is uniquely solvable if $\widehat{i} = s$, which means that we need s vectors $\boldsymbol{g}_i \in \mathcal{G}_j$ for $\boldsymbol{r}_{n+1} \in \mathcal{G}_{j+1}$.
Suppose that after n iterations we have *exactly* s vectors $\boldsymbol{g}_i \in \mathcal{G}_j,\ i = n-1, \dots n-s$, and s corresponding vectors $\boldsymbol{u}_i$ with $\boldsymbol{g}_i = \boldsymbol{A}\boldsymbol{u}_i$ at our disposal. Define the matrices

$$\boldsymbol{G}_n = \left(\boldsymbol{g}_{n-s}\ \boldsymbol{g}_{n-s+1}\ \cdots\ \boldsymbol{g}_{n-1}\right), \tag{6}$$
$$\boldsymbol{U}_n = \left(\boldsymbol{u}_{n-s}\ \boldsymbol{u}_{n-s+1}\ \dots\ \boldsymbol{u}_{n-1}\right). \tag{7}$$

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

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

According to (4), the new residual satisfies

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

Multiplying this expression with $\boldsymbol{A}^{-1}$ yields the corresponding recursion for the iterate:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \omega_{j+1}\boldsymbol{v}_n + \boldsymbol{U}_n\boldsymbol{c} .$$

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 natural 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. Minimizing $\|\boldsymbol{r}_{n+1}\|_2 = \|\boldsymbol{v}_n - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}_n\|_2$ yields

$$\omega_{j+1} = \frac{(\boldsymbol{A}\boldsymbol{v}_n)^H\boldsymbol{v}_n}{(\boldsymbol{A}\boldsymbol{v}_n)^H\boldsymbol{A}\boldsymbol{v}_n} .$$

In [15], Sleijpen and Van der Vorst propose in the context of Bi-CGSTAB an improvement on this choice. They 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 the cosine of angle between $\boldsymbol{A}\boldsymbol{v}_n$ and $\boldsymbol{v}_n$ is smaller than a threshold κ. This means that ω is increased if these vectors are too close to being perpendicular. A similar approach can be applied to the IDR(s) algorithm. In the setting of this algorithm the computation of ω_{j+1} according to the strategy of Sleijpen and Van der Vorst becomes:

```
$\omega_{j+1} = (\frac{\boldsymbol{A}\boldsymbol{v}_n)^H\boldsymbol{v}_n}{\boldsymbol{A}\boldsymbol{v}_n)^H\boldsymbol{A}\boldsymbol{v}_n}$,    $\rho = \frac{(\boldsymbol{A}\boldsymbol{v}_n)^H\boldsymbol{v}_n}{\|\boldsymbol{A}\boldsymbol{v}_n\|\|\boldsymbol{v}_n\|}$
IF $|\rho| < \kappa$
    $\omega_{j+1} = \omega_{j+1}\kappa/|\rho|$
ENDIF
```

Sleijpen and Van der Vorst recommend 0.7 as a suitable value for κ. In our experience, this "maintaining the convergence" choice for ω can greatly improve the rate of convergence of IDR(s), see Section 6.4 of [18] for an example. In Section 4 we will illustrate through numerical experiments that this choice for ω may also yield a more accurate solution, in particular for larger values of s. Note that the calculation of ω_{j+1} does not require an additional matrix multiplication, since the vector $\boldsymbol{A}\boldsymbol{v}_n$ can be re-used in the update of the residual.
Of course, other than the above discussed strategies for the computation of ω_{j+1} are possible, see for example [12] for a new approach based on precomputed Ritz values.
The above framework explains how a residual in $\mathcal{G}_{j+1}$ can be computed given $\boldsymbol{r}_n, \boldsymbol{g}_{n-i} \in \mathcal{G}_j,\ i = 1, \ldots s$. Next we will discuss a technique for computing these vectors.

2.4 Computing additional vectors in $\mathcal{G}_{j+1}$

The procedure that is outlined in the previous section can be used directly to compute a new residual $\boldsymbol{r}_{n+2} \in \mathcal{G}_{j+1}$: since $\boldsymbol{g}_i \in \mathcal{G}_j, i = n-1, \ldots n-s$ and $\boldsymbol{r}_{n+1} \in \mathcal{G}_{j+1} \subset \mathcal{G}_j$, the computations

$$\begin{aligned} \text{Calculate: } & \boldsymbol{c} \in \mathbb{C}^s \text{ from } (\boldsymbol{P}^H \boldsymbol{G}_n)\boldsymbol{c} = \boldsymbol{P}^H \boldsymbol{r}_{n+1}, \\ & \boldsymbol{v}_{n+1} = \boldsymbol{r}_{n+1} - \boldsymbol{G}_n \boldsymbol{c}, \\ & \boldsymbol{r}_{n+2} = \boldsymbol{v}_{n+1} - \omega_{j+1} \boldsymbol{A}\boldsymbol{v}_{n+1} \ . \end{aligned}$$

yield a residual that satisfies $\boldsymbol{r}_{n+2} \in \mathcal{G}_{j+1}$.
Furthermore, we observe that the *residual difference vector* $(\boldsymbol{r}_{n+2} - \boldsymbol{r}_{n+1})$ is in the space $\mathcal{G}_{j+1}$. Since $\boldsymbol{A}^{-1}(\boldsymbol{r}_{n+2} - \boldsymbol{r}_{n+1}) = -(\boldsymbol{x}_{n+2} - \boldsymbol{x}_{n+1})$ we have found a suitable pair of vectors $\boldsymbol{g}_{n+1}, \boldsymbol{u}_{n+1}$:

$$\boldsymbol{g}_{n+1} = -(\boldsymbol{r}_{n+2} - \boldsymbol{r}_{n+1}) \ , \quad \boldsymbol{u}_{n+1} = \boldsymbol{x}_{n+2} - \boldsymbol{x}_{n+1} \ .$$

In a practical algorithm, the computation of $\boldsymbol{g}_{n+1}$ and of $\boldsymbol{u}_{n+1}$ precedes the computation of $\boldsymbol{r}_{n+2}$ and of $\boldsymbol{x}_{n+2}$. First, the update vector for the iterate can be computed by

$$\boldsymbol{u}_{n+1} = \omega_{j+1} \boldsymbol{v}_{n+1} + \boldsymbol{U}_n \boldsymbol{c} \ ,$$

followed by the computation of $\boldsymbol{g}_{n+1}$ by

$$\boldsymbol{g}_{n+1} = \boldsymbol{A}\boldsymbol{u}_{n+1} \tag{8}$$

to preserve in finite precision arithmetic as much as possible the relation between $\boldsymbol{u}_{n+1}$ and $\boldsymbol{g}_{n+1}$. The iterate and residual are then updated through

$$\boldsymbol{x}_{n+2} = \boldsymbol{x}_{n+1} + \boldsymbol{u}_{n+1} \ , \qquad \boldsymbol{r}_{n+2} = \boldsymbol{r}_{n+1} - \boldsymbol{g}_{n+1} \ . \tag{9}$$

The vector $\boldsymbol{g}_{n+1}$ is in the space $\mathcal{G}_{j+1}$, and hence also in $\mathcal{G}_j$. This means that we can use this vector in the calculation of new vectors in $\mathcal{G}_{j+1}$, and discard an old vector, e.g., $\boldsymbol{g}_{n-s}$. This can be done by defining the matrices $\boldsymbol{G}_{n+2}$ and $\boldsymbol{U}_{n+2}$ as

$$\begin{aligned} \boldsymbol{G}_{n+2} &= (\boldsymbol{g}_{n+1}\ \boldsymbol{g}_{n-s+1}\ \cdots\ \boldsymbol{g}_{n-1}) \ , & (10) \\ \boldsymbol{U}_{n+2} &= (\boldsymbol{u}_{n+1}\ \boldsymbol{u}_{n-s+1}\ \ldots\ \boldsymbol{u}_{n-1}) \ . & (11) \end{aligned}$$

The advantage of this procedure is that it saves vector space: storage for exactly s $\boldsymbol{g}$-vectors and s $\boldsymbol{u}$-vectors is needed.

