DELFT UNIVERSITY OF TECHNOLOGY

REPORT 08-21

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

Martin B. van Gijzen and Peter Sonneveld

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 [7] is a very competitive 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 [7] is a direct translation of the theorem into an algorithm. This translation is not unique. This paper derives a new IDR(s) variant. This new variant imposes bi-orthogonalization conditions on the iteration vectors, which results in a very elegant method with 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 will show through numerical experiments that our new variant is more accurate than the original IDR(s) for large values of s. We will also present a numerical comparison with GMRES [3], Bi-CGSTAB [8], and BiCGstab(ℓ) [4] to illustrate the efficiency of IDR(s).

Keywords. Iterative methods, IDR, IDR(s), Krylov-subspace methods, bi-orthogonalization

AMS subject classification. 65F10, 65F50

1 Introduction

We consider the linear system

$$Ax = b$$

with $A \in \mathbb{C}^{N \times N}$ a large, sparse and non-symmetric matrix. In 1980, Sonneveld proposed the iterative method IDR [9] 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 $\mathcal{G}_j = (I - \omega_j A)(\mathcal{S} \cap \mathcal{G}_{j-1})$, where $\mathcal{S}$ is a fixed proper subspace of $\mathbb{C}^N$, and the ω_j's are non-zero scalars.

Recently, it was recognize 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) [7], a highly competitive method for solving large nonsymmetric linear systems. The examples that are described in [7] show that IDR(s), with $s > 1$ and not too big, outperforms the well-known Bi-CGSTAB method [8] 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 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[10] of Yeung and

---

*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`

Chan. We refer to [7] for the details. In [5], Bi-CGSTAB is considered as an IDR method, and this paper explains how IDR ideas can be incorporated into Bi-CGSTAB.

The prototype IDR(s) algorithm that is described in [7] 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. 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 are uniquely determined. They 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 bi-orthogonality conditions between the intermediate residuals and the pre-chosen vectors $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_s$, that define the subspace $\mathcal{S}$. Our new IDR variant uses less vector operations per iteration than the original IDR(s) method, and remains stable and accurate 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 new IDR(s) variant by filling in the freedom in generating the intermediate residuals by imposing bi-orthogonality conditions between the intermediate residuals and the vectors $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_s$.

Section 4 presents numerical experiments that compare the new IDR(s) variant with the original IDR(s) method presented in [7].

Section 5 evaluates the new IDR(s) variant for CFD problems and compares its performance with GMRES [3], Bi-CGSTAB, and BiCGstab(2) [4].

We make concluding remarks in Section 6.

2 Making an IDR-based algorithm

2.1 The IDR theorem

At the basis of every IDR algorithm is the IDR Theorem [7], 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*

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

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 $j \leq N$.

For the proof we refer to [7].

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 $\boldsymbol{Q}$:

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

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 $\mathcal{K}^n(\boldsymbol{A}, \boldsymbol{r}_0)$. 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

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha\boldsymbol{Av}_n - \sum_{i=1}^{\widehat{i}} \gamma_i \boldsymbol{g}_{n-i} \tag{1}$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha\boldsymbol{v}_n + \sum_{i=1}^{\widehat{i}} \gamma_i \boldsymbol{u}_{n-i}$$

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

$$\boldsymbol{g}_i = \boldsymbol{Au}_i . \tag{2}$$

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, $\boldsymbol{r}_n \in \{\boldsymbol{0}\}$ 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

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

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

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

which corresponds to (1).

Now suppose that $\boldsymbol{r}_n, \boldsymbol{g}_{n-i} \in \mathcal{G}_j$, $i = 1, \cdots, \widehat{i}$. This implies that $\boldsymbol{v}_n \in \mathcal{G}_j$. If we choose $\gamma_i, i = 1, \cdots, \widehat{i}$ 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 $\boldsymbol{v}_n \in \mathcal{S} = \mathcal{N}(\boldsymbol{Q}^H)$, it satisfies

$$\boldsymbol{Q}^H \boldsymbol{v}_n = \boldsymbol{0} . \tag{5}$$

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 $\boldsymbol{g}_i \in \mathcal{G}_j$, $i = n-1, \cdots n-s$, and s corresponding vectors $\boldsymbol{u}_i$ with $\boldsymbol{g}_i = \boldsymbol{A}\boldsymbol{u}_i$ at our disposal. Define the matrices

$$\boldsymbol{G}_n = \begin{pmatrix} \boldsymbol{g}_{n-s} \ \boldsymbol{g}_{n-s+1} \ \cdots \ \boldsymbol{g}_{n-1} \end{pmatrix}, \tag{6}$$

$$\boldsymbol{U}_n = \begin{pmatrix} \boldsymbol{u}_{n-s} \ \boldsymbol{u}_{n-s+1} \ \cdots \ \boldsymbol{u}_{n-1} \end{pmatrix}. \tag{7}$$

