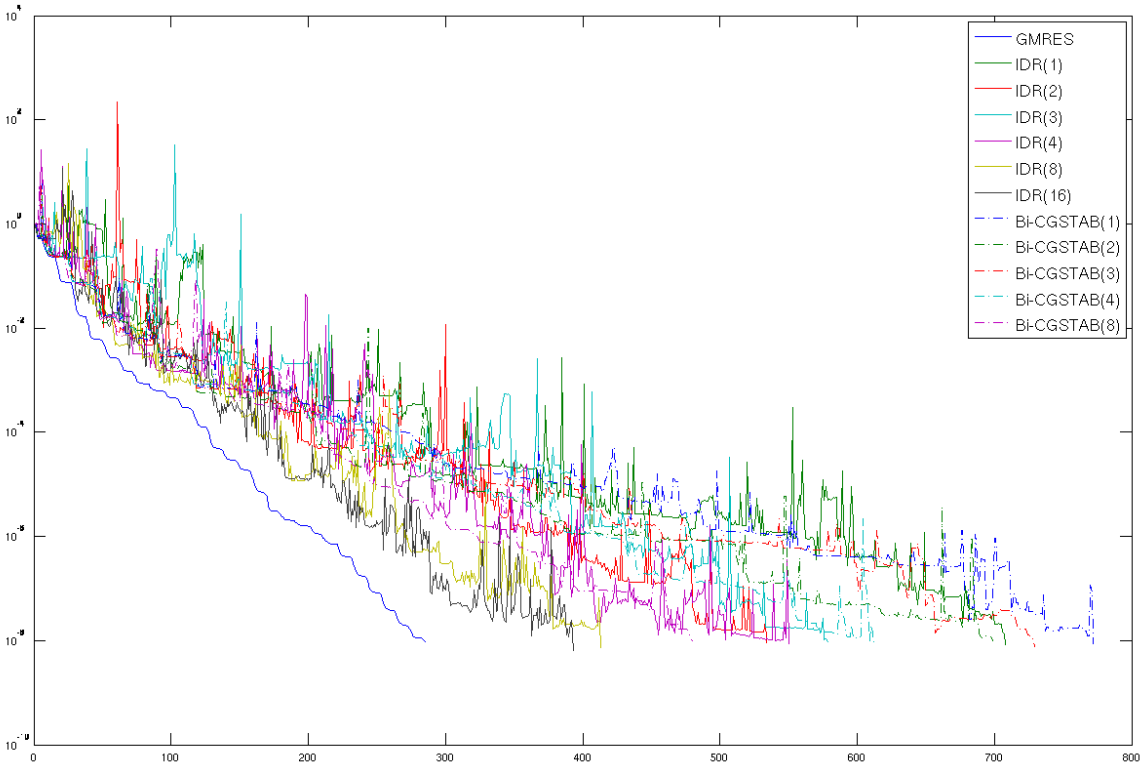
Induced Dimension Reduction
derivation and evaluation

Raymond van Venetië and Jan Westerdiep

February 1, 2015



Cover image. This chaotic graph illustrates the convergence of different (iterative) methods applied to some system $Ax = b$.

"The IDR(s) algorithm presented in this paper is quite promising and seems to outperform the state-of-the-art Bi-CG-type methods for important classes of problems." [PS08].

1. Introduction

In this paper, we are concerned with solving $Ax = b$ for a general (non-hermitian) matrix A. The main purpose is to derive and evaluate the so-called IDR method (*Induced Dimension Reduction*) against other wel established methods.

We start by introducing the concept of an iterave Krylov subspace solver. We assume the reader to have seen these methods before, so these first sections mostly serve to refresh the memory. For Hermitian matrices, the well known CG method is among the fastest iterative solvers. For non-Hermitian matrices, such a holy grail cannot exist. [VF84] The Bi-CG method and its Hybrid versions like Bi-CGstab and Bi-CGstab(ℓ) are algorithms for general matrices, derived from CG that share some of the powerful CG properties. For some time, most of the research has been in deriving CG-like algorithms for general matrices. We will discuss these in §3 and §4.

In 2008, Sonneveld and van Gijzen published an article about the IDR(s) method. [PS08] They generalize the old IDR method that was published in 1980 [WS80] and got more or less forgotten. This generalized method, according to the authors, leads to similar or even better results than Bi-CG derived algorithms. See the quote above, directly from their original paper.

The original IDR method actually was a predecessor to the Bi-CGstab method. To no surprise, one can prove that IDR(1) and Bi-CGstab are, under mild conditions, mathematically equivalent. We will derive a proof in §6.1.

To test the IDR(s) method, we conclude with our implementation and numerical results in §8 and §9. Here we will compare the perforance of IDR(s) for against other iterative methods: GMRES, Bi-CG and Bi-CGstab(ℓ) for $s, \ell \in \{1, 2, 4, 8\}$. GMRES produces the best residual in each Krylov iteration – albeit against a high computational cost – and is therefore used as a benchmark to show the relative convergence of other methods.

2. Krylov subspace methods

As noted in the introduction, we will look at *Iterative Methods* to solve $Ax = b$. These are methods that approximate the solution of $Ax = b$ iteratively by using an *initial guess* x_0 and generating a sequence of solutions x_k converging to x. They generally use two closely related quantities: the *error* of the approximate solution x_k is

$$e_k := x - x_k$$

and the *residual* is

$$r_k := Ae_k = A(x - x_k) = b - Ax_k.$$

In particular, we will look at *Krylov subspace* methods. These are methods that search for solutions in the *Krylov subspace*:

$$\mathcal{K}_{k+1}(A, r_0) := \text{span}\{r_0, Ar_0, \ldots, A^k r_0\} = \{q(A)r_0 \mid q \in \mathcal{P}_k\}.$$

Specifically, they find $x_k \in x_0 + \mathcal{K}_k(A, r_0)$, so $x_k = x_0 + y_k$ for some $y_k \in \mathcal{K}_k(A, r_0)$. We can write $y_k = q(A)r_0$ for some $q \in \mathcal{P}_{k-1}$, and

$$r_k = b - Ax_k = r_0 - Ay_k = r_0 - Aq(A)r_0 = (1 - Aq(A))r_0 =: p_k(A)r_0 \in \mathcal{K}_{k+1}(A, r_0). \tag{1}$$

Note that, per construction, $p_k(0) = 1$. On the other hand, for every polynomial $p_k \in \mathcal{P}_k$ with $p_k(0) = 1$ we find that

$$\begin{aligned} p_k(\lambda) = \gamma_k z^k + \ldots + 1 &\implies p_k(\lambda) = 1 - z(\gamma_k z^{k-1} + \ldots + \gamma_1) =: 1 - zq(\lambda) \\ &\implies p_k(A)r_0 = r_0 - Aq(A)r_0. \end{aligned}$$

In other words, r_k is a residual if and only if $r_k = p_k(A)r_0$ for some polynomial $p_k \in \mathcal{P}_k$ with $p_k(0) = 1$. Because of this characterization, we call

$$\mathcal{P}_k^0 := \{p \in \mathcal{P}_k \mid p(0) = 1\}$$

the set of *residual polynomials*. For Krylov methods the residual and approximate solution x_k are related by:

$$\begin{aligned} r_k &= r_0 - Aq(A)r_0 \\ x_k &= x_0 + q(A)r_0. \end{aligned}$$

Note that this convention – "$\mathcal{K}_k$ is a k-dimensional space"[1] instead of "$\mathcal{K}_k$ contains r_k" – is not shared among all sources.

Most Krylov-type methods work by using a recurrence relation for the residual updates, hoping to find residuals of decreasing norm. If such recurrence relations are given by

$$r_{k+1} = r_k + AU\vec{\gamma}$$

then one can find the corresponding approximate solution by

$$x_{k+1} = x_k - U\vec{\gamma}.$$

In this paper we derive some methods where we only state the recurrence relation for the residual. In such cases one can use the above relation to find the corresponding update formula for the approximate solution.

The initial guess x_0 initializes the Krylov methods. In practice the common choice is $x_0 = \vec{0}$. This can be justified by noting that if Krylov methods minimalize $\|b - Ax_k\|$ with $x_k = x_0 + q(A)r_0$ then this is equivalent to minimizing $\|(b - Ax_0) - Aq(a)r_0\|$, i.e. applying the Krylov solver on a different right hand side and the zero vector as initial guess.

One example of an iterative method that uses residual polynomials quite literally is LMR (*Local Minimal Residuals*). This method finds residuals of the form

$$r_{k+1} := p_{k+1}(A)r_0, \quad p_{k+1}(z) := (1 - \alpha_k A)p_k(z)$$

where α_k is chosen to minimize $\|r_{k+1}\|_2$. The residuals satisfy the recurrence relation $r_{k+1} = r_k - \alpha_k Ar_k$. Note that minimizing $\|r_{k+1}\|_2$ is equivalent to choosing α_k such that $r_{k+1} \perp Ar_k$. [Sle, Ex 3.16]

2.1. **GMRES.** In the previous section we saw that Krylov subspace methods search for solutions in the Krylov subspace. Some of these methods try to find residuals such that $\|r_k\|_2$ gets small in some sense. As $r_k \in \mathcal{K}_{k+1}(A, r_0)$ one may also wonder if we can find the best choice for r_k, i.e. the residual $r_k \in \mathcal{K}_{k+1}$ with the smallest 2-norm. We will now derive such a method.

[1]The case where the Krylov space does not expand is called a breakdown. In exact arithmetic, a breakdown of this type means that the exact solution x is in the space $x_0 + \mathcal{K}_k(A, r_0)$.

Suppose we have an orthonormal basis $V_k = \left[\ v_1 \mid \cdots \mid v_k\ \right]$ for $\mathcal{K}_k(A, r_0)$. We can compute v_{k+1} by:

$$\begin{aligned} \text{Expand:} \quad & w = Av_k, \\ \text{Orthogonalise:} \quad & \widetilde{v} = w - V_k \vec{h}'_k, h'_k := V_k^* w, \\ \text{Normalize:} \quad & v_{k+1} = \widetilde{v}/\|\widetilde{v}\|_2. \end{aligned}$$

If we define $\vec{h}_k := (\vec{h}'^{\top}_k, \|\widetilde{v}\|_2)^{\top}$, then can formulate the *Arnoldi relation*:

$$AV_k = V_{k+1}\underline{H}_k.$$

The matrix V_k is orthonormal with span(V_k)=$\mathcal{K}_k(A, r_0)$ and $\underline{H}_k$ is upper Hessenberg with columns $\vec{h}_j$ (appended with zeros to match the dimension). For details we refer to any introductionary text to Numerical Linear Algebra, e.g., [GHG13, LNT97, Sle].

One iterative method that uses this Arnoldi relation is GMRES (*Generalised Minimal Residual*). [SS86] Take $r_0 = b - Ax_0$ and $x_k = x_0 + V_k\vec{y}_k$ for some vector $\vec{y}_k \in \mathbb{C}^k$. In particular we have that $x_k - x_0 = V_k\vec{y}_k \in \mathcal{K}(A, r_0)$ and that [Sle, Lec. 6A]

$$\|r_k\|_2 = \|\|r_0\|_2 e_1 - \underline{H}_k\vec{y}_k\|_2.$$

One can prove that [Sle, Lec. 6]

$$\begin{aligned} \|r_k\|_2 = \min\{\|p_k(A)r_0\|_2 : p_k \in \mathcal{P}_k^0\} &\iff \|r_k\|_2 = \min\{\|r_0 - A\widetilde{x}\|_2 : \widetilde{x} \in \mathcal{K}_k(A, r_0)\} \\ &\iff r_k \perp \mathcal{K}_k(A, r_0) \\ &\iff \vec{y} = \arg\min\{\|\|r_0\|_2 e_1 - \underline{H}_k\vec{y}\|_2 : \vec{y} \in \mathbb{C}^k\}. \end{aligned}$$

In other words, solving the equation above is equivalent to finding the residual with minimal norm. This is exactly what GMRES does [Sle, Lec. 6]. This method is widely used as a reference point in numerical analysis, for it finds the optimal approximate solution in the given Krylov space. The method however relies on long recurrences as we have to orthoganalise against a space that grows every iteration, making practical uses limited.

2.2. **GCR.** In LMR, the updated residual r_{k+1} has smallest 2-norm with respect to the quantities of the previous iteration: $r_{k+1} = r_k - \alpha_k Ar_k$ with α_k that minimizes $\|r_{k+1}\|_2$. Instead of only looking at the last residual, we could generalise this by taking in consideration all the residuals produced so far. In this case you would calculate the next residual by

$$r_{k+1} = r_k - (\alpha_k Ar_k + \alpha_{k-1}Ar_{k-1} + \cdots + \alpha_0 Ar_0) \text{ with } \vec{\alpha} \text{ that minimizes } \|r_{k+1}\|_2.$$

This is what GCR does. [Sle, Lec. 5D] One can prove ([Sle, Lec. 6 Sl. 30]) that GCR and GMRES are mathematically equivalent, in that they produce the same residuals.

3. CG and Bi-CG

The *Conjugate Gradient method* (CG) is a beautiful iterative solver for $Ax = b$ with A a Hermitian positive definite matrix. It uses only a few inner products and one matrix-vector product per iteration. The algorithm is given in Algorithm 1.