We can repeat the above procedure s times to compute $\boldsymbol{r}_{n+s+1}, \boldsymbol{g}_{n+k} \in \mathcal{G}_{j+1},\ k = 1, \dots s$, and the corresponding vectors $\boldsymbol{x}_{n+s+1}, \boldsymbol{u}_{n+k},\ k = 1, \dots s$, which are the vectors that are needed to compute a residual in $\mathcal{G}_{j+2}$.
The above relations define (apart from initialization of the vectors) a complete IDR-method. In fact, the algorithm that is outlined above is almost the same as the IDR(s) method from [18]. The only difference is that the original IDR(s) method also computes $\boldsymbol{g}_n = -(\boldsymbol{r}_{n+1} - \boldsymbol{r}_n)$, which vector is then included in $\boldsymbol{G}_{n+k}, k = 1, \dots, s$. Leaving this vector out simplifies the IDR(s)-biortho algorithm that we present in the next section.
In the above algorithm, vectors in $\mathcal{G}_{j+1}$ are generated by direct application of the IDR theorem. The computations of the first residual in $\mathcal{G}_{j+1}$ is almost the same as the computation of the following s residuals in $\mathcal{G}_{j+1}$. However, in computing the intermediate residuals, there is more freedom that can be exploited. In the algorithm above, a residual is updated by

$$\boldsymbol{r}_{n+k+1} = \boldsymbol{r}_{n+k} - \boldsymbol{g}_{n+k} .$$

Here, $\boldsymbol{r}_{n+k+1}, \boldsymbol{r}_{n+k}$, and $\boldsymbol{g}_{n+k}$ are in $\mathcal{G}_{j+1}$. But in order to compute a new residual in $\mathcal{G}_{j+1}$ we could also have used a more general linear combination of vectors in $\mathcal{G}_{j+1}$:

$$\boldsymbol{r}_{n+k+1} = \boldsymbol{r}_{n+k} - \sum_{i=1}^{k} \beta_i \boldsymbol{g}_{n+i} .$$

Clearly, the vector $\boldsymbol{r}_{n+k+1}$ computed in this way is also in $\mathcal{G}_{j+1}$. We can choose the parameters β_i to give the intermediate residuals a desirable property, like minimum norm. In the algorithm that we present in the next section we will use the parameters β_i such that the intermediate residual $\boldsymbol{r}_{n+k+1}$ is orthogonal to $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$.
The same freedom that we have for computing a new residual in $\mathcal{G}_{j+1}$, we have for computing the vectors $\boldsymbol{g}_{n+k}$: linear combinations of vectors in $\mathcal{G}_{j+1}$ are still in $\mathcal{G}_{j+1}$. Let

$$\bar{\boldsymbol{g}} = -(\boldsymbol{r}_{n+k+1} - \boldsymbol{r}_{n+k}).$$

Then the vector

$$\boldsymbol{g}_{n+k} = \bar{\boldsymbol{g}} - \sum_{i=1}^{k-1} \alpha_i \boldsymbol{g}_{n+i}$$

is also in $\mathcal{G}_{j+1}$, and can be used in the subsequent computations. Again, the parameters α_i can be chosen such that the vector $\boldsymbol{g}_{n+k}$ gets some favorable properties. In the algorithm that we present in the next section we will chose the parameters α_i such that the vector $\boldsymbol{g}_{n+k}$ is made orthogonal to $\boldsymbol{p}_1 \dots \boldsymbol{p}_{k-1}$.
Apart from the initialization of the variables, we have now given a complete framework for an IDR-based solution algorithm. To initialize the recursions, values for $\boldsymbol{x}_s, \boldsymbol{r}_s, \boldsymbol{U}_0$, and $\boldsymbol{G}_0$ have to be computed. This can be done by any Krylov method. Figure 1 presents a framework, including an (implicit) initialization in the first s steps, for an IDR-based algorithm. The freedom that is left open is in the choice of the parameters α_i and β_i, and ω. In the algorithm we have omitted the indices for the iteration number. Vectors on the left are overwritten by vectors on the right.

```
Require: $\boldsymbol{A} \in \mathbb{C}^{N\times N}$; $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{C}^N$; $\boldsymbol{P} \in \mathbb{C}^{N\times s}$; $TOL \in (0,1)$;
Ensure: $x$ such that $\|\boldsymbol{b} - \boldsymbol{Ax}\| \le TOL \cdot \|\boldsymbol{b}\|$
  {Initialization.}
  Calculate $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$;
  $\boldsymbol{G} = \boldsymbol{O} \in \mathbb{C}^{N\times s}$; $\boldsymbol{U} = \boldsymbol{O} \in \mathbb{C}^{N\times s}$;
  $\boldsymbol{M} = \boldsymbol{I} \in \mathbb{C}^{s\times s}$; $\omega = 1$;

  {Loop over $\mathcal{G}_j$ spaces}
  while $\|\boldsymbol{r}\| > TOL$ do
      {Compute $s$ independent vectors $\boldsymbol{g}_k$ in $\mathcal{G}_j$ space}
      for $k = 1$ to $s$ do
          Compute $\boldsymbol{f} = \boldsymbol{P}^H\boldsymbol{r}$;
          {Note: $\boldsymbol{M} = \boldsymbol{P}^H\boldsymbol{G}$}
          Solve $\boldsymbol{c}$ from $\boldsymbol{Mc} = \boldsymbol{f}$;
          $\boldsymbol{v} = \boldsymbol{r} - \boldsymbol{Gc}$;
          $\boldsymbol{u}_k = \boldsymbol{Uc} + \omega\boldsymbol{v}$;
          $\boldsymbol{g}_k = \boldsymbol{Au}_k$;
          {Linear combinations of vectors $\in \mathcal{G}_j$ are still in $\mathcal{G}_j$:}
          Select $\alpha_i$ and $\beta_i$, $i = 1, \ldots k-1$;
          $\boldsymbol{g}_k = \boldsymbol{g}_k - \sum_{i=1}^{k-1}\alpha_i\boldsymbol{g}_i$ ;    $\boldsymbol{u}_k = \boldsymbol{u}_k - \sum_{i=1}^{k-1}\alpha_i\boldsymbol{u}_i$;
          $\boldsymbol{r} = \boldsymbol{r} - \sum_{i=1}^{k}\beta_i\boldsymbol{g}_i$ ;    $\boldsymbol{x} = \boldsymbol{x} + \sum_{i=1}^{k}\beta_i\boldsymbol{u}_i$;
          {Update of the $k$-th column of $\boldsymbol{M}$:}
          $\boldsymbol{M}_{:,k} = \boldsymbol{P}^H\boldsymbol{g}_k$;
          {Overwrite $k-$th column of $\boldsymbol{G}$ by $\boldsymbol{g}_k$, and of $\boldsymbol{U}$ by $\boldsymbol{u}_k$}
          $\boldsymbol{G}_{:,k} = \boldsymbol{g}_k$; $\boldsymbol{U}_{:,k} = \boldsymbol{u}_k$;
      end for
      { Entering $\mathcal{G}_{j+1}$}
      $\boldsymbol{f} = \boldsymbol{P}^H\boldsymbol{r}$;
      Solve $\boldsymbol{c}$ from $\boldsymbol{Mc} = \boldsymbol{f}$;
      $\boldsymbol{v} = \boldsymbol{r} - \boldsymbol{Gc}$;
      $\boldsymbol{t} = \boldsymbol{Av}$;
      Select $\omega$;
      $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{Uc} + \omega\boldsymbol{v}$;
      {Alternative computation: $\boldsymbol{r} = \boldsymbol{v} - \omega\boldsymbol{t}$)};
      $\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{Gc} - \omega\boldsymbol{t}$;
  end while
```

Figure 1: A framework for an IDR-based algorithm.