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 $(\boldsymbol{Q}^H \boldsymbol{G}_n)\boldsymbol{c} = \boldsymbol{Q}^H \boldsymbol{r}_n$,

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

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

Using (4), the new residual can also be computed by

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

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

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

Note that this calculation 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. We remark that the (local) minimal residual norm strategy for selecting ω_{j+1} may lead to small ω-values, which can result in numerical problems. In that case the more stable strategy that is described in [6] should be used to compute a suitable ω_{j+1}.

The above framework explains how a residual in $\mathcal{G}_{j+1}$ can be computed given $\boldsymbol{r}_n, \boldsymbol{g}_{n-i} \in \mathcal{G}_j$, $i = 1, \cdots s$. 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 $\boldsymbol{g}_i \in \mathcal{G}_j, i = n-1, \cdots n-s$ and $\boldsymbol{r}_{n+1} \in \mathcal{G}_{j+1} \subset \mathcal{G}_j$, the computations

Calculate: $\boldsymbol{c} \in \mathbb{C}^s$ from $(\boldsymbol{Q}^H \boldsymbol{G}_n)\boldsymbol{c} = \boldsymbol{Q}^H \boldsymbol{r}_{n+1}$,

$$\boldsymbol{v}_{n+1} = \boldsymbol{r}_{n+1} - \boldsymbol{G}_n \boldsymbol{c},$$

$$\boldsymbol{r}_{n+2} = \boldsymbol{v}_{n+1} - \omega_{j+1} \boldsymbol{A} \boldsymbol{v}_{n+1} .$$

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

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

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

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

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. Moreover, for stability reasons it is also advisable to use the most recent information in the iterative process.

We can repeat the above procedure s times to compute $\boldsymbol{r}_{n+s+1}, \boldsymbol{g}_{n+k} \in \mathcal{G}_j, \ k = 1, \cdots s$, and the corresponding vectors $\boldsymbol{x}_{n+s+1}, \boldsymbol{u}_{n+k}, \ k = 1, \cdots s$, 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 [7]. 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 $\boldsymbol{G}_{n+k}, k = 1, \cdots, s$.

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 $\mathcal{G}_{j+1}$:

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

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 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_k$.

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 $\boldsymbol{q}_1 \cdots \boldsymbol{q}_{k-1}$.

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. In the prototype algorithm that we present in Figure 1 this is (implicitly) done in the first s steps by Richardson iteration. In Figure 1 we present a framework for an IDR-based algorithm. The freedom that is left open is in the choice of the parameters α_i and β_i. In the algorithm we have omitted the indices for the iteration number. Vectors on the left are overwritten by vectors on the right.

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 orthogonality conditions:

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

and

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

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 orthogonality condition for the intermediate residuals (13) implies that the first intermediate residual is orthogonal to $\boldsymbol{q}_1$, the second to $\boldsymbol{q}_1$ and to $\boldsymbol{q}_2$, etc. Hence, the last intermediate residual before making a dimension reduction step, i.e. $\boldsymbol{r}_n$ is orthogonal to $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_s$. 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 $(\boldsymbol{Q}^H \boldsymbol{G}_{n+k})\boldsymbol{c} = \boldsymbol{Q}^H \boldsymbol{r}_{n+k}$ has to be solved. Using the conditions (12) and (13) this system gets a simple form. Let

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

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

  {*Loop over $\mathcal{G}_j$ spaces*}
  while $\|r\| > TOL$ **do**
      {*Compute s independent vectors g_k in $\mathcal{G}_j$ space*}
      for $k = 1$ to s **do**
          Compute $M = Q^H G$ and $f = Q^H r$;
          Solve c from $Mc = f$;
          $v = r - Gc$;
          $u_k = Uc + \omega v$;
          $g_k = Au_k$;
          {*Linear combination of vectors $\in \mathcal{G}_j$ are still in $\mathcal{G}_j$:*}
          $g_k = g_k - \sum_{i=1}^{k-1} \alpha_i g_i$; $u_k = u_k - \sum_{i=1}^{k-1} \alpha_i u_i$;
          $r = r - \sum_{i=1}^{k} \beta_i g_i$; $x = x + \sum_{i=1}^{k} \beta_i u_i$;
          $G = (g_1 \cdots g_k)$; $U = (u_1 \cdots u_k)$;
      end for
      { *Entering $\mathcal{G}_{j+1}$*}
      Compute $M = Q^H G$ and $f = Q^H r$;
      Solve c from $Mc = f$;
      $v = r - Gc$;
      $t = Av$;
      $\omega = (t^H v)/(t^H t)$;
      $x = x + Uc + \omega v$;
      $r = r - Gc - \omega t$;
  end while