It can be seen as the Hermitian positive definite variant of GCR. [Sle, Ex. 5.18] Mathematically, its approximate solutions x_k minimize the error in the A-norm: [LNT97, Thm. 38.2]

$$\text{CG finds } x_k \in x_0 + \mathcal{K}_k(A, r_0) \text{ such that } \|e_k\|_A \text{is minimised.}$$

Note that $\|x - x_k\|_A$ is minimal iff $x - x_k \perp_A \mathcal{K}_k(A, r_0)$ iff $r_k \perp \mathcal{K}_k(A, r_0)$. The residuals produced by CG form an orthogonal basis for the Krylov subspace: [LNT97, Thm. 38.1]

$$\mathcal{K}_k(A, r_0) = \operatorname{span}\{r_0, \ldots, r_{k-1}\}, \quad r_k^* r_j = 0 \quad \forall\, j < k. \tag{2}$$

One can introduce CG as an implementation of a basic iterative solver. In every step we have a search direcion u_k, and after choosing α_k carefully we update the residual by

$$r_{k+1} = r_k - \alpha_k c_k, \quad c_k := Au_k.$$

Remember that we the corresponding approximate solution is now given by $x_{k+1} = x_k + \alpha_k u_k$.

The search direction is given as $u_{k+1} = r_{k+1} - \beta_{k+1} u_k$. We determine α_k, β_{k+1} such that:

$$\begin{cases} r_{k+1} &= r_k - \alpha_k A u_k \perp r_k \\ u_{k+1} &= r_{k+1} - \beta_{k+1} u_k \text{ s.t. } Au_{k+1} \perp r_k. \end{cases} \tag{3}$$

Now for CG, (2) holds, expanding the above to

$$r_{k+1}, Au_{k+1} \perp r_k \implies r_{k+1}, Au_{k+1} \perp \operatorname{span}\{r_0, \ldots, r_k\} = \mathcal{K}_{k+1}(A, r_0).$$

In other words, we find an orthogonal basis of $\mathcal{K}_k$ at the cost of orthogonalizing against one vector.

3.1. **Bi-CG.** There are a few approaches for generalizing CG to non-Hermitian systems. One way[2] is to solve the normal equations

$$A^* A x = A^* b.$$

One disadvantage is that the condition number of this system is squared. Furthermore, the Krylov subspace generated by A^*A is hardly optimal: suppose A is Hermitian, then the Krylov subspace is generated by $A^*A = A^2$, so only even polynomials are considered.

Another method is considered in the Bi-CG (*Bi-orthogonal Conjugate Gradient*) method. [Sle, Ex. 8.3] It is an iterative solver for the equation $Ax = b$, where A does not need to be Hermitian. Instead of looking the normal equations, one can look at the following 'extended' system, for a given n-vector $\widetilde{b}$:

$$\begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix} \begin{bmatrix} \widetilde{x} \\ x \end{bmatrix} = \begin{bmatrix} b \\ \widetilde{b} \end{bmatrix}. \tag{4}$$

Note that the left hand side reduces to

$$\begin{bmatrix} Ax \\ A^* \widetilde{x} \end{bmatrix},$$

so a solution of this extended system provides us with a solution for the original system.

Furthermore, the matrix in (4) is Hermitian, so CG can be applied to this system. Without going into details, Bi-CG is just that: an implementation of CG applied to this extended system. [Sle, Ex. 8.3] The algorithm is given in Algorithm 2.

[2]This method is called CGLS, for CG applied to Least Squares. It solves the normal equations, thereby effectively solving the Least Squares problem. [Sle, Ex. 8.2]

Algorithm 1: Conjugate Gradient method

Select $\mathbf{x_0} \in \mathbb{C}^n$;
$\mathbf{x} = \mathbf{x_0},\ \mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$;
$\mathbf{u} = \mathbf{r},\ \rho = \|\mathbf{r}\|_2^2$;
while $\|\mathbf{r}\|_2 > tol$ **do**
  $\mathbf{c} = \mathbf{A}\mathbf{u}$;
  $\sigma = \mathbf{c}^*\mathbf{u},\ \alpha = \rho/\sigma$;
  $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{c}$;
  $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{u}$;
  $\rho' = \rho,\ \rho = \mathbf{r}^*\mathbf{r}$;
  $\beta = -\rho/\rho'$;
  $\mathbf{u} \leftarrow \mathbf{r} - \beta\mathbf{u}$;
end

Algorithm 2: Bi-orthogonal CG method

Select $\mathbf{x_0}, \widetilde{\mathbf{r}} \in \mathbb{C}^n$;
$\mathbf{x} = \mathbf{x_0},\ \mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$;
$\mathbf{u} = \mathbf{r},\ \rho = \|\mathbf{r}\|_2^2, \quad \widetilde{\mathbf{u}} = \widetilde{\mathbf{r}}$;
while $\|\mathbf{r}\|_2 > tol$ **do**
  $\mathbf{c} = \mathbf{A}\mathbf{u}, \quad \widetilde{\mathbf{c}} = \mathbf{A}^*\widetilde{\mathbf{u}}$;
  $\sigma = \widetilde{\mathbf{r}}^*\mathbf{c},\ \alpha = \rho/\sigma$;
  $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{c}, \quad \widetilde{\mathbf{r}} \leftarrow \widetilde{\mathbf{r}} - \overline{\alpha}\widetilde{\mathbf{c}}$;
  $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{u}$;
  $\rho' = \rho,\ \rho = \widetilde{\mathbf{r}}^*\mathbf{r}$;
  $\beta = -\rho/\rho'$;
  $\mathbf{u} \leftarrow \mathbf{r} - \beta\mathbf{u}, \quad \widetilde{\mathbf{u}} \leftarrow \widetilde{\mathbf{r}} - \overline{\beta}\widetilde{\mathbf{u}}$;
end

3.2. **Properties of Bi-CG.** Mathematically, Bi-CG finds

$$x_k \in x_0 + \mathcal{K}_k(A, r_0) \text{ such that } r_k \perp \mathcal{K}_k(A^*, \widetilde{r_0}).$$

Note that the residuals are orthogonal to a different set of residuals, the so-called *shadow* residuals. This *bi-orthogonality* (as opposed to CG's orthogonality) of residuals is reflected in the name – Bi-orthogonal CG or Bi-CG.

In case A is Hermitian and $\widetilde{r_0} = r_0$, we see that Bi-CG is equivalent to CG, as it finds x_k and r_k with the same properties. [Sle, Ex. 8.4]

Like in CG (cf. (3)), we find in Bi-CG that the update vector u_k and residual r_k in step k satisfy:

$$\left\{ \begin{array}{ll} r_{k+1} & = r_k - \alpha_k A u_k \perp \widetilde{r}_k \\ u_{k+1} & = r_{k+1} - \beta_{k+1} u_k \text{ s.t. } A u_{k+1} \perp \widetilde{r}_k. \end{array} \right. \tag{5}$$

Now let $\widetilde{r}_0, \ldots, \widetilde{r}_k$ be a Krylov basis of $\mathcal{K}_{k+1}(A^*, \widetilde{r}_0)$. One can prove [Sle, Ex. 8.9] that

$$r_{k+1}, A u_{k+1} \perp \widetilde{r}_k \implies r_{k+1}, A u_{k+1} \perp \mathcal{K}_{k+1}(A^*, \widetilde{r}_0). \tag{6}$$

In other words, if we choose α_k, β_{k+1} such that we are orthogonal to $\widetilde{r}_k$, then we are orthogonal to every shadow residual and equivalently $\mathcal{K}_{k+1}(A^*, \widetilde{r}_0)$.

The scalars α_k and β_{k+1} can be calculated in the following way: [Sle, Ex. 8.9]

$$\rho_k := \widetilde{r}_k^* r_k, \quad \sigma_k := \widetilde{r}_k^* A u_k, \quad \alpha_k := \frac{\rho_k}{\sigma_k}, \quad \beta_{k+1} := \frac{\rho_{k+1}}{\rho_k}.$$

3.3. **A basis for the shadow Krylov subspace.** In the last section we assumed to have a basis $\widetilde{r}_0, \ldots, \widetilde{r}_k$ of the shadow Krylov subspace $\mathcal{K}_{k+1}(A^*, \widetilde{r}_0)$. In fact, without knowing it, we have already constructed such a basis!

Bi-CG relies on the following argument. One can prove that the following update formulas produce a basis of the shadow space: [Sle, Ex. 8.10]

$$\left\{ \begin{array}{ll} \widetilde{r}_{k+1} & = \widetilde{r}_k - \overline{\alpha_k} A^* \widetilde{u}_k. \\ \widetilde{u}_{k+1} & = \widetilde{r}_{k+1} - \overline{\beta_{k+1}} \widetilde{u}_k \end{array} \right. \tag{7}$$

so to produce $\widetilde{r}_{k+1}$ and $\widetilde{u}_{k+1}$ we only need $\widetilde{u}_k$ and $\widetilde{r}_k$.

Remember that we could write $r_k = p_k(A) r_0$ for some residual polynomial of degree k. Using this notation, we see that above formulas reduce to $\widetilde{r}_k = \overline{p_k}(A^*)\widetilde{r}_0$ – they use the same residual polynomial as r_k, only conjugated. This way, we get a basis for $\mathcal{K}_k(A^*, \widetilde{r}_0)$ for 'free'.

3.4. **Other shadow Krylov space bases.** Of course, there are other possibilities for a basis of this shadow Krylov space. With another basis than the one in the previous section, one has

$$r_k = p_k(A)r_0, \quad \widetilde{r}_k = \overline{q}_k(A^*)\widetilde{r}_0 \tag{8}$$

with q_k some residual polynomial of degree exactly k (so that the regular and shadow residuals use different polynomials).

One such method – inspired by the LMR approach (see [Sle, Ex. 8.10]) – creates a shadow residual $\widetilde{r}_{k+1} = (I - \omega_{k+1}A^*)\widetilde{r}_k$ with ω_{k+1} such that $\widetilde{r}_{k+1}$ has minimal norm:

$$\widetilde{r}_{k+1} = (I - \omega_{k+1}A^*)\widetilde{r}_k = \widetilde{r}_k - \omega_{k+1}A^*\widetilde{r}_k, \quad \omega_{k+1} = \arg\min_{\omega} \|\widetilde{r}_{k+1}\|_2 = \frac{\langle \widetilde{r}_k, A^*\widetilde{r}_k\rangle}{\langle A^*\widetilde{r}_k, A^*\widetilde{r}_k\rangle}. \tag{9}$$

This last equality follows from, e.g., [Sle, Ex. 3.18].

3.5. **Transpose-free Bi-CG.** One of the drawbacks of Bi-CG is the fact that it requires us to compute with A^*. In real-life applications, many matrices are so-called *black boxes*, in the sense that there is a routine f for which $f(x) = Ax$. In other words, there is no explicit matrix A available. In such cases, computation with A^* might be hard or even impossible.

One can avoid the calculation of A^*. First, note that A^* is only used in the computation of $\widetilde{r}_k$, and with this, in the computation of ρ_k and σ_k only. Expand $\widetilde{r}_k = \overline{p}_k(A^*)\widetilde{r}_0$, and move the residual polynomial to the other side of the inner product:

$$\rho_k := \widetilde{r}_k^* r_k = (\overline{p}_k(A^*)\widetilde{r}_0^*)r_k = \widetilde{r}_0^* p_k(A)r_k, \quad \sigma_k := \widetilde{r}_k^* A u_k = \widetilde{r}_0^* A p_k(A)u_k. \tag{10}$$

In Bi-CG, one uses $\widetilde{r}_k, r_k$ and u_k to find ρ_k and σ_k. In transpose-free Bi-CG, one uses $r_k, p_k(A)r_k$, $p_k(A)Au_k$ and the above identities to find ρ_k and σ_k without using A^*.[3]

4. Hybrid Bi-CG methods

Note that as α_k and β_{k+1} in (7) are the *unique* scalars that orthogonalize their respective quantities, a change of the shadow basis does not change the residuals r_{k+1}^{Bi-CG} that Bi-CG produces. Moreover, going from regular Bi-CG to transpose-free Bi-CG does not change these residuals either. Combining these results (in mathematical notation, combining (8) and (10)) can yield interesting variations.

Given some residual polynomial q_k of degree exactly k as in (8), transpose-free Bi-CG computes inner products using the quantity $q_k(A)r_k^{Bi-CG}$. If we define the residual to be *this* quantity, as opposed to r_k^{Bi-CG} itself, then we call this a *Hybrid Bi-CG method*. In more mathematical terms, define

$$r_k := q_k(A)r_k^{Bi-CG}, \quad u_k := q_k(A)u_k^{Bi-CG}, \quad r'_k := q_k(A)r_{k+1}^{Bi-CG}, \quad u'_k := q_k(A)u_{k+1}^{Bi-CG}. \tag{11}$$

Hybrid Bi-CG methods use these four quantities to generate approximate solutions x_k, x'_k subject to

$$r_k = b - Ax_k, \quad r'_k = b - Ax'_k.$$

Note that these approximate solutions are different from the ones that Bi-CG generates, but because they do in fact use r_k^{Bi-CG} in their computations, we call them Hybrid Bi-CG methods.

[3] One can accomplish this by saving both Bi-CG residuals and so-called *CGS* (CG squared) residuals. See [TFC97] and [Sle, Ex 8.12].

4.1. **Bi-CGstab.** Bi-CGstab [VdV92] is a Hybrid Bi-CG method, where $q_k = q_k^{stab}$ is taken to be a *stabilisation* or *acceleration* polynomial.[4] This immediately explains the name of the method and the effect it is hoped to have.

Let $q_{k+1}^{stab}(\lambda) := (1 - \omega_{k+1}\lambda)q_k^{stab}(\lambda)$ be the uniquely defined residual polynomial of the LMR approach from (9). We will use the notation from (11).

