Recycling Techniques for Induced Dimension Reduction Methods Used for Solving Large Sparse Non-Symmetric Systems of Linear Equations

July 26, 2017

Master thesis, Mathematical Sciences, Utrecht University

Abstract

Induced Dimension Reduction (IDR) methods are among the most stable and efficient iterative methods for solving large, sparse, non-symmetric systems of linear equations known today. However, the exact reasons for their effectiveness are only partially understood. In recent work on recycling techniques for IDR it is shown that greater efficiency than theoretically possible for a Krylov subspace-method can be achieved by altering the auxiliary input of an IDR algorithm, perhaps turning the methods into a class of their own. In this thesis a self-contained derivation of IDR(s) and IDR(s)Stab(ℓ) is presented, prior to an experiment driven investigation of the possibilities of recycling techniques. Issues with the numerical stability of these techniques will be addressed and theory is presented providing more insight in the matter. Moreover, an attempt is made to apply theory on spectral aspects of IDR-methods to justify the effectiveness of recycling, which also might be a starting point in gaining a further understanding of the convergence behavior of IDR methods in general.

Acknowledgements

I would first of all like to thank Gerard Sleijpen for his supervision and guidance throughout the process. This thesis would not have come to be without his expertise in the field numerical linear algebra and our valuable discussions (whatever the subject). One only needs to take a glimpse at the bibliography at the end of this document to see how much this work, and that of many others, has been influenced by his knowledge. I would also like to thank Tristan van Leeuwen for acting as my co-supervisor. Furthermore I wish to thank E.J.M Beije, L. Stronks, H.V. Koops and D. Speelman, as well as my parents and brother, Sjip, for taking the time to teach me about thesis writing and prevent many errors from seeing the light of day.

Contents

Glossary of Symbols

Symbol	Description
N	A capital letter N will denote a dimension (for example, the dimension of the input problem)
$\mathbb{N}, \mathbb{N}_+$	We distinguish between the set of natural numbers including zero, $\mathbb{N}$, and the strictly positive $\mathbb{N}_+ \equiv \mathbb{N} \setminus \{0\}$
$\mathbf{A}, \mathbf{V}, ...$	Uppercase letters in bold denote matrices with columns in $\mathbb{C}^N$
$\mathbf{I}$	The uppercase $\mathbf{I}$ in bold is reserved for the identity matrix, whose size may be dependent on the context
$\mathbf{O}$	The uppercase $\mathbf{O}$ in bold is reserved for the zero matrix, whose size may be dependent on the context
$\mathbf{A}^*$	A matrix super scripted with an asterisk will denote the matrix's conjugate transposed
$\mathbf{x}, \mathbf{y}, ...$	Lowercase letters in bold denote vectors in $\mathbb{C}^N$
$[\mathbf{x}, \mathbf{y}, ...]$	We may put vectors between square brackets to denote the matrix with columns $\mathbf{x}, \mathbf{y}, ...$
$\mathbf{x}_k, \mathbf{r}_k, ...$	We will index vectors do emphasize that they are the k-th iterates in some iterative process
$\vec{\gamma}, \vec{\alpha}...$	Greek symbols with an arrow op top denote vectors of length smaller than N (Sometimes to denote a set of coefficients, denoting their i-th entry by a subscript.)
$(\omega)_{k\in\mathbb{N}_+}, (\mu)_{k=1}^J, ...$	We use bracketed ω and μ to denote sequences of scalars in $\mathbb{C}$. Their elements are indexed by a subscript, whose range is always denoted together with the sequence. That is, we have $(\omega)_{k\in\mathbb{N}_+} \in \mathbb{C}^{\mathbb{N}}$, such that $\forall_{k\in\mathbb{N}}\omega_k \in \mathbb{C}$. Likewise we have a finite sequence by writing $(\mu)_{k=1}^J$, such that $\forall_{k=1,...,J}\mu_k \in \mathbb{C}$.
$\mathcal{G}, \mathcal{R}^\perp$	Calligraphic uppercase letters will denote vector (sub)spaces in $\mathbb{C}^N$. The $\perp$ in superscript denotes the orthogonal complement of the space
$\mathbf{R} \in \mathcal{G}$	We will use '$\in$' to denote that the columns of a matrix, $\mathbf{R}$, are elements of the vector space $\mathcal{G}$
$\mathbf{W} \perp \mathcal{G}$	We will use '$\perp$' to denote that the columns of a matrix, $\mathbf{W}$, are orthogonal to the vector space $\mathcal{G}$
$P_m(\xi), Q_n(\xi)$	Non-bold uppercase letters denote polynomials. We will always accompany polynomials with a subscript denoting its degree. For example we might have $P_3(\xi) = (1-\alpha_1\xi)(1-\alpha_2\xi)(1-\alpha_3\xi)$. (for some coefficients $\alpha_1, \alpha_2, \alpha_3$). We will freely apply the polynomial structure to either complex numbers or matrices. Hence, $P_k(\mathbf{A})$ will denote a matrix that is the result of evaluating P_k in the matrix $\mathbf{A}$.
DW2048, SHERMAN4,...	Terms in 'small caption' denote a coefficient matrix used as a test problem. See Appendix B for a list of all the matrices used in this wirh and some elementary properties that may be relevant from a numerical perspective.

Chapter 1

Introduction and Preliminaries

One of the most common computations to be done in both scientific research and engineering is the solving of linear systems. Typical contexts giving rise to such systems are applying finite element methods to discretized differential equations, optimization in electronic circuit board design and inverse problems in geo-sciences to name only a few. Also, the so-called *linearization* of non-linear problems should not be overlooked when one tries to get a sense of the importance of effectively solving linear systems. Over the years, the size of the systems to be solved has grown rapidly, and systems consisting of well over a million equations are no rarity at all. Although the availability of computational power has grown as well, the development of increasingly efficient methods over the last few decades has enabled its users to take a leap forward enabling better bridge designs, more accurate weather-forecasts, and so on. In a sense, the availability of faster methods speeds up a significant part of research in so many fields, that researchers will always be eager to obtain a better understanding of the methods used hoping the be able to improve them even further. For an important class of these linear systems, *iterative Krylov subspace methods* have shown to be unmatched. In particular, we will be focusing systems in which $\mathbf{x}$ is to be solved from

$$\mathbf{Ax} = \mathbf{b} \quad \text{, with} \quad \mathbf{A} \in \mathbf{C}^{N \times N} \text{ and } \mathbf{x}, \mathbf{b} \in \mathbb{C}^N,$$

in which $\mathbf{A}$ is **sparse**, and possibly non-symmetric. That is, many (if not most) entries in $\mathbf{A}$ are equal to zero, and $\mathbf{A}$ may have complex eigenvalues. For these systems, Krylov subspace methods have gained an increasing interest over the years as they have a natural way of exploiting the sparsity, explaining their efficiency. The convergence speed and stability of solving algorithms are strongly dependent on the input size, algebraic properties of $\mathbf{A}$ and underlying method used. Well-known Kyrlov subspace-methods range from the simple *(modified) Richardson iteration* ([10]) to the elegant, and very effective, *Conjugate Gradient* ([4]) for symmetric $\mathbf{A}$, eventually resulting in the, more generally applicable, methods such as *Bi-Conjugate Gradient* ([3]), *Conjugate Gradient Squared* ([22]) and *Bi-Conjugate Gradient Stabilized* ([24]). Less well known was the method of *Induced Dimension Reduction*, proposed by Sonneveld and introduced in [26] in 1980 , until it was revived as *IDR(s)* in a paper by Sonneveld & Gijzen from 2008. It is this development that led to one of the most robust and efficient methods for non-symmetric sytems known today called *IDRStab* ([16]).

After this introductory chapter motivating our research and acquainting the reader[1] with

[1] Who, if already familiar with the subject of numerical linear algebra and Krylov subspace methods, may very well skip the rest of this chapter.

the relevant elementary concepts, the work is split in to two parts: Part I is devoted to a self-contained derivation of both IDR(s) and IDRStab. We will start from the underlying theory and work our way up to actual implementations. We will briefly study some of the design choices involved that are partly specific for IDR and set a standard for the rest of the chapters. Part I is concluded with a chapter on additional remarks aimed at giving the reader a better understanding of how IDR methods relate to other (Krylov subspace) methods. In Part II, we will focus on techniques for IDR methods used to solve a sequence of linear systems that share the coefficient matrix $\mathbf{A}$ in we attempt to **recycle** information from one system and exploit it for the solving of the next. These *sequences* of systems sharing the coefficient matrix $\mathbf{A}$ occur for example in the context of time-dependent problems, optimization and inverse problems. Experiments have also shown the possible effective application to **shifted systems**, in which $\mathbf{A}$ is replaced by $\mathbf{A}+\sigma\mathbf{I}$ for a multitude of values for σ, arising for example when using Tikhonov regularization. Focusing on the method rather than all its possible applications, we will see how these techniques might turn IDR into what is 'not really' a Krylov subspace method. Not discouraged by this fact, experiments giving plenty of motivation in terms of convergence speed, justifying the exploration of the possibilities are presented. After that we will study two related variations, their advantages and disadvantages in theory and practice. Peculiarities of one of these will be presented and we will try to theoretically justify both welcome and unwelcome phenomena observed in experiments. In both parts, algorithms are implemented in Matlab, and program code of the two main algorithms used can be found in Appendix A. The code is written to be readable, and is flexible in the sense that it allows easy experimentation in the way that is done to write this thesis. In order to make replication of experiments easier, and for those who wonder about the applicability of recycling IDR techniques, a glossary of test problems used to obtain the bulk of the experimental results together with their elementary numerical properties that might be of interest is included in Appendix B.

1.1 Elementary notions

We fix the following notation for the elementary properties of some $N \times k$-matrix $\mathbf{A}$ and some vector space $\mathcal{V} \subset \mathbb{C}^N$.

The *determinant* of $\mathbf{A}$

$$\mathbf{Det}(\mathbf{A})$$

The *span* of $\mathbf{A}$

$$\mathbf{Span}(\mathbf{A}) \equiv \left\{ \mathbf{A}\vec{\alpha} \;\middle|\; \vec{\alpha} \in \mathbb{C}^k \right\}$$

The *dimension* of $\mathcal{V}$

$$\mathbf{Dim}(\mathcal{V}) \equiv \min \left\{ d \in \mathbb{N} \;\middle|\; \exists \mathbf{V} \in \mathbb{C}^{N \times d} \text{ s.t. } \mathbf{Span}(\mathbf{V}) = \mathcal{V} \right\}$$

The *null space* (or *kernel*) of $\mathbf{A}$

$$\mathcal{N}(\mathbf{A}) \equiv \left\{ \vec{\alpha} \in \mathbb{C}^n \;\middle|\; \mathbf{A}\vec{\alpha} = \mathbf{0} \right\}$$

The *rank* of $\mathbf{A}$

$$\mathbf{Rank}(\mathbf{A}) \equiv \mathbf{Dim}(\mathbf{Span}(\mathbf{A})) = n - \mathbf{Dim}(\mathcal{N}(\mathbf{A}))$$

The *spectrum* of $\mathbf{A} \in \mathbb{C}^{N \times N}$

$$\Lambda(\mathbf{A}) \equiv \left\{ \lambda \in \mathbb{C} \;\middle|\; \exists \mathbf{x} \text{ s.t. } \mathbf{x} \neq \mathbf{0}, \mathbf{A}\mathbf{x} = \lambda \mathbf{x} \right\}$$

The *sum* of two vector spaces $\mathcal{V}_1, \mathcal{V}_2 \subseteq \mathbb{C}^N$

$$\mathcal{V}_1 + \mathcal{V}_2 \equiv \left\{ \mathbf{v}_1 + \mathbf{v}_2 \;\middle|\; \mathbf{v}_1 \in \mathcal{V}_1, \mathbf{v}_2 \in \mathcal{V}_2 \right\}$$

We call $\mathbf{A}$ *sparse* if most of its entries equal zero.

The number of non-zero entries of $\mathbf{A}$ will be denoted by $nz_{\mathbf{A}}$.

And we will use the following definitions of common concepts.

Definition 1.1. Krylov subspace
Define the k-th order Krylov subspace generated by a square matrix $\mathbf{A}$ and a vector of appropriate size $\mathbf{r}$ by $\mathcal{K}_0(\mathbf{A}, \mathbf{r}) \equiv \left\{ \mathbf{0} \right\}$, and

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}) \equiv \mathbf{Span}([\mathbf{r}, \mathbf{A}\mathbf{r}, ..., \mathbf{A}^{k-1}\mathbf{r}]),$$

or equivalently,

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}) \equiv \left\{ \sum_{i=1}^{k} \alpha_i \mathbf{A}^{i-1}\mathbf{r} \;\middle|\; \vec{\alpha} \in \mathbb{C}^k \right\}.$$

By a **power basis** of $\mathcal{K}_k(\mathbf{A}, \mathbf{r})$ we shall mean the set consisting of the vectors $\left\{ \mathbf{r}, \mathbf{A}\mathbf{r}, \ldots, \mathbf{A}^{k-1}\mathbf{r} \right\}$. Note that this set strictly may not form a basis as commonly defined in linear algebra, for the vectors of the power basis might be dependent. For this reason, we also define an **Arnoldi basis** to be a power basis that has been orthnormalized using the Gram-Schmidt process.

We can generalize over definition 1.1 and define the block Krylov subspace:

Definition 1.2. Block Krylov Subspace
Let $\mathbf{A}$ and $\mathbf{R}$ be $N \times N$ and $N \times s$ sized matrices respectively. Then define the k-th order block Krylov subspace by $\mathcal{K}_0(\mathbf{A}, \mathbf{R}) \equiv \{\mathbf{0}\}$, and

$$\mathcal{K}_k(\mathbf{A}, \mathbf{R}) \equiv \Big\{ \sum_{i=1}^{k} \mathbf{A}^{i-1}\mathbf{R}\vec{\alpha_i} \;\Big|\; \vec{\alpha_i} \in \mathbb{C}^s \Big\}.$$

Note that for $s = 1$ the definitions of the block Krylov subspace and regular Krylov subspace coincide.

Definition 1.3. Condition number The condition number of a square, non-singular matrix, $\mathcal{C}(\mathbf{A})$ will be defined as

$$\mathcal{C}(\mathbf{A}) \equiv \|\mathbf{A}\|_2 \left\|\mathbf{A}^{-1}\right\|_2 .$$

The condition number of a matrix $\mathbf{A}$ is used as a measure for how sensitive a system $\mathbf{Ax} = \mathbf{b}$ is to perturbations. A condition number equal to 1 makes for very stable computations whereas high values for $\mathcal{C}(\mathbf{A})$ imply roughly that one should not expect a very accurate solution. For a brief introduction to this matter, see [18], and for a more elaborate work see [5]

Computational terminology

When studying the computational costs of iterative methods, it is common practice to express these in terms of the following terminology.

FLOP - Floating point operation. We do now discriminate between addition, subtraction, multiplication and division of scalars (floating point numbers).

Vector updates are a crucial part of iterative methods, and a computation of the form $\alpha\mathbf{x}+\mathbf{y}$ will be called an 'AXPY' (alpha $\mathbf{x}$ plus $\mathbf{y}$). One AXPY accounts for $2N$ FLOP.

A dot-product like $\mathbf{x}^*\mathbf{x}$ will be counted as a 'DOT'. Similarly, for an $N \times n$-matrix $\mathbf{V}$, $\mathbf{V}^*\mathbf{x}$ is equivalent to performing n DOT. One DOT accounts for $2N - 1$ FLOP.

Usually, the most expensive part of the iterative methods are the multiplications involving the input matrix $\mathbf{A}$ and some N-vector $\mathbf{v}$ or $N \times n$-matrix $\mathbf{V}$. We will count the former multiplication as one 'MV', implying that the latter should be seen as n MV. Since we are focusing on sparse systems, we differ from the usual $N^2 - N$ FLOP for one MV, and use $nz_{\mathbf{A}} \cdot N - N$ as an estimate. It should be noted though, that when the system to be solved is **preconditioned** with a matrix $\mathbf{B}$, this may no longer be accurate, as $\mathbf{BA}$ need not be sparse.

Tables containing the costs in terms of the above mentioned units are presented along with pseudo code of derived algorithms.

1.2 Computational notation

In order to emphasize and convey certain computational aspects of the derivations found in this work, we will employ a means of notation to distinguish between a mathematical equivalence and, what should be thought of as, a variable assignment in the context of an actual algorithm, involving quantities that are subject to rounding errors.