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

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

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

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

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

Clearly, $\gamma_1, \cdots, \gamma_{k-1}$ 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

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

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

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

$$\alpha = \frac{\boldsymbol{q}_i^H \boldsymbol{g}_{n+k}}{\mu_{i,i}};$$

$$\boldsymbol{g}_{n+k} = \boldsymbol{g}_{n+k} - \alpha \boldsymbol{g}_{n+i};$$

$$\boldsymbol{u}_{n+k} = \boldsymbol{u}_{n+k} - \alpha \boldsymbol{u}_{n+i}.$$

End for

The next step in the algorithm is to compute the next intermediate residual $\boldsymbol{r}_{n+k+1}$ that is orthogonal to $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_k$. It is easy to check that such a residual can be computed by

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

The corresponding approximate solution then becomes

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

From the outline of the algorithm it seems that we have to compute the inner products $\phi_i, i = k, \ldots, s$, and $\mu_{i,k}, i = k, \cdots, s$. From (14) it is clear that, given the inner products $\mu_{i,k} = \boldsymbol{q}_i^H \boldsymbol{g}_{n+i}$, the new value of the inner products $\phi_i = \boldsymbol{q}_i^H \boldsymbol{r}_{n+k+1}$ can be computed via the scalar update

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

Require: $A \in \mathbb{C}^{N \times N}$; $x, b \in \mathbb{C}^N$; $Q \in \mathbb{C}^{N \times s}$; $TOL \in (0, 1)$;
Ensure: x_n such that $\|b - Ax\| \leq TOL$;
  {*Initialization.*}
  Calculate $r = b - Ax$;
  $g_i = u_i = 0, i = 1, \cdots, s$; $M = I$; $\omega = 1$;

  {*Loop over $\mathcal{G}_j$ spaces, for $j = 0, 1, 2, 3, \ldots$*}
  while $\|r\| > TOL$ **do**
      {*Compute s independent column vectors for $\mathcal{G}_j$*}
      $f = Q^H r$, $f = (\phi_1, \cdots, \phi_s)^T$;
      for $k = 1$ to s **do**
          {*Compute $v \in \mathcal{G}_j \cap \mathcal{S}$*}
          Solve c from $Mc = f$, $c = (\gamma_1, \cdots, \gamma_s)^T$;
          $v = r - \sum_{i=k}^s \gamma_i g_i$
          $u_k = \omega v + \sum_{i=k}^s \gamma_i u_i$;
          {*Compute $g_k \in \mathcal{G}_j$*}
          $g_k = A u_k$;
          {*Make g_k orthogonal to $q_1 \cdots q_{k-1}$*}
          for $i = 1$ to $k - 1$ **do**
              $\alpha = \dfrac{q_i^H g_k}{\mu_{i,i}}$;
              $g_k = g_k - \alpha g_i$;
              $u_k = u_k - \alpha u_i$;
          end for
          {*Compute $q_i^H g_k, i = k \cdots s$ update M*}.
          $\mu_{i,k} = q_i^H g_k, i = k \cdots s$, $M_{i,k} = \mu_{i,k}$
          {*Make the residual orthogonal to $q_1 \cdots q_k$*};
          $\beta = \dfrac{\phi_k}{\mu_{k,k}}$;
          $r = r - \beta g_k$:
          $x = x + \beta u_k$;
          {*Update $f = Q^H r$*};
          if $k + 1 \leq s$ **then**
              {*The first k entries in f are 0*}
              $\phi_i = 0, i = 1, \cdots k$
              $\phi_i = \phi_i - \beta \mu_{i,k}, i = k + 1, \cdots, s$
              $f = (\phi_1, \cdots, \phi_s)^T$
          end if
      end for
      {*Entering $\mathcal{G}_{j+1}$*}
      $t = Ar$;
      $\omega = (t^H r)/(t^H t)$;
      $r = r - \omega t$;
      $x = x + \omega r$;
  end while

Figure 2: IDR(s) with bi-orthogonalization of intermediate residuals.

Figure 2 presents the algorithm that is outlined above. In the algorithm we have omitted the indices for the iteration number.

The above algorithm is quite efficient in terms of vector operations and even more efficient than the original IDR(s) method, despite the additional orthogonalization operations. The operation count for the main operations to perform the dimension reduction step yields: one matrix-vector product, two vector updates and two inner products. For the intermediate steps we get: s 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: $s+1$ matrix-vector products, s^2+s+2 inner products and $2s^2+2s+2$ vector updates. The new IDR(s) variant requires slightly less vector updates than the original IDR(s) algorithm, and the same number of inner products and 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 matrix-vector multiplication for some values of s for the new IDR(s) variant, 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).