3 An efficient IDR(s) variant that exploits bi-orthogonality properties

3.1 General idea

In this section we will fill in the freedom that we have left in the framework IDR algorithm. As in the previous section we assume that $\boldsymbol{r}_{n+1}$ is the first residuals in $\mathcal{G}_{j+1}$. We fill in the freedom by constructing vectors that satisfy the following bi-orthogonality conditions:

$$\boldsymbol{g}_{n+k} \perp \boldsymbol{p}_i,\ i = 1, \ldots k-1,\ k = 2, \ldots, s, \tag{12}$$

and

$$\boldsymbol{r}_{n+k+1} \perp \boldsymbol{p}_i,\ i = 1, \ldots, k,\ k = 1, \ldots, s. \tag{13}$$

As we will see, these relations lead to important simplifications in the algorithm.

3.2 A dimension-reduction step: computing the first residual in $\mathcal{G}_{j+1}$

The bi-orthogonality condition for the intermediate residuals (13) implies that the first intermediate residual is orthogonal to $\boldsymbol{p}_1$, the second to $\boldsymbol{p}_1$ and to $\boldsymbol{p}_2$, etc. Hence, the last intermediate residual before making a dimension reduction step, i.e., $\boldsymbol{r}_n$ is orthogonal to $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_s$. Consequently,

$$\boldsymbol{r}_n \in \mathcal{G}_j \cap \mathcal{S} \ .$$

Now, by Theorem 1

$$\boldsymbol{r}_{n+1} = (\boldsymbol{I} - \omega_{j+1}\boldsymbol{A})\boldsymbol{r}_n \in \mathcal{G}_{j+1} \ .$$

With the standard choice for ω_{j+1}, the dimension reduction step simplifies to a standard minimal residual step.

3.3 Computing additional vectors in $\mathcal{G}_{j+1}$

In order to calculate a vector $\boldsymbol{v}_{n+k} \in \mathcal{G}_j \cap \mathcal{S}$, a system of the form $(\boldsymbol{P}^H\boldsymbol{G}_{n+k})\boldsymbol{c} = \boldsymbol{P}^H\boldsymbol{r}_{n+k}$ has to be solved. Using the conditions (12) and (13) this system gets a simple form. Let

$$\mu_{i,k} = \boldsymbol{p}_i^H \boldsymbol{g}_{n+k}, \quad i = 1, \ldots, s \ .$$

Then, because of (12), $\mu_{ik} = 0$ for $i < k$. Furthermore, let

$$\phi_i = \boldsymbol{p}_i^H \boldsymbol{r}_{n+k}, \quad i = 1, \ldots, s \ .$$

Then, because of (13), $\phi_i = 0$ for $i < k$. Consequently, the system $(\boldsymbol{P}^H\boldsymbol{G}_{n+k})\boldsymbol{c} = \boldsymbol{P}^H\boldsymbol{r}_{n+k}$ has the following structure

$$\begin{pmatrix} \mu_{1,1} & 0 & \ldots & \ldots & 0 \\ \mu_{2,1} & \mu_{2,2} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ \mu_{s,1} & \mu_{s,2} & \ldots & \ldots & \mu_{s,s} \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \vdots \\ \vdots \\ \vdots \\ \gamma_s \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \phi_k \\ \vdots \\ \phi_s \end{pmatrix} \ .$$

Clearly, $\gamma_1, \ldots, \gamma_{k-1}$ are zero, and the update for $\boldsymbol{v}_{n+k}$ becomes

$$\boldsymbol{v}_{n+k} = \boldsymbol{r}_{n+k} - \sum_{i=k}^{s} \gamma_i \boldsymbol{g}_{n+i-s-1}$$

Next, we compute 'temporary' vectors $\boldsymbol{u}_{n+k}$ and $\boldsymbol{g}_{n+k}$ by

$$\boldsymbol{u}_{n+k} = \omega_{j+1}\boldsymbol{v}_{n+k} + \sum_{i=k}^{s} \gamma_i \boldsymbol{u}_{n+i-s-1}; \quad \boldsymbol{g}_{n+k} = \boldsymbol{A}\boldsymbol{u}_{n+k} \ .$$

Next, the vector $\boldsymbol{g}_{n+k} \in \mathcal{G}_{j+1}$ is made orthogonal to $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{k-1}$ by the following procedure, that includes the corresponding updates to compute $\boldsymbol{u}_{n+k}$:

For $i = 1$ to $k - 1$

```
        $\alpha = \frac{\mathbf{p}_i^H \mathbf{g}_{n+k}}{\mu_{i,i}}$;
        $\mathbf{g}_{n+k} = \mathbf{g}_{n+k} - \alpha \mathbf{g}_{n+i}$;
        $\mathbf{u}_{n+k} = \mathbf{u}_{n+k} - \alpha \mathbf{u}_{n+i}$.
    End for
```

The above algorithm is similar to the modified Gram-Schmidt procedure for orthogonalizing a vector with respect to a set of orthogonal vectors. Alternatively, we could have used the classical Gram-Schmidt-like procedure as we have used for the computation of a vector $\mathbf{v}_{n+k}$ that is orthogonal to $\mathbf{p}_1, \ldots, \mathbf{p}_s$.
The next step in the algorithm is to compute the next intermediate residual $\mathbf{r}_{n+k+1}$ that is orthogonal to $\mathbf{p}_1, \ldots, \mathbf{p}_k$. It is easy to check that such a residual can be computed by

$$\mathbf{r}_{n+k+1} = \mathbf{r}_{n+k} - \frac{\phi_k}{\mu_{k,k}} \mathbf{g}_{n+k} \ . \tag{14}$$

The corresponding approximate solution then becomes

$$\mathbf{x}_{n+k+1} = \mathbf{x}_{n+k} + \frac{\phi_k}{\mu_{k,k}} \mathbf{u}_{n+k} \ .$$

The outline of the algorithm suggest that we have to compute the inner products as $\phi_i = \mathbf{p}_i^H \mathbf{r}_{n+k+1}, i = k, \ldots, s$. From (14), however, follows that

$$\mathbf{p}_i^H \mathbf{r}_{n+k+1} = \mathbf{p}_i^H \mathbf{r}_{n+k} - \frac{\phi_k}{\mu_{k,k}} \mathbf{p}_i \mathbf{g}_{n+k} \quad i = k+1, \ldots, s,$$

and hence that

$$\phi_i = \phi_i - \frac{\phi_k \mu_{i,k}}{\mu_{k,k}}, \quad i = k+1, \ldots, s \ .$$

So given $\mu_{i,k}, i = k+1, \ldots, s$, the new value for ϕ_i can be computed via a scalar update.

3.4 A preconditioned version of IDR(s)-biortho.

Preconditioning can simply be implemented by applying the unpreconditioned algorithm to an explicitly preconditioned problem. In the case of right preconditioning with preconditioner $\mathbf{B}$ this means that IDR(s) is applied to the system

$$\mathbf{A}\mathbf{B}^{-1}\mathbf{y} = \mathbf{b} \ .$$

and every matrix-vector multiplication becomes a multiplication with $\mathbf{A}\mathbf{B}^{-1}$, i.e., an application of the preconditioner followed by a multiplication with $\mathbf{A}$. In order to get the solution of the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$, we have to scale back the solution

$$\mathbf{x} = \mathbf{B}^{-1}\mathbf{y} \ ,$$

so one extra preconditioning step has to be performed after the iterative process has terminated.
As is shown in [9] for IDR(s)-proto, it is possible to avoid this extra operation and to make preconditioning implicit. The same ideas that are described in [9] can also be applied to IDR(s)-biortho.