Variable assignment

When calculating a value, and assigning it to a variable (i.e. writing it to the memory of the computer), we will use the symbol '$\leftarrow\!\shortmid$'. As an example we could write

$$a \leftarrow\!\shortmid\, 3^2 + 7^2$$

when we mean to compute the right-hand side and assign it to a variable 'a'.
When we would like to emphasize that, for example, a equals $b^2 + c^2$ with $b = 3$ and $c = 7$, and we would like to compute it as such, we could write:

$$a = b^2 + c^2 \leftarrow\!\shortmid\, 3^2 + 7^2.$$

This way we can mix mathematical derivations with an practically implementable computation. Using available quantities stored in memory As research in iterative methods is mostly about achieving a higher efficiency, we would like to clearly distinguish, within calculations, between quantities that need to be computed, and quantities that are readily available. In order to do so, we will denote available vectors, matrices (or even products of these) by placing them inside a box, for example within a line expressing a variable assignment. That is, a 'boxed' term denotes a quantity stored in the computers memory, that has been obtained by actual computation in finite arithmetic. Note that in terms of memory, it does not matter whether we store a matrix $\mathbf{A}$, or its conjugate transposed, $\mathbf{A}^*$. For any calculations with $\mathbf{A}^*$ can be done by using the values of $\mathbf{A}$, albeit using some internal re-indexing which, in our case, is taken care of by Matlab. As an example, consider a situation in which we have three matrix-valued variables $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ of appropriate sizes, and we have available in memory the actual values of $\mathbf{C}$, $\mathbf{C}^2$ and $\mathbf{C}^3$. When we wish to assign the result of $\mathbf{C}^2 + \mathbf{C}^*$ to $\mathbf{A}$, and subsequently assign the result of $\mathbf{C}^3 + \mathbf{C}^2 + \mathbf{C}^*$ to $\mathbf{B}$. We could write

$$\mathbf{A} \leftarrow\!\shortmid\, \boxed{\mathbf{C}^2} + \boxed{\mathbf{C}^*}$$

$$\mathbf{B} = \mathbf{C}^3 + \mathbf{C}^2 + \mathbf{C}^* = \mathbf{C}^3 + \mathbf{A} \leftarrow\!\shortmid\, \boxed{\mathbf{C}^3} + \boxed{\mathbf{A}}$$

to emphasize the fact that there is no need to recalculate $\mathbf{C}^2 + \mathbf{C}^*$ in order to compute $\mathbf{B}$.

Inner products and MV's

When applying the iterative methods considered in this work, the bulk of the computational work in terms of FLOP can usually be largely attributed to the MV's (multiplication of an $N \times N$-matrix with an N-vector) required to yield a certain result. It is therefore that we will be focused in minimizing the required MV, since it is the goal to keep the computational costs as low as possible. Also, as the theoretical effectiveness of a method is easily judged by the number of MV required, we will, to avoid any confusion on this matter, distinguish between an actual MV, and a matrix-vector multiplication involving a matrix that is significantly

smaller in size than $N \times N$. We will see, for example, matrix-vector multiplications of the form $\mathbf{V}\vec{a}$, with $\mathbf{V}$ an $N \times s$-matrix with $s \ll N$, and $\vec{a}$ a vector of length s. When studying the costs of an algorithm, one does not want to confuse a multiplication of the latter type, which should rather be thought of as performing N dot-products, with an MV. Therefore we will distinguish between the two by using an '$*$' for a full (sparse[2]) MV, and a '$\cdot$' for a multiplication involving a significantly smaller matrix. Note that this notation also allows us to easily distinguish between the quantity $\mathbf{A}^{i+1}\mathbf{x}$ and its computation from, say, $\mathbf{A}$ and $\mathbf{A}^i\mathbf{x}$:

$$\mathbf{A}^{i+1}\mathbf{x} \leftarrow \boxed{\mathbf{A}} * \boxed{\mathbf{A}^i\mathbf{x}},$$

which is a natural computation to be done in the context of Krylov subspace methods.

1.3 Preliminaries on Krylov subspace methods

As IDR(s) and IDRStab can be, under certain conditions, seen as Krylov Subspace methods, it makes sense to do a short review of the basic concepts and associated terminology. Here we will only very briefly cover the essential concepts and terminology needed for the upcoming chapters. To obtain a thorough understanding of Krylov subspace methods we would like to refer to [23] and [11], which are both works dedicated to the subject containing many methods and their underlying theoretical foundations. To roughly define what a Krylov subspace method is, consider some iterative process that tries to solve $\mathbf{Ax} = \mathbf{b}$ step by step, generating **approximate solutions** $\mathbf{x}_k$ for increasing k with $\mathbf{x}_0$ being some initial guess. We define the k-th **residual**, $\mathbf{r}_k$ as

$$\mathbf{r}_k \equiv \mathbf{b} - \mathbf{A}\mathbf{x}_k.$$

The interest in Krylov subspaces in the context of solving linear systems is motivated by the following theorem and its corollaries. We will omit the proof.

Theorem 1.1. ***The Cayley-Hamilton Theorem*** *Let* $\mathbf{A} \in \mathbb{C}^{N\times N}$ *such that* $\mathbf{Det}(\mathbf{A}) \neq 0$ *(i.e.,* $\mathbf{A}$ *is non-singular), and let* $P(\xi)$ *be its characteristic polynomial, i.e.,*

$$P(\xi) \equiv \mathbf{Det}(\xi\mathbf{I} - \mathbf{A}).$$

Then $P(\mathbf{A}) = \mathbf{O}$.

Let $P(\xi)$ be a polynomial such that $P(\mathbf{A}) = \mathbf{O}$. From theorem 1.1 it can be seen that we can write

$$P(\mathbf{A}) = \sum_{i=0}^{N} \alpha_i \mathbf{A}^i = \mathbf{O},$$

or equivalently

$$\sum_{i=1}^{N} \alpha_i \mathbf{A}^i = -\alpha_0 \mathbf{I}.$$

[2]Concerning this matter, note that software packages like Matlab or BLAS enable the distinction between a regular MV and a sparse MV and provide corresponding data structures for sparse matrices, significantly cutting computational costs by disregarding multiplications with zero's, etc.

Assume that $\alpha_0 \neq 0$, (note that if this is not the case, one could use the polynomial $\xi^{-1}P(\xi)$ in the following observations.) multiplying both sides of the latter equation with $\mathbf{A}^{-1}$ and dividing by $-\alpha_0$ yields

$$\mathbf{A}^{-1} = \frac{1}{-\alpha_0} \sum_{i=1}^{N} \alpha_i \mathbf{A}^{i-1}.$$

In other words, we can express the inverse of $\mathbf{A}$ as a polynomial of degree $m \leq N-1$, evaluated in $\mathbf{A}$. And, in particular, we see that the exact solution, $\mathbf{x}$ of $\mathbf{Ax} = \mathbf{b}$ it must hold that $\mathbf{x} \in \mathcal{K}_{m+1}(\mathbf{A}, \mathbf{b})$.

For the moment assuming that the initial guess $\mathbf{x}_0$ equals the zero-vector, $\mathbf{0}$, Krylov methods generally construct $\mathbf{x}_k$ (with their corresponding residuals) such that

$$\mathbf{x}_k \in \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$$

This concept leaves a great amount of freedom as to what a method might precisely do in order to achieve this, and over the years many methods meant for just as many specialized purposes have been developed. Here, we will discuss the global workings of the *Generalized Minimal Residual* method, albeit without going into the technicalities, as we will refer to it as a benchmark in later chapters.

Example: The Generalized Minimal Residual method

In the Generalized Minimal Residual method, or GMRES for short, every k-th iteration, a basis for $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ is constructed. In fact, every, k-th step the readily available basis is expanded with one basis vector $\mathbf{v}_k$ that is obtained by multiplying $\mathbf{A}$ with the last constructed basis vector $\mathbf{v}_{k-1}$, resulting in $\tilde{\mathbf{v}}_k$. This is followed by an orthonormalization of $\tilde{\mathbf{v}}_k$ against all previously obtained basis vectors $\mathbf{v}_i$ with $i \leq k$, the result of which forms $\mathbf{v}_k$. GMRES differs a bit from most other Krylov subspace methods in that it does not explicitly maintains a residual vector $\mathbf{r}_k$, and neither does it maintain an approximate solution $\mathbf{x}_k$. However, by clever construction, it does calculate the **residual norm** $\|\mathbf{r}_k\|_2$, based on which (comparison with some given tolerance) it may construct the actual approximation $\mathbf{x}_k$.What sets GMRES apart from most practically used methods is firstly the fact that it maintains a **full** orthonormal Krylov basis. This is costly in terms of memory usage and since every, in every step, a new basis vector must be orthogonalized against all previous ones, the amount of FLOP needed per iteration grows every step. The advantage of GMRES is, however, that the maintenance of the full orthonormal basis enables the method to accurately minimize the residual norm with respect to the full space $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. Hence yielding an 'optimal' method in terms of residual norm in each step. It therefore also acts as a lower bound for all Krylov subspace solvers in terms of how small the residual norm might be after a given number of used MV. It is this property that makes it useful as a bench-marking method.

A short history of iterative Krylov subspace methods

The practical downside of GMRES are already mentioned; The memory and computational requirements increase with every iteration. Therefore, so called **short recurrence methods** have gained most interest for practical applications, 'short recurrence' meaning that in every iteration, the recurrence relations for constructing a $\mathbf{r}_k$ and $\mathbf{x}_k$ consist of a (preferably small,) fixed number of terms. A widely used short recurrence method for systems in which

A is (preferably) positive semi-definite (or at least hermitian) is Conjugate Gradient (CG), in which recurrences of length two are used:

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{v}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{v}_k$$

$$\mathbf{v}_{k+1} = \mathbf{r}_k + \beta_k \mathbf{v}_{k+1}$$

for some (cleverly chosen) scalars α_k, β_k and some initial conditions. The catch of CG is that one must make (symmetry) assumptions on the properties of **A** in order to prevent the method from breaking down[3]. Later, Bi-CG was introduced as a short recurrence method that does not need such properties on **A**, but instead performs explicit calculations with $\mathbf{A}^*$, which might very well not be available, or costly to use. Further research led to CGS (CG Squared) that does not need multiplications with $\mathbf{A}^*$, making it even more universally applicable. In CGS, the explicit multiplication with $\mathbf{A}^*$ is made redundant by cleverly squaring a polynomial expression. It is therefore said to be **transpose free**, but suffers from possibly very irregular convergence. Continuing in this line of work, Bi-CGStab emerged, which is, generally applicable, more stable than CGS and transpose free whilst still being a short recurrence method. Since short recurrence methods can not minimize their residuals over an entire Krylov basis, concessions need to be made on the speed of convergence. It is therefore that one of the major goals in research to iterative solvers is to find a method having all the nice properties like Bi-CGStab, but with the convergence (per MV) as much as possible as GMRES. In 1984, Faber&Mantueffel ([2]) proved that there can not be a short recurrence method for general **A**.

Orthogonality Conditions

Generally speaking, the idea of Krylov subspace methods is to construct a basis of $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ and extract an approximation $\mathbf{x}_k$ that has favorable properties. For example, GMRES constructs $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$, and extracts $\mathbf{x}_k$ from $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ such that $\|\mathbf{r}_k\|_2$ is as small as possible with respect to this basis. Note that this amounts to $\mathbf{r}_k$ having the following property:

$$\mathbf{r}_k \perp \mathbf{A}\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0).$$

Hence, orthogonality conditions arise in a natural way from optimality conditions on the residuals or approximations. Iterative methods can, based on the properties of the residuals they construct, roughly be divided in a few classes. GMRES is part of the **Petrov-Galerkin** class of methods, in which a **test space**, $\mathcal{L}$ (the space to which the residuals should be orthogonal), is used that is not equal to the **search space** (the space from which the vector updates for $\mathbf{x}_k$ are taken). For GMRES we have $\mathcal{L} = \mathbf{A}\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. We will see that IDR-methods are also of the Petrov-Galerkin class.

[3]More specific, formally **A** would need to be positive semi-definite in order to guarantee an implicit LU-decomposition from breaking down. In practice, a stable implementation may be successfully used on **A** that is merely symmetric.

Part 1: IDR Methods

Chapter 2

IDR(s)

2.1 Basic Definitions and Framework

In this section we will discuss the theoretical foundations needed for the derivation of actual IDR methods. The core of this foundation consists of what we will call the IDR Theorem (as defined in [19]), which implicitly states a means of creating a sequence of subspaces, ever shrinking in terms of inclusion, and thus, dimension.

Theorem 2.1. ***The IDR Theorem***
Let $\mathbf{A} \in \mathbb{C}^{N\times N}$ a matrix, $\mathcal{G}_0 \subseteq \mathbb{C}^N$ a subspace that is invariant under multiplication by $\mathbf{A}$, $(\omega)_{k\in\mathbb{N}_+} \in \mathbb{C}^{\mathbb{N}}$ a sequence of non-zero scalars and $\mathcal{R}^\perp = \mathbf{Span}(\mathbf{R})^\perp$ for some $\mathbf{R} \in \mathbb{C}^{N\times s}$ for some s. Define a sequence of subspaces recursively by

$$\mathcal{G}_k^{'} \equiv \mathcal{G}_k \cap \mathcal{R}^\perp, \quad \mathcal{G}_{k+1} \equiv (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathcal{G}_k^{'},$$

then we have

$$\mathcal{G}_{k+1} \subseteq \mathcal{G}_k \quad \textit{and} \quad \mathbf{A}\mathcal{G}_k^{'} \subseteq \mathcal{G}_k.$$

If, additionally, we have that $\mathcal{R}^\perp$ does not contain a non-trivial invariant subspace of $\mathbf{A}$, we have

$$\mathcal{G}_{k+1} = \mathcal{G}_k \Leftrightarrow \mathcal{G}_k = \{\mathbf{0}\}.$$

Proof. We first show that $\mathcal{G}_{k+1} \subset \mathcal{G}_k$ by induction. First, note that by assumption we have $\mathbf{A}\mathcal{G}_0 \subset \mathcal{G}_0$, hence $\mathbf{A}\mathcal{G}_0^{'} \subset \mathcal{G}_0$, and thus for any scalar ω_0 we have $\mathcal{G}_1 \subset \mathcal{G}_0$. Next, assume that we have $\mathcal{G}_k \subset \mathcal{G}_{k-1}$ for some k, then if $\mathbf{x} \in \mathcal{G}_{k+1}$ then by definition we have $\mathbf{x} = (\mathbf{I}-\omega_{k+1}\mathbf{A})\mathbf{y}$ for some $\mathbf{y} \in \mathcal{G}_k^{'} = \mathcal{G}_k \cap \mathcal{R}^\perp$. Since $\mathcal{G}_k \subset \mathcal{G}_{k-1}$ by our induction hypothesis, it follows that $\mathbf{y} \in \mathcal{G}_{k-1} \cap \mathcal{R}^\perp$, hence we have that $(\mathbf{I} - \omega_k\mathbf{A})\mathbf{y} \in \mathcal{G}_k$. This implies that $\mathbf{A}\mathbf{y} \in \mathcal{G}_k$, and thus $\omega_{k+1}\mathbf{A}\mathbf{y} \in \mathcal{G}_k$. Therefore we can conclude that $\mathbf{x} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{y} \in \mathcal{G}_k$. So $\mathcal{G}_{k+1} \subseteq \mathcal{G}_k$.

Note that $\mathcal{G}_{k+1} \subset \mathcal{G}_k$ also implies the second result: Let $\mathbf{y} \in \mathcal{G}_k^{'}$. Then, since $\mathcal{G}_{k+1} \subseteq \mathcal{G}_k$, we have $(\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{y} \in \mathcal{G}_k$, hence $\mathbf{A}\mathbf{y} \in \mathcal{G}_k$.

Let us now assume that $\mathcal{R}^\perp$ does not contain an non-trivial invariant space under multiplication with $\mathbf{A}$, and that we have $\mathcal{G}_{k+1} = \mathcal{G}_k$ for some k. Note that since $\mathbf{Dim}(\mathcal{G}_{k+1}) = \mathbf{Dim}(\mathcal{G}_k)$, we must have that $\mathcal{G}_k \cap \mathcal{R}^\perp = \mathcal{G}_k$, and hence that $\mathcal{G}_k \subseteq \mathcal{R}^\perp$. Because

$$\mathcal{G}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})(\mathcal{G}_k \cap \mathcal{R}^\perp) = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathcal{G}_k = \mathcal{G}_k,$$

we must have that $\mathbf{A}\mathcal{G}_k \subseteq \mathcal{G}_k$, which, by our assumption, can only hold if $\mathcal{G}_k = \{\mathbf{0}\}$. □

We will call the matrix $\mathbf{R}$ the **test matrix** (and its columns the **test vectors**). For completeness, we include the following statement on the reduced dimensions when transitioning from $\mathcal{G}_k$ to $\mathcal{G}_{k+1}$ which we will, in line with [21], call the 'Extended IDR Theorem'.