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-CGSTAB	2	3	7
BiCGstab(2)	$2\frac{1}{4}$	$3\frac{3}{4}$	9

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

4   Numerical comparison of the original IDR(s) and IDR(s) with bi-orthogonalization of the intermediate vectors

In this section we present experiments to compare the original IDR(s) method and the new variant. In all our experiments we take for $q_1, \cdots, q_s$ orthogonalized random vectors. The parameter ω_j is computed via the standard minimum residual strategy.

4.1   Mathematical equivalence of the original IDR(s) and IDR(s) with bi-orthogonalization of the intermediate vectors

The first numerical example validates that standard IDR(s) and IDR(s) with bi-orthogonalization of the intermediate vectors in theory yield the same residual at every $s + 1$-st iteration. To investigate this numerically we consider the the ADD20 matrix from the MATRIX MARKET collection. We have taken a right-hand-side vector corresponding to a solution vector that consists of ones. Figure 3 shows the convergence of the two IDR(s) variants for $s = 4$.

The numerical equivalence of the two variants is confirmed by the convergence curves for the first 25 iterations, that are presented in the lower part of Figure 3. The residual norms

coincide at the crucial iterations $5, 10, 15, \cdots$.

The numerical stability properties of the two methods are different, resulting in a different ultimate convergence behaviour. In this case, the IDR(s) variant with bi-orthogonalization converges slightly faster.
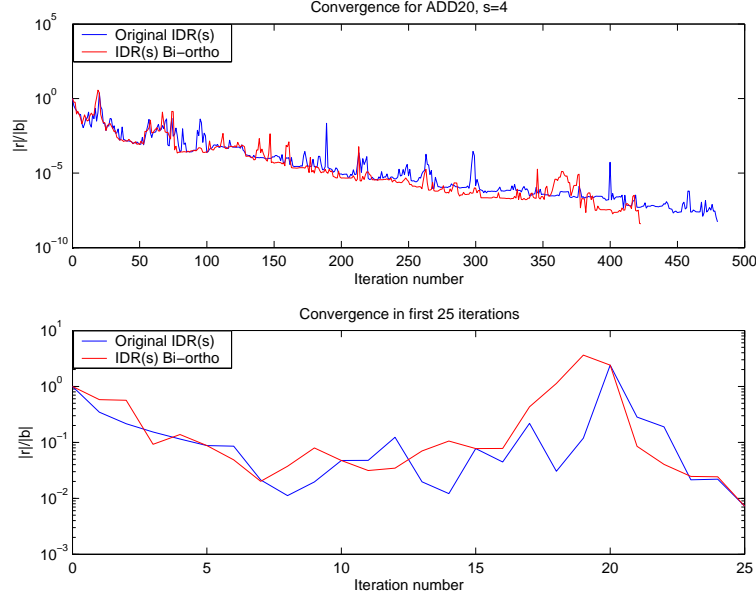

Figure 3: Convergence of the two IDR(4) variants.

4.2 Numerical stability of standard IDR(s) and IDR(s) with bi-orthogonalization of intermediate vectors 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 was taken from [2, 11]. The system matrix of this test problem is complex Toeplitz 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 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}. The upper part of Figure 4 shows the norm of the final true residual. As is apparent, both methods yield an accurate solution for small values of s. For large values of s, however, the original IDR(s) 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 $\boldsymbol{Q}^H \boldsymbol{G}_n \boldsymbol{c} = \boldsymbol{Q}^H \boldsymbol{r}_n$ inaccurate. The additional orthogonalizations in the new variant clearly improve the accuracy of the algorithm.
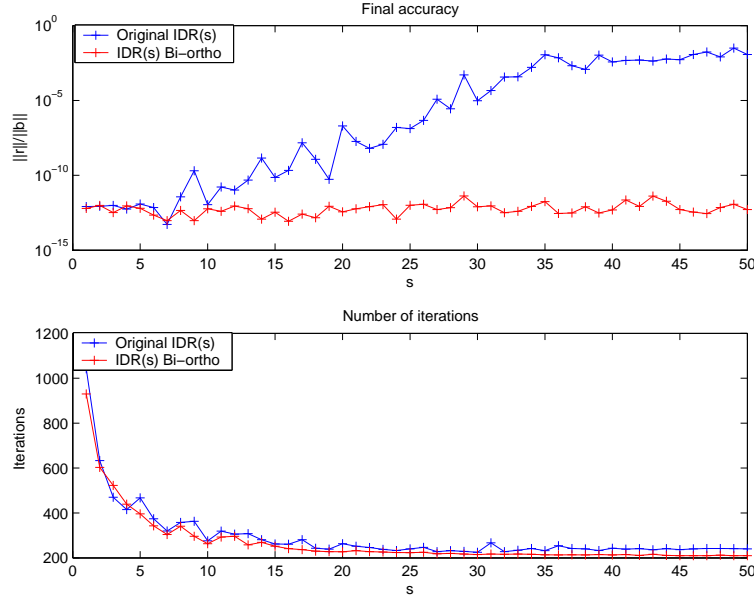