Note that A and $q_k^{stab}(A)$ commute, so the scalars ρ_k and σ_k become

$$\rho_k := \widetilde{r}_k^* r_k^{Bi-CG} = \widetilde{r}_0^* q_k^{stab}(A) r_k^{Bi-CG} = \widetilde{r}_0^* r_k$$
$$\sigma_k := \widetilde{r}_k^* A u_k^{Bi-CG} = \widetilde{r}_0^* q_k^{stab}(A) A u_k^{Bi-CG} = \widetilde{r}_0^* A u_k.$$

Now, familiar recurrence relations hold for u'_k and r'_k (cf. (5)):

$$\begin{cases} r'_k &= r_k - \alpha_k A u_k \text{ s.t. } r'_k \perp \widetilde{r}_0 \\ u'_k &= r'_k - \beta_{k+1} u_k \text{ s.t. } A u'_k \perp \widetilde{r}_0 \end{cases}$$

and for r_{k+1} and u_{k+1} (cf. (9)):

$$\begin{cases} \omega_{k+1} &:= \arg\min_w \|r'_k - \omega A r'_k\|_2 \\ r_{k+1} &= (I - \omega_{k+1} A) r'_k \\ u_{k+1} &= (I - \omega_{k+1} A) u'_k \end{cases}.$$

These recurrence relations yield the following Bi-CGstab Algorithm 3. [GLS10, Alg. 2] A slightly more mathematical version with different notation, Algorithm 4, was also found in [GLS10, Alg. 1] and will come in handy in §6.

Algorithm 3: BiCGstab method

Select $\mathbf{x_0}, \widetilde{\mathbf{r}} \in \mathbb{C}^n$;
$\mathbf{x} = \mathbf{x_0}$, $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$;
$\mathbf{u} = \mathbf{r}$;
while $\|\mathbf{r}\|_2 > tol$ **do**
  $\mathbf{c} = \mathbf{A}\mathbf{u}$;
  $\sigma = \widetilde{\mathbf{r}}^*\mathbf{c}$;
  $\alpha = \sigma^{-1}\widetilde{\mathbf{r}}^*\mathbf{r}$;
  $\mathbf{r}' = \mathbf{r} - \alpha\mathbf{c}$; $\mathbf{x}' = \mathbf{x} + \alpha\mathbf{u}$;
  $\mathbf{s} = \mathbf{A}\mathbf{r}'$;
  $\omega = \mathbf{s}^*\mathbf{r}'/\mathbf{s}^*\mathbf{s}$;
  $\beta = \sigma^{-1}\widetilde{\mathbf{r}}^*\mathbf{s}$;
  $\mathbf{u}' = \mathbf{r}' - \beta\mathbf{u}$; $\mathbf{u} = \mathbf{u}' - \omega(\mathbf{s} + \beta\mathbf{c})$;
  $\mathbf{r} = \mathbf{r}' - \omega\mathbf{s}$; $\mathbf{x} = \mathbf{x}' + \omega\mathbf{r}'$;
end

Algorithm 4: Bi-CGstab (revised)

Select $\mathbf{x_0}, \widetilde{\mathbf{r}} \in \mathbb{C}^n$;
$\mathbf{x} = \mathbf{x_0}$, $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$;
$\mathbf{u} = \mathbf{r}$;
while $\|\mathbf{r}\|_2 > tol$ **do**
  $\mathbf{s} = \mathbf{A}\mathbf{u}$;
  α such that
    $\mathbf{v} = \mathbf{r} - \mathbf{s}\alpha \perp \widetilde{\mathbf{r}}_\mathbf{0}$;
  Select ω
  $\mathbf{r} = (\mathbf{I} - \omega\mathbf{A})\mathbf{v}$;

  β such that
    $\mathbf{A}\mathbf{u}' = \mathbf{A}(\mathbf{v} - \beta\mathbf{u}) \perp \widetilde{\mathbf{r}}_0$;
  $\mathbf{u}' = \mathbf{v} - \beta\mathbf{u}$;
  $\mathbf{u} = (\mathbf{I} - \omega\mathbf{A})\mathbf{u}'$;
end

4.2. **Bi-CGstab(ℓ).** Bi-CGstab uses the first degree LMR polynomial as q_k to (hopefully) stabilize the Bi-CG method. Of course, this can be generalized. In Bi-CGstab(ℓ), the polynomial q_k is chosen as the product of Minimal Residuals of degree ℓ. This means that we stabilize the residual every ℓ steps, by a polynomial of degree ℓ: for $k = m\ell + \ell$ we have $q_k = q_{m\ell+\ell} = o_m q_{m\ell}$ with o_m an ℓ-degree Minimal Residual polynomial. This can be implemented by looping ℓ times over the 'Bi-CG' part and then applying the stabilizing polynomial o_m. Details can be found in the original

[4]The theoretical foundations for the hybrid Bi-CG methods seem to be sparse. We rely on empirical evidence that this method accelerates the convergence.

artice. [SF93] For $\ell = 1$, Bi-CGstab(ℓ) is exactly the Bi-CGstab method. Note that Bi-CGstab(ℓ) requires more memory and has additional computation cost per MV if ℓ grows.[5] For some $\ell \geq 2$ it appears that Bi-CGstab(ℓ) produces better convergence behaviour than Bi-CGstab. [SF93, §5]

5. IDR Theory

We will now explain the IDR (*Induced Dimension Reduction*) method. Suppose we have a sequence of nested subspaces such that

$$\{0\} = G_K \subset \cdots \subset G_2 \subset G_0 = \mathcal{K}_n(A, r_0)$$

and a function that transfers our residual $r_k \in G_j$ to $r_{k+1} \in G_{j+1}$ using linear transformations, then we will eventually find a residual of $r_k = 0$. As the residual is updated in a linear way, we can apply these transformations to the initial solution x_0 to find approximate solutions x_k corresponding to r_k. This naturally leads to an iterative method that finds the exact solution x in fintely many steps.

The IDR Theorem gives us exactly such a sequence of nested subspaces.

5.1. The IDR Theorem.

Theorem 1 (IDR Theorem)**.** *Let A be a matrix in $\mathbb{C}^{n\times n}$, $\widetilde{R}$ a full rank matrix in $\mathbb{C}^{n\times s}$ and (ω_j) be a sequence of non-zero scalars. Let r_0 be a nonzero vector in $\mathbb{C}^n$ and let $\mathcal{G}_0 = \mathcal{K}_n(A, r_0)$ be the full Krylov space. For the sequence $(\mathcal{G}_j)$ of subspaces defined by*

$$\mathcal{G}'_j := \mathcal{G}_j \cap \widetilde{R}^{\perp}, \quad \mathcal{G}_{j+1} := (I - \omega_{j+1}A)\mathcal{G}'_j,$$

we have that

(1) $\mathcal{G}_{j+1} \subset \mathcal{G}_k$;

(2) If $\widetilde{R}^{\perp}$ does not contain an Eigenvector of A, then

$$\mathcal{G}_{j+1} = \mathcal{G}_j \iff \mathcal{G}_j = \{0\}.$$

This IDR theorem (cf. [PS08, §2]), of which the proof is beneath here, shows that we can create a sequence of subspaces $(\mathcal{G}_k)$ all contained in the previous. Under some mild condition – $R^{\perp}$ not containing an Eigenvector of A – it holds that this nesting is strict, meaning that there is a finite n such that $\mathcal{G}_n = \{0\}$. The probability of this happening is zero when we take $\widetilde{R}$ to be a random matrix.

If we manage to create some iterative solver which finds residuals $r_k \in \mathcal{G}_k$, the IDR theorem tells us that we will find an exact solution in finitely many steps.

Proof. (1) We proceed by induction. As $\mathcal{G}_0$ is a full Krylov subspace, it is an A-invariant subspace, so we find that

$$(I - \omega_0 A)\mathcal{G}_0 \subset \mathcal{G}_0.$$

Now as $\mathcal{G}_0 \cap \widetilde{R}^{\perp} \subset \mathcal{G}_0$ we see that

$$\mathcal{G}_1 = (I - \omega_0 A)(\mathcal{G}_0 \cap \widetilde{R}^{\perp}) \subset (I - \omega_0 A)\mathcal{G}_0 \subset \mathcal{G}_0$$

completing the basis step.

Assume $\mathcal{G}_k \subset \mathcal{G}_{k-1}$ holds for some $k > 0$. Then $\mathcal{G}'_k \subset \mathcal{G}'_{k-1}$ and for $x \in \mathcal{G}_{k+1}$ we find per definition:

$$x = (I - \omega_k A)y \text{ with } y \in \mathcal{G}'_k \subset \mathcal{G}'_{k-1} \implies (I - \omega_{k-1}A)y \in \mathcal{G}_k.$$

[5] The number of AXPY and DOT operations grows if ℓ becomes larger, see [SF93, Tbl. 3.1].

As both $y \in \mathcal{G}_k$ and $(I - \omega_{k-1}A)y \in \mathcal{G}_k$, we have that $Ay \in \mathcal{G}_k$, implying that also $x = (I - \omega_k A)y \in \mathcal{G}_k$. From here, it follows that $\mathcal{G}_{k+1} \subset \mathcal{G}_k$.

(2) the implication from right to left is trivial, so we will only look at the left to right part of the bi-implication.

Given that $\mathcal{G}_{k+1} = \mathcal{G}_k$, suppose that $\mathcal{G}_k \neq \{0\}$. The proof will consist of two parts: first, we will show that $\mathcal{G}_k = \mathcal{G}'_k = \mathcal{G}_{k+1}$. Next, we will prove that under the assumption, $\mathcal{G}_k$ contains an Eigenvector.

We know that $\mathcal{G}'_k = \mathcal{G}_k \cap \widetilde{R}^{\perp}$, so that $\dim \mathcal{G}'_k \leq \dim \mathcal{G}_k$.

Let $r = \dim \mathcal{G}'_k$. Then there is a $n \times r$ matrix B whose columns form a basis of $\mathcal{G}'_k$ so that rank $B = r$. As $\mathcal{G}_{k+1} = C\mathcal{G}'_k$ for $C = I - \omega_k A$, we see that rank $CB = \dim \mathcal{G}_{k+1}$. Using that rank $CB \leq$ rank B, it must hold that $\dim \mathcal{G}_{k+1} \leq \dim \mathcal{G}'_k$.

We find

$$\dim \mathcal{G}_{k+1} \leq \dim \mathcal{G}'_k \leq \dim \mathcal{G}_k \implies \dim \mathcal{G}'_k = \dim \mathcal{G}_k \implies \mathcal{G}'_k = \mathcal{G}_k.$$

Using this, we see that $\mathcal{G}_{k+1} = (I - \omega_k A)\mathcal{G}_k = \mathcal{G}_k$, meaning that $\mathcal{G}_k$ is $(I - \omega_k A)$-invariant. Now, recall the basis matrix B. Every column of the product $(I - \omega_k A)B$ is a vector in $\mathcal{G}_k$ and thus can be written in terms of its basis, to find an $r \times r$ matrix U with $(I - \omega_k A)B = BU$. Let x be an Eigenvector of U with $Ux = \lambda x$ and $x \neq 0$. Then

$$\lambda x = Ux \implies B\lambda x = \lambda(Bx) = BUx = (I - \omega_k A)Bx$$

so that (Bx, λ) is an Eigenpair of $I - \omega_k A$, so that $(Bx, (1 - \lambda)/\omega_k)$ is an Eigenpair of A with Bx in $\mathcal{G}_k$.

When $\widetilde{R}^{\perp} \supset \mathcal{G}'_k$ does not contain an Eigenvector of A, we thus must have $\mathcal{G}_k = \{0\}$, completing the proof.

□

The IDR theorem can be given in a more general form (see [Sle, Thm 11.1]), but the above formulation will suit our purposes. Furthermore, we can extend the IDR theorem to include the rate at which the dimensions of G_j shrink. For the proof we refer to the literature, e.g., [PS08, Thm 3.1].

Theorem 2. *Use the same notation and conditions as in Theorem 1. Let dim$(\mathcal{G}_j) = d_j$; then the sequence d_j is monotically nonincreasing and satisfies*

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

As noted in the original IDR(s) article [PS08] this implies that (in exact arithmetic) the dimension reduction per step is between 0 and s. If the dimension reduction is always equal to s, we find that IDR(s) needs at most $(s+1)\frac{n}{s} = n + \frac{n}{s}$ matrix-vector products to find the exact solution. In practice the dimension reduction is 'almost always' equal to s, since we have finite precision.

In the next sections we will use the IDR theorem to derive a Krylov-type iterative solver: IDR(s).

5.2. **Finding residuals: IDR.** Recall that Krylov solvers find approximate solutions x_k for which the residuals $r_k = b - Ax_k$ satisfy $r_k = q_k(A)r_0$ (see (1)). Here, q_k is a residual polynomial of exact degree k. From this, it follows that $r_k \in \mathcal{K}_{k+1}(A, r_0) \setminus \mathcal{K}_k(A, r_0)$.

One can prove that general Krylov-type solvers satisfy the following recurrence relations:[6]

$$\text{(12)}\qquad \begin{cases} r_{k+1} &= r_k - \alpha A v_k - \sum_{i=1}^{\ell} \gamma_i \Delta r_{k-i} \\ x_{k+1} &= x_k - \alpha v_k - \sum_{i=1}^{\ell} \gamma_i \Delta x_{k-i} \end{cases}, \quad v_k \in \mathcal{K}_{k+1}(A, r_0) \setminus \mathcal{K}_k(A, r_0)$$

with $\Delta u_k := u_{k+1} - u_k$ the forward difference operator. Here ℓ is the depth of recursion. If $\ell = k$ then we have deep recurions like GMRES. We are interested in methods that require small recursion, with ℓ fixed and small, like CG.

As stated before, we want to find residuals in the shrinking spaces given by the IDR theorem. These spaces ultimately shrink to $\{0\}$, so we will find the exact solution in a finite number of steps. Let a residual r_k be given in $\mathcal{G}_j$. How can we find a residual r_{k+1}, and when is it in the next space $\mathcal{G}_{j+1}$?