Theorem 2.2. ***Extended IDR Theorem***
Define $d_k \equiv \mathbf{Dim}(\mathcal{G}_k)$ *and assume* $(\mathbf{I} - \omega_k\mathbf{A})$ *to be non-singular. Then, for a choice of* s *and* $\mathbf{R}$ *as defined above, with* $\mathbf{Rank}(\mathbf{R}) = s$*, the sequence of* $d_0, d_1, d_2, ...$ *is monotonically non-decreasing and satisfies*

$$0 \leq d_k - d_{k+1} \leq d_{k-1} - d_k \leq s.$$

Proof. Let, again, for any k, $\mathcal{G}_k^{'} \equiv \mathcal{G}_k \cap \mathcal{R}^{\perp}$, and let $\mathbf{G}_k$ be a matrix whose columns form a basis for $\mathcal{G}_k$. Let $\mathbf{x} \in \mathcal{G}_{k-1}$ and note that we have $\mathbf{x} = \mathbf{G}_{k-1}\mathbf{v}$ for some $\mathbf{v}$. Now assume that also $\mathbf{x} \in \mathcal{G}_{k-1}^{'}$, implying that we have that $\mathbf{R}^*\mathbf{G}_{k-1}\mathbf{v} = \mathbf{0}$. We can therefore write

$$\mathcal{G}_{k-1}^{'} = \mathbf{G}_{k-1}(\mathcal{N}(\mathbf{R}^*\mathcal{G}_{k-1})),$$

and hence

$$\mathcal{G}_k = (\mathbf{I} - \omega_{k-1}\mathbf{A})\mathbf{G}_{k-1}(\mathcal{N}(\mathbf{R}^*\mathcal{G}_{k-1})).$$

Observe that, since we assumed the non-singularity of $(\mathbf{I} - \omega_{k-1}\mathbf{A})$, we have that

$$d_k = \mathbf{Dim}(\mathcal{G}_k) = \mathbf{Dim}(\mathcal{G}_{k-1}^{'}).$$

Since $\mathbf{R}^*\mathbf{G}_{k-1}$ is an $s \times d_{k-1}$-matrix, it follows that

$$d_k = \mathbf{Dim}(\mathcal{N}(\mathbf{R}^*\mathbf{G}_{k-1})) = d_{k-1} - \mathbf{Rank}(\mathbf{R}^*\mathbf{G}_{k-1}).$$

Note that we also have $\mathbf{Rank}(\mathbf{R}^*\mathbf{G}_{k-1}) = s - \mathbf{Dim}(\mathcal{N}(\mathbf{G}_{k-1\mathbf{R}}^*))$.
Next, set $j = \mathbf{Dim}(\mathcal{N}(\mathbf{G}_{k-1}^*\mathbf{R}))$. We then have

$$d_k = d_{k-1} - \mathbf{Rank}(\mathbf{R}^*\mathbf{G}_{k-1}) = d_{k-1} - s + j \quad \text{with} \quad 0 \leq j \leq s,$$

which we can equally state as

$$0 = d_{k-1} - d_k - s + j,$$

hence

$$0 \leq d_{k-1} - d_k \leq s.$$

Next, suppose that $\mathbf{v} \in \mathcal{N}(\mathbf{G}_{k-1}^*\mathbf{R})$ and $v \neq \mathbf{0}$, then $\mathbf{R}\mathbf{v} \in \mathcal{N}(\mathcal{G}_{k-1}^*)$ and hence $\mathbf{R}\mathbf{v} \perp \mathcal{G}_{k-1}$. Since $\mathcal{G}_k \subset \mathcal{G}_{k-1}$, we also have $\mathbf{R}\mathbf{v} \perp \mathcal{G}_k$, and hence $\mathbf{v} \in \mathcal{N}(\mathbf{G}_k^*\mathbf{R})$. Therefore, $\mathcal{N}(\mathbf{G}_{k-1}^*\mathbf{R}) \subset \mathcal{N}(\mathbf{G}_k^*\mathbf{R})$. This yields us

$$\mathbf{Dim}(\mathbf{G}_{k+1}^*\mathbf{R}) \leq \mathbf{Dim}(\mathcal{N}(\mathbf{G}_k^*\mathbf{R})),$$

from which it follows that, for $j' = \mathbf{Dim}(\mathcal{N}(\mathbf{G}_k^*\mathbf{R}))$,

$$d_{k+1} = d_k - s + j' \quad \text{with} \quad j \leq j' \leq s.$$

Therefore, we can indeed conclude that

$$0 \leq d_k - d_{k+1} \leq d_{k-1} - d_k \leq s.$$

□

The Extended IDR Theorem, in conjunction with the IDR Theorem, basically states that the dimension of $\mathcal{G}_{k+1}$ will always be at least one smaller than the dimension of $\mathcal{G}_k$, unless they are both equal to $\{\mathbf{0}\}$. However, in practice, a 'dimension reduction' of less than s is very rare. In a paper on the convergence of IDR methods by Sonneveld [20] it is in fact proven that, for mild conditions on $\mathbf{R}$, the probability of such an event equals zero. Therefore, we may in general expect that the number of subsequent IDR spaces we ought to generate before arriving at the zero-space is $\lceil \frac{d_0}{s} \rceil$, with $d_0 = \mathbf{Dim}(\mathcal{G}_0)$.

Next, we introduce the concept of Sonneveld Subspaces as introduced in [16], which, in the context of IDR-methods seems a more natural structure to work with than Krylov subspaces.

Definition 2.1. Sonneveld subspace
Given a k-th order polynomial $P_k(\xi)$, an $N \times N$ matrix $\mathbf{A}$, a $N \times s$ matrix $\mathbf{R}$ and $\mathcal{G}_0$ as in the IDR theorem. Define the **Sonneveld subspace** generated hereby as

$$\mathcal{S}(P_k, \mathbf{A}, \mathbf{R}) \equiv \{P_k(\mathbf{A})\mathbf{v} \mid \mathbf{v} \in \mathcal{G}_0, \mathbf{v} \perp \mathcal{K}_k(\mathbf{A}^*, \mathbf{R}))\}.$$

The following lemma states that the spaces mentioned in the IDR theorem are in fact Sonneveld subspaces.

Lemma 2.1. *Let $\mathbf{A}$, $\mathcal{S}$, $\mathcal{G}_0$ and $\omega \in \mathbb{C}^{\mathbb{N}}$ be as described in the IDR theorem. Then, the polynomial $P_k(\xi) \equiv \prod_{i=1}^{k}(1 - \omega_i \xi)$, we have*

$$\mathcal{G}_k = \mathcal{S}(P_k, \mathbf{A}, \mathbf{R}).$$

Proof. We proof the statement by induction. First note that we, indeed, have for $P_0(\xi) = \alpha_0$

$$\mathcal{G}_0 = \{\alpha_0 \mathbf{v} \mid \mathbf{v} \in \mathcal{G}_0 \wedge \mathbf{v} \perp \{\mathbf{0}\}\} = \mathcal{S}(P_0, \mathbf{A}, \mathbf{R}).$$

Now assume that the statement holds for a $k \in \mathbb{N}$, then for $k+1$ we have

$$\mathcal{G}_{k+1} = \{(\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{y} \mid \mathbf{y} \in (\mathcal{G}_k \cap \mathcal{R}^{\perp})\},$$

which by our induction hypothesis equals

$$\mathcal{G}_{k+1} = \{(\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{y} \mid \mathbf{y} \in (\mathcal{S}(P_k, \mathbf{A}, \mathbf{R}) \cap \mathcal{R}^{\perp})\}.$$

Using the definition of the Sonneveld subspace, we obtain, for

$$P_k(\mathbf{A})\mathbf{v} \perp \mathcal{R}^{\perp} \Rightarrow \mathbf{v} \perp \mathcal{K}_{k+1}(\mathbf{A}^*, \mathbf{R})$$

(To see this, note that we can write $P_k(\mathbf{A}) \sum_{i=1}^{k} \alpha_i \mathbf{A}^i$, and moreover
that $\mathbf{v} \perp \mathcal{K}_k(\mathbf{A}^*, \mathbf{R}) \forall_{i=0,\ldots,k-1} \mathbf{R}^* \mathbf{A}^i \mathbf{v} = 0$)
Hence,

$$\mathcal{G}_{k+1} = \{(\mathbf{I} - \omega_{k+1}\mathbf{A})P_k(\mathbf{A})\mathbf{v} \mid \mathbf{v} \in \mathcal{K}_N(\mathbf{A}, \mathbf{R}), \mathbf{v} \perp \mathcal{K}_{k+1}(\mathbf{A}^*, \mathbf{R})\}.$$

Hence

$$\mathcal{G}_{k+1} = \{P_{k+1}(\mathbf{A})\mathbf{v} \mid \mathbf{v} \in \mathcal{K}_N(\mathbf{A}^*, \mathbf{R}), \mathbf{v} \perp \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{R})\} = \mathcal{S}(P_{k+1}, \mathbf{A}, \mathbf{R})$$

as desired. This concludes the proof. □

In IDR methods, the idea is to generate residuals $\mathbf{r}_k$ in the shrinking sequence of $\mathcal{G}_k$ rather than in a growing sequence of Krylov subspaces of the form $\mathcal{K}_k(\mathbf{A}, \mathbf{R})$, as is the case for most Krylov subspace type methods. An important observation to be made at this point is that we have already proven the termination at a finite number of iterations of such a method by the IDR- and Extended IDR Theorem (albeit in exact arithmetic). Hence, we can now focus on constructing these residuals.

Given a residual $\mathbf{r}_k \in \mathcal{G}_k$, the means to construct $\mathbf{r}_{k+1} \in \mathcal{G}_{k+1}$ will loosely be structured as in the definition of $\mathcal{G}_{k+1}$ in the IDR theorem. That is, we will first construct

$$\mathbf{r}'_k \in \mathcal{G}_k \cap \mathcal{R}^{\perp},$$

and subsequently 'lift' it into the next Sonneveld subspace by using

$$\mathbf{r}_{k+1} \equiv (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{r}'_k \in \mathcal{G}_{k+1}.$$

For this approach, we will need a way of constructing $\mathbf{r}'_k$. Note that simply using orthogonal projections like

$$\mathbf{r}'_k = (\mathbf{I} - \mathbf{R}(\mathbf{R}^*\mathbf{R})^{-1}\mathbf{R}^*)\mathbf{r}_k$$

such that

$$\mathbf{R}^*\mathbf{r}'_k = \mathbf{R}^*\mathbf{r}_k - \mathbf{R}^*\mathbf{R}(\mathbf{R}^*\mathbf{R})^{-1}\mathbf{R}^*\mathbf{r}_k = \mathbf{0}$$

won't do the trick, for it would need that $\mathbf{R} \in \mathcal{G}_k$, which might, by the IDR theorem, ruin[1] the shrinking property of the IDR spaces, $\mathcal{G}_k$.

Therefore, we will use a set of vectors in $\mathcal{G}_k$ other than $\mathbf{r}_k$, that will lead us to the following oblique projection. Assume we have $\mathbf{U}_k \in \mathbb{C}^{N\times s}$, such that $\mathbf{A}\mathbf{U}_k \in \mathcal{G}_k$. Then

$$\mathbf{r}'_k \equiv (\mathbf{I} - \mathbf{A}\mathbf{U}_k(\mathbf{R}^*\mathbf{A}\mathbf{U}_k)^{-1}\mathbf{R}^*)\mathbf{r}_k \in \mathcal{G}_k \cap \mathcal{R}^{\perp}.$$

In line with common literature on IDR-methods, we will refer to the columns of $\mathbf{U}_k$ as a set of **auxiliary vectors**, and as **search matrix** to the matrix as a whole. As we will make extensive use of projections like these in IDR-methods, we define

$$\sigma \equiv \mathbf{R}^*\mathbf{A}\mathbf{U}_k,$$

and

$$\mathbf{\Pi}_1 \equiv \mathbf{I} - \mathbf{A}\mathbf{U}_k\sigma^{-1}\mathbf{R}^* \quad \text{, and} \quad \mathbf{\Pi}_0 \equiv \mathbf{I} - \mathbf{U}_k\sigma^{-1}\mathbf{R}^*\mathbf{A}.$$

the latter of which will find its use later in this chapter. Note that the three of these (σ, Π_1, Π_0) are dependant on the iteration number k. We have chosen not to index the projections, nor σ, with this parameter for readability reasons, as there will be a need for more indexing in a later chapter. Also note that σ will need to be non-singular, we will discuss this aspect later (see Section 3.1) in this work in the context of numerical stability.

The following properties of $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_0$ can be easily verified by the reader.

$$\mathbf{R}^*\mathbf{\Pi}_1 = \mathbf{0} \ \ (\ \text{or } \mathbf{Span}(\mathbf{\Pi}_1)\perp\mathbf{Span}(\mathbf{R})\ ), \text{and} \quad \mathbf{\Pi}_1\mathbf{A} = \mathbf{A}\mathbf{\Pi}_0 \tag{2.1}$$

$$\forall \mathbf{s} \in \mathcal{G}_k : \ \ \mathbf{\Pi}_1\mathbf{s} \in \mathcal{G}_k \cap \mathcal{R}^{\perp}. \tag{2.2}$$

[1] To see this, note that if for some k we have $\mathbf{R} \in \mathcal{G}_k$, then $\mathbf{R}\perp\mathcal{G}'_k = \mathcal{G}_k \cap \mathcal{R}^{\perp}$. In order to have $\mathbf{Dim}(\mathcal{G}_{k+1}) = \mathbf{Dim}(\mathcal{G}_k) - s$ we would need that $\mathbf{R} \in \mathcal{G}_{k+1}$. This would require the space spanned by $\mathbf{R}$ to be reintroduced in the step $\mathcal{G}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathcal{G}'_k$, which is very unlikely.

Also note that the way in which we defined $\mathbf{r}'_k$, $\mathbf{r}_{k+1}$ and $\mathbf{\Pi}_1$, also makes for an easy way to update $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$: let

$$\mathbf{x}'_k \equiv \mathbf{x}_k + \mathbf{U}_k(\mathbf{R}^*\mathbf{A}\mathbf{U}_k)^{-1}\mathbf{R}^*\mathbf{r}_k,$$

and

$$\mathbf{x}_{k+1} \equiv \mathbf{x}'_k + \omega_{k+1}\mathbf{r}'_k.$$

Note that we indeed have as desired: $\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1}$ (also implying that $\mathbf{r}'_k = \mathbf{b} - \mathbf{A}\mathbf{x}'_k$). Since, if we would like to repeat this process in order to construct $\mathbf{r}_{k+2} \in \mathcal{G}_{k+2}$, we will be needing a new $\mathbf{U}_{k+1} \in \mathcal{G}_{k+1}$. The idea will be to construct the s columns of $\mathbf{U}_{k+1}$ based on the just obtained $\mathbf{r}'_k$ that lies in $\mathcal{G}_k \cap \mathcal{R}^\perp$.
We first note that multiplying a vector $\mathbf{s} \in \mathcal{G}_k \cap \mathcal{R}^\perp$ with $\mathbf{\Pi}_1\mathbf{A}$ yields

$$\mathbf{\Pi}_1\mathbf{A}\mathbf{s} \in \mathcal{G}_k \cap \mathcal{R}^\perp, \tag{2.3}$$

by the **IDR theorem** and **Property 2.2**.
We will use this knowledge to first construct a matrix $\mathbf{V}_k$ such that if we put $\mathbf{U}_{k+1} \equiv (\mathbf{I} - \omega_k\mathbf{A})\mathbf{V}_k$, we have that $\mathbf{A}\mathbf{U}_{k+1} \in \mathcal{G}_{k+1}$, as desired. (Note that we will be needing the explicit availability of both $\mathbf{U}_{k+1}$ and $\mathbf{A}\mathbf{U}_{k+1}$ in order to project $\mathbf{r}_{k+1}$ and update $\mathbf{x}_{k+1}$ in the same manner as above.)
To this end, we let $\mathbf{V}_k$ be a power basis of $\mathcal{K}_s(\mathbf{\Pi}_0\mathbf{A}, \mathbf{\Pi}_0\mathbf{r}'_k)$:

$$\mathbf{V}_k \equiv [\mathbf{\Pi}_0\mathbf{r}'_k, ..., (\mathbf{\Pi}_0\mathbf{A})^{s-1}\mathbf{\Pi}_0\mathbf{r}'_k].$$

Observe that, applying **Property 2.1** and **2.3**

$$\mathbf{A}\mathbf{V}_k = \mathbf{A}[\mathbf{\Pi}_0\mathbf{r}'_k, ..., (\mathbf{\Pi}_0\mathbf{A})^{s-1}\mathbf{\Pi}_0\mathbf{r}'_k] = [\mathbf{\Pi}_1\mathbf{A}\mathbf{r}'_k, ..., (\mathbf{\Pi}_1\mathbf{A})^s\mathbf{r}'_k] \in \mathcal{G}_k \cap \mathcal{R}^\perp,$$

such that, indeed,

$$\mathbf{A}\mathbf{U}_{k+1} = \mathbf{A}(\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{V}_k = (\mathbf{I} - \omega_k\mathbf{A})\mathbf{A}\mathbf{V}_k \in \mathcal{G}_{k+1}.$$

With the main definitions of this chapter in place, we can now introduce terminology for the transition from the quantities $\mathbf{r}_k, \mathbf{A}\mathbf{U}_k \in \mathcal{G}_k$ and $\mathbf{x}_k$ to $\mathbf{r}_{k+1}, \mathbf{A}\mathbf{U}_{k+1} \in \mathcal{G}_{k+1}$ with $\mathbf{x}_{k+1}$, which we will call an **IDR cycle**. Within this cycle we will further distinguish two separate phases: The **dimension reduction step**, in which we construct $\mathbf{r}'_k, \mathbf{x}'_k, \mathbf{V}_k$ and $\mathbf{A}\mathbf{V}_k$, and the **polynomial step**, in which we lift our quantities into $\mathcal{G}_{k+1}$ by multiplication with $(\mathbf{I} - \omega_k\mathbf{A})$ and update $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ accordingly. Note that this last step requires the determination of the scalar ω_k, which will be our final point of attention before continuing to the efficient computation of the IDR cycle.