Figure 4: Accuracy and number of iterations for the two IDR(s) variants

The lower part of Figure 4 shows the number of iterations to achieve 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. It is hardly ever necessary to choose s larger than 10.

5 Performance comparison with GMRES and BiCGstab(ℓ) using IFISS

5.1 IFISS

The performance comparison that we present in this section have been carried out with the Finite Element software package IFISS. IFISS is a MATLAB open source package associate with the book [1] by Howard Elman, David Silvester and Andy Wathen. The open source code has been developed by Alison Rammage, David Silvester and Howard Elman, and can be downloaded from the web[1]. The program can be used to model a range of incompressible fluid flow problems and provides an ideal testing environment for iterative solvers and preconditioners. The package has implementations of some of the most powerful iterative methods, like Bi-CGstab(ℓ) [4], and GMRES [3], and of many advanced preconditioners. In the experiments we will use the new variant of IDR(s) and focus on the comparison of IDR(4) and IDR(8) with BiCGstab(2), the default variant of BiCGstab(ℓ) in IFISS, with full GMRES, and with Bi-CGSTAB. For Bi-CGSTAB we simply use the mathematically equivalent method BiCGstab(1).

---

[1]`http://www.manchester.ac.uk/ifiss` and `http://www.cs.umd/ elman/ifiss.html`

In all experiments we take for $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_s$ orthogonalized random vectors. The parameter ω_j is computed using the technique described in [6], which strategy is used by default in the BiCGstab(ℓ) implementation of IFISS.

5.2 A convection-diffusion problem

5.2.1 Description of the test problem

The first problem we consider is example 3.1.3 in [1]. This is a convection diffusion problem with zero source term,

$$-\epsilon \nabla^2 u + \boldsymbol{w} \cdot \nabla u = 0 \quad (x, y) \in (-1, 1) \times (-1, 1)$$

with constant wind $\boldsymbol{w}$ at a 30° angle to the left of the vertical, i.e.

$$\boldsymbol{w} = \begin{pmatrix} -\sin \frac{\pi}{6} \\ \cos \frac{\pi}{6} \end{pmatrix} .$$

Dirichlet boundary conditions are imposed on all sides of the domain and are as follows:

$$u = 0 \ \text{ if } x = -1 \text{ or } y = -1, x < 0 \text{ or } y = 1$$

and

$$u = 1 \ \text{ if } y = -1, x \geq 0 \text{ or } x = 1 .$$

The solution solution has a boundary layer near $y = 1$ and an internal boundary layer due to the jump discontinuity at (0,-1). The problem is discretised with square bi-linear Q_1 elements, using a mesh size of $h = 2^{-7}$, which yields a nonsymmetric linear system of 65025 equations.

5.2.2 On the choice of s

Before we present the actual comparison of IDR(4) and IDR(8) with the other methods, we will first show two examples to motivate our choice for $s = 4$ and $s = 8$. For the first example we take $\epsilon = 0.01$. We solve the resulting linear system with IDR(s), with $s = 1$, $s = 2$, $s = 4$, and $s = 8$. The convergence behavior of the different IDR(s) variants is shown in Figure 5. The results show that a considerable reduction in number of iterations is obtained for $s = 2$ and for $s = 4$ but only a modest reduction is obtained by taking $s = 8$. As a results $s = 4$ is the optimal value with respect to computing time.
For the second example we take $\epsilon = 0.001$. This example is convection dominated. Since we do not use any stabilization technique like SUPG, the resulting system matrix is strongly a-symmetric. For this example, IDR(s) does not converge for $s = 1, 2$ and 4. For $s = 8$ and $s = 16$, however, convergence of the method is satisfactory. In this case, IDR(s) is more robust for larger values of s.

