数理解析研究所講究録
第1614巻 2008年 53-63

The IDR(s) method for solving nonsymmetric systems: a performance study for CFD problems.

Martin B. van Gijzen and Peter Sonneveld*

Abstract

The recently proposed method IDR(s) [6] for solving large and sparse nonsymmetric systems of linear equations has proven to be highly efficient for important classes of applications. This paper presents an performance comparison of IDR(s) with GMRES [3], Bi-CGSTAB [7] and BiCGstab(ℓ) [4] for representative fluid dynamics test problems. The computations are done with the MATLAB finite element program IFISS [1]. In our computations we only consider a default value of $s = 4$ for the parameter in IDR(s). Our experimental results show that IDR(4) is a promising method for solving the type of incompressible flow problems that we consider in this paper. The method is based on short recurrences and therefore more efficient than GMRES in memory consumption and computing time if many GMRES-iterations have to be performed. Furthermore, IDR(4) is competitive or faster than Bi-CGSTAB and BiCGstab(ℓ) for all physically relevant examples.

1 Introduction.

We consider the linear system

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

with $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ a large, sparse and nonsymmetric matrix. In 1980, Sonneveld proposed the iterative method IDR [8] for solving such systems. IDR has several favorable features: it is simple, uses short recurrences, and computes the exact solution in at most $2N$ steps (matrix-vector multiplications) in exact arithmetic. Analysis of IDR revealed a close relation with Bi-CG [2]. It was shown in [8] that the iteration polynomial constructed by IDR is the product of the Bi-CG polynomial with another, locally minimizing polynomial. Sonneveld's observation that the Bi-CG polynomial could be combined with another polynomial without matrix-vector multiplications with the transpose $\boldsymbol{A}^T$ led to the development first of CGS [5], and later of Bi-CGSTAB [7].

Over the years, CGS and Bi-CGSTAB have completely overshadowed IDR. This was unfortunate since, although there is a clear relation between Bi-CG-type methods and the original IDR method, the underlying ideas are completely different. This suggests that by exploiting the differences new methods may be developed. This idea has led to the development of IDR(s) [6], which is a generalization of the original IDR method.

Bi-CG, CGS, Bi-CGSTAB and their descendants are essentially based on the computation of two mutually bi-orthogonal bases for the Krylov subspaces $\mathcal{K}^n(\boldsymbol{A}, r_0)$ and $\mathcal{K}^n(\boldsymbol{A}^T, \tilde{r}_0)$. The IDR family, on the other hand, generates residuals that are forced to be in nested subspaces of decreasing dimension. These subspaces are related by $\mathcal{G}_j = (\boldsymbol{I} - \omega_j \boldsymbol{A})(\mathcal{S} \cap \mathcal{G}_{j-1})$,

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

where $\mathcal{S}$ is a fixed proper subspace of $\mathbb{R}^N$, and the ω_j's are nonzero scalars. It can be shown [6] that in general the reduction in dimension of $\mathcal{G}_j$ is the co-dimension s of $\mathcal{S}$. Hence, after at most N/s dimension reductions $\mathcal{G}_m = \{\mathbf{0}\}$, for some $m \leq N/s$.
The remainder of this paper is organized as follows. In the following section, we describe the IDR(s) algorithm. Section 3 gives a guideline of how to choose the parameter s, and compares the memory requirements and vector operations of IDR(s) with the methods we will use in the performance study: GMRES [3], Bi-CGSTAB, [7], and BiCGstab(ℓ) [4], with $\ell = 2$. Section 4 presents the actual comparison. This study is done using the MATLAB code IFISS [1], a finite element program for fluid dynamics computations. Section 5 summarizes our findings.