In the original paper describing IDR(s) ([21]), the ω_k are chosen to minimize the norm of the next residual, similar to BiCGSTAB ([24]), i.e. given $\mathbf{r}'_k$

$$\omega_{k+1} \equiv \arg\min_{\omega_{k+1}}(\|\mathbf{r}_{k+1}\|_2) = \arg\min_{\omega_{k+1}}(\|\mathbf{r}'_k - \omega_{k+1}\mathbf{A}\mathbf{r}'_k\|_2).$$

Note that this amounts to putting $\mathbf{r}_{k+1} \perp \mathbf{A}\mathbf{r}'_k$, and hence we can determine ω_{k+1} by using the same technique as we would in case of an orthogonal projection:

$$\omega_{k+1} \equiv \frac{(\mathbf{A}\mathbf{r}'_k)^*\mathbf{r}'_k}{(\mathbf{A}\mathbf{r}'_k)^*(\mathbf{A}\mathbf{r}'_k)}.$$

2.2 Efficient computation of an IDR cycle

We will break down the computational aspects of the IDR cycle, in to the previously discussed separate steps.

Computation of the Dimension Reduction step

In this section we assume the iterative method to have completed the k-th IDR cycle for some $k \geq 0$, thus having available the vectors $\mathbf{r}_k, \mathbf{x}_k$ and the matrices $\mathbf{U}_k$ and $\mathbf{AU}_k$. We will here describe how to efficiently compute $\mathbf{r}_{k+1}, \mathbf{x}_{k+1}$ and $\mathbf{U}_{k+1}$, $\mathbf{AU}_{k+1}$.

As discussed in previous section, we will first compute $\mathbf{r}'_k$ using $\mathbf{\Pi}_1$. Since we do not have $\mathbf{\Pi}_1$ available, we need to compute it (or actually its missing parts), by computing

$$\sigma = \mathbf{R}^*\mathbf{AU}_k \leftarrow\!\shortmid \boxed{\mathbf{R}^*} \cdot \boxed{\mathbf{AU}_k},$$

which allows us to compute

$$\alpha \equiv \sigma^{-1}\mathbf{R}^*\mathbf{r}_k \leftarrow\!\shortmid \boxed{\sigma}^{-1}\boxed{\mathbf{R}^*} \cdot \boxed{\mathbf{r}_k}.$$

Subsequently we can compute

$$\mathbf{r}'_k = \mathbf{\Pi}_1\mathbf{r}_k \leftarrow\!\shortmid \boxed{\mathbf{r}_k} - \boxed{\mathbf{AU}_k} \cdot \boxed{\alpha}$$

$$\mathbf{x}'_k \leftarrow\!\shortmid \boxed{\mathbf{x}_k} + \boxed{\mathbf{U}_k} \cdot \boxed{\alpha}.$$

Next we would like to compute $\mathbf{V}_k$, for which we will proceed per column $q \leq s$. Since we defined $\mathbf{V}_k$ as a power basis of $\mathcal{K}_s(\mathbf{\Pi}_0\mathbf{A}, \mathbf{\Pi}_0\mathbf{r}'_k)$, we assign first

$$\mathbf{Ar}'_k \leftarrow\!\shortmid \boxed{\mathbf{A}} * \boxed{\mathbf{r}'_k}$$

(Using a full matrix-vector multiplication) which we will use to compute

$$\beta = \sigma^{-1}\mathbf{R}^*\mathbf{Ar}'_k \leftarrow\!\shortmid \boxed{\sigma}^{-1}\boxed{\mathbf{R}^*} \cdot \boxed{\mathbf{Ar}'_k}.$$

Then the first columns of $\mathbf{V}_k$ and $\mathbf{AV}_k$ will then be computable as

$$\mathbf{V}_k e_1 \leftarrow\!\shortmid \boxed{\mathbf{r}'_k} - \boxed{\mathbf{U}_k} \cdot \boxed{\beta}$$

$$\mathbf{AV}_k e_1 \leftarrow\!\shortmid \boxed{\mathbf{Ar}'_k} - \boxed{\mathbf{AU}_k} \cdot \boxed{\beta}.$$

We can now, using a for-loop, compute for $q = 2, ..., s$

$$\mathbf{A}^2\mathbf{V}_k e_{q-1} \leftarrow\!\shortmid \boxed{\mathbf{A}} * \boxed{\mathbf{AV}_k e_{q-1}}$$

$$\beta = \sigma^{-1}\mathbf{R}^*\mathbf{A}^2\mathbf{V}_k e_{q-1} \leftarrow\!\shortmid \boxed{\sigma}^{-1}\boxed{\mathbf{R}^*} \cdot \boxed{\mathbf{A}^2\mathbf{V}_k e_{q-1}}$$

$$\mathbf{V}_k e_q = \mathbf{\Pi}_0\mathbf{AV}_k e_{q-1} \leftarrow\!\shortmid \boxed{\mathbf{AV}_k e_{q-1}} - \boxed{\mathbf{U}_k} \cdot \boxed{\beta}$$

$$\mathbf{AV}_k e_q = \mathbf{A}\mathbf{\Pi}_0\mathbf{AV}_k e_{q-1} \leftarrow\!\shortmid \boxed{\mathbf{A}^2\mathbf{V}_k e_{q-1}} - \boxed{\mathbf{AU}_k} \cdot \boxed{\beta}$$

The above is a complete description of the dimension reduction step, in which we have used a total of $s + 1$ MV's.

Computation of the Polynomial Step

With the dimension reduction step completed, we have available $\mathbf{r}'_k, \mathbf{A}\mathbf{r}'_k, \mathbf{x}'_k, \mathbf{V}_k, \mathbf{A}\mathbf{V}_k$ and $\mathbf{A}^2\mathbf{V}_k$. As mentioned in previous section, we can compute ω_{k+1} as the scalar that minimizes $\|\mathbf{r}_{k+1}\|_2$. We then want to compute:

$$\mathbf{r}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{r}'_k \hookleftarrow \boxed{\mathbf{r}'_k} - \boxed{\omega_k} \cdot \boxed{\mathbf{A}\mathbf{r}'_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}'_k + \omega_{k+1}\mathbf{r}'_k \hookleftarrow \boxed{\mathbf{x}'_k} + \boxed{\omega_k} \cdot \boxed{\mathbf{r}'_k}$$

$$\mathbf{U}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{V}_k \hookleftarrow \boxed{\mathbf{V}_k} - \boxed{\omega_k} \cdot \boxed{\mathbf{A}\mathbf{V}_k}$$

$$\mathbf{A}\mathbf{U}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{A}\mathbf{V}_k \hookleftarrow \boxed{\mathbf{A}\mathbf{V}_k} - \boxed{\omega_k} \cdot \boxed{\mathbf{A}^2\mathbf{V}_k}$$

Note how the availability of $\mathbf{A}^2\mathbf{V}_k$ allows for completing the step by mere use of vector updates and no matrix-vector products.

Termination

Note that in the above scheme we have used only $s+1$ MV's per IDR cycle. In conjunction with previously made comment on the convergence of the IDR spaces to $\{\mathbf{0}\}$ in at most $\frac{\mathbf{Dim}(\mathcal{G}_0)}{s}$steps, we have, assuming that $\mathbf{Dim}(\mathcal{G}_0) = N$, that in exact arithmetic the above described IDR method's residual $\mathbf{r}$ equals zero after at most

$$\frac{\mathbf{Dim}(\mathcal{G}_0)}{s}(s+1) = \frac{s+1}{s}N \quad \text{MV}.$$

We will see, for example in Figure 3.1 (Section 3.2), that this upper-bound is usually rather pessimistic in practice. Similar to, for example, GMRES, theoretically needing N iterations (and thus MV) for a general $\mathbf{A}$ in order to terminate, this upper-bound is sharp only on rare occasions. In many cases in fact, the excellent performance of Krylov Subspace methods is not well understood.

Chapter 3

Implementation and Stability

The algorithm presented in previous chapter is merely an illustration of how to globally implement an IDR method as described in the theory and is often not suited for practical use yet. It is already numerically unstable for $s > 3$ and there are still choices to make regarding the first set of auxiliary vectors. In this section we discuss several different aspects of the implementation that influence the efficiency of the method and fix what we will be using in most experiments. We will refer to this version as **standard IDR(s)**[1].

3.1 Design Choices

Enhancing stability

The method as described in previous chapter will loose numerical accuracy quickly for $s > 1$ because σ will usually become ill-conditioned or even singular. This is due to the fact that we are constructing $\mathbf{V}_k$ as a power basis of $\mathcal{K}_s(\Pi_0\mathbf{A}, \Pi_0\mathbf{r}'_k)$. As is the case with all power bases, repeated multiplication with the same linear operator (in this case $\Pi_0\mathbf{A}$) results in vectors that increasingly point in the direction of the dominant eigenvector, exactly as exploited in the power method for finding such eigenvectors. The result of which is a set of vectors that is increasingly linearly dependant, implying that the solving of the $s \times s$ systems

$$\sigma\vec{\alpha} = \mathbf{R}^*\mathbf{r}_k \quad \text{and} \quad \sigma\vec{\beta} = \mathbf{R}^*\mathbf{A}\mathbf{r}'_k$$

for $\vec{\alpha}$ and $\vec{\beta}$ will be unstable, introducing errors in the residual and auxiliary vectors.
There are several ways known for dealing with this problem, the easiest of which being the use of an orthonormalization scheme for $\mathbf{V}_k$. Note that if we have $\mathcal{C}(\mathbf{V}_k) = 1$, we will also have bounded $\mathcal{C}(\sigma) = \mathcal{C}(\mathbf{R}^*\mathbf{A}(\mathbf{I} - \omega_{k+1})\mathbf{V}_k)$. This scheme should be applied after each change made to the columns of $\mathbf{V}$, and one should note that, in order to maintain the relations

$$\boxed{\mathbf{AV}} \approx \boxed{\mathbf{A}} * \boxed{\mathbf{V}} \quad \text{and} \quad \boxed{\mathbf{AAV}} \approx \boxed{\mathbf{A}} * \boxed{\mathbf{A}} * \boxed{\mathbf{V}},$$

the values of $\boxed{\mathbf{AV}}$ and $\boxed{\mathbf{AAV}}$ should be altered as well. Although different schemes may be used, in this thesis we will stick with Arnoldi's approach to orthonormalization by intermediately applying modified Gram-Schmidt, as it is stable enough for our intentions using only moderate values for s. An other option is to use repeated Gram-Schmidt, which

[1] This naming is specific for this work, as our approach differs from most found in common literature.

is know to achieve a higher numerical accuracy. In [25] a method of bi-orthogonalization is presented that is in several respects superior to our rather simplistic approach, which is therefore better suited for practical use.

Choosing the first set of Auxiliary vectors

Up till now, we have only assumed the first set of auxiliary vectors (the columns of $\mathbf{U}_0$) to lie in $\mathcal{G}_0 = \mathcal{K}_N(\mathbf{A}, \mathbf{r}_0)$, the full Krylov subspace of $\mathbf{A}$ and $\mathbf{r}_0$. However, we have not given a specific way of generating these vectors. To our best knowledge, there are no clear theoretical results that predict the effectiveness of a particular choice, whereas the choice does certainly has its influence as we will see in later chapters. As stated in the original paper describing IDR(s) [21], choosing the columns of $\mathbf{U}_0$ to be an Arnoldi basis of $\mathcal{K}_s(\mathbf{A}, \mathbf{A}\mathbf{r}_0)$ generally yields fine results. This in contrast to, for example a random choice, or a choice for differently structured sets, for which the convergence behavior turned out the be less prosperous. The initial auxiliary vectors will be the subject of Part II of this work. For now, we continue in the line of [21], and by default set $\mathbf{U}_0$ to be an orthonormalized power basis of $\mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$. In Part II we shall also prove that this choice is an example for which the here described IDR-method is in fact a Krylov subspace method, for this is not necessarily the case. See Figure 3.2 for a comparison in convergence behaviour between our default choice for $\mathbf{U}_0$ and a randomly generated (and also orthonormalized) $\mathbf{U}_0$.

The relaxation parameters ω_k

The sequence of relaxation parameters ω is not governed by the IDR theorem, and we are free to choose them to our liking in any way. The choice to select a ω_k that minimizes $\|\mathbf{r}_{k+1}\|_2$ will be our default choice throughout this work. However some comments should be made. Firstly, the ω_k only perform a local minimization, not giving any guaranty of leading to the fastest possible convergence, as is the case in GMRES. In fact, in some cases, fixing all ω_k to some value may yield significantly faster convergence than our default choice, see Figure 3.3. Another criticism on this choice is that in the case of $\mathbf{A}$ being skew-symmetric, we will have[2] $\forall_{k\in\mathbb{N}}\omega_k = 0$.As we will see next chapter, we can adapt IDR(s) to select $\omega_{k+1}, \ldots, \omega_{k+\ell}$ such that they perform a minimization of residual norm over ℓ dimension reduction steps, leading to IDRStab ([16]), which yields a clear improvement. Yet another way of exploiting the freedom of ω is discussed in [12], where the the field of values of $\mathbf{A}$ to choose relaxation parameters. The underlying theory to the observations made in [12] can be found in [14]. In Part II we will discuss other effects that are the result of the chosen ω_k, we will see that they will actually influence the convergence behavior on a global scale.

The test vectors

An other freedom left to (possibly) exploit by the designer of an IDR-method lies in the choice of the test vectors, i.e. the columns of $\mathbf{R}$. In [20] it is proven that if we choose $\mathbf{R}$ to be (directionally) random, the probability of $\mathcal{R}^\perp$ containing a non-trivial invariant subspace w.r.t. $\mathbf{A}$ is zero. In our standard IDR(s) implementation we choose the columns of $\mathbf{R}$ entry-wise random[3], and subsequently orthonormalize them for the sake of stable computations. In case of an input problem with coefficient matrix $\mathbf{A}$ that has complex entries, it should

[2] To see this: Note that if $\mathbf{A}$ is skew symmetric, we have: $(\mathbf{A}\mathbf{r}'_k)^*\mathbf{r}'_k = -\mathbf{r}'^*_k(\mathbf{A}\mathbf{r}'_k)$, implying that if $\mathbf{r}'$ and $\mathbf{A}$ are both non-zero, we have $(\mathbf{A}\mathbf{r}')^*\mathbf{r}' = 0$, and hence $\omega_{k+1} = 0$.

[3] That is, random from a uniform distribution of the interval $[-1, 1]$.

be considered to choose the test vectors complex as well, as it increases the effectiveness of orthogonalization steps ([21]).

3.2 Pseudo-code of IDR(s) and first test results

Below we present IDR(s) in pseudo-code, heavily based on the above described theory. Although the code would yield a fine running algorithm for many inputs, it is mainly written with educational purposes and readability in mind, and is not optimized in terms of stability or computational costs. It has been however, the main algorithm for our experiments, as it stable enough for modest input sizes.
Remark on notation: with $\alpha \leftarrow\!\shortmid\, \sigma \setminus \mathbf{v}$ with σ a matrix and $\mathbf{v}$ a vector of appropriate sizes, we denote that α is computed to be the approximate solution of the system

$$\sigma\alpha = \mathbf{v}.$$

We use the Matlab-style '\'-notation to emphasize that we do not explicitly calculate inverses (although Matlab may do so for modest values of s.).

function IDRS($\mathbf{A}, \mathbf{b}, \mathbf{x}_0, \mathbf{U}, \mathbf{R}, \mathbf{tol}, \mathbf{maxit}, s$)
  $\mathbf{k} = 0$
  $\mathbf{x} \leftarrow\!\shortmid \mathbf{x}_0$
  $\mathbf{r}_0 \leftarrow\!\shortmid \mathbf{b} - \mathbf{A} * \mathbf{x}_0$
  $\mathbf{AU} \leftarrow\!\shortmid \mathbf{A} * \mathbf{U}$

  while $(\|\mathbf{r}_k\|_2 > \mathbf{tol}) \wedge (\mathbf{k} < \mathbf{maxit})$ **do**
    %Dimension reduction step
    $\sigma \leftarrow\!\shortmid \mathbf{R}^* \cdot \mathbf{AU}$
    $\alpha \leftarrow\!\shortmid \sigma \setminus (\mathbf{R}^* \cdot \mathbf{r})$
    $\mathbf{r}' \leftarrow\!\shortmid \mathbf{r} - \mathbf{AU} \cdot \alpha$
    $\mathbf{x}' \leftarrow\!\shortmid \mathbf{x} - \mathbf{U} \cdot \alpha$
    $\mathbf{Ar}' \leftarrow\!\shortmid \mathbf{A} * \mathbf{r}'$

    $\beta \leftarrow\!\shortmid \sigma \setminus (\mathbf{R}^* \cdot \mathbf{Ar}')$
    $\mathbf{V}e_1 \leftarrow\!\shortmid \mathbf{r}' - \mathbf{U} \cdot \beta$
    $\mathbf{AV}e_1 \leftarrow\!\shortmid \mathbf{Ar}' - \mathbf{AU} \cdot \beta$
    $\mathbf{AAV}e_1 \leftarrow\!\shortmid \mathbf{A} * \mathbf{AV}e_1$
    Orth($\mathbf{V}, \mathbf{AV}, \mathbf{AAV}, 1$)

    for $q = 2 \ldots (s-1)$ **do**
      $\beta \leftarrow\!\shortmid \sigma \setminus (\mathbf{R}^* \cdot \mathbf{AAV}e_{q-1})$
      $\mathbf{V}e_q \leftarrow\!\shortmid \mathbf{AV}e_{q-1} - \mathbf{U} \cdot \beta$
      $\mathbf{AV}e_q \leftarrow\!\shortmid \mathbf{AAV}e_{q-1} - \mathbf{AU} \cdot \beta$
      $\mathbf{AAV}e_q \leftarrow\!\shortmid \mathbf{A} * \mathbf{AV}e_q$
      Orth($\mathbf{V}, \mathbf{AV}, \mathbf{AAV}, q$)
    end for
    %Polynomial step
    $\omega \leftarrow\!\shortmid \frac{(\mathbf{Ar}')^* \cdot \mathbf{r}'}{(\mathbf{Ar}')^* \cdot \mathbf{Ar}'}$