The idea is to slightly re-arrange the update of the solution. For example, the dimension-reduction step for the explicitly right-preconditioned problem reads

$$\begin{aligned} \boldsymbol{t} &= \boldsymbol{A}\boldsymbol{B}^{-1}\boldsymbol{r}_n \ , \\ \boldsymbol{r}_{n+1} &= \boldsymbol{r}_n - \omega_{j+1}\boldsymbol{t} \ , \\ \boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + \omega_{j+1}\boldsymbol{r}_n \ . \end{aligned}$$

Multiplying the recursion for $\boldsymbol{y}$ with $\boldsymbol{B}^{-1}$ gives the recursion for $\boldsymbol{x}$:

$$\begin{aligned} \boldsymbol{t} &= \boldsymbol{A}\boldsymbol{B}^{-1}\boldsymbol{r}_n \ , \\ \boldsymbol{r}_{n+1} &= \boldsymbol{r}_n - \omega_{j+1}\boldsymbol{t} \ , \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \omega_{j+1}\boldsymbol{B}^{-1}\boldsymbol{r}_n \ , \end{aligned}$$

which can be implemented as

$$\begin{aligned} \boldsymbol{v} &= \boldsymbol{B}^{-1}\boldsymbol{r}_n \ , \\ \boldsymbol{t} &= \boldsymbol{A}\boldsymbol{v} \ , \\ \boldsymbol{r}_{n+1} &= \boldsymbol{r}_n - \omega_{j+1}\boldsymbol{t} \ , \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \omega_{j+1}\boldsymbol{v} \ . \end{aligned}$$

The same technique can be used to make the preconditioning operation in the intermediate steps implicit.

3.5 The IDR(s)-biortho algorithm

Figure 2 presents the complete IDR(s)-biortho algorithm, including preconditioning and the computation of ω_{j+1} according to the "maintaining the convergence" strategy. In the algorithm we have omitted the indices for the iteration number.
The algorithm is quite efficient in terms of vector operations and even more efficient than IDR(s)-proto, despite the additional orthogonalization operations. The operation count for the main operations to perform the dimension reduction step yields: one preconditioned matrix-vector product, two vector updates and two inner products. For the intermediate steps we get: s preconditioned matrix-vector products, $2s+2$ vector updates and $s+1$ inner products. Hence, for a full cycle of $s+1$ IDR(s) steps we get: $s+1$ preconditioned matrix-vector products, s^2+s+2 inner products and $2s^2+2s+2$ vector updates. IDR(s)-biortho requires slightly less vector updates than IDR(s)-proto, and the same number of inner products and preconditioned matrix-vector multiplications. The original IDR(s) method requires $2s^2+\frac{7}{2}s+\frac{5}{2}$ vector-updates.
Table 1 gives an overview of the number of vector operations per preconditioned matrix-vector multiplication for some values of s for IDR(s)-biortho, and for comparison also for the Krylov methods that we will use in the numerical experiments. 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 vector and the solution).

4 Numerical experiments.

In this section we present numerical examples to compare the numerical behaviour of IDR(s)-proto and IDR(s)-biortho. For another evaluation of the performance of the two IDR(s) variants, based on selected test matrices from the Florida Sparse Matrix Collection

```
Require: $\boldsymbol{A} \in \mathbb{C}^{N\times N}$; $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{C}^N$; $\boldsymbol{P} \in \mathbb{C}^{N\times s}$; $TOL \in (0,1)$;
Ensure: $\boldsymbol{x}_n$ such that $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\| \leq TOL \cdot \|\boldsymbol{b}\|$;
  $\{$Initialization$\}$
  Calculate $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$;
  $\boldsymbol{g}_i = \boldsymbol{u}_i = \boldsymbol{0} \in \mathbb{C}^N, i = 1, \dots, s$; $\boldsymbol{M} = \boldsymbol{I} \in \mathbb{C}^{s\times s}$; $\omega = 1$;

  $\{$Loop over $\mathcal{G}_j$ spaces, for $j = 0, 1, 2, 3, \dots\}$
  while $\|\boldsymbol{r}\| > TOL$ do
      $\boldsymbol{f} = \boldsymbol{P}^H\boldsymbol{r},\ (\phi_1, \dots, \phi_s)^T = \boldsymbol{f}$;
      for $k = 1$ to $s$ do
          Solve $\boldsymbol{c}$ from $\boldsymbol{M}\boldsymbol{c} = \boldsymbol{f},\ (\gamma_1, \dots, \gamma_s)^T = \boldsymbol{c}$;
          $\boldsymbol{v} = \boldsymbol{r} - \sum_{i=k}^{s} \gamma_i \boldsymbol{g}_i$;
          $\{$Preconditioning operation$\}$
          $\boldsymbol{v} = \boldsymbol{B}^{-1}\boldsymbol{v}$;
          $\boldsymbol{u}_k = \omega\boldsymbol{v} + \sum_{i=k}^{s} \gamma_i \boldsymbol{u}_i$;
          $\boldsymbol{g}_k = \boldsymbol{A}\boldsymbol{u}_k$;
          for $i = 1$ to $k-1$ do
              $\alpha = \frac{\boldsymbol{p}_i^H \boldsymbol{g}_k}{\mu_{i,i}}$;
              $\boldsymbol{g}_k = \boldsymbol{g}_k - \alpha\boldsymbol{g}_i$;
              $\boldsymbol{u}_k = \boldsymbol{u}_k - \alpha\boldsymbol{u}_i$;
          end for
          $\mu_{i,k} = \boldsymbol{p}_i^H \boldsymbol{g}_k,\ \boldsymbol{M}_{i,k} = \mu_{i,k},\ i = k \dots s$;
          $\beta = \frac{\phi_k}{\mu_{k,k}}$;
          $\boldsymbol{r} = \boldsymbol{r} - \beta\boldsymbol{g}_k$:
          $\boldsymbol{x} = \boldsymbol{x} + \beta\boldsymbol{u}_k$;
          if $k + 1 \leq s$ then
              $\phi_i = 0, i = 1, \dots k$;
              $\phi_i = \phi_i - \beta\mu_{i,k}, i = k+1, \dots, s$;
              $\boldsymbol{f} = (\phi_1, \dots, \phi_s)^T$;
          end if
      end for
      $\{$ Entering $\mathcal{G}_{j+1}\}$
      $\{$Preconditioning$\}$
      $\boldsymbol{v} = \boldsymbol{B}^{-1}\boldsymbol{r}$;
      $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{v}$;
      $\{$Calculation of $\omega$ using "maintaining the convergence" strategy$\}$
      $\omega = \boldsymbol{t}^H\boldsymbol{r}/\boldsymbol{t}^H\boldsymbol{t};\ \rho = (\boldsymbol{t}^H\boldsymbol{r})/(\|\boldsymbol{t}\|\|\boldsymbol{r}\|)$;
      if $|\rho| < \kappa$ then
          $\omega = \omega\kappa/|\rho|$;
      end if
      $\boldsymbol{r} = \boldsymbol{r} - \omega\boldsymbol{t}$;
      $\boldsymbol{x} = \boldsymbol{x} + \omega\boldsymbol{v}$;
  end while
```

Figure 2: Preconditioned IDR(s)-biortho, the preconditioner is denoted by $\boldsymbol{B}$.