5.2.3 Performance comparison for the convection-diffusion problem

In the next experiments we consider increasingly small values of the diffusion parameter ϵ, with values ranging from 1 to 10^{-4}. It is well known that if ϵ is too small with respect to the mesh size a stabilization procedure like Streamline Upwind Petrov-Galerkin (SUPG) should be applied in order to avoid unwanted numerical oscillations in the solution. We give the numerical results both for the in practice more relevant stabilized case (if necessary),
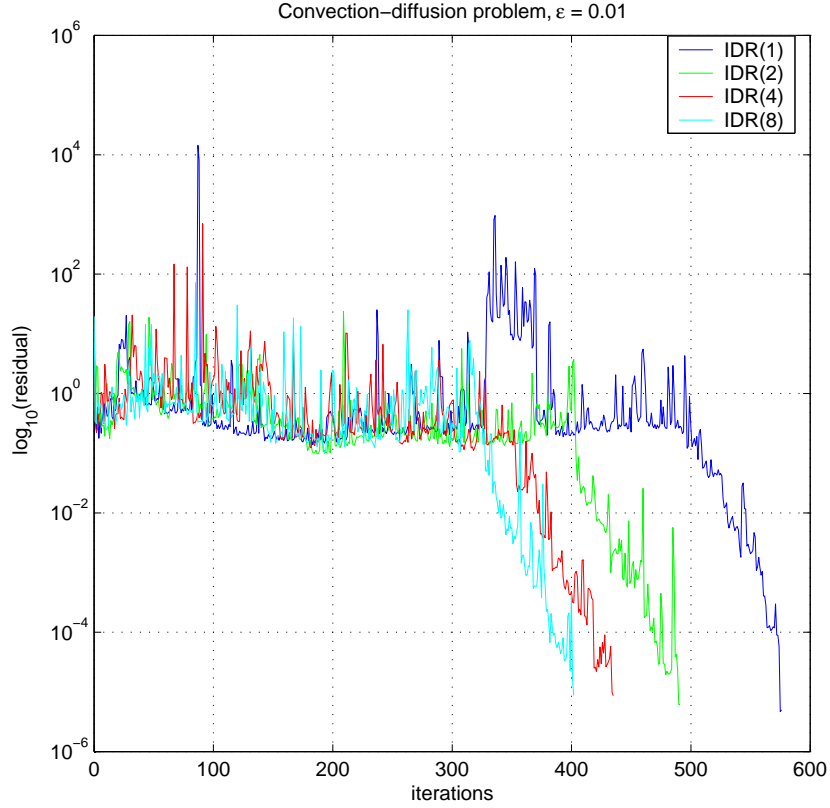
Figure 5: Convergence history of IDR(s), for different values of s

and for the unstabilized case to investigate the performance of the iterative methods for increasingly skew-symmetric systems.

Figure 7 shows for each of the five iterative methods the required number of matrix-vector products (MATVECS) to solve a system with a given diffusion parameter ϵ to a tolerance (= reduction of the residuals norm) of 10^{-6}. The solid lines show the results if no stabilization is used, and the dashed lines for the systems with SUPG stabilization. Note that for the larger values of ϵ SUPG stabilization is not necessary and therefore not used. No preconditioner is applied in the experiments.

Since full GMRES is optimal with respect to the number of MATVECS, this method always needs the least amount of MATVECS. But, as was remarked before, the overhead in vector operations and memory requirements is much larger for this method. IDR(4) and Bi-CGSTAB do not converge for the strongly a-symmetric systems, i.e. for small values of ϵ without SUPG stabilization. Also IDR(8) does not converge for ϵ too small, but as we saw before, the method still converges for $\epsilon = 0.001$, this in contrast to IDR(4) and Bi-CGSTAB. In the strongly a-symmetric case, the system matrix has complex eigenvalues with large imaginary parts. For such problems the linear minimization steps that are used in both IDR(s) and in Bi-CGSTAB do not work well. BiCGstab(2), however, uses quadratic minimization polynomials that also work well in the strongly a-symmetric case. As a result BiCGstab(2) converges in all the cases. For the physically realistic problems with SUPG stabilization, however, IDR(4) and IDR(8) are always faster then both Bi-CGSTAB and Bi-CGSTAB(2), and for $\epsilon = 1$ even much faster then Bi-CGSTAB.