The IDR Theorem states that $r_{k+1} \in \mathcal{G}_{j+1}$ if we can write

$$r_{k+1} = (I - \omega_{j+1} A) v_k \text{ with } v_k \in \mathcal{G}_j \cap \widetilde{R}^{\perp}.$$

On the other hand, we want the residual r_{k+1} to be of the general Krylov form given in (12), as this allows for easy updating of the approximate solution. A quick inspection reveals that for this to happen, the v_k must be of the form

$$\text{(13)}\qquad v_k = r_k - \sum_{i=1}^{\ell} \gamma_i \Delta r_{k-i}.$$

Subsituting this into our previous relation of r_{k+1} we find that

$$r_{k+1} = (I - w_{j+1} A) v_k = r_k - \omega_{j+1} A v_k - \sum_{i=1}^{\ell} \gamma_i \Delta r_{k-i}.$$

The equations above describe a way to transfer residuals from $r_k \in \mathcal{G}_j$ to $r_{k+1} \in \mathcal{G}_{j+1}$. We first construct a residual vector $v_k \in \mathcal{G}_j \cap \widetilde{R}^{\perp}$, which we then lift to $\mathcal{G}_{j+1}$ by applying the operator $I - \omega_{j+1} A$.

The question arises how and when we can find v_k such that $v_k \in \mathcal{G}_j \cap \widetilde{R}^{\perp}$. To answer this question, we introduce the residual (difference) matrix

$$S_k = \begin{bmatrix} \Delta r_{k-1} & \cdots & \Delta r_{k-s} \end{bmatrix},$$

reducing the relation of r_{k+1} to

$$r_{k+1} = r_k - \omega_{j+1} A v_k - S_k \vec{\gamma}.$$

The requirement that $v_k \in \widetilde{R}^{\perp} \cap \mathcal{G}_j$ gives us two conditions:

(1) $v_k \in \widetilde{R}^{\perp}$, which reduces to the constraint $\widetilde{R}^* v_k = 0$. This is an $s \times \ell$ linear system for the coefficients γ_i, which is uniquely solvable iff $\ell = s$. Now per definition of v_k in (13), solving $\widetilde{R}^* v_k = 0$ is equal to solving the equation $\widetilde{R}^* S_k \vec{\gamma} = \widetilde{R}^* r_k$ for $\vec{\gamma}$. After solving this equation for $\vec{\gamma}$ we find v_k as $v_k = r_k - S_k \vec{\gamma}$.

[6]The source used here is [PS08, p. 1038]; note that their definition of $\mathcal{K}_k$ impies that $r_k \in \mathcal{K}_k$, while ours asserts $\dim \mathcal{K}_k \leq k$. Compare with caution.

(2) $v_k = r_k - S_k\vec{\gamma} \in \mathcal{G}_j$, which happens if $\text{span}(S_k) \subset \mathcal{G}_j$ and $r_k \in \mathcal{G}_j$, or equivalently, if $r_{k-i} \in \mathcal{G}_j$ for $i = 0, \ldots, s$. With 'normal' dimension reduction (cf. Thm. 2) we expect this to happen only for $k \geq (j+1)(s+1)$.

This gives rise to the following Lemma.

Lemma 1. *Use the notation introduced above. If $r_{k-i} \in \mathcal{G}_j$ for $i = 0, \ldots, s$, then we can find $r_{k+1} \in \mathcal{G}_{j+1}$ in three steps:*

- *Calculate the $\vec{\gamma}$ that solves $\widetilde{R}^* S\vec{\gamma} = \widetilde{R}^* r_k$;*
- *Create $v_k = r_k - S\vec{\gamma} \in \mathcal{G}_j \cap \widetilde{R}^\perp$;*
- *Lift v_k to find a residual in $\mathcal{G}_{j+1}$: $r_{k+1} = (I - \omega_{j+1}A)v_k$.*

This method leads to the IDR(s) algorithm. Remember that $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, so the above Lemma still holds if some of the residuals r_{k-i} are contained in $\mathcal{G}_{j+1}$.

Let $s+1$ residuals r_{k-i} in $\mathcal{G}_j$ be given. We can lift these to $s+1$ residuals in $\mathcal{G}_{j+1}$ by applying the Lemma to each r_{k-i} (and of course updating the matrix S_k accordingly). This way, we find a dimension reduction every $s+1$ steps.[7] Every time a dimension is reduced, (or equivalently, we find the first residual r_k in $\mathcal{G}_{j+1}$) we can choose a new ω_{j+1}. Later residuals $r_{k+1}, \ldots, r_{k+s}$ in this space must use the same ω_{j+1}. One must be careful when $\omega_{j+1} \approx 0$ or $\widetilde{R}^* S$ is (close to) singular, as this causes breakdowns or stagnation. See §5.4 and §5.5 for a little more detail on these subjects.

5.3. **Algorithm derivation.** We formalise the previous idea so that we can derive an explicit IDR(s) algorithm. A relation between each residual and its approximate solution is given by $\Delta r_k = -A\Delta x_k$.[8] Define the matrix U_k:

$$U_k = \begin{bmatrix} -\Delta x_{k-1} & \cdots & -\Delta x_{k-s} \end{bmatrix} \implies S_k = AU_k.$$

The algorithm for calculating r_{k+1} from previous residuals can now be introduced.

Lemma 2. *Suppose we have $S_k = AU_k$, $r_k = b - Ax_k$ and $\omega \in \mathbb{C}$. Let $\vec{\gamma}$ be such that*

$$v := r_k - S_k\vec{\gamma} \perp \widetilde{R}.$$

Then with

$$r_{k+1} := (I - \omega A)v, \quad s_k := \Delta r_k, \quad u_k := -U_k\vec{\gamma} - \omega v, \quad x_{k+1} := x_k - u_k$$

we have that

$$s_k = Au_k \text{ and } r_{k+1} = b - Ax_{k+1}.$$

Proof. This follows easily. We have that

$$Au_k = -AU_k\vec{\gamma} - \omega Av = -S_k\vec{\gamma} - \omega Av = v - r_k - \omega Av = r_{k+1} - r_k = \Delta r_k = s_k$$

and

$$r_{k+1} = r_k + s_k = b - Ax_k + Au_k = b - A(x_k - u_k) = b - Ax_{k+1}. \qquad \square$$

As noted in [GLS10] the above lemma gives us multiple ways of computing s_k and r_{k+1}, all with their own stability properties.

---

[7] For the actual implementation of the algorithm, we also need to update the approximate solution x_k, therefore needing the update vectors u_k.

[8] $\Delta r_k = r_{k+1} - r_k = A(x - x_{k+1} - x + x_k) = A(-\Delta x_k)$.

Corollary 1. *Using the definitions given in Lemma 2, we find that*

$$s_k = Au_k = -S_k\vec{\gamma} - \omega Av_k = \Delta r_k$$

and

$$r_{k+1} = v - \omega Av = \Delta r_k + r_k = b - Ax_{k+1}.$$

Lemma 1 tells us under what conditions r_{k+1} is in the 'next' subspace $\mathcal{G}_{j+1}$. To initalize the algorithm, we need matrices U_s and $S_s = AU_s$ with the given properties. We will discuss different approaches in the implementation details of §8. Now we will discuss how to choose ω.

5.4. **Choice of ω.** By the IDR Theorem, we are free to choose ω_{j+1} when lifting the first residual to $\mathcal{G}_{j+1}$. The other s residuals must be lifted using this same ω_{j+1}. As suggested in [PS08, §4.2] we select w_{j+1} such that the norm of r_{k+1} is minimized, using the LMR appoach (cf. (9)). There are also other possibilities: see, e.g., [vGS11].[9]

With our LMR strategy, we get for $s = 1$ an algorithm that is mathematically equivalent to Bi-CGstab: see §6.

5.5. **Orthogonalising U and S.** Numerical instabilities can occur if the matrix $\widetilde{R}^*S$ is badly conditioned. One can help the conditioning in two ways: first, we can orthogonalise $\widetilde{R}$, which is cheap because this matrix is constant. The second is more involving: one can orthogonalise the matrices S and U after every new residual. This is implemented in [vG]; we chose not to investigate this method in more detail. See [vGS11] for a lot more details.

5.6. **Algorithm.** Based on Lemma 2, we are now ready to give the IDR(s) algorithm. The straightforward and most intuitive implementation is given in Algorithm 5. [GLS10, Alg. 1] As noted in Corollary 1, we have a lot of freedom in the way we calculate the vectors s_k and r_{k+1}. Another (equivalent) method is presented in Algorithm 6. [PS08, Fig. 3.1] It is mentioned in [PS08, §4.3] that this choice of freedom is numerically more stable.

[9]This method, called the "maintaining the convergence" strategy resolves stagnation in systems where ω is so small that the other iteration parameters cannot be computed with sufficient accuracy.

Algorithm 5: IDR(s)

Select $\mathbf{x}$;
Select matrices $\widetilde{\mathbf{R}}, \mathbf{S}, \mathbf{U}$ such that $\mathbf{S} = \mathbf{AU}$;
$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$;
$i = 1$; $j = 0$;
while $\|r\|_2 >$ `tol` **do**
  Solve $\vec{\gamma}$ from $\widetilde{\mathbf{R}}^*\mathbf{S}\vec{\gamma} = \widetilde{\mathbf{R}}^*\mathbf{r}$;
  $\mathbf{v} = \mathbf{r} - \mathbf{S}\vec{\gamma}$;
  $\mathbf{c} = \mathbf{Av}$;
  if $j = 0$ **then**
    $\omega = \mathbf{c}^*\mathbf{v}/\mathbf{c}^*\mathbf{c}$
  $\mathbf{U}e_i = -\mathbf{U}\vec{\gamma} + \omega\mathbf{v}$;
  $\mathbf{x} = \mathbf{x} + \mathbf{U}e_i$;
  $\mathbf{r}_{\text{old}} = \mathbf{r}$;
  $\mathbf{r} = \mathbf{v} - \omega\mathbf{c}$;
  $\mathbf{S}e_i = \mathbf{r} - \mathbf{r}_{\text{old}}$;
  $i = i + 1$;
  if $i > s$ **then**
    $i = 1$;
  $j = j + 1$;
  if $j > s$ **then**
    $j = 0$;
end

Algorithm 6: IDR(s)

Select $\mathbf{x}$;
Select matrices $\widetilde{\mathbf{R}}, \mathbf{S}, \mathbf{U}$ such that $\mathbf{S} = \mathbf{AU}$;
$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$;
$i = 1$; $j = 0$;
while $\|r\|_2 >$ `tol` **do**
  Solve $\vec{\gamma}$ from $\widetilde{\mathbf{R}}^*\mathbf{S}\vec{\gamma} = \widetilde{\mathbf{R}}^*\mathbf{r}$;
  $\mathbf{v} = \mathbf{r} - \mathbf{S}\vec{\gamma}$;
  if $j = 0$ **then**
    $\mathbf{c} = \mathbf{Av}$;
    $\omega = \mathbf{c}^*\mathbf{v}/\mathbf{c}^*\mathbf{c}$;
    $\mathbf{U}e_i = -\mathbf{U}\vec{\gamma} - \omega\mathbf{v}$;
    $\mathbf{S}e_i = -\mathbf{S}\vec{\gamma} - \omega\mathbf{c}$;
  else
    $\mathbf{U}e_i = -\mathbf{U}\vec{\gamma} - \omega\mathbf{v}$;
    $\mathbf{S}e_i = \mathbf{AU}e_i$;
  end
  $\mathbf{x} = \mathbf{x} - \mathbf{U}e_i$;
  $\mathbf{r} = \mathbf{r} + \mathbf{S}e_i$;
  $i = i + 1$;
  if $i > s$ **then**
    $i = 1$;
  $j = j + 1$;
  if $j > s$ **then**
    $j = 0$;
end

6. IDR FOR $s = 1$

Notice the similarity between Bi-CGstab (Algorithm 4 on page 7) and the IDR(s) algorithm above. In this section, we will prove that they are mathematically the same for $s = 1$. Before we can prove this, we will derive an IDR(s) algorithm for the special case where $s = 1$.

Remember that the residuals in the IDR(s) algorithm are lifted to the next space after $s + 1$ steps. In Bi-CGstab, the residuals are taken from a smaller space in *every* iteration, as we take them orthogonal to some growing shadow space. We therefore can only 'hope' that every first lifted residual in the IDR(s) algorithm corresponds to the residual produced by Bi-CGstab. As $s = 1$, we want to prove that

$$r_{2k}^{\text{IDR}} \equiv r_k^{\text{Bi-CGstab}}.$$

To solve this indexing problem, we will derive an explicit algorithm for $s = 1$ where the residual is only updated every 'lifted' iteration.

Look at Lemma 2. As $s = 1$, the matrices S_k and U_k used in the algorithm reduce to the single vectors s_{k-1} and u_{k-1} respectively, while the matrix $\widetilde{R}$ reduces to a vector which we denote by $\widetilde{r}_0$. In Algorithm 7 we give the IDR(1) algorithm where we unroll the for-loop needed to lift a single residual to the next space. We (indeed) see that the residual is updated twice for every iteration of IDR(1).

We can use the identities in Corollary 1 to write Algorithm 7 in an alternative form, where we only store the 'lifted' residual. Using indices $k+1$ and $k+2$ for the first and second vector updates respectively, we find that

$$\begin{aligned} v_{k+1} &= r_k - s_{k-1}\alpha_{k+1} \\ &= r_{k-1} + s_k - s_k\alpha_{k+1} \\ &= r_{k-1} + s_k \underbrace{(\alpha_k - 1)}_{\alpha'_k} \end{aligned}$$

which leads to the update formula $u_k = (\alpha'_k + 1)u_{k-1} + \omega_{j+1}v_{k+1}$. Similarly, we can calculate s_k by $s_k = (\alpha'_k + 1)s_{k-1} + \omega_{j+1}Av_{k+1}$. Note that this allows us to compute r_{k+1} by only using r_{k-1}, i.e., the intermidate residual update r_k is not needed. The vectors u and s still need two updates every iteration: we suggest naming the intermidiate results with a prime (u' and s'). This leads to Algorithm 8.