    $\mathbf{r} \leftarrow\!\shortmid \mathbf{r}' - \omega \cdot \mathbf{Ar}'$
    $\mathbf{x} \leftarrow\!\shortmid \mathbf{x}' + \omega \cdot \mathbf{r}'$
    $\mathbf{U} \leftarrow\!\shortmid \mathbf{V} - \omega \cdot \mathbf{AU}$
    $\mathbf{AU} \leftarrow\!\shortmid \mathbf{AV} - \omega \cdot \mathbf{AAV}$
  end while
end function

Algorithm 1: *A straight forward implementation of IDR(s) based on the theory of Chapter 1 using local residual minimization. Note the added orthonormalization steps in gray for* $\mathbf{V}$*, which changes* $\mathbf{AV}$ *and* $\mathbf{AAV}$ *accordingly. Any orthonormalizing subroutine could be used here but we have added an example in Algorithm 2.*

1: **function** ORTH($\mathbf{V}, \mathbf{AV}, \mathbf{AAV}, q$)
2:
3: **if** $q > 1$ **then**
4: $\vec{\mu} \leftarrow \mathbf{V}^*_{(:,1:q-1)} \cdot \mathbf{V}_{(:,q)}$
5: $\mathbf{V}_{(:,q)} \leftarrow \mathbf{V}_{(:,q)} - \mathbf{V}_{(:,1:q-1)} \cdot \vec{\mu}$
6: $\mathbf{AV}_{(:,q)} \leftarrow \mathbf{AV}_{(:,q)} - \mathbf{AV}_{(:,1:q-1)} \cdot \vec{\mu}$
7: $\mathbf{AAV}_{(:,q)} \leftarrow \mathbf{AAV}_{(:,q)} - \mathbf{AAV}_{(:,1:q-1)} \cdot \vec{\mu}$
8: **end if**
9:
10: $\rho \leftarrow \sqrt{\mathbf{V}^*_{(:,q)} \cdot \mathbf{V}_{(:,q)}}$
11: $\mathbf{V}_{(:,q)} \leftarrow \mathbf{V}_{(:,q)}/\rho$
12: $\mathbf{AV}_{(:,q)} \leftarrow \mathbf{AV}_{(:,q)}/\rho$
13: $\mathbf{AAV}_{(:,q)} \leftarrow \mathbf{AAV}_{(:,q)}/\rho$
14: **end function**

Algorithm 2: *Example of an Orthonormalization Subroutine, in this case Arnoldi's approach using modified Gram-Schmidt, which we will use in standard IDR(s).*

Algorithm 1 in conjunction with Algorithm 2 lead to the computational costs as summarized in Table 3.1. It should be noted that costs for the initialization are subject to change when one chooses $\mathbf{U}_0$ differently, so the focus should be on the IDR cycle. It should be mentioned that in [25] an IDR method is presented that is both more stable and more efficient in terms of required DOT and AXPY per cycle.

	AXPY	DOT	MV
Initialization	5	0	$s+1$
Dimension Reduction Step	$(2s+2)(s+1)$	$2s^2+s$	$s+1$
Polynomial Step	$2s+2$	2	0
IDR Cycle total	$\mathbf{(2s+2)(s+2)}$	$\mathbf{2s^2+s+2}$	$\mathbf{s+1}$
Orthonormalization	$3(\frac{s^2+s}{2})$	$\frac{s^2+s}{2}$	0

Table 3.1: *Computational costs of standard IDR(s) per iteration phase. Note that after initialization is done, the costs per iteration equal the total costs for an IDR-cycle, plus the costs for orthonormalization during the IDR-cycle, for which one might want to use a different orthonormalization scheme.*

To explore the practical usability on the theory on the termination of IDR(s) and the comments made on the implementation choices, the results of three experiments are shown below.

Termination

To give the reader an idea of the effect of increasing s in practice, Figure 3.1 shows the results of of an experiment on two different problems for increasing values of s.

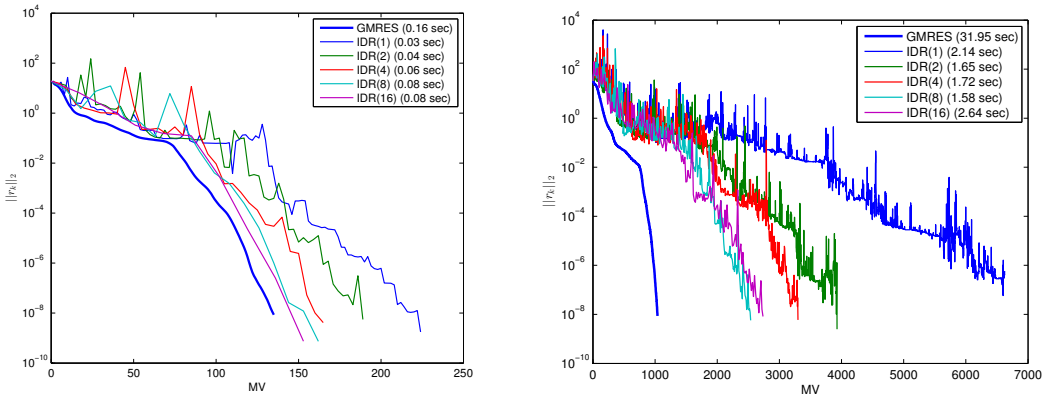


Figure 3.1: *Residual norm history for different values of s on the* SHERMAN4 *($N = 1104$, left)* SHERMAN5 *($N = 3312$, right) test matrices. We can see how on* SHERMAN4 *the number of required MV indeed seems to converge to the GMRES curve for increasing s. On* SHERMAN5 *this effect is far less apparent.*

Random search matrix

The only structured choice for $\mathbf{U}_0$ that consistently leads to fine convergence results is the choice of $\mathbf{U}_0$ being a power basis of $\mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$. In Figure 3.2 we see a typical improvement over running IDR(s) with a randomly generated $\mathbf{U}_0$.

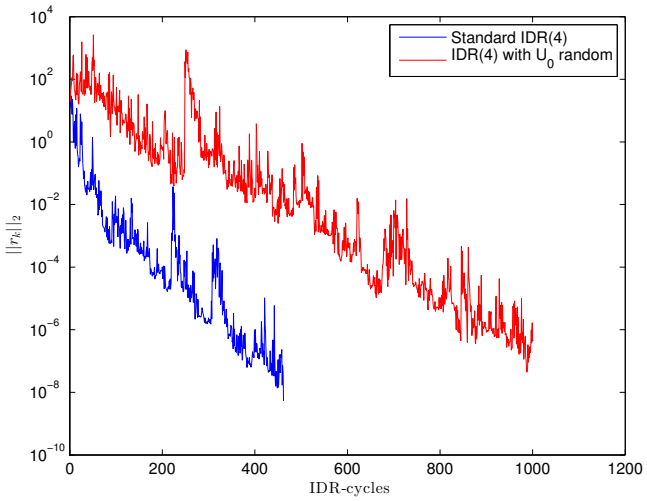


Figure 3.2: *Comparison of residual norm history for standard IDR(4) and IDR(4) with $\mathbf{U}_0$ random and orthonormalized on* MEIER01*.*

Optimality of the relaxation parameters

The relaxation parameters in standard IDR(s) are merely a local 'best choice'. In some cases, simply fixing a hand-picked value for all ω_k clearly outperforms our standard approach, as is shown in Figure 3.3; apparently there is plenty of room for improvement.

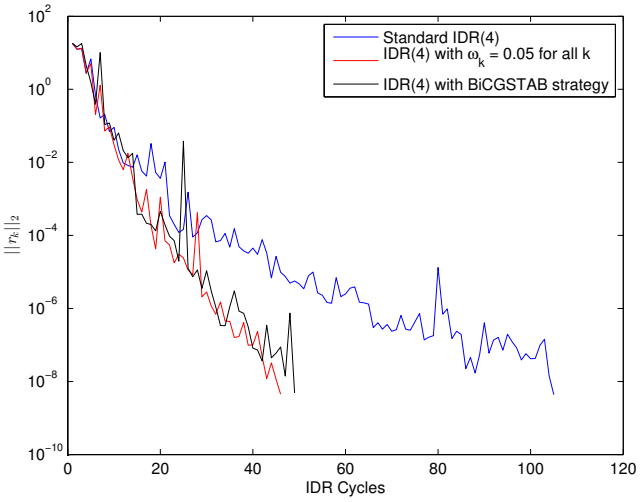


Figure 3.3: *Comparison using* MEIER01 *of standard IDR(8), IDR(8) with* $(\omega)_{k\in\mathbb{N}} = (0.05, 0.05, 0.05, \dots)$ *and IDR(8) using the* ω *selection technique as used in BiCGSTAB (see also [14]). Standard IDR(8) is clearly outperformed.*

Chapter 4

IDRStab

In the IDR algorithm as derived in previous chapters, we used the freedom offered by the IDR framework to choose our values for $(\omega)_{k\in\mathbb{N}_+}$ such that ω_{k+1} minimizes $\|r_{k+1}\|_2 = \|\mathbf{r}' - \omega_{k+1} v\mathbf{A}\mathbf{r}'\|_2$. Since this minimization is local (i.e., it only considers the current next residual), this does not guarantee optimal convergence. IDRStab, first presented in [16], very much like BiCGStab(ℓ) (see [13]), postpones the polynomial step of the algorithm for a given number of iterations, ℓ, and then calculates the values for $\omega_{k+1}, ..., \omega_{k+\ell}$ by minimizing the residual norm over all these iterations. (Actually, BiCGStab(ℓ) is a bit more refined, and cleverly averages between minimizing polynomials and orthogonalizing polynomials to achieve even faster convergence.) In order to do so we will be needing a scheme that repeats the dimension reduction step an additional $\ell - 1$ times over the original, without the need of the values of the relaxation parameters. For each j-th dimension reduction step, with $j = 1...\ell$, we will generalize first our definitions and provide lemma's that indeed show that this postponing of the polynomial step is possible.

4.1 Postponing Minimization

Let k again be the iteration number, we then perform ℓ dimension reduction steps during this iteration, for which we will define the oblique projections for $j = 1, ..., \ell$ and $i = 0, ..., j$

$$\mathbf{\Pi}_i^j \equiv \mathbf{I} - \mathbf{A}^i \mathbf{V}_{j-1} \sigma_j^{-1} \mathbf{R}^* \mathbf{A}^{j-i} \quad \text{with} \quad \mathbf{V}_0 \equiv \mathbf{U}_k \quad \text{and} \quad \sigma_j \equiv \mathbf{R}^* \mathbf{A}^j \mathbf{V}_{j-1}$$

We will later define the $\mathbf{V}_j$ for $j = 1...\ell$ by recursion. Note that $\mathbf{\Pi}_1^1$ and $\mathbf{\Pi}_0^1$ are equivalent to the projections $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_0$ defined earlier. Moreover, we have similar properties as before:

$$\mathbf{R}^* \mathbf{A}^i \mathbf{\Pi}_{j-i}^j = \mathbf{0} \quad (i \leq j) \quad \text{and} \quad \mathbf{\Pi}_{i+1}^j \mathbf{A} = \mathbf{A} \mathbf{\Pi}_i^j \quad (i \leq j-1). \tag{4.1}$$

We will, for the remainder of this chapter, suppose that we have fixed values for s and ℓ, and that the algorithm has arrived at iteration k, thus has obtained $\mathbf{r}_k$, $\mathbf{x}_k$ and $\mathbf{U}_k$. We define for the entire dimension reduction step

$$\mathbf{r}'_0 \equiv \mathbf{r}_k \quad \text{and} \quad \mathbf{x}'_0 \equiv \mathbf{x}_k,$$

and for $j = 1, ..., \ell$

$$\mathbf{r}'_j \equiv \mathbf{\Pi}_1^j \mathbf{r}'_{j-1}$$
$$\mathbf{x}'_j \equiv \mathbf{x}'_{j-1} + \mathbf{A}^{j-1} \mathbf{V}_{j-1} \sigma_j^{-1} \mathbf{R}^* \mathbf{r}'_j.$$

Note that the structure of the definitions reflect the fact that that we are repeatedly projecting the $\mathbf{r}'_j$ orthogonal to the space spanned by $(\mathbf{A}^*)^{j-1}\mathbf{R}$, and updating $\mathbf{x}'_j$ accordingly as was the case in regular IDR. As we will see later, a similar statement will hold for the columns of the $\mathbf{V}_j$, which we define as an Krylov basis of $\mathcal{K}_s(\mathbf{\Pi}_0^j\mathbf{A}, \mathbf{\Pi}_0^j\mathbf{r}'_j)$:

$$\mathbf{V}_j \equiv [\mathbf{\Pi}_0^j\mathbf{r}'_j, ..., (\mathbf{\Pi}_0^j\mathbf{A})^{s-1}\mathbf{\Pi}_0^j\mathbf{r}'_j].$$

Note that this implies that $\mathbf{AV}_j$ is a basis of $\mathcal{K}_s(\mathbf{\Pi}_1^j\mathbf{A}, \mathbf{\Pi}_1^j\mathbf{A}\mathbf{r}'_j)$:

$$\mathbf{AV}_j = \mathbf{A}[\mathbf{\Pi}_0^j\mathbf{r}'_j, ..., (\mathbf{\Pi}_0^j\mathbf{A})^{s-1}\mathbf{\Pi}_0^j\mathbf{r}'_j] = [\mathbf{\Pi}_1^j\mathbf{A}\mathbf{r}'_j, ..., (\mathbf{\Pi}_1^j\mathbf{A})^s\mathbf{r}'_j].$$

The intuition backing these definition is that we want to end up with an $\mathbf{r}'_\ell$ and $\mathbf{AV}_\ell$ that we can easily lift into $\mathcal{G}_{k+\ell}$ by mere multiplication with a matrix polynomial $Q_\ell(\mathbf{A})$ in such a way that enables us to perform residual minimization over all the intermediate residuals.

We now give give some lemma's that form the theoretical basis which allows IDRStab to indeed postpone the polynomial step by giving the following lemma's, each one being applied applied in the proof of the next.

Lemma 4.1. ***Retaining orthogonality***
Assume that $\mathbf{AV}_j \in \mathcal{G}_k \cap \mathcal{K}_j(\mathbf{A}^*, \mathbf{R})^\perp$*, and let* $\mathbf{s} \in \mathcal{G}_k \cap \mathcal{K}_{j+1}(\mathbf{A}^*, \mathbf{R})^\perp$*, then* $\mathbf{\Pi}_1^{j+1}\mathbf{As} \in \mathcal{G}_k \cap \mathcal{K}_{j+1}(\mathbf{A}^*, \mathbf{R})^\perp$.

Proof. Note that $\mathbf{s} \in \mathcal{G}_k$ implies (by the definition of the Sonneveld subspace) that for some polynomial P_k and $\mathbf{w} \perp \mathcal{K}_{k+(j+1)}(\mathbf{A}^*, \mathbf{R})$ we have $\mathbf{s} = P_k(\mathbf{A})\mathbf{w}$. Hence,

$$\mathbf{As} = \mathbf{A}P_k(\mathbf{A})\mathbf{w} = P_k(\mathbf{A})\mathbf{Aw} \quad \text{with} \quad \mathbf{Aw} \perp \mathcal{K}_{k+j}(\mathbf{A}^*, \mathbf{R}).$$

Applying $\mathbf{\Pi}_1^{j+1}$ then yields

$$\mathbf{\Pi}_1^{j+1}\mathbf{As} = \mathbf{As} - \mathbf{AV}_j(\mathbf{R}^*\mathbf{A}^{j+1}\mathbf{V}_j)^{-1}\mathbf{R}^*\mathbf{A}^j\mathbf{s} \in \mathcal{G}_k \cap \mathcal{K}_j(\mathbf{A}^*, \mathbf{R})^\perp,$$

for $\mathbf{As}, \mathbf{AV}_j \in \mathcal{G}_k \cap \mathcal{K}_j(\mathbf{A}^*, \mathbf{R})^\perp$.

Moreover, we have by the properties mentioned in 4.1

$$\mathbf{R}^*\mathbf{A}^j\mathbf{\Pi}_1^{j+1}\mathbf{As} = 0,$$

and hence

$$\mathbf{\Pi}_1^{j+1}\mathbf{As} \in \mathcal{G}_k \cap \mathcal{K}_{j+1}(\mathbf{A}^*, \mathbf{R})^\perp.$$

□

The next lemma shows that we are repeatedly orthogonalizing our vectors, in preparation of the polynomial step.