[1], we refer to [10]. We also present experiments to evaluate the performance of the biortho

Method	DOT	AXPY	Memory Requirements (vectors)
IDR(1)	2	3	7
IDR(2)	$2\frac{2}{3}$	$4\frac{2}{3}$	10
IDR(4)	$4\frac{2}{5}$	$8\frac{2}{5}$	16
IDR(8)	$8\frac{1}{4}$	$16\frac{2}{9}$	28
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
Bi-CGSTAB	2	3	7
BiCGstab(2)	$2\frac{1}{4}$	$3\frac{3}{4}$	9

Table 1: Memory requirements and vector operations per preconditioned matrix-vector product

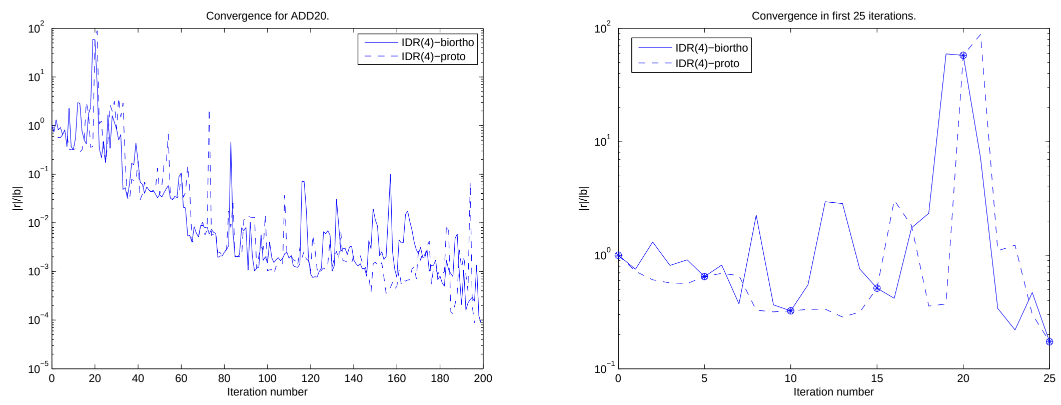
variant of IDR(s) in comparison with a number of state-of-the-art Krylov methods. Other performance comparisons can be be found in [18], [20] and in [9], the latter reference uses a test set from the University of Florida Sparse Matrix Collection. In these three references the IDR(s)-proto variant is used.
The experiments that are presented in this section have been performed on a standard desktop computer with an Intel Core 2 duo processor and 4 Gb of RAM using MATLAB 7.5.
In all our experiments we take for $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_s$ the orthogonalization of s normally distributed random vectors, with mean 0 and standard deviation 1.

4.1 Mathematical equivalence of IDR(s)-proto and IDR(s)-biortho

The first numerical example validates that IDR(s)-proto and IDR(s)-biortho (in exact arithmetic) yield the same residual at every $s+1$-st iteration. This property is shown to be true under mild conditions in [18], Section 5.1.
To investigate this numerically we consider the the ADD20 matrix and corresponding right-hand-side vector from the MATRIX MARKET collection. The parameter ω_j is computed via the "maintaining the convergence" strategy of Sleijpen and Van der Vorst.
Figure 3(a) shows the convergence of the two IDR(s) variants for $s = 4$. The iterative processes are terminated if $\|\boldsymbol{r}_i\|/\|\boldsymbol{b}\| < 10^{-4}$. Clearly, the convergence of the two variants is quite similar for this well-conditioned problem.
The mathematical equivalence of the two variants is confirmed by the convergence curves for the first 25 iterations, that are presented in the Figure 3(b). The residual norms coincide at the crucial iterations $5, 10, 15, \ldots$.

4.2 Numerical stability of IDR(s)-proto and IDR(s)-biortho for large values of s

In order to investigate the accuracy of the two IDR(s) variants for increasingly large values of s, we consider a test problem that is taken from [6, 23]. The system matrix of this test



(a) Convergence for ADD20. (b) Zoom on the first 25 iterations.

Figure 3: Convergence of IDR(s)-proto and IDR(s)-biortho for ADD20.

problem is a complex Toeplitz matrix of order 200 and given by

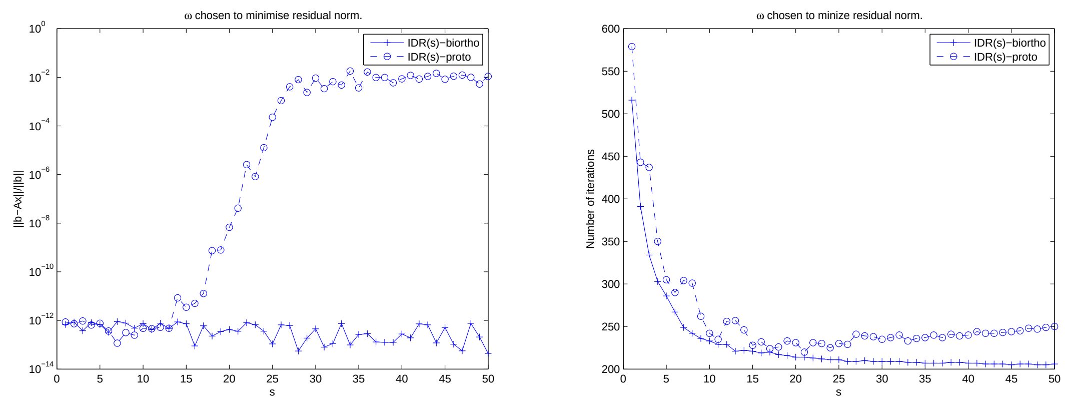
$$\boldsymbol{A} = \begin{pmatrix} 4 & 0 & 1 & 0.7 & & \\ \gamma i & \ddots & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & \ddots & 0.7 \\ & & \ddots & \ddots & \ddots & 1 \\ & & & \ddots & \ddots & 0 \\ & & & & \gamma i & 4 \end{pmatrix}$$

and the right-hand-side vector by $\boldsymbol{b} = (i, i, \ldots, i)^T$. Here, i is the imaginary unit number. For γ we take the value 3.6. The system is solved with the two IDR(s) variants with values for the parameter s ranging from 1 to 50. The iterative process is stopped if the norm of the scaled recursively computed residual drops below 10^{-12}.

In the first experiments we use the minimal residual strategy for ω_j. Figure 4(a) shows the norm of the final true residual divided by the norm of the right-hand-side vector as a function of s. Both methods yield an accurate solution for small values of s. For large values of s, however, the IDR(s)-proto method produces an inaccurate solution. The reason is that the $\boldsymbol{g}$-vectors in the original method are computed in a power-method-like way. As a result, the matrix $\boldsymbol{G}_n$ becomes ill conditioned and the solution of the systems $\boldsymbol{P}^H\boldsymbol{G}_n\boldsymbol{c} = \boldsymbol{P}^H\boldsymbol{r}_n$ inaccurate. The additional orthogonalizations in the new variant clearly improve the accuracy of the algorithm: the required accuracy for IDR(s)-biortho is always achieved.

Figure 4(b) shows the number of iterations to achieve the required tolerance. We have repeated the same experiments with the "maintaining the convergence" choice for ω_j. The most striking difference with the experiments with the "minimum residual choice" for ω_j is that the IDR(s)-proto algorithm remains accurate for much larger s. Only for s larger than 27 the accuracy is lower than expected, whereas for the minimum residual choice for ω_j the accuracy is lower than the expected level for s larger than 12. Again the final accuracy for the IDR(s)-biortho algorithm always satisfies the required tolerance.