Algorithm 7: IDR(s) for $s = 1$

Select $\mathbf{x}, \widetilde{\mathbf{r}}_\mathbf{0} \in \mathbb{C}^n$;
$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$;
$\mathbf{u} = \mathbf{r}, \quad \mathbf{s} = \mathbf{A}\mathbf{u}$;
while *Condition* **do**
  α such that
    $\mathbf{v} = \mathbf{r} - \mathbf{s}\alpha \perp \widetilde{\mathbf{r}}_\mathbf{0}$;
  $\mathbf{r}_{\text{old}} = \mathbf{r}$;
  Select ω
  $\mathbf{r} = (\mathbf{I} - \omega\mathbf{A})\mathbf{v}$;
  $\mathbf{u} = \mathbf{u}\alpha + \omega\mathbf{v}$;
  $\mathbf{s} = \mathbf{r} - \mathbf{r}_{\text{old}}$;

  β such that
    $\mathbf{v} = \mathbf{r} - \mathbf{s}\beta \perp \widetilde{\mathbf{r}}_\mathbf{0}$;
  $\mathbf{r}_{\text{old}} = \mathbf{r}$;
  $\mathbf{r} = (\mathbf{I} - \omega\mathbf{A})\mathbf{v}$;
  $\mathbf{u} = \mathbf{u}\beta + \omega\mathbf{v}$;
  $\mathbf{s} = \mathbf{r} - \mathbf{r}_{\text{old}}$;
end

Algorithm 8: IDR(s) for $s = 1$

Select $\mathbf{x}, \widetilde{\mathbf{r}}_\mathbf{0} \in \mathbb{C}^n$;
$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$;
$\mathbf{u} = \mathbf{r}, \quad \mathbf{s} = \mathbf{A}\mathbf{u}$;
while *Condition* **do**
  $\alpha = (1 + \alpha')$ such that
    $\mathbf{v} = \mathbf{r} - \mathbf{s}\alpha' \perp \widetilde{\mathbf{r}}_\mathbf{0}$;

  Select ω
  $\mathbf{r} = (\mathbf{I} - \omega\mathbf{A})\mathbf{v}$;
  $\mathbf{u}' = \mathbf{u}\alpha + \omega\mathbf{v}$;
  $\mathbf{s}' = \mathbf{s}\alpha + \omega\mathbf{A}\mathbf{v}$;

  β such that
    $\mathbf{v}' = \mathbf{r} - \mathbf{s}'\beta \perp \widetilde{\mathbf{r}}_\mathbf{0}$;

  $\mathbf{u} = \mathbf{u}'\beta + \omega\mathbf{v}'$;
  $\mathbf{s} = \mathbf{A}\mathbf{u}$;
end

6.1. **Comparison Bi-CGstab.** Algorithm 8 clearly 'looks' like Bi-CGstab given in Algorithm 4 (page 7).[10]

We are now ready to compare IDR(1) (given in Algorithm 8) with Bi-CGstab (Algorithm 4). Assuming that $v_{k+1}^{\text{IDR}} = v_{k+1}^{\text{Bi-CGstab}}$ and that the same selection procedure is used for ω_{k+1}, we immediately see that $r_{k+1}^{\text{IDR}} = r_{k+1}^{\text{Bi-CGstab}}$. What is left to check is the relation in v_{k+1} and u_{k+1}. We start with checking the latter.

[10]This Bi-CGstab algorithm was given in a more IDR suggestive form.

For IDR we find

$$\begin{aligned} u_{k+1}^{\text{IDR}} &= u'_{k+1}\beta_{k+1} + \omega_{k+1}v'_{k+1} \\ Au_{k+1}^{\text{IDR}} &= Au'_{k+1}\beta_{k+1} + \omega_{k+1}Av'_{k+1} \\ &= s'_{k+1}\beta_{k+1} + \omega_{k+1}Av'_{k+1} \\ &= r_{k+1} - v'_{k+1} + \omega_{k+1}Av'_{k+1} \\ &= (I - \omega_{k+1}A)v_{k+1} - (I - \omega_{k+1}A)v'_{k+1} \\ &= (I - \omega_{k+1}A)(v_{k+1} - v'_{k+1}). \end{aligned}$$

The IDR(1) algorithm ensures that $v'_{k+1} \perp \widetilde{r}_0$ and $v'_{k+1} \perp \widetilde{r}_0$. This implies that $v_{k+1} - v'_{k+1} \in r_0^{\perp}$, while on the other hand we have

$$\begin{aligned} v'_{k+1} - v_{k+1} &= r_{k+1} - s'_{k+1}\beta_{k+1} - v_{k+1} \\ &= v_{k+1} - \omega_{k+1}Av_{k+1} - (s_k\alpha_{k+1} + \omega_{k+1}Av_{k+1})\beta_{k+1} - v_{k+1} \\ &= -\omega_{k+1}Av_{k+1} - Au_k\alpha_{k+1}\beta_{k+1} - \omega_{k+1}Av_{k+1}\beta_{k+1}. \end{aligned}$$

This implies that $v'_{k+1} - v_{k+1} \in \text{span}(Av_{k+1}, Au_k) \cap \widetilde{r}_0^{\perp}$.

A similar statement holds for Bi-CGstab. We find that

$$\begin{aligned} Au_{k+1}^{\text{Bi-CGstab}} &= (I - \omega_{k+1}A)Au'_{k+1} \\ &= (I - \omega_{k+1}A)(Av_{k+1} - Au_k\beta_k). \end{aligned}$$

The Bi-CGstab algorithm ensures that we have $Av_{k+1} - Au_k\beta_k \in \text{span}(Av_{k+1}, Au_k) \cap \widetilde{r}_0^{\perp}$.

Now if v_{k+1} and u_k in IDR and Bi-CGstab coincide, then $v'_{k+1} - v_{k+1}$ in IDR is a multiple of $Av_{k+1} - Au_k$ in Bi-CGstab.[11]

So, if ω_{k+1} is chosen in the same fashion we find that $Au_{k+1}^{\text{Bi-CGstab}}$ is a multiple of Au_{k+1}^{IDR}. By construction we then have that v_{k+2} is the same in IDR and Bi-CGstab. Using an induction argument one now finds that IDR and Bi-CGstab produce the same vectors r_k, v_k while the vectors u_k are co-linear.

We summarize with the following corollary.

Corollary 2. *If IDR(1) and Bi-CGstab use the same selection procedure for ω and they have the same initial vectors u, v then Bi-CGstab and IDR(1) coincide (in exact arithmetic).*

7. Preconditioning

A (left-) preconditioner P of a matrix A is a matrix such that $P^{-1}A$ has a smaller condition number than A. This condition number plays a role in numerical stability and convergence rate. Instead of solving $Ax = b$, we premultiply by P^{-1} and solve the (left-)*preconditioned system*

$$P^{-1}Ax = P^{-1}b.$$

This preconditioner should satisfy:

- Convergence should be faster for the preconditioned system. Normally, this means that P is constructed as an "easily invertible" approximation to A;
- Operations with P^{-1} should be cheap to perform;
- P should be (relatively) easy to construct.

[11] The space $\text{span}(Av_{k+1}, Au_k) \cap \widetilde{r}_0$ is one-dimensional if Av_{k+1} or $Au_k$is not perpendicular to $\widetilde{r}_0$. This is the case, as $Av_{k+1} \perp \widetilde{r}_0$ implies a breakdown of both algorithms.

These matrices P can also form right-preconditioned systems

$$AP^{-1}y = b, \quad x = P^{-1}y$$

or central preconditioned systems

$$P = LU, \quad L^{-1}AU^{-1}y = L^{-1}b, \quad x = U^{-1}y.$$

As one can see, left preconditioning does not change the solution of the system, and is therefore easy to compute. Some methods (such as CG) rely on symmetry, and for these systems, the central preconditioning (given $L = U^*$) is more natural. An advantage of right preconditioning is that it does not affect the residual norm.

In Section 9 we will gather numerical results using the ILU(0) (*incomplete* LU with zero fill-in) preconditioner. It is constructed by making a standard LU-decomposition $A = LU$. However, during the process, some nonzero entries in the factors are discarded. This leads to $P = L'U'$ with L' and U' the incomplete L- and U-factors. We will discard entries based on the sparsity pattern of A: a nonzero is only kept if it corresponds to a nonzero entry in A. More details can be found in the literature, e.g., [Sle, Lec. 10B].

8. IDR implementation

The IDR(s) implementation takes ideas from [PS08] and [GLS10]. The former source suggests doing s steps of LMR to create an initial matrix U and S. The latter suggests setting

$$U = \texttt{orth}(\begin{bmatrix} r_0 & \cdots & A^{s-1}r_0 \end{bmatrix}), \quad S = AU.$$

Another source [Kis11, §3.3] suggests a few different options:

- Set $U = \widetilde{R}$, then solve $S = AU$;
- Set $U = \widetilde{R}$, solve $S = AU$, then set $\widetilde{R} = S$.

These options yield valid IDR methods, but they seem like an odd choice. Suddenly your residuals r_k will not be in the Krylov space $\mathcal{K}_{k+1}(A, r_0)$ anymore, but rather some iteratively expanded space initialized by columns of U.[12] We chose not to pursue these options.

We compared the different options, but there were minor differences in convergence. For our implementation we chose to use the LMR approach, this choice was arbitary. With this construction of U, we implemented both Algorithm 5 and Algorithm 6 in Matlab (using the ω selection described in §5.4). Also, for comparing Bi-CGstab and IDR we implemented the explicit IDR(1) algorithm given in Algorithm 8. The Matlab code is given in the appendix.

8.1. **Preconditioned IDR.**

In §7 we introduced the principle of a preconditioner. The preconditioner we will consider (ILU(0)) is of the form $P = P_1P_2$, where we do not explicitely calculate P. We implement the left preconditioner by taking the right hand side $P^{-1}b$ and replacing the matrix-vector product Av with $P^{-1}Av$. Here we split the calculating of $y = P^{-1}u$ in two steps. First we solve $P_1\hat{y} = u$ for $\hat{y}$ and then we find the solution y by solving $P_2y = \hat{y}$. As this is a left preconditioner we do not have to edit the solution afterwards. [Sle, Lec. 10 Sl. 8]

[12] Visual proof for this claim can be found in [Kis11, Fig. 5]. The residual vector for IDR(8) started with $U = \widetilde{R}$, $S = AU$ dips under the solution vector of GMRES for k between 50 and 100. This is not possible if $r_{k+1} \in K_k(A, r_0)$, as GMRES finds the *optimal* residual in this Krylov space. Hence, this method finds residuals outside our Krylov space.

Matrix	**# rows**	**Pattern sym**	**Value sym**	**Spectrum**	$\kappa_2(A)$
PDE	729	100%	44%	$\tau = 4.3 \cdot 10^0$	$2.4 \cdot 10^4$
STEAM2	600	100%	19%	$\tau = 0$ (real)	$3.8 \cdot 10^6$
diag(1 : 200)	200	100%	100%	$\tau = 0$ (real)	$2.0 \cdot 10^2$
SHERMAN4	1104	100%	0%	$\tau = 0$ (real)	$2.2 \cdot 10^2$
SHERMAN5	3312	74%	15%	$\tau = 4.0 \cdot 10^{-4}$	$1.9 \cdot 10^5$
ADD20	2395	100%	53%	$\tau = 1.0 \cdot 10^{-1}$	$1.2 \cdot 10^4$

TABLE 1. Some quantitative information on the matrices used. Sparsity pattern symmetry and numerical value symmetry measure how symmetric the matrix is. We define $\tau := \max_{\lambda \in \Lambda(A)} \left(\frac{Im(\lambda)}{Re(\lambda)} \right)$.

9. Numerical Results

In this section, we will look at a Matlab implementation of IDR(s) compared to a number of algorithms. This comparison will be done based on the amount of matrix-vector products (MV's) needed to converge, rather than computation time: we are comparing many different implementations from many different sources. Timing depends on skill of the programmer (among others). We simply cannot verify that each implementation is "optimal". Therefore, the amount of MV's needed is a good measure reliant on mathematics rather than implementation. Nevertheless, in §10 we will briefly discuss some computation time comparisons.

Our benchmark for this will be GMRES discussed in §2.1, as this produces the residual with minimal norm in each Krylov space. In other words, one cannot beat GMRES using a Krylov subspace solver in terms of MV's needed to converge.[13] Our comparison will include:

(1) The default Matlab implementation of GMRES, Bi-CG (§3.1), Bi-CGstab (§4.1), Bi-CGstab(ℓ) (§4.2);
(2) Our own implementation of IDR(s) and IDR for the explicit case with $s = 1$ as discussed in §8. From here on we will indicate the IDR(1) Algorithm 8 by IDR.

As Bi-CG produces a residual every second matrix-vector multiplication, we doubled its residual vector by repeating each value.

We will use various systems; see Table 1. We included the sparsity pattern symmetry and numerical value symmetry to measure the symmetry of the matrix. Large τ denotes a large relative imaginary component of the Eigenvalues. We know that Bi-CGstab and IDR perform badly on matrices with large τ.[SF93, p. 15]

In every case, we will use a relative tolerance of 10^{-8} and where possible, initialize

$$\widetilde{R} = \texttt{orth}([\texttt{r}, \texttt{randn}(\texttt{n}, \texttt{s} - \texttt{1})])$$

with a seed of zero, as suggested by [PS08]. Note that this choice allows us to directly compare IDR, Bi-CGstab, IDR($s = 1$) and Bi-CGstab($\ell = 1$), as these are equivalent when one chooses $\widetilde{r} = r$: see Corollary 2.