2 IDR(s)

The residual $\boldsymbol{r}_{n+1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{n+1}$ corresponding to the iterate $\boldsymbol{x}_{n+1}$ after $n+1$ iterations is in $\mathcal{G}_{j+1}$ if

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

We may assume, without loss of generality, that the space $\mathcal{S}$ is the left null space of some $N \times s$ matrix $\boldsymbol{P}$, i.e.:

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

The vector $\boldsymbol{v}_n$ is a linear combination of the residuals in $\mathcal{G}_j$. In order to be able to update the solution with the residuals, we express $\boldsymbol{v}$ as linear combination with residual difference vectors $\Delta\boldsymbol{r}_i = \boldsymbol{r}_{i+1} - \boldsymbol{r}_i$:

$$\boldsymbol{v}_n = \boldsymbol{r}_n - \sum_{i=1}^{s} \gamma_i \Delta\boldsymbol{r}_{n-i} \tag{1}$$

The residuals are then computed by

$$\boldsymbol{r}_{n+1} = \boldsymbol{v}_n - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}_n = \boldsymbol{r}_n - \sum_{i=1}^{s} \gamma_i \Delta\boldsymbol{r}_{n-i} - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}_n .$$

Since $\boldsymbol{A}\Delta\boldsymbol{x}_n = -\Delta\boldsymbol{r}_n$, with $\Delta\boldsymbol{x}_n = \boldsymbol{x}_{n+1} - \boldsymbol{x}_n$, we get the following recursion for the iterates:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \sum_{i=1}^{s} \gamma_i \Delta\boldsymbol{x}_{n-i} + \omega_{j+1}\boldsymbol{v}_n .$$

Having defined the recursion formulae for $\boldsymbol{x}_n$ and $\boldsymbol{r}_n$, we now explain how to compute the coefficients γ_i. Since $\boldsymbol{v}$ is in $\mathcal{S} = \mathcal{N}(\boldsymbol{P}^T)$, we also have

$$\boldsymbol{P}^T\boldsymbol{v} = \boldsymbol{0} \, . \tag{2}$$

Combining (1) and (2) yields an $s \times s$ linear system which can be solved for the γ_j's, so that we can determine $\boldsymbol{v}$ and $\boldsymbol{r}_{n+1}$.
Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, repeating these calculations will produce new residuals $\boldsymbol{r}_{n+2}, \boldsymbol{r}_{n+3}, \cdots$, in $\mathcal{G}_{j+1}$. Once $s+1$ residuals in $\mathcal{G}_{j+1}$ have been computed, we can expect the next residual to be in $\mathcal{G}_{j+2}$. This process is repeated until $\boldsymbol{r}_n \in \mathcal{G}_m = \{0\}$ for some n, m. This implies that the exact solution can be computed in exact arithmetic using $\frac{N}{s}(s+1)$ matrix-vector products.
We may choose ω_{j+1} freely in the calculation of the first residual in $\mathcal{G}_{j+1}$. However, the same value must be used in the calculation of the subsequent residuals in $\mathcal{G}_{j+1}$. A suitable

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

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

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

Figure 1: The IDR(s) Algorithm.

choice for ω_{j+1} is the value that minimizes the norm of $\boldsymbol{r}_{n+1}$. These ideas lead to the IDR(s) algorithm. We present the algorithm in Figure 1.
It can be shown that the most basic variant of our algorithm, IDR(1), is mathematically equivalent to Bi-CGSTAB. Higher-order IDR(s) methods are related to the Bi-CGSTAB generalization ML(k)BiCGSTAB of Yeung and Chan [9].

3 Practical issues

In this section we comment on a number of issues and choices that are of importance in a practical implementation.

Preconditioning

In the algorithm we did not explicitly include the preconditioning operation. Preconditioning can be included by performing a preconditioned matrix-vector product. In the experiments we use right preconditioning, which means that we actually solve the system

$$\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{y} = \boldsymbol{b} \quad \boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{y} \,,$$

in which $\boldsymbol{M}$ is the preconditioner. A preconditioned matrix-vector product $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{v}$ is performed in two steps. First, the preconditioning system $\boldsymbol{M}\boldsymbol{w} = \boldsymbol{v}$ is solved, and then the matrix-vector product $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{w}$ is performed.

Memory requirements and vector operations

The operation count for the main operations to perform a full cycle of $s+1$ IDR(s) iterations yields: $(s+1)$ matrix-vector products, s^2+s+2 inner products, and $2s^2+\frac{7}{2}s+\frac{5}{2}$ vector updates. We have counted scaling of a vector and a simple addition of two vectors as half an update each. Apart from storage for the system matrix and the preconditioner, space is needed for $5+3s$ vectors of length N.