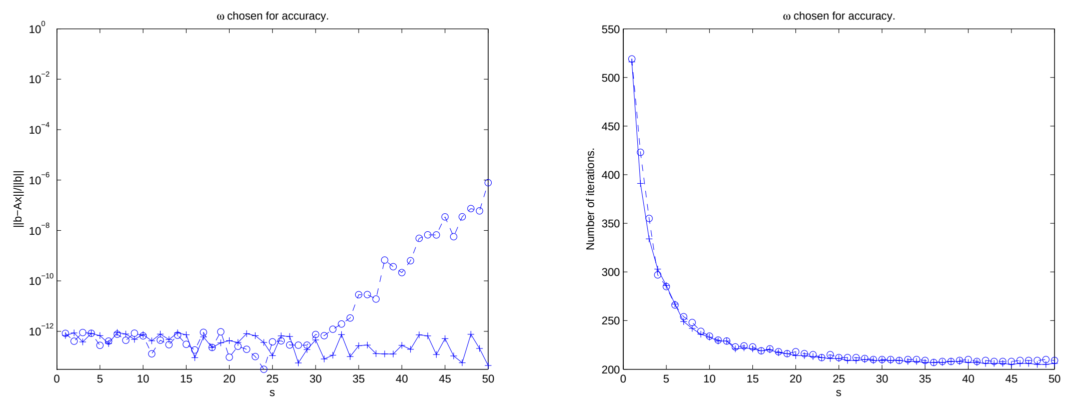
This figure shows that for s small, say up to $s = 10$, the number of iterations drops significantly with increasing s. However, for larger values of s no gain can be obtained. This is an observation, that we have often made for reasonably well-conditioned problems, has recently been explained in [17].



(a) Final accuracy.

(b) Required number of iterations.

Figure 4: Accuracy and number of iterations for the two IDR(s) variants with "minimal residual" choice for ω.



(a) Final accuracy.

(b) Required number of iterations.

Figure 5: Accuracy and number of iterations for the two IDR(s) variants with "maintaining the convergence" choice for ω.

4.3 A problem that requires a large value for s

For most problems, it is for efficiency reasons advisable to take a small value for s, e.g., $s = 4$. Problems with an ill-conditioned system matrix, however, may require a large value for s. The following example, the matrix SHERMAN2 from the MATRIX-MARKET collection with corresponding right-hand-side vector, is such a problem. We solve this problem with the two IDR(s) variants, with values of s ranging from 20 to 140. The required tolerance is $\|\boldsymbol{r}_i\|/\|\boldsymbol{b}\| < 10^{-4}$. The norm of the recursively updated residual is used in the termination criterion. Table 2 gives for the two IDR(s) variants the number of iterations and the accuracy that is reached (the norm of the true residual divided by the norm of the right-hand-side vector $\boldsymbol{b}$) for the different values of s. The maximum number of iterations is set to 2160, which is two times the problem size of 1080.

Method	Iterations	$\|\boldsymbol{b}-\boldsymbol{Ax}\|/\|\boldsymbol{b}\|$	Method	Iterations	$\|\boldsymbol{b}-\boldsymbol{Ax}\|/\|\boldsymbol{b}\|$
IDR(20)-proto	2160	$3.5 \cdot 10^{-2}$	IDR(20)-biortho	2160	$1.5 \cdot 10^{-3}$
IDR(40)-proto	1501	$1.5 \cdot 10^{-1}$	IDR(40)-biortho	969	$8.2 \cdot 10^{-5}$
IDR(60)-proto	1791	$6.6 \cdot 10^{-2}$	IDR(60)-biortho	945	$9.8 \cdot 10^{-5}$
IDR(80)-proto	1684	$5.1 \cdot 10^{-1}$	IDR(80)-biortho	883	$9.8 \cdot 10^{-5}$
IDR(100)-proto	2145	$9.3 \cdot 10^{-1}$	IDR(100)-biortho	536	$9.8 \cdot 10^{-5}$
IDR(120)-proto	1480	$5.7 \cdot 10^{-1}$	IDR(120)-biortho	473	$8.2 \cdot 10^{-5}$
IDR(140)-proto	2028	$1.7 \cdot 10^{-1}$	IDR(140)-biortho	166	$7.6 \cdot 10^{-5}$

Table 2: Sherman 2: Iterations and final accuracy for increasing s

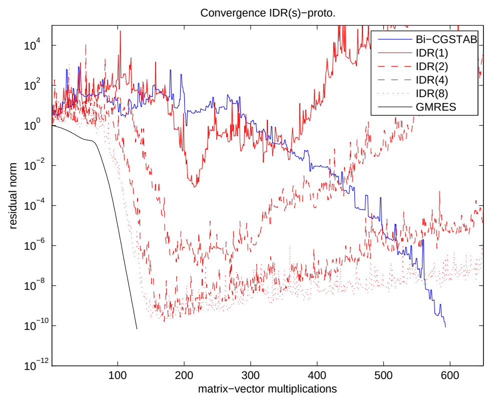
The table shows that IDR(s)-proto never achieves the required accuracy. IDR(s)-biortho, on the other hand, satisfies the accuracy for all the tested values for s, except for $s = 20$. We remark that the MATLAB built-in methods Bi-CGSTAB, QMR, Bi-GC and CGS do not converge for this problem. Full GMRES converges after 119 iterations.

4.4 A problem for which IDR(s)-proto is inaccurate for small s

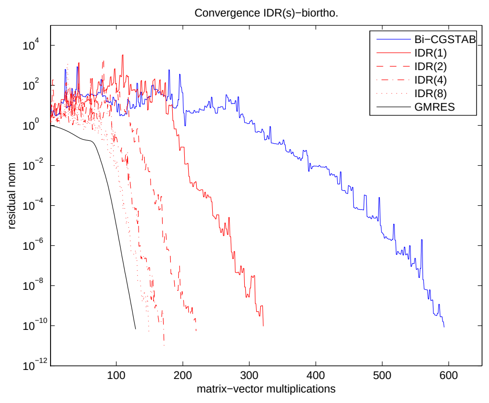
The next example that we consider shows a rare case where IDR(s)-proto does not converge for *small* s, whereas IDR(s)-biortho converges as expected. The example is an case example where IDR(s)-proto stagnates at a certain level, after which it gradually starts to diverge. We have observed this phenomenon more often if very high accuracy (close to machine precision) is required. In this example, however, stagnation of IDR(1)-proto and IDR(2)-proto occurs at a much higher level.
The example is the finite difference discretisation of the following convection-diffusion-reaction equation with homogeneous Dirichlet boundary conditions on the unit cube:

$$-\epsilon \Delta u + \vec{\beta} \cdot \nabla u - ru = F$$

The right-hand-side vector F is defined by the solution $u(x, y, z) = x(1-x)y(1-y)z(1-z)$. The problem is discretised using central differences with as grid size $h = 0.025$. The resulting linear system consists of approximately 60,000 equations. We take the following values for the parameters: $\epsilon = 1$ (diffusion), $\vec{\beta} = (0/\sqrt{5} \;\; 250/\sqrt{5} \;\; 500/\sqrt{5})^T$ (convection), and $r = 400$ (reaction). The resulting matrix is highly nonsymmetric and indefinite, properties that make the resulting system difficult to solve with an iterative solver.
The ω_j's are again computed using the "maintaining the convergence" strategy. The required tolerance is $\|\boldsymbol{r}_i\|/\|b\| < 10^{-10}$. The maximum number of matrix-vector multiplications is set to 650.
Figure 6(a) shows the convergence for IDR(s)-proto, for $s = 1, 2, 4$, and 8, together with the convergence for GMRES (optimal). For comparison we have also included the convergence of Bi-CGSTAB (MATLAB implementation). We remark that IDR(1) and the MATLAB implementation of Bi-CGSTAB are not mathematically equivalent due to the different choice for $\boldsymbol{p}_1$ (random in IDR(1) and $\boldsymbol{p}_1 = \boldsymbol{r}_0$ in Bi-CGSTAB), and because of the different strategy for computing ω_j ("maintaining the convergence" in IDR(1) and "minimum residual" in Bi-CGSTAB). These choices are standard for Bi-CGSTAB.
As can be seen, the *initial* convergence of all IDR(s) variants is satisfactory, for larger s the convergence is close to the optimal GMRES convergence, and the convergence is significantly faster than for Bi-CGSTAB. However, the convergence stagnates for all IDR(s) variants above the required, quite strict tolerance. Once stagnation occurs, IDR(s)-proto gradually starts to diverge. For the higher values of s the required tolerance is almost



(a) Convergence IDR(s)-proto.