Lemma 4.2. ***Postponing Polynomial step 1***
Given $\mathbf{r}_k, \mathbf{U}_k, \mathbf{AU}_k$*, then we have*

$$\forall j \in \mathbb{N}: \quad \mathbf{AV}_j, \mathbf{r}'_j \in \mathcal{G}_k \cap \mathcal{K}_j(\mathbf{A}^*, \mathbf{R})^\perp$$

Proof. We give a proof by induction. Note that, by definition, we have for $j = 0$

$$\mathbf{AV}_0 (= \mathbf{AU}_k), \mathbf{r}'_0 (= \mathbf{r}_k) \in \mathcal{G}_k = \mathcal{G}_k \cap \mathcal{K}_0(\mathbf{A}^*, \mathbf{R})^\perp.$$

Now assume that the statement is proven for some $j \geq 0$. We have, again by definition,

$$\mathbf{r}'_{j+1} = \mathbf{\Pi}_1^{j+1} \mathbf{r}'_j = \mathbf{r}'_j - \mathbf{AV}_j (\mathbf{R}^* \mathbf{A}^{j+1} \mathbf{V}_j)^{-1} \mathbf{R}^* \mathbf{A}^j \mathbf{r}'_j$$

Now note that since we have assumed that $\mathbf{AV}_j, \mathbf{r}'_j \in \mathcal{G}_k \cap \mathcal{K}_j(\mathbf{A}^*, \mathbf{R})^\perp$, we have

$$\mathbf{r}'_{j+1} \in \mathcal{G}_k \cap \mathcal{K}_j(\mathbf{A}^*, \mathbf{R})^\perp.$$

Moreover, we have

$$\mathbf{R}^* \mathbf{A}^j \mathbf{r}'_{j+1} = \mathbf{R}^* \mathbf{A}^j \mathbf{r}'_j - \mathbf{R}^* \mathbf{A}^{j+1} \mathbf{V}_j (\mathbf{R}^* \mathbf{A}^{j+1} \mathbf{V}_j)^{-1} \mathbf{R}^* \mathbf{A}^j \mathbf{r}'_j = 0,$$

hence $\mathbf{r}'_{j+1} \perp (\mathbf{A}^*)^j \mathbf{R}$, and hence

$$\mathbf{r}'_{j+1} \in \mathcal{G}_k \cap \mathcal{K}_{j+1}(\mathbf{A}^*, \mathbf{R})^\perp.$$

In order to yield the same result for the columns of $\mathbf{AV}_{j+1}$, we utilize the earlier made observation that

$$\mathbf{AV}_{j+1} = [\mathbf{\Pi}_1^{j+1} \mathbf{A} \mathbf{r}'_{j+1}, ..., (\mathbf{\Pi}_1^{j+1} \mathbf{A})^s \mathbf{r}'_{j+1}],$$

apply Lemma 4.1 inductively with with

$$s = \mathbf{r}'_{j+1}, \mathbf{\Pi}_1^{j+1} \mathbf{A} \mathbf{r}'_{j+1}, ..., (\mathbf{\Pi}_1^{j+1} \mathbf{A})^{s-1} \mathbf{r}'_{j+1}$$

indeed yielding

$$\mathbf{AV}_{j+1} \in \mathcal{G}_k \cap \mathcal{K}_{j+1}(\mathbf{A}^*, \mathbf{R})^\perp.$$

This completes the proof. □

Lemma 4.3. ***Postponing Polynomial step 2***
Let, for any j, $\mathcal{G}_{k+j} = S(Q_j P_k, \mathbf{A}, \mathbf{R})$ for some determined polynomials Q_j, P_k of degrees j and k respectively. Then for $\mathbf{r}_k, \mathbf{AU}_k \in \mathcal{G}_k$, we have

$$Q_j(\mathbf{A}) \mathbf{r}'_j, Q_j(\mathbf{A}) \mathbf{AV}_j \in \mathcal{G}_{k+j}.$$

Proof. Assume we indeed have polynomials Q_j, P_k and $\mathbf{r}_k, \mathbf{AU}_k \in \mathcal{G}_k$. Then, by Lemma 4.2 and the definition of Sonneveld subspaces we have $\mathbf{r}'_j, \mathbf{AV}_j \in \mathcal{G}_k \cap \mathcal{K}_{j+1}(\mathbf{A}^*, \mathbf{R})^\perp$, so that we can write

$$\mathbf{r}'_j = P_k(\mathbf{A}) \mathbf{w} \quad \text{with} \quad \mathbf{w} \perp \mathcal{K}_{k+j}(\mathbf{A}^*, \mathbf{R}),$$

and similarly we have

$$\mathbf{AV}_j = P_k(\mathbf{A}) \mathbf{W} \quad \text{with} \quad \mathbf{W} \perp \mathcal{K}_{k+j}(\mathbf{A}^*, \mathbf{R}).$$

Now multiplying with $Q_j(\mathbf{A})$ and again applying the definition of the Sonneveld subspace yields the result:

$$Q_j(\mathbf{A}) \mathbf{r}'_j = Q_j(\mathbf{A}) P_k(\mathbf{A}) \mathbf{w} \in \mathcal{G}_{k+j},$$
$$Q_j(\mathbf{A}) \mathbf{AV}_j = Q_j(\mathbf{A}) P_k(\mathbf{A}) \mathbf{W} \in \mathcal{G}_{k+j}.$$

□

The lemma's express the fact that we can postpone the polynomial step in our algorithm, enabling IDRStab to first obtain $\mathbf{r}'_\ell$ and $\mathbf{V}_\ell$, and after that determine scalars $\omega_k, ..., \omega_{k+\ell}$ that define the polynomial

$$Q_j(\xi) \equiv \prod_{i=1,...,\ell} (1 - \omega_{k+i}\xi)$$

enabling us to define

$$\mathbf{r}_{k+\ell} \equiv Q_j(\mathbf{A})\mathbf{r}'_\ell \quad \text{and} \quad \mathbf{U}_{k+\ell} \equiv Q_j(\mathbf{A})\mathbf{V}_\ell,$$

such that, using Lemma 4.3, we have, as desired:

$$\mathbf{r}_{k+\ell}, \mathbf{A}\mathbf{U}_{k+\ell} \in \mathcal{G}_{k+\ell}.$$

IDRStab will, at the end of every j-th dimension reduction step have available the following quantities:

$$\mathbf{x}'_j, \mathbf{A}^i\mathbf{r}'_j \quad (i = 0...j) \quad \text{and} \quad \mathbf{A}^i\mathbf{V}_j \quad (i = 0...j+1).$$

Note that this will, in terms of computation, reduce the multiplication with $Q_\ell(\mathbf{A})$ to mere vector updates, rather than matrix-vector multiplications. The efficient computation of these quantities will be the subject of the next part of this chapter.

4.2 Computation of the Dimension Reduction step

Next, we will focus on setting up a setting up a efficient computational scheme, after which, the polynomial step will turn out to be straight forward. Per dimension reduction step we can distinguish two parts: The calculation of $\mathbf{x}'_j$ and $\mathbf{A}^i\mathbf{r}'_j$ being the first part, and the calculation of $\mathbf{A}^i\mathbf{V}_j$ being the second.

Recall that at the start of each j-th step, we have available $\mathbf{A}^i\mathbf{r}'_{j-1}$ and $\mathbf{A}^i\mathbf{V}_{j-1}$. Using these quantities, we can calculate

$$\alpha \equiv \sigma_j^{-1}(\mathbf{R}^*\mathbf{A}^{j-1}\mathbf{r}'_{j-1}) \leftarrow\!\shortmid \boxed{\sigma_j}^{-1}\boxed{\mathbf{R}^*} \cdot \boxed{\mathbf{A}^{j-1}\mathbf{r}'_{j-1}},$$

enabling the calculation of $\mathbf{A}^i\mathbf{r}'_j (i = 0, ..., j-1)$ by computing, without any matrix-vector products:

$$\mathbf{A}^i\mathbf{r}'_j = \mathbf{\Pi}_{i+1}^j(\mathbf{A}^i\mathbf{r}'_{j-1}) = \mathbf{A}^i\mathbf{r}'_{j-1} - \mathbf{A}^{i+1}\mathbf{V}_{j-1}\alpha \leftarrow\!\shortmid \boxed{\mathbf{A}^i\mathbf{r}'_{j-1}} - \boxed{\mathbf{A}^{i+1}\mathbf{V}_{j-1}} \cdot \boxed{\alpha}.$$

We will use an explicit matrix-vector product to compute the final $\mathbf{A}^j\mathbf{r}'_j$:

$$\mathbf{A}^j\mathbf{r}'_j \leftarrow\!\shortmid \boxed{\mathbf{A}} * \boxed{\mathbf{A}^{j-1}\mathbf{r}'_j}$$

The vector $\mathbf{x}'_j$ can be yielded by using α again:

$$\mathbf{x}'_j \leftarrow\!\shortmid \boxed{\mathbf{x}'_{j-1}} + \boxed{\mathbf{V}_{j-1}} \cdot \boxed{\alpha}.$$

This concludes the calculations for this part, having used one matrx-vector product in total. Continuing with the second part, we proceed per column $q = 1, ..., s$ of the $\mathbf{A}^i\mathbf{V}_j$. Recall that we defined $\mathbf{V}_j$ as an Arnoldi basis of $\mathcal{K}_s(\mathbf{\Pi}_0^j\mathbf{A}, \mathbf{\Pi}_0^j\mathbf{r}'_j)$, hence the first column of $\mathbf{A}^i\mathbf{V}_j$ is computable setting

$$\beta \leftarrow \boxed{\sigma_j}^{-1}\boxed{\mathbf{R}^*}\cdot\boxed{\mathbf{A}^j\mathbf{r}'_j}$$

and then putting

$$\mathbf{A}^i\mathbf{V}_j e_1 = \mathbf{A}^i\mathbf{\Pi}_0^j\mathbf{r}'_j = \mathbf{A}^i\mathbf{r}'_j - \mathbf{A}^i\mathbf{V}_{j-1}\sigma_j^{-1}\mathbf{R}^*\mathbf{A}^j\mathbf{r}'_j \leftarrow \boxed{\mathbf{A}^i\mathbf{r}'_j} - \boxed{\mathbf{A}^i\mathbf{V}_{j-1}}\cdot\boxed{\beta},$$

which again takes no matrix-vector multiplications. To compute $\mathbf{A}^{j+1}\mathbf{V}_j$ in order to meet the requirements for the $j+1$-th dimension reduction step however, it does:

$$\mathbf{A}^{j+1}\mathbf{V}_j e_1 \leftarrow \boxed{\mathbf{A}} * \boxed{\mathbf{A}^j\mathbf{V}_j e_1}.$$

For the next columns, let $q > 1$,we wish to calculate $V_j e_q = \mathbf{\Pi}_0^j\mathbf{A}\mathbf{V}_j e_{q-1}$, so we set

$$\beta \leftarrow \boxed{\sigma_j}^{-1}\boxed{\mathbf{R}^*}\cdot\boxed{\mathbf{A}^{j+1}\mathbf{V}_j e_{q-1}},$$

and calculate

$$\mathbf{A}^i\mathbf{V}_j e_q = \mathbf{A}^i\mathbf{\Pi}_0^j\mathbf{A}\mathbf{V}_j e_{q-1} \leftarrow \boxed{\mathbf{A}^{i+1}\mathbf{V}_j e_{q-1}} - \boxed{\mathbf{A}^i\mathbf{V}_{j-1}}\cdot\boxed{\beta}.$$

We then use a matrix-vector multiplication to calculate the remaining $\mathbf{A}^{j+1}\mathbf{V}_j e_q$ which will be needed for the next column:

$$\mathbf{A}^{j+1}\mathbf{V}_j e_q \leftarrow \boxed{\mathbf{A}} * \boxed{\mathbf{A}^j\mathbf{V}_j e_q}.$$

The above is a complete description of how to efficiently compute $\mathbf{A}^i\mathbf{r}'_\ell$ $(i \leq \ell)$ and $\mathbf{A}^i\mathbf{V}_\ell$ $(i \leq \ell+1)$. We now only need to determine the scalars which will define the polynomial $\mathbf{Q}_j$ in order tot lift the results into $\mathcal{G}_{k+\ell}$.

4.3 The Polynomial Step

Assuming that we want to minimize $\|\mathbf{r}_{k+\ell}\|_2$, we can state the problem as follows. Find $\vec{\gamma} \in \mathbb{C}^\ell$ such that

$$\|\mathbf{r}_{k+\ell}\|_2 = \left\|\mathbf{r}'_\ell - \gamma_1\mathbf{A}\mathbf{r}'_\ell - \cdots - \gamma_\ell\mathbf{A}^\ell\mathbf{r}'_\ell\right\|_2 \tag{4.2}$$

is minimal. Similarly to the case in standard IDR(s), this problem can be reformulated as the problem to find $\vec{\gamma} \in \mathbb{C}^\ell$, such that

$$\forall i \in \{1, ..., \ell\} : \mathbf{r}_{k+\ell} \perp \mathbf{A}^i\mathbf{r}'_\ell. \tag{4.3}$$

Define

$$\mathbf{B} \equiv [\mathbf{A}\mathbf{r}'_\ell, ..., \mathbf{A}^\ell\mathbf{r}'_\ell].$$

Now note that solving to following $\ell \times \ell$-system for $\vec{\gamma}$

$$\mathbf{B}^*\mathbf{B}\vec{\gamma} = \mathbf{B}^*\mathbf{r}'_\ell$$

yields us $\vec{\gamma}$ such that

$$\mathbf{B}^*\mathbf{r}_{k+\ell} = \mathbf{B}^*(\mathbf{r}'_\ell - \sum_{i=1,\ldots,\ell} \gamma_i \mathbf{A}^i \mathbf{r}'_\ell) = \mathbf{B}^*(\mathbf{r}'_\ell - \mathbf{B}\vec{\gamma}) = 0.$$

Hence, for this $\vec{\gamma}$, we indeed have 4.3, and thus minimized 4.2. Since, once arrived at the polynomial step, we have already obtained $\mathbf{A}^i\mathbf{r}'_\ell$ $(i \leq \ell)$ and $\mathbf{A}^i\mathbf{V}_\ell$ $(i \leq \ell + 1)$, we can compute the polynomial multiplication without matrix-vector multiplications, like so:

$$\gamma \leftarrow \boxed{\mathbf{R}^*} \cdot \boxed{\mathbf{R}} \setminus \boxed{\mathbf{R}^*} \cdot \mathbf{r}'_\ell$$

$$\mathbf{r}_{k+\ell} \leftarrow \boxed{\mathbf{r}'_\ell} - \boxed{\gamma_1} \cdot \boxed{\mathbf{A}^1\mathbf{r}'_\ell} - \boxed{\gamma_2} \cdot \boxed{\mathbf{A}^2\mathbf{r}'_\ell} - \ldots - \boxed{\gamma_\ell} \cdot \boxed{\mathbf{A}^\ell\mathbf{r}'_\ell}$$

$$\mathbf{x}_{k+\ell} \leftarrow \boxed{\mathbf{x}'_\ell} + \boxed{\gamma_1} \cdot \boxed{\mathbf{r}'_\ell} + \boxed{\gamma_2} \cdot \boxed{\mathbf{A}^1\mathbf{r}'_\ell} + \ldots + \boxed{\gamma_\ell} \cdot \boxed{\mathbf{A}^{\ell-1}\mathbf{r}'_\ell}$$

$$\mathbf{U}_{k+\ell} \leftarrow \boxed{\mathbf{V}_\ell} - \boxed{\gamma_1} \cdot \boxed{\mathbf{A}^1\mathbf{V}_\ell} - \boxed{\gamma_2} \cdot \boxed{\mathbf{A}^2\mathbf{V}_\ell} - \ldots - \boxed{\gamma_\ell} \cdot \boxed{\mathbf{A}^\ell\mathbf{V}_\ell}$$

$$\mathbf{A}\mathbf{U}_{k+\ell} \leftarrow \boxed{\mathbf{A}\mathbf{V}_\ell} - \boxed{\gamma_1} \cdot \boxed{\mathbf{A}^2\mathbf{V}_\ell} - \boxed{\gamma_2} \cdot \boxed{\mathbf{A}^3\mathbf{V}_\ell} - \ldots - \boxed{\gamma_\ell} \cdot \boxed{\mathbf{A}^{\ell+1}\mathbf{V}_\ell}$$

4.4 Implementation

As with the derivation with IDR(s), we can now, with all computational steps covered, set up a basic implementation of IDRStab. Again, the pseudocode presented in this section (Algorithm 3) is mainly written with readability in mind. A few words on notation are in place here: we have chosen to put the vectors $\mathbf{A}^i\mathbf{r}'_j$ with $i = 0, \ldots, j$ all stacked in what one might think of as a 'vector array' called $\mathbf{r}'$. The same goes for $\mathbf{A}^i\mathbf{V}_j$ with $i = 0, \ldots, j+1$. During iteration k, and dimension reduction step j, we refer to $\mathbf{A}^i\mathbf{r}'_j$ by $\mathbf{r}'_i$, and likewise, to $\mathbf{A}^i\mathbf{V}_j$ by $\mathbf{V}_i$. Figure 4.1 shows the performance for different values of ℓ, and since IDR(s)Stab(1) is equal to IDR(s), compares it to IDR(s). It should be noted that this implementation of IDRStab is rather slow in terms of CPU-time. Table 4.1 summarizes again the costs, as we did for standard IDR(s). Note that there is some overhead caused by the higher order minimization as compared to IDR(s).