Choice of s and of $\boldsymbol{P}$

One can expect the number of iterations to decrease if s is chosen larger. However, the memory requirements and overhead due to vector operations increase and it is clear that if s is chosen too large, the increase in overhead will be more important than the reduction in the number of iterations. In order to make an a priori choice for s, it is necessary to have some heuristic on how the number of iterations depends on s. We can base such an heuristic on our knowledge of how the termination at the *exact* solution depends on s: in exact arithmetic IDR(s) terminates at the exact solution in at most $N + N/s$ iterations (= matrix-vector multiplications). This means for $s = 2$ we need 25% less iterations to compute the exact solution than for $s = 1$. For $s = 4$ we need 37.5% less, and for large s we need about 50% less iterations. From this it is clear that for computing the *exact* solution, it is almost never useful to chose a large value for s. Of course, this reasoning is based on the exact termination behavior, in practice the algorithm will converge long before this theoretical limit. However, in many experiments we have seen a similar dependence on s for the convergence behaviour. Increasing s from 1 to a modest value like 4 or 8 gives for most problems a significant reduction, but not much can be gained by choosing s larger. Since for $s = 4$ the required number of vector operations and memory requirements are modest, we choose this value as our default in all the experiments.

An other important question is how to choose the matrix $\boldsymbol{P}$. For reasons of robustness we take an orthogonalized set of random vectors as columns of $\boldsymbol{P}$, see [6] for a further motivation of this choice.

Comparison of overhead with other Krylov methods

In the numerical experiments we will compare the performance of IDR(s) with $s = 4$, with the well known Krylov methods full GMRES [3], Bi-CGSTAB [7], and BiCGstab(ℓ) [4] with $\ell = 2$. The table below gives numbers of inner products (DOT) and vector updates (AXPY) per matrix-vector product for each of the methods. Also given is the required memory space for the vectors needed to carry out the iterative process. The number of

Method	DOT	AXPY	Memory Requirements
IDR(4)	$4\frac{2}{5}$	$9\frac{7}{10}$	17
Full 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

vector operations and the memory requirements for IDR(4) are slightly higher than for Bi-CGSTAB and BiCGstab(2). However, the overhead is modest and fixed. The overhead for GMRES, on the other hand, grows with the number of iterations. For this method, the vector operations will eventually dominate the process if many iterations need to be performed, and also the memory requirements may become too large.

4 Numerical experiments

4.1 IFISS

The numerical experiments 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 focus on the comparison of IDR(4) with BiCGstab(2), the default variant of 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`

4.2 A convection-diffusion problem

4.3 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^o 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 discretized with square bi-linear Q1 elements, using a mesh size of $h = 2^{-7}$, which yields a nonsymmetric linear system of 65025 equations.

4.4 A motivation example for the default $s = 4$.

Before we present the actual comparison of IDR(4) with the other methods, we will first show an example that clearly illustrates why $s = 4$ is a good default value. For this 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 2. 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.

4.5 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-Gallerkin (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), and for the unstabilized case to investigate the performance of the iterative methods for increasingly skew-symmetric systems.
Figure 3 shows for each of the four 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 number of steps. But, as was remarked before, the overhead in