(b) Convergence IDR(s)-biortho.

Figure 6: Accuracy and number of iterations for the two IDR(s) variants, minimal residual choice for ω.

achieved, but for small s the stagnation occurs at a level that is much higher than to be expected.

Figure 6(b) shows the convergence for IDR(s)-biortho. In this case, no stagnation and subsequent divergence occurs. Also, the difference in rate of convergence with Bi-CGSTAB is quite striking. The convergence of IDR(4) is close to the optimal GMRES convergence, while the convergence of Bi-CGSTAB lags considerably behind. Such behaviour is typical for difficult (nonsymmetric indefinite) problems.

Table 3 shows for all methods the different the number of matrix-vector multiplications methods and the final accuracy. The table shows that IDR(s)-biortho is accurate: the required tolerance is always achieved. In terms of matrix-vector multiplications, IDR(s)-

Method	MATVECS	$\|\boldsymbol{b}-\boldsymbol{Ax}\|/\|\boldsymbol{b}\|$	Method	MATVECS	$\|\boldsymbol{b}-\boldsymbol{Ax}\|/\|\boldsymbol{b}\|$
IDR(1)-proto	650	$1.4\cdot10^{+6}$	IDR(1)-biortho	321	$9.3\cdot10^{-11}$
IDR(2)-proto	650	$1.7\cdot10^{+3}$	IDR(2)-biortho	220	$5.4\cdot10^{-11}$
IDR(4)-proto	650	$1.0\cdot10^{-4}$	IDR(4)-biortho	172	$8.6\cdot10^{-12}$
IDR(8)-proto	650	$1.8\cdot10^{-7}$	IDR(8)-biortho	149	$5.7\cdot10^{-11}$
GMRES	129	$6.6\cdot10^{-11}$	Bi-CGSTAB	593	$5.1\cdot10^{-11}$

Table 3: Convection-diffusion-reaction problem: iterations and final accuracy for increasing s

biortho is much faster than Bi-CGSTAB for all values of s, and almost as fast as GMRES for $s = 4$ and $s = 8$. Note that we use full GMRES, which is optimal with respect to the number of matrix-vector multiplications, at the expense of a large overhead in vector operations and high storage requirements. The resulting CPU-times are also much lower for IDR(s)-biortho: 1.2s for IDR(4) against 10.6s for GMRES and 6.0s for Bi-CGSTAB. For completeness we also give the time for MATLABs sparse direct solve ("\"), which takes 10.2s for this example.

4.5 A performance comparison for a Navier-Stokes problem using IFISS.

IFISS is a MATLAB open source package associate with the book [2] by Elman, Silvester and Wathen. The open source code is described in [3], and can be downloaded from

`http://www.manchester.ac.uk/ifiss` and `http://www.cs.umd/ elman/ifiss.html`. IFISS can be used to model a range of incompressible fluid flow problems and provides an ideal testing environment for iterative solvers and preconditioners.
We compare the performance of IDR(s)-biortho with Bi-CGstab(ℓ), and with full GMRES. We have performed our experiments with version 3.0 of IFISS, which has implementations of all three methods. We also report on results for Bi-CGSTAB, which were actually obtained for the mathematically equivalent method BiCGstab(1).
The test problem that we consider describes flow over a step (see [2], example 7.1.2), which is modeled as a Navier-Stokes problem with zero forcing term . The steady-state Navier-Stokes equations are given by

$$-\eta\nabla^2\boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} + \nabla p = 0,$$

$$\nabla\cdot\boldsymbol{u} = 0,$$

where $\eta > 0$ is a given constant called the kinematic viscosity. The domain is L-shaped. The x- coordinate ranges from -1 to 5. The y-coordinate ranges from 0 to 1 for x between -1 and 0, and between -1 and 1 elsewhere: there is a step in the domain at $x = 0$. A Poiseuille flow profile is imposed on the inflow boundary $x = -1, 0 \leq y \leq 1$ and a zero velocity condition on the walls. The Neumann condition

$$\eta\frac{\partial u_x}{\partial x} - p = 0 \qquad \frac{\partial u_y}{\partial x} = 0$$

is applied at the outflow boundary $x = 5, -1 \leq y \leq 1$. The problem is discretised with bi-quadratic Q_2 elements for the velocities and bi-linear Q_1 elements for the pressures. The resulting nonlinear system can be solved with Newton's method, which implies that in every iteration a linear system has to be solved to compute the Newton updates. This system has the following form:

$$\begin{pmatrix} \boldsymbol{F} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{O} \end{pmatrix}\begin{pmatrix} \Delta\boldsymbol{u} \\ \Delta\boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{pmatrix} .$$

Here, the submatrix $\boldsymbol{F}$ is nonsymmetric, and becomes increasingly more nonsymmetric if η is decreased.
As a test problem we consider the linear system after one Newton iteration. A block-triangular preconditioner of the form

$$\begin{pmatrix} \boldsymbol{F} & \boldsymbol{B}^T \\ \boldsymbol{O} & -\boldsymbol{M}_s \end{pmatrix}$$

is applied to speed-up the convergence of the iterative methods. Here, $\boldsymbol{M}_s$ is an approximation to the Schur complement $\boldsymbol{S} = \boldsymbol{B}\boldsymbol{F}^{-1}\boldsymbol{B}^T$. The specific preconditioner we have selected for our experiments is the modified pressure-correction preconditioner [2]. Each application of this preconditioner requires three solves of subsystems: one solve with $\boldsymbol{F}$ and two solves with the approximate Schur complement $\boldsymbol{M}_s$. These solves are approximately done with one sweep of the AMG solver of IFISS.
In the numerical experiments, we have taken a mesh size $h = 2^{-6}$, which yields a system of 102,659 equations. We have performed experiments with increasing Reynolds numbers Re. The Reynolds number is related to the kinematic viscosity by $Re = 2/\eta$. All systems are solved to a tolerance (= reduction of the residuals norm) of 10^{-6}. Table 4 gives the numbers of matrix-vector multiplications, and in between brackets the computing times. In comparison with Bi-CGSTAB and BiCGstab(2), IDR(4), and especially IDR(8) are

Re	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)	IDR(8)
100	59 (21.8s)	106 (36.8s)	92 (32.0s)	72 (24.9s)	71 (25.1s)
200	76 (31.9s)	142 (57.2s)	144 (56.2s)	97 (37.8s)	92 (37.1s)
400	114 (52.2s)	280 (115.9s)	272 (113.6s)	156 (65.1s)	143 (60.6s)
800	181 (100.0s)	486 (284.6s)	652 (317.7s)	284 (138.6s)	231 (114.3s)

Table 4: Navier-Stokes test problem: Matrix-vector multiplications and computing times

considerably faster, in particular for large Reynolds numbers. The difference in solution time for the higher Reynolds numbers is a factor of two to three.
The preconditioner described above is quite effective in reducing the number of iterations, but is expensive to apply. This makes the preconditioned matrix-vector multiplication expensive. As a result, the time per iteration is basically determined by the preconditioned matrix-vector multiplication, and overhead for vector operations is small compared to the cost of the preconditioned matrix-vector multiplications. This situation is particularly advantageous for GMRES, since this method gives an optimal reduction of the residual norm for a given number of iterations (= preconditioned matrix-vector multiplications).
The computing times for GMRES are slightly lower than for IDR(s). However, this comes at the price of a much higher memory consumption: for every iteration an extra vector of dimension N has to be stored, whereas the storage costs for IDR(s) are fixed. For comparison: for the test problem with $Re = 800$, GMRES requires the storage of 184 vectors, IDR(4) of 16 vectors and IDR(8) of 28.