The computing times are shown in Table 2. The results show that GMRES, although the
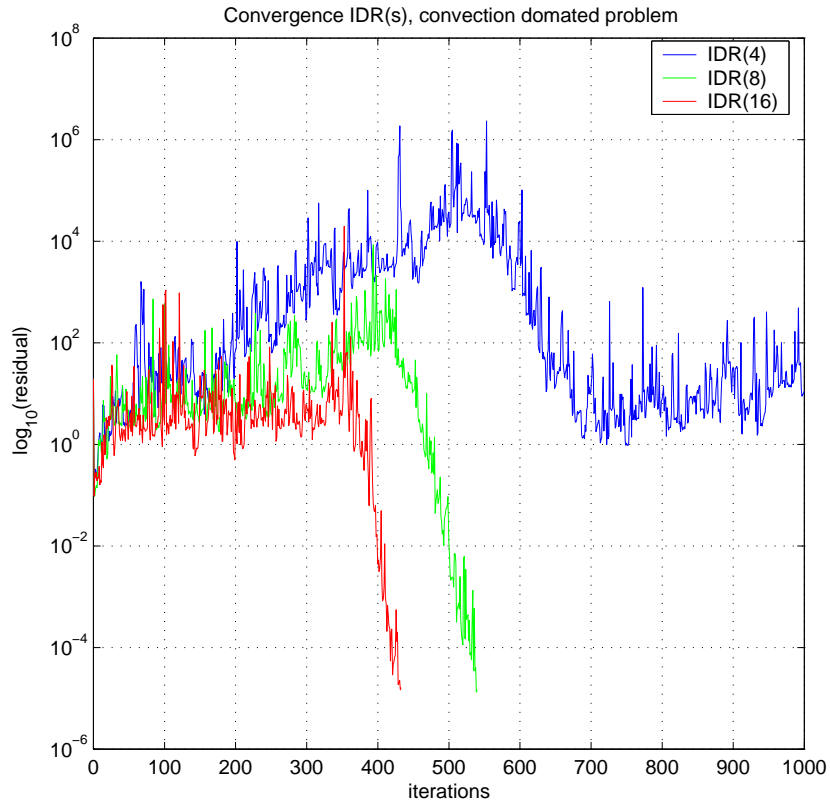
14

Figure 6: Convergence history of IDR(s), for different values of s

Diffusion ϵ	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)	IDR(8)
1	187	45	42	29	42
0.1	152	28	23	26	35
0.01	124	25	26	24	33
0.001	102	n.c.	27	n.c.	43
0.0001	245	n.c.	32	n.c.	n.c.
0.001 (SUPG)	88	26	25	22	28
0.0001 (SUPG)	91	26	25	21	29

Table 2: Computing times [s] for solving the convection-diffusion problem

fastest in terms of MATVECS, is much slower in computing time than the other methods. This is due to the fact that the matrix-vector product (without preconditioning) is cheap in this example, and the number of iterations is large. This is an unfavorable situation for GMRES. Also, due to the overhead in vector operations, IDR(8) is slower than the other methods. IDR(4) is always faster in time than Bi-CGSTAB, and for $\epsilon = 1$ considerably faster. In this case IDR(4) is also significantly faster than BiCGstab(2). As was remarked before, BiCGstab(ℓ) is the preferred method for the strongly a-symmetric problems, in which cases neither Bi-CGSTAB nor IDR(s) converge.
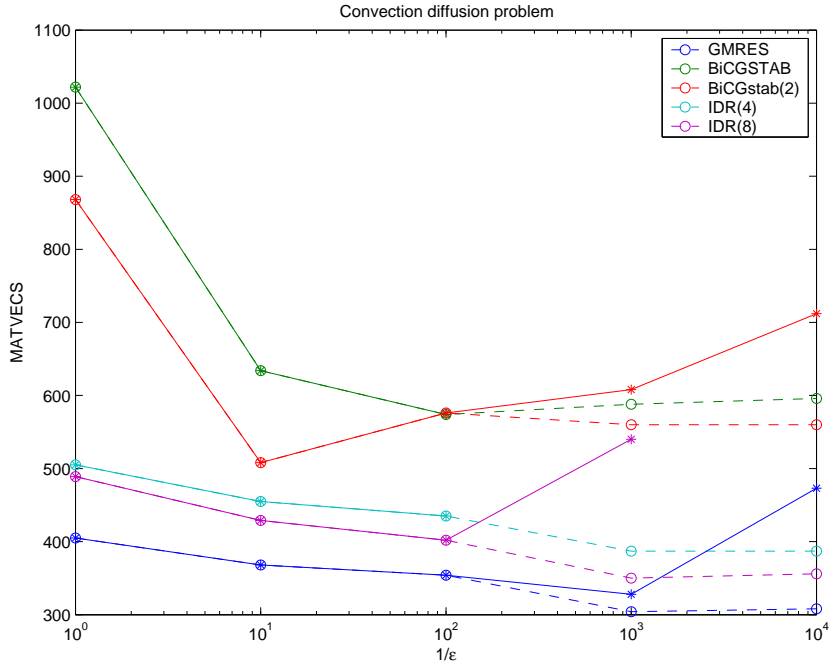
Figure 7: Number of matvecs for GMRES, Bi-CGSTAB, BiCGstab(2), IDR(4), and IDR(8) for different diffusion parameters, solid without SUPG, dashed with SUPG.