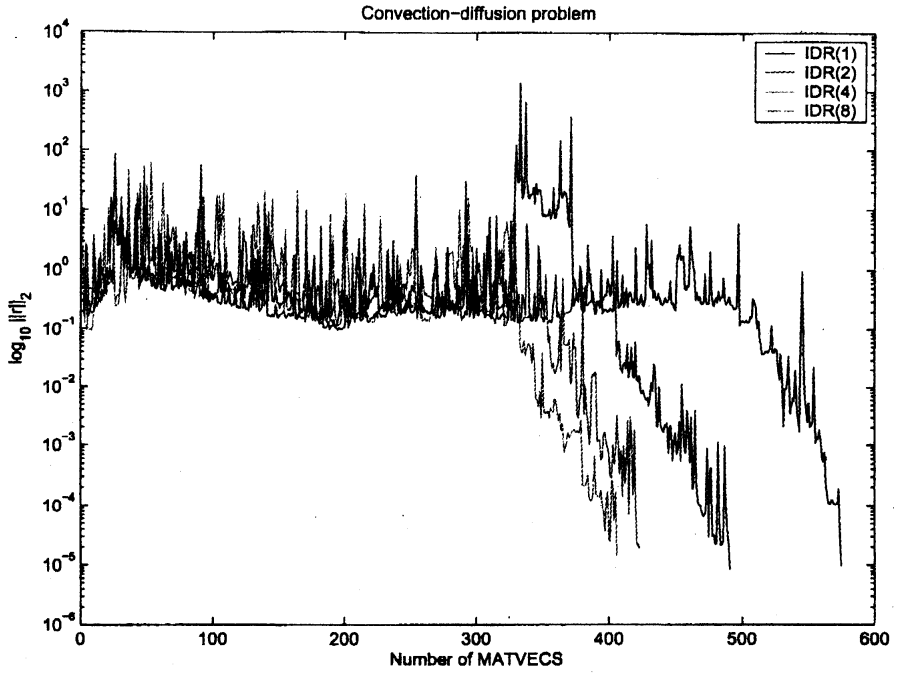

Figure 2: Convergence history of IDR(s), for different values of s

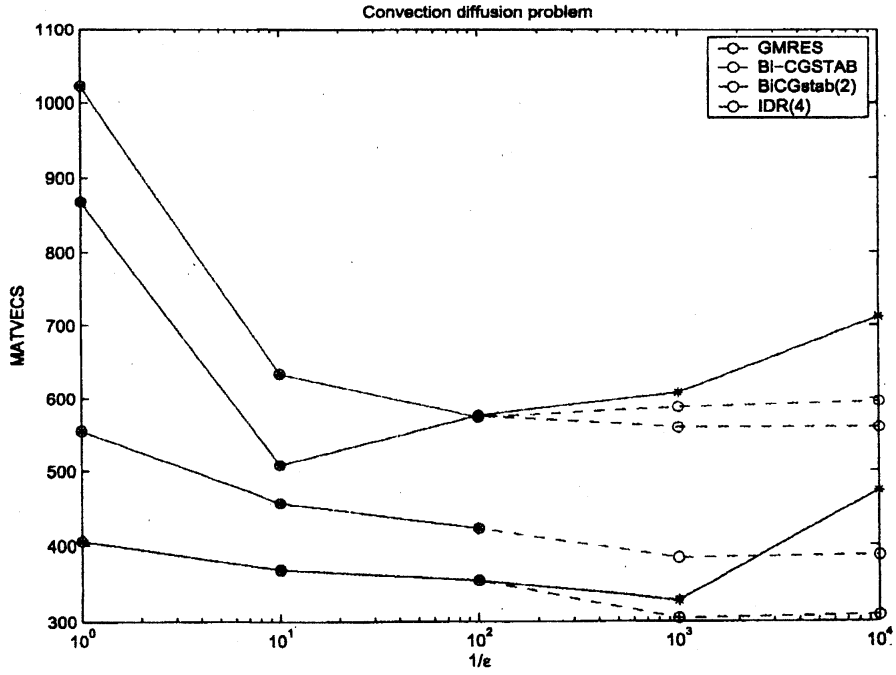

Figure 3: Number of matvecs for GMRES, IDR(4), Bi-CGSTAB and BiCGstab(2) for different diffusion parameters, solid without SUPG, dashed with SUPG.

vector operations and memory requirements is much larger for this method. IDR(4) and Bi-CGSTAB do not converge for the strongly nonsymmetric systems, i.e. for small values of ϵ without SUPG stabilization. In these cases 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[2]. BiCGstab(2), however, uses quadratic minimization polynomials that also work well in the strongly nonsymmetric case. As a result BiCGstab(2) converges in all the cases. For the physically realistic problems with SUPG stabilization, however, IDR(4) is 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

Diffusion ϵ	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)
1	218	47	40	30
0.1	186	30	26	27
0.01	176	27	26	23
0.001	154	n.c.	30	n.c.
0.0001	473	n.c.	33	n.c.
0.001 (SUPG)	138	28	27	23
0.0001 (SUPG)	137	29	26	23

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. 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 nonsymmetric problems, in which cases neither Bi-CGSTAB nor IDR(4) converges.

5 A Navier-Stokes problem

5.1 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 from -1 to 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 discretized 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

[2]This problem can be overcome by chosing $\boldsymbol{P}$ *complex* instead of real, see [6].

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 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 ideal pressure-convection diffusion preconditioner. Each application of this preconditioner requires solving three subsystems: one with $\boldsymbol{F}$ and two with the approximate Schur complement $\boldsymbol{M}_s$. These systems are solved 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.