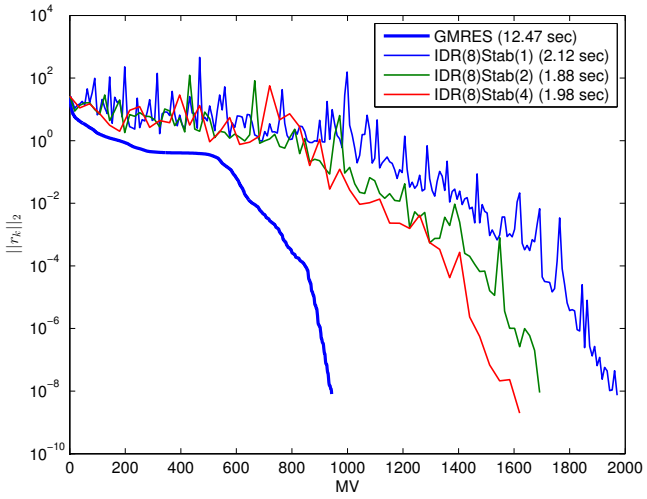


Figure 4.1: *Residual norm history for different values of ℓ in IDRStab on the* DW2048 *matrix ($N = 3312$). We can see the increased performance per MV as ℓ increases.*

	AXPY	DOT	MV
Initialization	5	0	$s+1$
Dimension Reduction Step	$(\frac{\ell^2+\ell}{2})(s^2+2s+1)+\ell$	$\ell(2s^2+s)$	$\ell(s+1)$
Polynomial Step	$(\ell+1)(2+2s)$	2ℓ	0
IDR Cycle total	$(\frac{\boldsymbol{\ell^2+\ell}}{\mathbf{2}})(\mathbf{s^2+2s+1})+(\boldsymbol{\ell+1})(\mathbf{2+2s})+\ell$	$\ell(\mathbf{2s^2+s+2})$	$\mathbf{s+1}$
Orthonormalization	$(\frac{\ell^2+\ell}{2}+2\ell)\frac{s^2+s}{2}$	$\ell\frac{s^2+s}{2}$	0

Table 4.1: *Computational costs of standard IDRStab per iteration. We can distinguish some overhead compared to IDR(s), but in theory get a better converging method in return.*

```
1: function IDRSTAB(A, b, x_0, U, R, tol, maxit, s, ℓ)
2:     k = 0
3:     x ↤ x_0
4:     r_0 ↤ b − A ∗ x_0
5:     AU ↤ A ∗ U
6:     while (‖r‖_2 > tol) ∧ (k < maxit) do
7:         Ṽ ↤ [U; AU], r̃ ↤ r
8:         x′ ↤ x
9:         %Dimension Reduction Step
10:        for j = 1, ..., ℓ do
11:            σ ↤ R* · Ṽ_j
12:                Part I
13:            α ↤ σ \ R* · r̃_{j−1}
14:            r′ ↤ r̃ − Ṽ_{(1:j)} · α
15:            r′ ↤ [r′; A ∗ r′_j]
16:            x′ ↤ x′ + Ṽ_0 · α
17:                Part II
18:            β ↤ σ \ R* · r′_j
19:            Ve_1 ↤ r′ − Ṽ · β
20:            Ve_1 ↤ [Ve_1; A ∗ V_j e_1]
21:            Ve_1 ↤ Orth(V,1)
22:            for q = 2, ..., s do
23:                β ↤ σ \ R* · V_{j+1} e_{q−1}
24:                V_{(0:j)} e_q ↤ V_{(1:j+1)} e_{q−1} − Ṽ · β
25:                Ve_q ↤ [V; A ∗ V_j e_q]
26:                Ve_q ↤Orth(V,q)
27:            end for
28:            Ṽ ↤ V, r̃ ↤ r′
29:        end for
30:        % Polynomial Step
31:        B ↤ [Ar′_ℓ, ..., A^ℓ r′_ℓ]
32:        γ⃗ ↤ (B* · B)^{−1} B* · r′_ℓ
33:        r ↤ r′_ℓ − B · γ⃗
34:        for i = 0, ..., ℓ do
35:            x ↤ x′ + γ⃗_i · r′_i
36:            U ↤ V_0 − γ⃗_i · V_i
37:            AU ↤ V_1 − γ⃗_i · V_{i+1}
38:        end for
39:        k ↤ k + 1
40:    end while
41: end function
```

Algorithm 3: *An implementation of IDR(s)Stab(ℓ) based on the theory of Chapter 4, using altered notation as compared to the previous algorithms. Again, note the added orthonormalization steps in gray for* $\mathbf{V}_j$*, which which should alter* $\mathbf{A}^i\mathbf{V}_j$ *accordingly.*

Chapter 5

Additional remarks on IDR-methods

For completeness and to provide possible extra insight, we conclude this part with making some additional notes on the IDR-methods derived in previous chapters.

Basic properties

Note that the recurrence relations for computing $\mathbf{r}_k$, $\mathbf{x}_k$, $\mathbf{U}_k$ and $\mathbf{AU}_k$ do not grow in length during the iterative process. We may therefore conclude that IDR is a short recurrence method, using s-term recurrences to be precise. Moreover note that there is no need for explicit computation with $\mathbf{A}^*$, making it a transpose-free method, generalizing its applicability. Finally, we have not made any assumptions on the algbraic properties of $\mathbf{A}$, making IDR theoretically very generally applicable.

IDR(1) and Bi-CGSTAB

In [17], the mathematical equivalence between IDR(1) and Bi-CGSTAB is proven, on its turn implying the the equivalence between IDR(1)Stab(ℓ) and BiCGStab(ℓ). This relationship could be of interest in doing further research concerning the (experimental) results that we will see in Part II. It might also provide insight in the convergence properties of IDR-methods, as they are then proven to have their roots in CG, for which convergence bounds based on spectral properties of $\mathbf{A}$ are available.[1]

Standard IDR(s) and IDRStab are Krylov subspace-methods

As experimental results from previous part have shown, the initial choice of the auxiliary vectors are one of the key factors in making the most of IDR-methods. The results stated below provide some insight in the influence of these vectors on the spaces in which the $\mathbf{r}_k$ and the columns of $\mathbf{U}_k$ will reside in future iterations.

Lemma 5.1. ***Involved Spaces***
Let $\mathbf{r}_k$ and $\mathbf{U}_k$ be as defined in the derivation of IDR(s), and let $\mathcal{S}, \mathcal{L} \subset \mathbb{C}^N$ be linear subspaces. Assume $\mathbf{r}_k \in \mathcal{S}, \mathbf{U}_k \in \mathcal{L}$, then

$$\mathbf{r}_{k+1} \in \mathcal{S} + \mathbf{A}\mathcal{S} + \mathbf{A}\mathcal{L} + \mathbf{A}^2\mathcal{L},$$

[1] See, for example [**?**].

and

$$\mathbf{U}_{k+1} \in \big(\sum_{i=0}^{s+1} \mathbf{A}^i\mathcal{L}\big) + \mathbf{A}^{s-1}\mathcal{S} + \mathbf{A}^s\mathcal{S}.$$

Proof. Assume $\mathbf{r}_k \in \mathcal{S}, \mathbf{U}_k \in \mathcal{L}$. We will follow the construction of $\mathbf{r}_{k+1}$ and $\mathbf{U}_{k+1}$ to obtain the desired result. By definition we have

$$\mathbf{r}'_k = \Pi_1\mathbf{r}_k = \mathbf{r}_k - \mathbf{A}\mathbf{U}\sigma^{-1}\mathbf{R}^*\mathbf{r}_k \in \mathcal{S} + \mathbf{A}\mathcal{L},$$

and hence

$$\mathbf{r}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{r}'_k \in \mathcal{S} + \mathbf{A}\mathcal{S} + \mathbf{A}\mathcal{L} + \mathbf{A}^2\mathcal{L}.$$

Next, we inspect $\mathbf{V}_k e_1$;

$$\mathbf{V}_k e_1 = \Pi_0\mathbf{r}'_k = \mathbf{r}'_k - \mathbf{U}_k\sigma^{-1}\mathbf{R}^*\mathbf{A}\mathbf{r}'_k \in \mathcal{S} + \mathcal{L} + \mathbf{A}\mathcal{L},$$

since $\mathbf{Span}(\mathbf{U}_k) \subseteq \mathcal{L}$. Note that when we repeatedly apply the operator $\Pi_0\mathbf{A}$ in order to construct $\mathbf{V}_k e_2, \ldots, \mathbf{V}_k e_s$, the space in which the columns of these matrices reside is, from a Kyrlov subspace perspective, mainly determined by the intermediate multiplications by $\mathbf{A}$ and the $\mathbf{I}$-term in Π_0. That is, for any $1 < q \leq s$ we have

$$\mathbf{V}_k e_q = \Pi_0\mathbf{A}\mathbf{V}_k e_{q-1} = \mathbf{A}\mathbf{V}_k e_{q-1} - \mathbf{U}_k\sigma^{-1}\mathbf{R}^*\mathbf{A}^2\mathbf{V}_k e_{q-1} \in \mathbf{A}\mathcal{Q} + \mathcal{L}$$

with $\mathcal{Q}$ the space in which $\mathbf{V}_k e_{q-1}$ lies. Applying this observation inductively on the result for $\mathbf{V}_k e_1$ yields

$$\mathbf{V}_k e_q \in \big(\sum_{i=0}^{q} \mathbf{A}^i\mathcal{L}\big) + \mathbf{A}^{q-1}\mathcal{S}, \quad \text{hence, } \mathbf{V}_k \in \big(\sum_{i=0}^{s} \mathbf{A}^i\mathcal{L}\big) + \mathbf{A}^{s-1}\mathcal{S}$$

This indeed implies that we have

$$\mathbf{U}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\mathbf{V}_k \in \big(\sum_{i=0}^{s+1} \mathbf{A}^i\mathcal{L}\big) + \mathbf{A}^{s-1}\mathcal{S} + \mathbf{A}^s\mathcal{S}.$$

□

Now as this lemma on itself seems not very informative, we will apply it to the standard case, in which we have $\mathbf{r}_0 \in \mathcal{K}_1(\mathbf{A}, \mathbf{r}_0), \mathbf{U}_0 \in \mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$, to show its use.

Corollary 5.1. ***Standard IDR(s) is a Krylov-subspace method***
With $\mathbf{r}_0 \in \mathcal{K}_1(\mathbf{A}, \mathbf{r}_0), \mathbf{U}_0 \in \mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$, the IDR(s) method as defined this work yields

$$\mathbf{r}_k \in \mathcal{K}_{k(s+1)+1}(\mathbf{A}, \mathbf{r}_0),$$

and

$$\mathbf{A}\mathbf{U}_k \in \mathcal{K}_{(k+1)(s+1)}(\mathbf{A}, \mathbf{r}_0).$$

Proof. Note that for $k = 0$ we indeed have

$$\mathbf{r}_0 \in \mathcal{K}_{k(s+1)+1}(\mathbf{A}, \mathbf{r}_0), \quad \text{and} \quad \mathbf{A}\mathbf{U}_0 \in \mathcal{K}_{(k+1)(s+1)}(\mathbf{A}, \mathbf{r}_0).$$

We proceed by induction and assume that for some k the statement holds. Note this implies that $\mathbf{U}_k \in \mathcal{K}_{(k+1)(s+1)-1}(\mathbf{A}, \mathbf{r}_0)$. For the rest of the proof, we will omit the generators of the involved Kyrlov subspaces for readability reasons as they do not change during the proof, i.e., we write $\mathcal{K}_k$ instead of $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. We have, by applying Lemma 5.1,

$$\mathbf{r}_{k+1} \in \mathcal{K}_{k(s+1)+1} + \mathbf{A}\mathcal{K}_{k(s+1)+1} + \mathbf{A}\mathcal{K}_{(k+1)(s+1)-1} + \mathbf{A}^2\mathcal{K}_{(k+1)(s+1)-1}.$$

Which, for $\mathbf{A}\mathcal{K}_k \subset \mathcal{K}_{k+1}$, simplifies to

$$\mathbf{r}_{k+1} \in \mathcal{K}_{k(s+1)+2} + \mathcal{K}_{(k+1)(s+1)+1} = \mathcal{K}_{(k+1)(s+1)+1}$$

for all $s \geq 1$. Hence,

$$\mathbf{U}_{k+1} \in \big(\sum_{i=0}^{s+1} \mathbf{A}^i \mathcal{K}_{(k+1)(s+1)-1}\big) + \mathbf{A}^{s-1}\mathcal{K}_{k(s+1)+1} + \mathbf{A}^s\mathcal{K}_{k(s+1)+1}.$$

This, on its turn, simplifies to

$$\mathbf{U}_{k+1} \in \mathcal{K}_{(k+1)(s+1)+s} + \mathcal{K}_{k(s+1)+s+1} = \mathcal{K}_{(k+1)(s+1)+s}.$$

Thus implying

$$\mathbf{A}\mathbf{U}_{k+1} \in \mathcal{K}_{(k+1)(s+1)+s+1} = \mathcal{K}_{(k+2)(s+1)}.$$

□

The assurance of standard IDR(s) being an Krylov subspace method basically implies two things:

> Firstly, we may rely on theorem 1.1 (the Cayley-Hamilton theorem) in that the exact solution will be an element of the involved spaces for some k.
>
> Secondly, the convergence of the residual $\mathbf{r}_k$ per MV is bounded from below by GMRES applied to the same problem.

Sonneveld actually gives a proof of IDR(s) being equal to GMRES in terms of convergence for $s \to \infty$. Note, on the other hand, that if we would choose our initial set of auxiliary vectors differently, we may lose the Krylov subspace properties of the method, but at the same time it would (theoretically) enable convergence per MV faster than GMRES. We will see some examples of this in later chapters, as well as references to work that actually predicts this behaviour.

IDR(s) and IDRStab are Petrov-Galerkin methods

In order to position standard IDR(s) and IDRStab in the landscape of Krylov subspace-methods, we prove, based on [12], that they are of the Petrov-Galerkin type. Basically meaning that although the residuals in IDR(s) and IDR(s)Stab are elements of a shrinking sequence of IDR spaces, they are at the same time orthogonal to a growing sequence of Krylov subspaces.

Lemma 5.2. ***Sonneveld spaces and orthogonality***
For Sonneveld spaces we have the following equivalence[2]*:*

$$\mathcal{S}(\mathbf{A}, P_k, \mathbf{R}) = (\mathcal{K}_k(\mathbf{A}^*, \bar{P}_k(\mathbf{A}^*)^{-1}\mathbf{R}))^{\perp}.$$

[2]Here, for a polynomial $P_k(\xi) = \prod_{i=1}^{k}(1+\omega_i\xi)$, we mean with $\bar{P}_k(\xi)$ the polynomial $\bar{P}_k(\xi) = \prod_{i=1}^{k}(1+\bar{\omega}_i\xi)$, where the bar denotes complex conjugation.

Proof. Let $\mathbf{y} \in \mathcal{S}(\mathbf{A}, P_k, \mathbf{R})$, then by definition there exists an $\mathbf{v}$ such that

$$\mathbf{y} = P_k(\mathbf{A})\mathbf{v} \quad \text{with} \quad \mathbf{v} \perp \mathcal{K}_k(\mathbf{A}^*, \mathbf{R}).$$

In particular, we have

$$\forall_{i=1,\ldots,k-1} \mathbf{R}^* \mathbf{A}^i \mathbf{v} = 0.$$

Now note that we also have, as $P_k(\mathbf{A})$ and $\mathbf{A}$ commute with eachother, that for $i = 1, \ldots, k-1$,

$$\mathbf{R}^* P_k(\mathbf{A})^{-1} \mathbf{A}^i P_k(\mathbf{A})\mathbf{v} = \mathbf{R}^* P_k(\mathbf{A})^{-1} P_k(\mathbf{A}) \mathbf{A}^i \mathbf{v} = \mathbf{R}^* \mathbf{A}^i \mathbf{v} = 0$$

Conversely, let $\mathbf{y} \perp \mathcal{K}_k(\mathbf{A}^*, \bar{(P)}_k(\mathbf{A}^*)^{-1}\mathbf{R})$. Then, it follows that

$$\forall_{i=1,\ldots,k-1} \mathbf{R}^* P_k(\mathbf{A})^{-1} \mathbf{A}^i \mathbf{y} = 0.$$

Again, this time exploiting the commutativity of $P_k(A)^{-1}$ and $\mathbf{A}$, we have

$$\forall_{i=1,\ldots,k-1} \mathbf{R}^* \mathbf{A}^i P_k(\mathbf{A})^{-1} \mathbf{y} = 0.$$

The non-singularity now implies that indeed, there exists $\mathbf{v} \perp \mathcal{K}_k(\mathbf{A}^*, \mathbf{R})$ such that $\mathbf{y} = P_k(\mathbf{A})\mathbf{v}$, hence

$$\mathbf{y} \in \mathcal{S}(\mathbf{A}, P_k, \mathbf{R}).$$

□

It is a matter of recalling that in IDR-methods, we have $\mathbf{r}_k \in \mathcal{G}_k = \mathcal{S}(\mathbf{A}, P_k, \mathbf{R})$, to observe that they are indeed of the Petrov-Galerkin type Krylov subspace-methods.

Residual norm reduction

In standard IDR(s), we choose our the relaxation parameters to minimize the next residual norm. It is, however, usually *not* this minimization that may take for account the bulk of the residual norm reduction in an IDR-cycle. More specifically, the convergence properties of IDR-methods are thought to be mainly dominated by the dimension reduction, occurring when constructing $\mathbf{r}'_k = \Pi_1 \mathbf{r}_k$. Figure 5.1 shows the convergence history of both $\mathbf{r}_k$ and $\mathbf{r}'_k$ of the same experiment.