The test-matrices proposed in the original assignment all perform reasonably well, with a few exceptions. We found that when IDR(s) fails, it often has to do with a bad conditioning of the

[13] Please note however that GMRES is far from optimal in the computation time sense: it relies on long recurrences, so the cost of inner products is dominant in the long run. Other methods are crafted to use short recurrences, providing a method that scales better.

s	**Algorithm 5**		**Algorithm 6**		**van Gijzen**	
1	4374	(5.7×10^{-09})	5368	(8.8×10^{-09})	3838	(8.7×10^{-09})
2	2809	(8.6×10^{-09})	2621	(8.2×10^{-09})	2395	(7.9×10^{-09})
3	3166	(6×10^{-09})	2863	(3.5×10^{-09})	2386	(7.3×10^{-09})
4	2603	(7.3×10^{-09})	2760	(8.2×10^{-09})	1928	(8.1×10^{-09})
8	2437	(8.8×10^{-09})	2703	(9.1×10^{-09})	1687	(9.1×10^{-09})
16	2777	(9×10^{-09})	3178	(7.3×10^{-09})	1477	(8×10^{-09})

FIGURE 1. The matrix is SHERMAN5 with $b = A\vec{1}$. This table shows the amount of MV's needed and relative residual norm of the approximate solution for different values of s, comparing three algorithms.

matrix $\widetilde{R}^*S$ so that the $\vec{\gamma}$ vector is inaccurate.[14] When Bi-CGstab(ℓ) fails, it has to do with very slow convergence, warning us that the input tolerance might not be achieved. We chose to denote errors like these in the computation with an asterisk ($*$) in the figures ahead.

9.1. **IDR(s) comparison.** We start off by comparing the two different implementations of IDR(s) given in §8. We will complement this comparison with the (non-equivalent) implementation found in [vG] (cf. §5.4). The two main differences are (1) choice of ω and (2) the orthogonalisation of the matrices involved. Our system of choice is SHERMAN5 with right-hand side $b = A\vec{1}$. Later on we will see this system again with a different right-hand side.

See Figure 1. The algorithm of van Gijzen converges quicker than our implementations in all cases. We attribute this to the superior ω-strategy. Inspection showed that for large values of s, the matrix R^*S is ill-conditioned (as can be expected) in both our implementations, with the effect that $s = 16$ actually takes more iterations than $s = 8$. Van Gijzen has no such problem.

Opposed to what [PS08, §4.3] states, we see that Algorithm 6 converges faster than Algorithm 5 in most cases. We therefore decided to use Algorithm 6 as implementation of IDR(s) for the comparisons in this section.

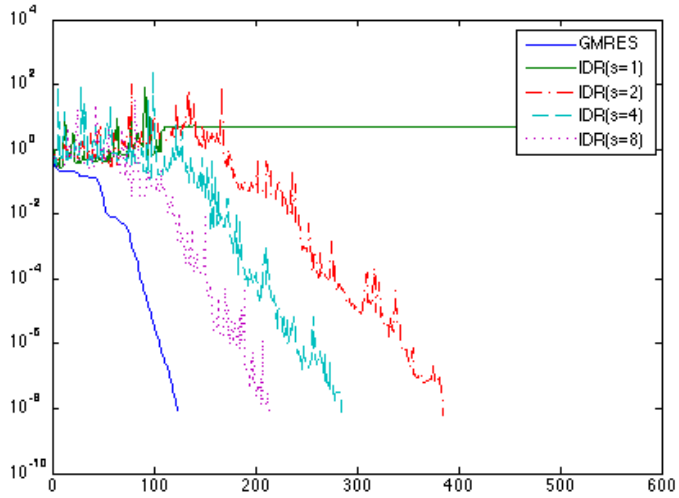
9.2. **A real-life example.** We will look at an instructional example taken from [vG, §3.1] that (albeit in a slightly different form) appears in [Kis11, §3.4.3] and [PS08, §6.3]. It is an example for which Bi-CGstab does not work well, due to the strong nonsymmetry of the system matrix. Specifically, the problem is caused by the fact that the matrix has Eigenvalues with large imaginary parts: see the right of Figure 3 and Table 1 – the matrix PDE has $\tau = 4.3$. Recall that Bi-CGstab uses the LMR method to stabilize the residual. It does not work well for this type of problem because the LMR steps produce a polynomial q_k that is a product of linear factors. Therefore, q_k has real roots, and hence it is unsuited as a residual minimizing polynomial as this should have roots close to the Eigenvalues.

The test problem is the finite difference discretization of the following convection-diffusion equation on the unit cube, with homogenous Dirichlet boundary conditions:

$$-\epsilon\Delta u + \vec{\beta} \cdot \nabla u - ru = F, \quad u(x, y, z) := x(1-x)y(1-y)z(1-z).$$

We used a grid size of $h = 0.1$ (yielding a system with 729 unknowns). The other parameters were left unchanged: the diffusion parameter $\epsilon := 0.02$, convection vector $\vec{\beta} := (0, 1, 2)/\sqrt{5}$ and reaction term $r = 6$. See Figure 2. We see the convergence for IDR(s) and Bi-CGstab(ℓ) for different values,

[14] This can be solved by orthogonalizing the matrices involved in IDR(s): see §5.5.



Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	123	8.8×10^{-09}
Bi-CG	314	8.6×10^{-09}
IDR		stagnate
IDR($s=1$)		stagnate
IDR($s=2$)	418	6.5×10^{-09}
IDR($s=4$)	311	9.9×10^{-09}
IDR($s=8$)	222	8.7×10^{-09}
Bi-CGstab		stagnate
Bi-CGstab($\ell=1$)		stagnate
Bi-CGstab($\ell=2$)	246	7.2×10^{-09}
Bi-CGstab($\ell=4$)	226	9.2×10^{-09}
Bi-CGstab($\ell=8$)	221*	1.1×10^{-08}

FIGURE 2. Convergence of different methods on the finite difference discretization of a convection-diffusion PDE. The x-axis in the image represents the amount of MV's executed with the corresponding relative residual norm on the y-axis. Note the stagnation for all methods mathematically equivalent to IDR.

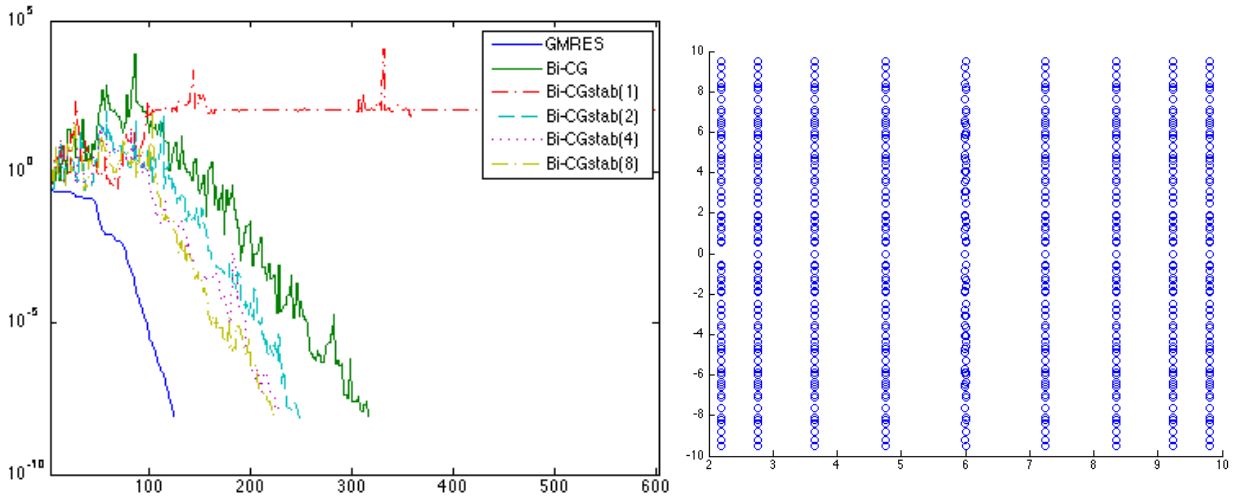
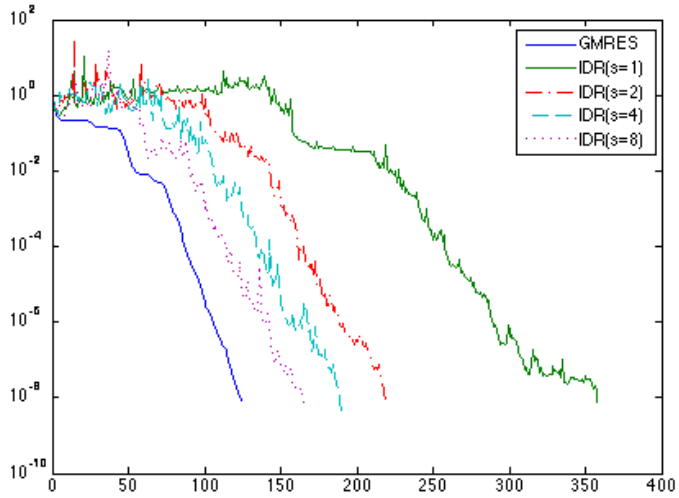


FIGURE 3. Again, left shows the convergence of different Bi-CGstab methods on this PDE matrix. Right shows the spectrum of this matrix: note that the relative imaginary part is large ($\tau = 4.3$).

and see (in the left pane of Figures 2 and 3) stagnation of all methods mathematically equivalent to Bi-CGstab for the reasons above. As second-degree polynomials can have complex roots, the fact that Bi-CGstab(ℓ) for $\ell \geq 2$ does not stagnate is no surprise, nor is the fact that Bi-CGstab(ℓ) performs better than IDR(s).

Note that the convergence of Bi-CG is quite good; this method does not rely on LMR polynomials.



Method	**MV**	$\|b - Ax\|/\|b\|$
GMRES	123	8.8×10^{-09}
Bi-CG	314	8.6×10^{-09}
IDR($s = 1$)	379	7.6×10^{-09}
IDR($s = 2$)	213	7.5×10^{-09}
IDR($s = 4$)	185	8.3×10^{-09}
IDR($s = 8$)	170	9.9×10^{-09}

FIGURE 4. (cf. Figure 2). The stagnation is gone if we use a complex random matrix $\widetilde{R}$ instead of a real one.

Beside increasing s or ℓ, there are other ways to improve performance. Recall that IDR(1) stagnates because the LMR polynomial is unable to approximate the Eigenvalues well, as $\omega \in \mathbb{R}$. If we take $\widetilde{R}$ to be a complex matrix rather than real, we find that $\vec{\gamma}$ is a complex vector so that ω becomes a complex value, solving the problem. The results, shown in Figure 4, are quite interesting. Suddenly, IDR($s = 2$) converges quicker than any Bi-CGstab(ℓ) method tested.

Do note however that computations are now done on complex values rather than reals, increasing the computational cost.

9.3. **IDR(1) comparison.** As we know from the theory of this paper, we have multiple methods that are mathematically equivalent, namely

$$\text{Bi-CGstab} \iff \text{Bi-CGstab}(\ell = 1) \iff \text{IDR}(s = 1) \iff \text{IDR}.$$

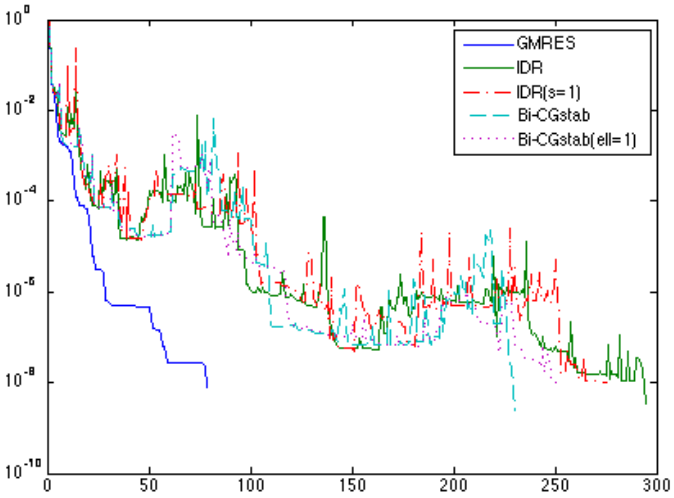
We decided to compare these using the matrix STEAM2 provided by the MatrixMarket. STEAM2 models a 3D oil reservoir. See Table 1. We chose $b = A\vec{1}$ to be the right hand side, so we know the exact solution.

See Figure 5. We immediately see that, while not exactly the same, the convergence pattern for these four methods is visibly equivalent. We also see that they perform quite badly; up to 3.8 times the optimal amount of MV's needed. This can (in part) be explained by the high condition number. This matrix also benefits a lot from using higher-dimensional shadow space; $s = 8$ only requires 1.2 times the optimal.

9.4. **Bi-CG comparison.** We took our matrix A to be the very simple diag(1 : 200). See Table 1; this matrix is very well-conditioned. Using preconditioning on this system solves it in one step.

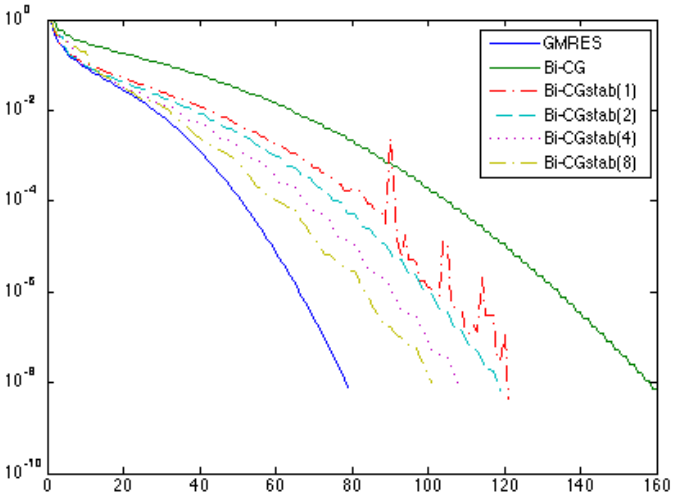
The result is Figure 6, showing beautifully how GMRES produces the best residuals, and that each enlargement of ℓ produces better results than the previous. It is around $\ell = 8$ that the optimum seems to have been reached; we are very close to the optimal MV count.