5.2 Performance comparison for the convection-diffusion 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)
10	23 (2.1s)	34 (3.0s)	36 (3.1s)	28 (2.4s)
100	47 (4.2s)	106 (8.9s)	116 (9.7s)	72 (6.2s)
200	76 (6.9s)	242 (20.9s)	236 (20.3s)	120 (10.4s)

Table 3: Matrix-vector multiplications and computing times, $h = 2^{-3}$, 1747 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. There, approximations to the direct solves have to be used. This will result in a cheaper, but less effective preconditioner. In this situation a short-recurrence method like IDR(4) may be competitive again, or possibly even be required because of the limited memory consumption.

Reynolds number Re	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)
10	27 (14.8s)	36 (19.6s)	36 (19.6s)	28 (18.8s)
100	50 (26.7s)	146 (76.7s)	128 (67.5s)	87 (45.8s)
200	72 (38.4s)	365 (196s)	276 (150s)	140 (73.4s)

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

Reynolds number Re	GMRES	Bi-CGSTAB	BiCGstab(2)	IDR(4)
10	31 (112s)	44 (167s)	44 (163s)	37 (134s)
100	59 (204s)	214 (736s)	208 (715s)	115 (398s)
200	81 (292s)	398 (1399s)	420 (1429s)	209 (705s)

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

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

6 Concluding remarks

We have presented a performance comparison of IDR(s) with full GMRES, Bi-CGSTAB, and BiCGstab(ℓ) (with $\ell = 2$) for two classes of fluid dynamics problems. We have restricted our evaluation to the default value $s = 4$. Restricting the evaluation to one default choice for s mimics the way IDR(s) would be used in a practical implementation: a system would be solved once, with a pre-chosen default value.
Our main findings are:

- IDR(4) is for all our test problems faster than Bi-CGSTAB, both in numbers of MATVECS and in computing times. This difference in time is about a factor of two for the Navier-Stokes problems with a high Reynolds number.

- IDR(4) is faster than BiCGstab(2) for all test problems for which IDR(4) (and Bi-CGSTAB) converge. However, for almost skew-symmetric convection-diffusion problems IDR(4) and Bi-CGSTAB did not converge. This is due to the fact that both IDR(4) and Bi-CGSTAB use linear local minimization polynomials, which do not work well for matrices with complex eigenvalues with large imaginary parts. BiCGstab(2) uses quadratic minimization polynomials, which also work well for strongly nonsymmetric matrices. BiCGstab(2) is therefore more robust for such problems. We remark, however, that the strongly nonsymmetric matrices are based on an unstable discretization of the convection-diffusion problem, and the resulting solution is non-physical. If a stable discretization is used, IDR(4) converges faster than BiCGstab(2).

- The preconditioner for the Navier-Stokes problem that we applied is powerful, but also very costly. As a result, the number of GMRES iterations is limited and overhead for vector operations is negligible with respect to the cost of the matrix-vector multiplications. In such a situation, full GMRES is the method of choice as long as memory consumption is not an issue. If memory consumption becomes too high, one has to resort to another method. In that case IDR(4) seems to be the best option of the methods under consideration. For our examples, IDR(4) is significantly faster than Bi-CGSTAB and BiCGstab(2), in particular for higher Reynolds numbers.

We conclude that IDR(s) with $s = 4$ is a promising method for solving the type of incompressible flow problems that we have considered in this paper. The method is based on short recurrences and therefore more efficient than GMRES in memory consumption and computing time if many GMRES-iterations have to be performed. Furthermore, IDR(4) is competitive or faster than Bi-CGSTAB and BiCGstab(ℓ) for all physically relevant examples that we have considered.

Acknowledgments

The authors thank the developers of IFISS for making this code available. Part of this research has been funded by the Dutch BSIK/BRICKS project.

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

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

[6] P. Sonneveld and M.B. van Gijzen. IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems. Technical Report 07-07, Department of Applied Mathematical Analysis, Delft University of Technology, Delft, The Netherlands, 2007.

[7] 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.

[8] 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.

[9] M-C. Yeung and T.F. Chan. ML(k)BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors. *SIAM J. Sci. Comput.*, 21(4):1263–1290, 1999.