4.6 A performance comparison for large atmospheric test problems using built-in MATLAB routines

The final set of test problems is part of the University of Florida Sparse Matrix collection [1]. The four test problems come from numerical weather prediction and (three dimensional) atmospheric modeling. The systems are large, in excess of a million of equations, and have a block tri-diagonal structure with seven nonzero diagonals. The matrices are mildly nonsymmetric, and strictly diagonally dominant. As a result of these properties, the systems are well suited for iterative solution methods.
In the experiments we do not apply a preconditioner. This is realistic for this type of applications, where the matrix is normally not explicitly available. We compare the performance of IDR(s) with that of the built-in MATLAB routines Bi-CGSTAB, CGS [16], Bi-CG [4], QMR [5] and restarted GMRES, with restarts every 50th iteration. The systems are solved to a tolerance (= reduction of the residuals norm) of 10^{-8}. Table 5 gives the number of matrix-vector products that each of the methods requires, and in between brackets the required CPU times. The results show that IDR(2) and IDR(4) are both considerably faster than all the other methods, both in terms of matrix-vector multiplications and in terms of CPU-time. The above problems cannot be solved on our computer with the MATLAB direct solution method due to insufficient memory.

5 Concluding remarks

We have presented a new variant of IDR(s), called IDR(s)-biortho, that is slightly cheaper in vector overhead and according to our experiments more accurate and more stable than the original IDR(s), in particular for large s.

Iterative \ Matrix Method \ Size	ATMOSMODD $N = 1270432$	ATMOSMODJ $N = 1270432$	ATMOSMODL $N = 1489752$	ATMOSMODM $N = 1489752$
GMRES(50)	690 (980s)	1069 (1515s)	64 (95s)	38 (50s)
Bi-CGSTAB	479 (145s)	439 (131s)	89 (35s)	78 (28s)
CGS	No convergence	No convergence	No convergence	No convergence
Bi-CG	448 (79s)	430 (75s)	112 (24s)	76 (16s)
QMR	446 (94s)	436 (91s)	112 (28s)	76 (19s)
IDR(1)	520 (45s)	550 (48s)	106 (11s)	78 (9s)
IDR(2)	351 (40s)	311 (36s)	83 (12s)	59 (8s)
IDR(4)	268 (48s)	267 (48s)	69 (15s)	49 (11s)
IDR(8)	247 (80s)	248 (80s)	63 (25s)	43 (18s)

Table 5: Atmospheric model problems: Matrix-vector multiplications and computing times

For most problems it is not necessary to choose the parameter s large. In our experience $s = 4$ is a good default value. For this value there is normally little difference in numerical behaviour between IDR(s)-biortho and IDR(s)-proto. However, some problems require larger values of s, such as the ill-conditioned problem SHERMAN2 that we have presented in the numerical examples. In particular for such cases we consider the new IDR(s) variant an important improvement. In rare cases there is also a difference in numerical behaviour between IDR(s)-proto and IDR(s)-biortho for small values of s. We have presented such an example for which IDR(s)-proto failed to converge up to the required accuracy for small values of s, whereas IDR(s)-biortho reached the required accuracy for all the tested values of s.
We have also compared the performance of IDR(s)-biortho with state-of-the-art Krylov methods like GMRES, Bi-CGSTAB and BiCGstab(ℓ). These experiments confirm that IDR(s) is quite competitive and outperforms the other methods for important problem classes.

Acknowledgment: Professor Fujino of the Kyushu University is acknowledged for pointing us at the Toeplitz test problem. We thank the referees who made many detailed and constructive comments to an earlier version of this manuscript. Part of this research has been funded by the Dutch BSIK/BRICKS project.

References

[1] T. A. Davis and Y. F. Hu, The University of Florida Sparse Matrix Collection. Submitted to *ACM Transactions on Mathematical Software.*

[2] H. Elman, D. Silvester and A. Wathen. Finite Elements and Fast Iterative Solvers with application in incompressible fluid dynamics. *Oxford Science Publications*, Oxford University Press, 2005.

[3] H. Elman, A. Rammage, and D. Silvester. Algorithm 866: IFISS, a Matlab toolbox for modelling incompressible flow. *ACM Transactions on Mathematical Software*, 33(2) Article 14, 2007.

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

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

[6] M.H. Gutknecht. Variants of BICGSTAB for Matrices with Complex Spectrum. *SIAM J. Sci. Comp.* 14(5):1020–1033, 1993.

[7] M.H. Gutknecht. IDR Explained. *Electr. Trans. Numer. Anal.*, 36:126–148, 2010.

[8] M.H. Gutknecht, and J.P.M Zemke. Eigenvalue computations based on IDR. Research Report No. 2010–13, SAM, ETH Zürich, Switzerland, 2010.

[9] Y. Onoue, S. Fujino, and N. Nakashima. A Difference between Easy and Profound Preconditionings of IDR(s) Method. *Transactions of the Japan Society for Computational Engineering and Science* Paper 20080023, 2008 (in Japanese).

[10] Y. Onoue, S. Fujino, and N. Nakashima. An overview of a family of new iterative methods based on IDR theorem and its estimation. *Proceedings of the International Multi-Conference of Engineers and Computer Scientists 2009* Vol II IMECS 2009, March 18 - 20, 2009, Hong Kong, pp. 129–2134, 2009.

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

[12] V. Simoncini and D.B. Szyld, Interpreting IDR as a Petrov-Galerkin method, *SIAM J. Sci. Comp.*, 32:1898-1912, 2010.

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

[14] G.L.G. Sleijpen, P. Sonneveld and M.B. van Gijzen. Bi-CGSTAB as an induced dimension reduction method. *Applied Numerical Mathematics* (Article in Press), doi:10.1016/j.apnum.2009.07.001, 2009

[15] G.L.G. Sleijpen and H.A. van der Vorst. Maintaining convergence properties of BiCGstab methods in finite precision arithmetic. *Numerical Algorithms*, 10:203–223, 1995.

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

[17] P. Sonneveld. On the convergence behaviour of IDR(s). Technical Report 10-08, Department of Applied Mathematical Analysis, Delft University of Technology, Delft, The Netherlands, 2010.

[18] P. Sonneveld and M.B. van Gijzen. IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems. *SIAM J. Sci. Comp.*, 31(2):1035–1062, 2008.

[19] 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.*, 13:631–644, 1992.

[20] M.B. van Gijzen and P. Sonneveld. The IDR(s) method for solving nonsymmetric systems: a performance study for CFD problems. In: *High Performance Algorithms for Computational Science and Their Applications.* Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan, 2008

[21] 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.

[22] M. Yeung and T.F. Chan. ML(k)BiCGSTAB: A BiCGSTAB Variant based on multiple Lanczos starting vectors. *SIAM J. Sci. Comp.*, 21:1263–1290, 1999.

[23] S. L. Zhang. GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 18(2):537–551, 1997.