9.5. **IDR(s) comparison.** We took the SHERMAN4 matrix from the Matrix Market. It again models an oil reservoir. See Table 1. We chose $b = A\vec{1}$ to be compliant with results from [GLS10].



Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	78	8.2×10^{-09}
Bi-CG	156	8.6×10^{-09}
IDR	294	3.5×10^{-09}
IDR($s=1$)	274	9.8×10^{-09}
IDR($s=2$)	141	9.5×10^{-09}
IDR($s=4$)	170	9.9×10^{-09}
IDR($s=8$)	96	4.8×10^{-09}
Bi-CGstab	229	2.5×10^{-09}
Bi-CGstab($\ell=1$)	251	9.6×10^{-09}
Bi-CGstab($\ell=2$)	196	7.5×10^{-09}
Bi-CGstab($\ell=4$)	158*	6.1×10^{-08}
Bi-CGstab($\ell=8$)	162	1.3×10^{-09}

FIGURE 5. A comparison of different methods mathematically equivalent to Bi-CGstab, tested on the matrix STEAM2 from the Matrix Market.



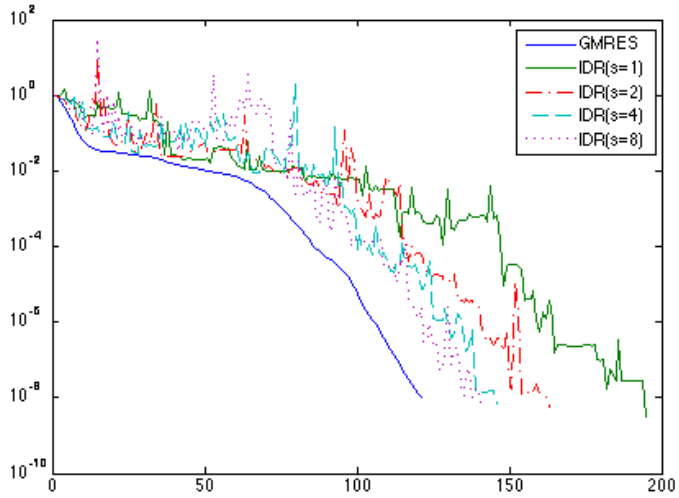
Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	78	8×10^{-09}
Bi-CG	158	7×10^{-09}
IDR	106	8.5×10^{-09}
IDR($s=1$)	106	8.5×10^{-09}
IDR($s=2$)	97	5.9×10^{-09}
IDR($s=4$)	90	7.2×10^{-09}
IDR($s=8$)	88	3×10^{-09}
Bi-CGstab	121	5.8×10^{-09}
Bi-CGstab($\ell=1$)	120	4.3×10^{-09}
Bi-CGstab($\ell=2$)	118	6.8×10^{-09}
Bi-CGstab($\ell=4$)	107	8.8×10^{-09}
Bi-CGstab($\ell=8$)	100	1×10^{-08}

FIGURE 6. A comparison of different Bi-CG(-like) methods, tested on the matrix diag(1 : 200).

This matrix is very well conditioned with a real spectrum, so we can expect good convergence for all methods (cf. §9.2).

See Figure 7. We see that larger s creates faster convergence (which was to be expected given the theory). We see that IDR(s) indeed works well for this matrix: the amount of MV's needed by IDR(s) is 1.6 times the optimal amount. For $s = 8$, this is a factor 1.2. Comparing results with [GLS10, Tbl. 1] shows that results are very similar. We also see that Bi-CGstab(ℓ) benefits from increasing ℓ, although less so.

9.6. **Preconditioning comparison.** We will look at the SHERMAN5 matrix (cf. Table 1), which produces results with very slow convergence, even for large values of ℓ and s. More testing



Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	120	9.8×10^{-09}
Bi-CG	272	9.1×10^{-09}
IDR	198	4×10^{-09}
IDR($s=1$)	194	3.2×10^{-09}
IDR($s=2$)	162	5.7×10^{-09}
IDR($s=4$)	145	7.1×10^{-09}
IDR($s=8$)	140	6.6×10^{-09}
Bi-CGstab	201	8.8×10^{-09}
Bi-CGstab($\ell=1$)	199	7.9×10^{-09}
Bi-CGstab($\ell=2$)	187*	1.1×10^{-08}
Bi-CGstab($\ell=4$)	183*	1×10^{-08}
Bi-CGstab($\ell=8$)	177	8.2×10^{-09}

FIGURE 7. A comparison of IDR(s) for different s, tested on the matrix SHERMAN4 from the Matrix Market.

yielded that even using smarter algorithms (like orthogonalizing the matrices, §5.5 or using another strategy for ω, §5.4) does not affect the convergence very much.

By Theorem 2, $(1+1/s)3312$ MV's should be enough for the methods to converge. While the convergence is very slow – see Figure 8 – IDR(s) for $s>1$ does manage to complete 'in time'. The exact reason for this very slow convergence is unclear: the matrix isn't ill-conditioned. We do see however that the conditioning of the matrix $\widetilde{R}^*S$ used in finding $\vec{\gamma}$ is very bad with a reciprocal condition number of 10^{-16}.

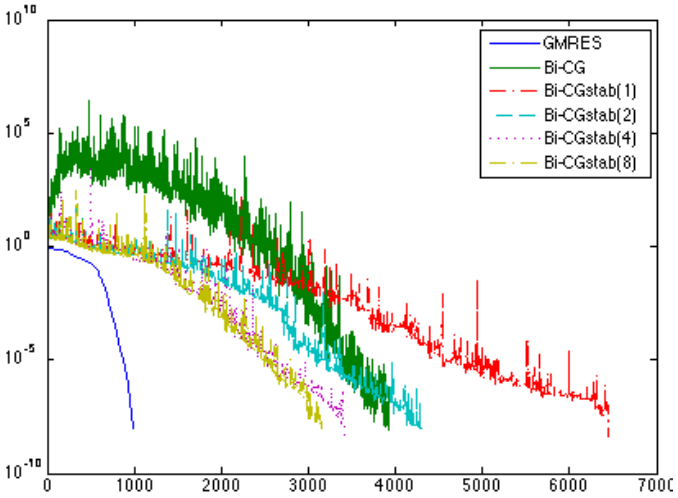
Figure 9 shows the convergence in the case where an ILU(0) preconditioner was used. Note the incredible difference. Looking at the condition number $\kappa_2(P^{-1}A) = 2.8 \cdot 10^3$ does not give much insight. Even looking at the spectrum ($\tau(P^{-1}A) = 7.4 \cdot 10^{-2}$) does not tell us much.

We can neither explain the immense computation time needed without preconditioning, nor the immense speedup from using a preconditioner.

9.7. **ADD20.** To complete our test case, we chose another MatrixMarket matrix: ADD20 – see Table 1. This matrix was also used in [GLS10, Tbl. 2]. See Figure 10 for a comparison of the convergence for regular and ILU(0)-preconditioned methods. We can see that IDR(s) performs better than Bi-CGstab(ℓ) in the non-preconditioned version, but Bi-CGstab(ℓ) seems to benefit more from preconditioning than IDR. The (non-preconditioned) results from [GLS10, Tbl. 2] are (thankfully) comparable.

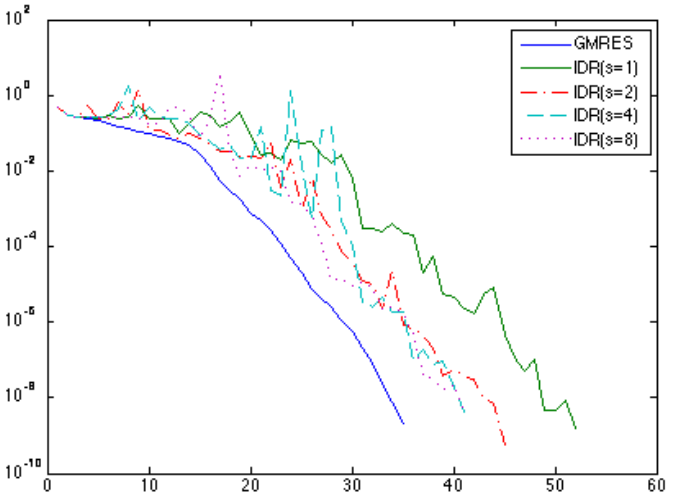
10. TIMING RESULTS

As noted before, we used the MV to compare the performance of different methods. In reality, the methods also spent time computing inner products and vector updates. GMRES uses long recurrences to update the residuals so iterations get slower. As IDR(s) and Bi-CGSTAB use short recurrence relations, the time spent in each iteration is more or less the same. We timed the iterations of different methods to get an idea of the computation time per iteration. We did this by editing Matlab's GMRES and Bi-CGSTAB(ℓ) implementation to return a vector with the time elasped between two consecutive MV's. Similarly, we edited our IDR(s) implementation.



Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	986	9.7×10^{-09}
Bi-CG	3928	7.7×10^{-09}
IDR	7404	9.1×10^{-09}
IDR($s=1$)	10461	5.3×10^{-09}
IDR($s=2$)	3919	9.1×10^{-09}
IDR($s=4$)	3002	1×10^{-08}
IDR($s=8$)	3098*	7.8×10^{-09}
Bi-CGstab	1328*	0.47
Bi-CGstab($\ell=1$)	6452	4.2×10^{-09}
Bi-CGstab($\ell=2$)	4294	1×10^{-08}
Bi-CGstab($\ell=4$)	3410	5.1×10^{-09}
Bi-CGstab($\ell=8$)	3149*	1×10^{-08}

FIGURE 8. A comparison of different methods, tested on the matrix SHERMAN5 ($n = 3312$) from the Matrix Market.



Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	34	2.1×10^{-09}
Bi-CG	74	5.5×10^{-09}
IDR	52	6×10^{-10}
IDR($s=1$)	51	1.6×10^{-09}
IDR($s=2$)	44	5.8×10^{-10}
IDR($s=4$)	40	4.4×10^{-09}
IDR($s=8$)	40	3.7×10^{-09}
Bi-CGstab	49	4×10^{-09}
Bi-CGstab($\ell=1$)	49	4×10^{-09}
Bi-CGstab($\ell=2$)	49	7.4×10^{-09}
Bi-CGstab($\ell=4$)	49	2.7×10^{-09}
Bi-CGstab($\ell=8$)	49*	3.2×10^{-08}

FIGURE 9. Preconditioned convergence for the same matrix as Figure 8. Notice the immense difference in convergence speed.

An example of the time per iteration is given in Figure 11 (we used the ADD20 matrix). We clearly see behaviour one would expect. For GMRES the time per MV (or per iteration) increases linearly. For IDR(16) the time spent between the first 16 MV's is small, this corresponds to the initialization stage. In this stage we construct the matrices U and S, in 16 iterations. Then upon entering the main loop, we see a spike in computation time every 17th MV. This corresponds to the case where our residuals get lifted to the next space; the new ω must be calculated. For Bi-CGSTAB(8) the graph clearly shows 8-periodic beheaviour. The first 8 MV's correspond to the Bi-CG loop, while the next 8 MV's correspond to finding the stabilizing polynomial of degree 8.

Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	295	1×10^{-08}
Bi-CG	640	9.4×10^{-09}
IDR	826	7.2×10^{-09}
IDR($s=1$)	772	9.9×10^{-09}
IDR($s=2$)	550	8.9×10^{-09}
IDR($s=4$)	542	4.3×10^{-09}
IDR($s=8$)	380	8.7×10^{-09}
Bi-CGstab	750	9.8×10^{-09}
Bi-CGstab($\ell=1$)	815	8.7×10^{-09}
Bi-CGstab($\ell=2$)	738	7.4×10^{-09}
Bi-CGstab($\ell=4$)	587	9.9×10^{-09}
Bi-CGstab($\ell=8$)	567	9.3×10^{-09}

Method	**MV**	$\|b-Ax\|/\|b\|$
GMRES	138	6.3×10^{-08}
Bi-CG	276	9.7×10^{-09}
IDR	240	7.1×10^{-08}
IDR($s=1$)	247	4.7×10^{-08}
IDR($s=2$)	240	6.2×10^{-08}
IDR($s=4$)	215	5.7×10^{-08}
IDR($s=8$)	182	3.1×10^{-08}
Bi-CGstab	273	9.1×10^{-09}
Bi-CGstab($\ell=1$)	260	1×10^{-08}
Bi-CGstab($\ell=2$)	200	9.6×10^{-09}
Bi-CGstab($\ell=4$)	168	9.9×10^{-09}
Bi-CGstab($\ell=8$)	179	9.6×10^{-09}

FIGURE 10. Convergence for the matrix ADD20 from the Matrix Market. Left: non-preconditioned convergence table; right: preconditioned convergence table.

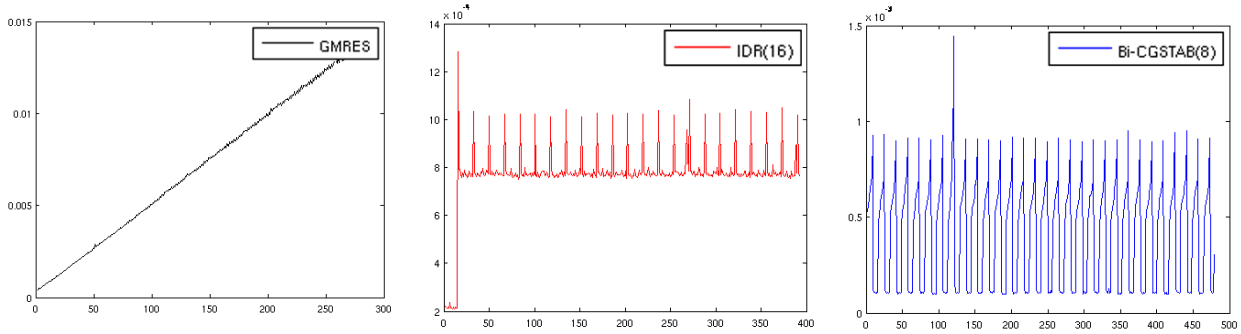


FIGURE 11. Timing results of three different (unpreconditioned) methods for the ADD20 matrix with the corresponding Matrix Market right hand side. On the x-axis we have the MV number. The y-axis indicates the time spent between two consecutive MV's.