5.3 A Navier-Stokes problem

5.3.1 Description of the test problem

The second example that we consider is a Navier-Stokes problem with zero forcing term. The example describes flow over a step (see [1], example 7.1.2). 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

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

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:

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

16

Here, the submatrix $\boldsymbol{F}$ is nonsymmetric, and becomes increasingly more a-symmetric of 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 ideal pressure-convection diffusion preconditioner [1]. 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 done with MATLAB's direct sparse solver.

The preconditioner described above is quite effective in reducing the number of iterations, but makes the preconditioned matrix-vector multiplication very expensive. As a result, the time per iteration is basically determined by the preconditioned matrix-vector multiplication, and overhead for vector operations is negligible. This situation is particularly advantages for GMRES, since this method gives an optimal reduction of the residual norm for a given number of iterations (= preconditioned matrix-vector multiplications). This is the opposite situation that we had for the convection-diffusion test problem, where many iterations had to be performed to achieve a required tolerance, but where the matrix-vector multiplication was a cheap operation.

5.3.2 Performance comparison for the Navier-Stokes problem

In the numerical experiments, we have varied two parameters in the test problem: the step size h, and the Reynolds number, which 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}. Tables 3 - 5 give the number of matrix-vector multiplications, and in between brackets the computing times.

Reynolds number Re	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)	IDR(8)
10	23 (2.1s)	34 (3.0s)	36 (3.1s)	28 (2.4s)	27 (2.3s)
100	47 (4.2s)	106 (8.8s)	116 (9.7s)	68 (5.5s)	62 (5.0s)
200	76 (6.9s)	242 (20.7s)	236 (20.6s)	119 (10.3s)	103 (8.9s)

Table 3: Matrix-vector multiplications and computing times, $h = 2^{-3}$, 1747 equations

Reynolds number Re	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)	IDR(8)
10	27 (16.7s)	36 (19.5s)	36 (19.5s)	32 (16.7s)	32 (16.6)
100	50 (26.5s)	146 (76.5s)	128 (67.1s)	84 (45.3s)	70 (36.5)
200	72 (38.1s)	356 (185s)	276 (143)	134 (69.4s)	106 (55.4)

Table 4: Matrix-vector multiplications and computing times, $h = 2^{-4}$, 6659 equations

The results show, as predicted, that GMRES is the best method for this set of test problems. We remark, however, that the implementation that we used for the (action of the) preconditioner uses three direct solves, which is too expensive in a realistic setting. In a realistic setting, approximations to the direct solves have to be used. This will result in a

Reynolds number Re	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)	IDR(8)
10	31 (113s)	44 (153s)	44 (154s)	38 (127s)	35 (117s)
100	59 (201s)	214 (724s)	208 (703s)	103 (346s)	86 (288s)
200	81 (276s)	398 (1343s)	420 (1414s)	172 (577s)	132 (444s)

Table 5: Matrix-vector multiplications and computing times, $h = 2^{-5}$, 25987 equations

cheaper, but less effective preconditioner. In this situation short-recurrence methods like IDR(4) and IDR(8) may be competitive again, or possibly even be required because of the limited memory consumption.

In comparison with Bi-CGSTAB and BiCGstab(2), IDR(4), and especially IDR(8) are considerably faster, in particular for large Reynolds numbers. The difference in solution time for $Re = 200$ is a factor of two to three for all three grid sizes.

6   Concluding remarks

We have presented a new variant of IDR(s) that is cheaper in vector overhead and according to our experiments more accurate and more stable than the original IDR(s) for large s. We have evaluated this new variant and compared it with state-of-the-art Krylov methods like GMRES, Bi-CGSTAB and BiCGstab(ℓ). The performance of IDR(s) is vary favorable, in particular in comparison with Bi-CGSTAB.

In general it is wise to choose the parameter s not too large. In our experience $s = 4$ is a good default value. For this value there is little difference in numerical behaviour between the original IDR(s) method and the variant that we have described in this paper. However, some problems require larger values of s, such as the highly a-symmetric convection-diffusion problem that we have discussed. In particular for such cases we consider the new IDR variant an important improvement.

References

[1] 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.

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

[3] 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.

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

[5] G.L.G. SLEIJPEN, P. SONNEVELD and M.B. VAN GIJZEN. Bi-CGSTAB as an induced dimension reduction method. Technical Report 08-07, Department of Applied Mathematical Analysis, Delft University of Technology, Delft, The Netherlands, 2008.

[6] 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

[7] 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. Comput.*, to appear

[8] 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.

[9] 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.

[10] 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.

[11] 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.