The graphs corresponding to the preconditioned timings look more or less the same, as one would expect.

In Figure 12, a comparison of the total computation time is given. The real power of Bi-CGSTAB and IDR is clearly visible in this data. Especially in the unpreconditioned version we see that both methods outperform GMRES in computation time by a factor ≈ 10. Another interesting thing is the comparison of Bi-CGSTAB(1) and IDR(1). As these methods are mathematically equivalent one should expect their executing time to be more or less the same. From the Figure we see however that Bi-CGSTAB(1) is almost twice as slow in execution time. This (most likely) due to the fact that the implementation of Bi-CGSTAB(ℓ) runs all kinds of different 'checks' each iteration whereas our implementation of IDR(s) has no such checks. This is exactly the reason we used MV to compare the performance of different methods.

Method	MV	Time in s
GMRES	285	2.1353
IDR($s = 1$)	708	0.14085
IDR($s = 2$)	534	0.1237
IDR($s = 3$)	579	0.14322
IDR($s = 4$)	550	0.1428
IDR($s = 8$)	413	0.13184
IDR($s = 16$)	393	0.21767
Bi-CGstab($\ell = 1$)	772	0.24351
Bi-CGstab($\ell = 2$)	699	0.22958
Bi-CGstab($\ell = 3$)	729	0.24744
Bi-CGstab($\ell = 4$)	612	0.21634
Bi-CGstab($\ell = 8$)	480	0.19322

Method	MV	Time in s
GMRES	126	0.57601
IDR($s = 1$)	229	0.074272
IDR($s = 2$)	192	0.069943
IDR($s = 3$)	249	0.097691
IDR($s = 4$)	159	0.062567
IDR($s = 8$)	159	0.069901
IDR($s = 16$)	145	0.088876
Bi-CGstab($\ell = 1$)	240	0.13531
Bi-CGstab($\ell = 2$)	171	0.10136
Bi-CGstab($\ell = 3$)	198	0.11736
Bi-CGstab($\ell = 4$)	196	0.12409
Bi-CGstab($\ell = 8$)	187	0.13054

FIGURE 12. Timing results for the ADD20 matrix with the corresponding right hand side. On the left we have the non-preconditioned verison, on the right wave have the ILU(0) preconditioned system.

11. Conclusion

In this paper, Iterative solvers were discussed. We started at GMRES in §2.1, before asserting that if A is Hermitian, GMRES is mathematically equivalent to CG. We discussed CG in §3 and its short-recurrence, non-Hermitian variant Bi-CG in §3.1. From this, we introduced Bi-CGstab in §4.1 and generalized this to §4.2.

We then proposed the idea of finding residuals not in some (growing) sequence of Krylov spaces, but rather some sequence of shrinking subspaces in Theorem 1. This *Induced Dimension Reduction* method or IDR(s) is the shining star of this paper. We saw that IDR(1) is in fact mathematically equivalent to Bi-CGstab in §6, providing us with the connection between IDR(s) and the previous.

We then used GMRES, Bi-CG, Bi-CGstab, Bi-CGstab(ℓ), IDR and IDR(s) to provide the reader with numerical results, some of which containing comparisons with articles of other authors. We saw that convergence improves when using higher s and ℓ. From these results, we can conclude that the quote at the very top of this paper holds: the IDR(s) algorithm indeed seems to outperform the state-of-the-art Bi-CG-type methods.

From §9.1, we can conclude that orthogonalisation of the matrices and using the ω-strategy of [vGS11] yields improved performance.

11.1. **Discussion.** One must admit that while it is interesting to compare IDR(s) with Bi-CGstab(ℓ), they are different methods entirely. The former increases the size of the shadow space while retaining the polynomial degree, while the former uses a one-dimensional shadow space with a polynomial of degree ℓ. In [GLS10], a method named Bi-CGSTAB(s) is suggested, which is mathematically equivalent to IDR(s). To not clutter our minds too much, we decided to leave this for another time.

Each article, paper, lecture and other source that we used has a slightly (mathematically equivalent) definition of whatever method they discuss. Some sources for instance find r_{k+1}, and then u_{k+1} while others find u_k and use this to find r_{k+1}. This makes translating algorithms very arduous. Moreover, our main two sources for the IDR(s) algorithm – [GLS10] and [PS08] – have opposing signs for some of the involved quantities. This again made translation troublesome.

The last point of discussion is the choice of matrices. The original assignment states that we should use multiple self-defined systems and a selection of found matrices. We found it very difficult to generate matrices with the desired properties, and chose to use existing examples from other sources as inspiration for our own numerical results. Using examples from more knowledgeable people instead of our own allowed us to understand much more of the underlying mechanics behind certain convergence behaviours.

References

[GHG13] Charles F. Van Loan Gene H. Golub. *Matrix Computations:: 4th Edition*. The Johns Hopkins University Press, 2013.

[GLS10] Martin B. van Gijzen Gerard L.G. Sleijpen, Peter Sonneveld. Bi-CGSTAB as an induced dimension reduction method. *Applied Numerical Mathematics*, 60, 2010.

[Kis11] Goushani Kisoensingh. IDR(s). Master's thesis, University of Utrecht, 2011.

[LNT97] David Bau Lloyd N. Trefethen. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[PS08] Martin B. van Gijzen Peter Sonneveld. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *Journal for Scientific Computing*, 30(2):1035–1062, 2008.

[SF93] Gerard L.G. Sleijpen and Diederik R. Fokkema. BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numerical Analysis*, 1(11):2000, 1993.

[Sle] Gerard L.G. Sleijpen. Numerical Linear Algebra — Course notes. `http://www.staff.science.uu.nl/~sleij101/Opgaven/NumLinAlg`.

[SS86] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[TFC97] Henk van der Vorst Tony F. Chan, Lisette de Pillis. Transpose-free formulations of Lanczos-type methods for nonsymmetric linear systems, 1997.

[VdV92] Henk A Van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992.

[VF84] Thomas A. Manteuffel Vance Faber. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 1984.

[vG] Martin van Gijzen. The Induced Dimension Reduction method. `http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html`.

[vGS11] Martin B. van Gijzen and Peter Sonneveld. An Elegant IDR(s) Variant that Efficiently Exploits Biorthogonality Properties. *ACM Transactions on Mathematical Software*, 2011.

[WS80] P. Wesseling and Peter Sonneveld. Numerical experiments with a multiple grid and a preconditioned Lanczos type method. *J. Comput. Phys.*, 1980.

Appendix A. Matlab Algorithms

Listing 1. IDR(s) Algorithm 5

```
function [x, k, resvec,timevec] = idrs_eigen_imp(A, b, s, tol, maxit, R, M1, M2 , x_0)
    tic;
    [n, ~] = size(b);
    if nargin < 5,
        maxit = n;
    end
    if nargin < 6,
        randn('state',0);
        R = randn(n, s);
        R = orth(R);
    end

    if nargin < 7 || isempty(M1)
        precM1 = 0;
    else
        precM1 = 1;
        b = M1\b;
    end
    if nargin < 8 || isempty(M2)
        precM2 = 0;
    else
        precM2 = 1;
        b = M2\b;
    end

    if nargin < 9,
        x = zeros(n,1);
        r = b;
    else
        x = x_0;
        r = b - A*x;
    end
    nb = norm(b);
    resvec = zeros(maxit+1,1);
    timevec = zeros(maxit+1,1);
    resvec(1) = nb;
    timevec(1) = toc;

    U = zeros(n,s);
    S = zeros(n,s);
    for k = 0:(s-1),
        v = A*r;
        if precM1
            v = M1\v;
        end
        if precM2
            v = M2\v;
        end
        omega = v'*r/(v'*v);
        U(:,k+1) = -omega*r;
        S(:,k+1) = - omega*v;
        r = r + S(:,k+1);
        resvec(k+2) = norm(r);
        timevec(k+2) = toc;
        x = x - U(:,k+1);
    end

    i = 1;
    j = 0;
    omega = 0;
    k = s;
    while k <= maxit && (resvec(k)/resvec(1) > tol),
        gamma = (R'*S)\(R'*r);
        v = r - S*gamma;
        if j == 0,
            c = A*v;
            if precM1
                c = M1\c;
            end
            if precM2
                c = M2\c;
```

```
            end
            omega = c'*v/(c'*c);
            U(:,i) = -U*gamma - omega*v;
            S(:,i) = -S*gamma - omega*c;
        else
            U(:,i) = -U*gamma - omega*v;
            S(:,i) = A*U(:,i);
            if precM1
                S(:,i) = M1\S(:,i);
            end
            if precM2
                S(:,i) = M2\S(:,i);
            end
        end
        x = x - U(:,i);
        r = r + S(:,i);
        resvec(k+1) = norm(r);
        timevec(k+1) = toc;
        i = i+1;
        if i > s,
            i = 1;
        end
        j = j+1;
        if j > s,
            j = 0;
        end
        k = k + 1;
    end
    if (k <= maxit)
        resvec = resvec(1:k);
        timevec = timevec(1:k);
    end
end
```

LISTING 2. IDR(s) Algorithm 6

```
function [x, k, resvec,timevec] = idrs_eigen(A, b, s, tol, maxit, R, M1, M2 , x_0)
    tic;
    [n, ~] = size(b);
    if nargin < 5,
        maxit = n;
    end
    if nargin < 6,
        randn('state',0);
        R = randn(n, s);
        R = orth(R);
    end

    if nargin < 7 || isempty(M1)
        precM1 = 0;
    else
        precM1 = 1;
        b = M1\b;
    end
    if nargin < 8 || isempty(M2)
        precM2 = 0;
    else
        precM2 = 1;
        b = M2\b;
    end

    if nargin < 9,
        x = zeros(n,1);
        r = b;
    else
        x = x_0;
        v = A*x;
        if precM1
            v = M1\v;
        end
        if precM2
            v = M2\v;
        end
        r = b - v;
    end
```

```
    nb = norm(b);
    resvec = zeros(maxit+1,1);
    timevec = zeros(maxit+1,1);
    resvec(1) = nb;
    timevec(1) = toc;

    U = zeros(n,s);
    S = zeros(n,s);
    for k = 0:(s-1),
        v = A*r;
        if precM1
            v = M1\v;
        end
        if precM2
            v = M2\v;
        end
        omega = v'*r/(v'*v);
        U(:,k+1) = -omega*r;
        S(:,k+1) = - omega*v;
        r = r + S(:,k+1);
        resvec(k+2) = norm(r);
        timevec(k+2) = toc;
        x = x - U(:,k+1);
    end

    i = 1;
    j = 0;
    omega = 0;
    k = s;
    while k <= maxit && (resvec(k)/resvec(1) > tol),
        gamma = (R'*S)\(R'*r);
        v = r - S*gamma;
        c = A*v;
        if precM1
            c = M1\c;
        end
        if precM2
            c = M2\c;
        end
        if j == 0,
            omega = c'*v/(c'*c);
        end
        U(:,i) = -U*gamma - omega*v;
        x = x -  U(:,i);
        r1 = v - omega*c;
        S(:,i) = r1 - r;
        r = r1;
        resvec(k+1) = norm(r);
        timevec(k+1) = toc;
        i = i+1;
        if i > s,
            i = 1;
        end
        j = j+1;
        if j > s,
            j = 0;
        end
        k = k + 1;
    end
    if (k <= maxit)
        resvec = resvec(1:k);
        timevec = timevec(1:k);
    end
end
```

LISTING 3. IDR(1) Algorithm 8

```
function [x, k, resvec] = idr_eigen( A, b, tol, rtilde, M1, M2, x_0)
    [n, ~] = size(A);
    if nargin < 4,
        randn('state',0);
        rtilde = randn(n, 1);
    end
    if nargin < 5 || isempty(M1)
        precM1 = 0;
```

```
    else
        precM1 = 1;
        b = M1\b;
    end
    if nargin < 6 || isempty(M2)
        precM2 = 0;
    else
        precM2 = 1;
        b = M2\b;
    end
    if nargin < 7,
        x = zeros(n,1);
        r = b;
    else
        x = x_0;
        r = b - A*x;
    end
    resvec = [norm(r)];
    u = r;
    c = A*u;
        if precM1
            c = M1\c;
        end
        if precM2
            c = M2\c;
        end
    k=0;
    while norm(r)/norm(b) > tol,
        rho = rtilde'*r;
        sigma = rtilde'*c;
        alpha = rho/sigma;
        rprime = r - alpha*c;
        resvec = [resvec; norm(rprime)];
        xprime = x + alpha*u;
        s = A*rprime;
        if precM1
            s = M1\s;
        end
        if precM2
            s = M2\s;
        end
        omega = rprime'*s/(s'*s);
        r = rprime - omega*s;
        resvec = [resvec; norm(r)];
        x = xprime + omega*rprime;
        mu = rtilde'*s;
        beta = mu/sigma;
        cprime = s - beta*c;
        uprime = r - beta*u;
        l = A*cprime;
        if precM1
            l = M1\l;
        end
        if precM2
            l = M2\l;
        end
        c = cprime - omega*l;
        u = uprime - omega*cprime;
        k = k+1;
    end
end
```