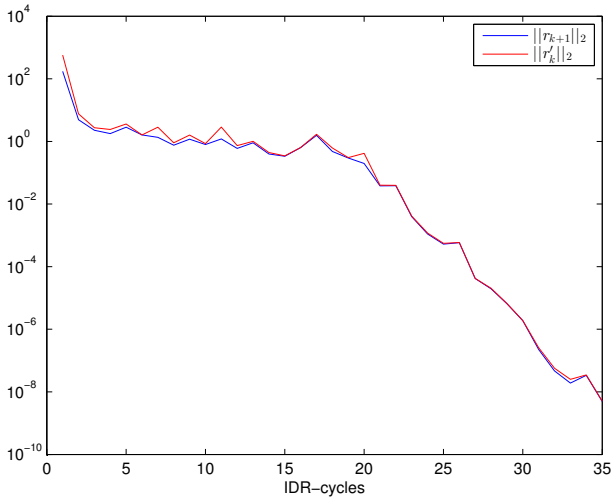


Figure 5.1: *Comparison of the norms of* $\mathbf{r}_{k+1}$ *and* $\mathbf{r}'_k$ *using* SHERMAN4. *Typically, they do not differ much, as shown here. We can infer from this observation that the minimizing property of the relaxation parameters are not the main reason for the fine convergence properties of standard IDR(s). They do however play an important role in terms of stability, as not having this minimization property will lead to a loss of numerical accuracy (see Section 6.1).*

Part II: Recycling

Introduction

Now we have introduced two IDR-methods, this part will be devoted to a technique that we will call **recycling** that may yield significant convergence speed-up in cases in which multiple systems using the same coefficient matrix $\mathbf{A}$ are to be solved. We will focus on only two such systems, but the results in this chapter will naturally translate to sequences of more systems. We will speak of a **seed system**, referring to the system

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

from which we will fetch quantities to recycle. And use the term **recycling system** for the system

$$\mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}},$$

in which we will try to make good use of these quantities. Moreover, in the solving of the recycling system, we will distinguish between quantities that are specific to this run, by placing a tilde above it. For example, the residual appearing in the k-th iteration of solving the recycling system will be denoted by $\tilde{\mathbf{r}}_k$. In particular, we will sometimes omit this tilde to emphasize that both the seed as the recycling run are using the very same quantity.

Chapter 6

Recycling Sonneveld Subspaces

By definition, the used Sonneveld subspaces used in our IDR-methods are not dependent on the right-hand side vector $\mathbf{b}$ of the input problem. This suggests that the used $\mathcal{G}_0, \ldots, \mathcal{G}_k$ may be used as well for the solving of the recycling system with the same $\mathbf{A}$ and $\mathbf{R}$, but different right-hand side, $\tilde{\mathbf{b}}$. In this chapter we will show a intuitive approach to restrict the residuals $\tilde{\mathbf{r}}_k$ of the alternative system to the J-th Sonneveld space from the seed system.

6.1 SRIDR(s)

In work by Neuenhofen [8], a method is presented that is called 'SRIDR(s)', an acronym for Short Recurrence IDR(s). Recall that in IDR(s), we use a matrix of the form $\mathbf{AU}_k$ whose columns lie in $\mathcal{G}_k$. The purpose of its column vectors is to restrict $\mathbf{r}_k$ to $\mathcal{G}_k \cap \mathcal{R}^\perp$ by projection with Π_1. By the nesting property of Sonneveld subspaces, one can derive that (for some $J \in \mathbb{N}$) $\mathbf{AU}_J$, we have $\forall_{k \leq J} \mathbf{AU}_J \in \mathcal{G}_k$. This implies that using $\mathbf{U}_J$ for all the projections Π_1 up to the J-th iteration, will yield us an operator

$$\Pi_1 : \forall_{k \leq J} \;\; \mathcal{G}_k \mapsto \mathcal{G}_k \cap \mathcal{R}^\perp.$$

Hence, if we manage to obtain a residual $\tilde{\mathbf{r}}_k \in \mathcal{G}_k$, this Π_1 meets all the requirements.

Next, note that if we assume that in the recycling run, $\tilde{\mathbf{r}}_0 \in \tilde{\mathcal{G}}_0 \subset \mathcal{G}_0$, and in addition, set $(\tilde{\omega}_k)_{k=1}^J = (\omega_k)_{k=1}^J$, we have

$$\Pi_1 \tilde{\mathbf{r}}_0 = \tilde{\mathbf{r}}_0 - \mathbf{AU}_J(\mathbf{R}^*\mathbf{AU}_J)^{-1}\mathbf{R}^*\tilde{\mathbf{r}}_0 \in \tilde{\mathcal{G}}_0 \cap \mathcal{R}^\perp \subset \mathcal{G}_0 \cap \mathcal{R}^\perp,$$

and hence

$$\tilde{\mathbf{r}}_1 = (\mathbf{I} - \tilde{\omega}_1 \mathbf{A})\Pi_1 \tilde{\mathbf{r}}_0 \in \tilde{\mathcal{G}}_1 \subset \mathcal{G}_1.$$

Similarly, by induction, we have

$$\forall_{k \leq J} \tilde{\mathbf{r}}_k \in \tilde{\mathcal{G}}_k \subset \mathcal{G}_k,$$

proving that, at least in exact arithmetic, by saving $\mathbf{U}_J$ and $(\omega_k)_{k=1}^J$, we can indeed restrict the first J residuals of the alternative run to the first J Sonneveld spaces of the seed run. Also note that this approach may use $\forall_{k \leq J} \tilde{\mathbf{U}}_k = \mathbf{U}_J$, eliminating the need for these iterations to construct a new search matrix. This saves s MV and one DOT per IDR-cycle, excluding

AXPY, resulting in a computationally very cheap method. Simply put, SRIDR(s) is equal to IDR(s), but is applied to the recycling system and uses $\forall_{k\leq J}$:

$$\boxed{\tilde{\mathbf{U}}_k} = \boxed{\mathbf{U}_J}$$

$$\boxed{\mathbf{A}\tilde{\mathbf{U}}_k} = \boxed{\mathbf{A}\mathbf{U}_J}$$

$$\boxed{\tilde{\omega}_k} = \boxed{\omega_k},$$

and will perform regular IDR-cycles for $k > J$.

As a severe drawback, this method does no longer have the freedom to choose any $(\tilde{\omega}_k)_{k=1}^J$, as the sequence is now already fixed. Therefore, we lose the assurance that $\tilde{\omega}_{k+1}$ minimizes $\|\tilde{\mathbf{r}}_{k+1}\|_2$.

Presented in Figure 6.1 are the results of two experiments in which $\mathbf{U}_J, \mathbf{A}\mathbf{U}_J$ and $(\omega)_{k=0}^J$ with from the seed run are recycled. The left plot shows an speed-up in terms of MV, recalculating based on the number of IDR-cycles for the right hand side plot reveals that this does not need to be the case. We can clearly see growth rather than convergence of the residual norm in the first J IDR-cycles, which can be accounted for by the non-minimizing nature of $(\tilde{\omega})_{k=1}^J$.

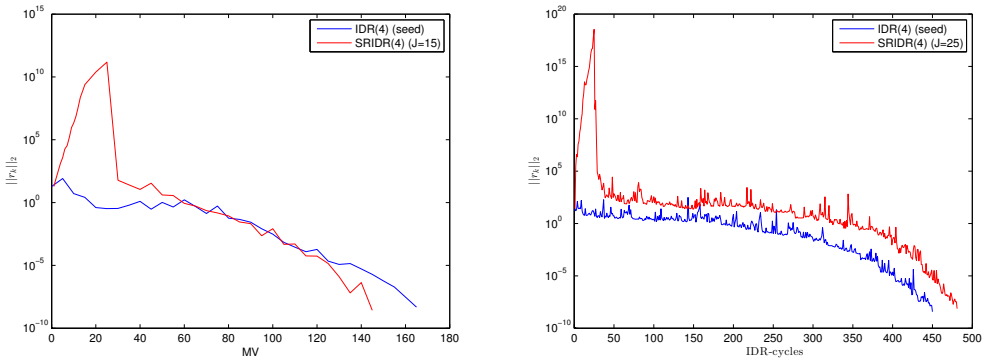


Figure 6.1: *Running SRIDR(s) on* SHERMAN4 *(left,* $J = 15$*) and* DW2048*,* $J = 25$ *(right) recycling* $\mathbf{U}_J, \mathbf{A}\mathbf{U}_J$ *and* $(\omega)_{k=1}^J$*. On the left we see that an actual speed-up is achieved in terms of MV. On the right the reader should focus on the residual growth; the spike of residual growth in the first* J *iterations can be accounted for by the fact that the values of* $\tilde{\omega}_{k+1}$ *are not chosen as minimizers of* $\|\tilde{\mathbf{r}}_{k+1}\|_2$.

An important observation concerning the growth of the residual to be made is that it, in finite arithmetic, leads to a loss of accuracy. Assume the algorithm is running on a computer using 16-decimal arithmetic (Matlab standard), in which the entries of $\tilde{\mathbf{r}}_0$ are represented. If after, say J, iterations the individual entries of $\tilde{\mathbf{r}}_J$ have increased in absolute value with a factor 10^m, the last m decimals of $\tilde{\mathbf{r}}_0$ can not be represented anymore. This implies that we may expect a loss of accuracy of the same factor per entry that is very likely to persist throughout the entire process. Below we see a plot containing the true residuals of the same experiments. Notice in Figure 6.2 how the 'convergence' of the true residuals

in SRDIR(s) stagnate at approximately $6 \cdot 10^3$, while the difference between residual and true residual is much smaller before the stagnation occurs. This can be explained by the difference between regular, recursively computed residuals, and the directly computed true residual; The regular residuals $\mathbf{r}_k$ are not influenced by approximate solutions $\mathbf{x}_k$, whereas the true residuals, $\mathbf{b} - \mathbf{A}\mathbf{x}_k$, are. This mechanism is also studied in [15].

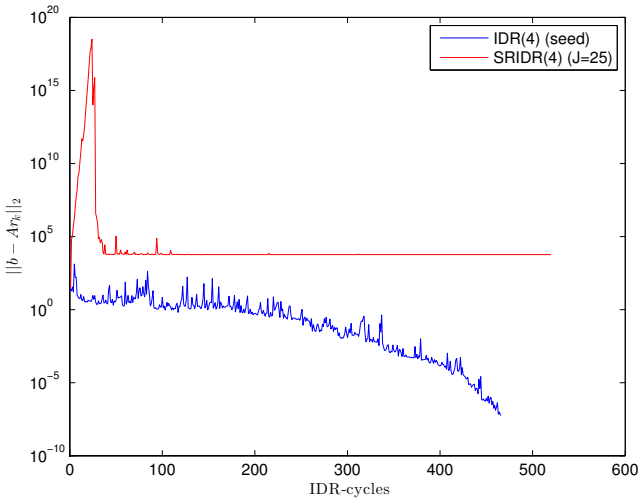


Figure 6.2: *The true residuals reveal the problem with radical growth of residuals when using finite precision arithmetic: The convergence of the true residual norm stagnates as the loss of accuracy in the first J iterations can not be reversed.*

This observation on residual growth will be a topic that we will elaborate on in Section 9.1, when we will be studying a different recycling technique in which we will try to prevent this growth from occurring.

Chapter 7

Recyling of the Auxiliary Vectors

7.1 Miltenbergers Approach and RIDR(s)

Although the recycling of Sonneveld spaces has some clear intuitive advantage, the results of SRIDR(s) are usually very disappointing due to the residual growth in the first J iterations. However, the technique gives rise to an even simpler approach, i.e.: Why not only use $\mathbf{U}_J$ as the initial search matrix of the alternative run(s) so that the minimizing property is still kept intact? This leads us to what we will call 'RIDR(s)', for Recycling IDR(s).

It should also be mentioned that this approach is not new, as it is also studied in [7], by Miltenberger, hence we might also refer to it, in line with [9] as 'Miltenbergers approach'. In our experiments we mainly use a slight variation on this, in which we recycle $\boxed{\mathbf{U}_J}$ as $\boxed{\tilde{\mathbf{U}}_0}$, and calculate $\boxed{\mathbf{A}\tilde{\mathbf{U}}_0}$ as $\boxed{\mathbf{A}} * \boxed{\mathbf{U}_J}$, rather than recycling both quantities, for stability reasons. Note that the intuitive justification of re-using (relatively low-dimensional) Sonneveld spaces does not seem to apply here, hence we will first motivate our investigations to this approach by presenting some numerical results.

Outperforming GMRES

When regarding this recycling technique purely as IDR(s) with a rather specific choice for its initial set of auxiliary vectors in $\mathcal{K}_{J(s+1)+1}(\mathbf{A}, \mathbf{r}_0)$, we see by Lemma 5.1 that $\tilde{\mathbf{r}}_k$ need not be an element of a Krylov subspace brought forth by $\mathbf{A}$ and $\mathbf{r}_0$, just as much as it does not strictly needs to be an element of the Krylov subspace brought forth by $\mathbf{A}$ and $\tilde{\mathbf{r}}_0$. This observation implies that the convergence is not necessarily bounded per MV by GMRES. The following results presented in Figure 7.1 confirm this. Note that as IDR(s) is a short-recurrence method, we also expect that the recycling runs will be outperforming GMRES even more than already is the case with the seed run in terms of CPU-time and memory requirements, but this is not the point here.

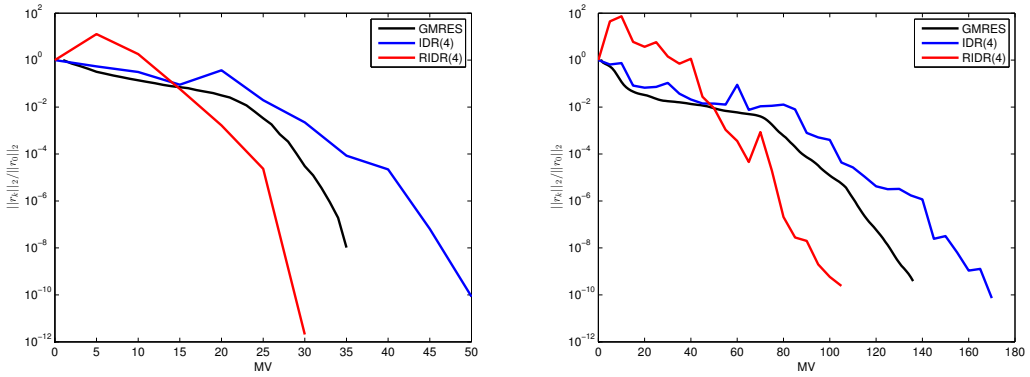


Figure 7.1: *On the left: An experiment showing the result of recycling U_{10} on the 35×35 diagonal matrix ($\mathbf{A} = diag([0.1 : 0.1 : 23 : 17])$, which isMatlab notation for the diagonal matrix with diagonal elements 0.1 up to 2 with step size 0.1, followed by the values 3 up to 17 with step size 1) compared to GMRES. On the right: The same experiment on* SHERMAN4 *and $J = 34$.*

Increasing s

As IDR(1) is equivalent to Bi-CGSTAB, one might wonder whether the possible speed-up is IDR(s) specific or not. The following plot shows the effect of increasing the number of test vectors, s. We can see that for $s = 1$, hardly, if any, any speed-up takes place, whereas for moderate values it occurs with increasing intensity. It might be due to stability issues that for higher s the speed-up is degraded again.

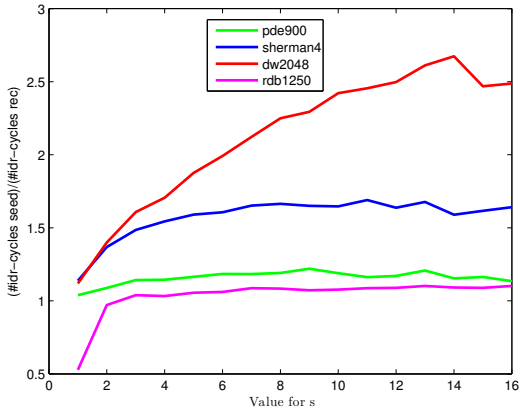


Figure 7.2: *Improvement ratios for different problems. We can witness how increasing s in general increases the positive (a higher ratio corresponds to a greater speed-up) effect of using RIDR(s).*

Varying the fetching point, J

In order to investigate whether the speed-up in the recycling run increases with J, an experiment in which, for equal conditions, multiple search matrices $\mathbf{U}_{J_1}, \ldots, \mathbf{U}_{J_p}$ are fetched,

and used for several runs of RIDR(s) on the recycling system. Intuitively, one might argue that higher values for J should result in more speed-up, more and more information from the previous system will be contained in $\mathbf{U}_J$ (whatever this may be exactly). However, as is pointed out in [9], in exact arithmetic, we expect $\mathbf{A}\mathbf{U}_J \in \mathbf{Span}(\mathbf{0})$ for some J. Hence, using values for J that are close to the total number of IDR-cycles used in the seed run might actually result in recycling merely round-off errors, rather than valuable directions corresponding to $\mathcal{G}_J$.

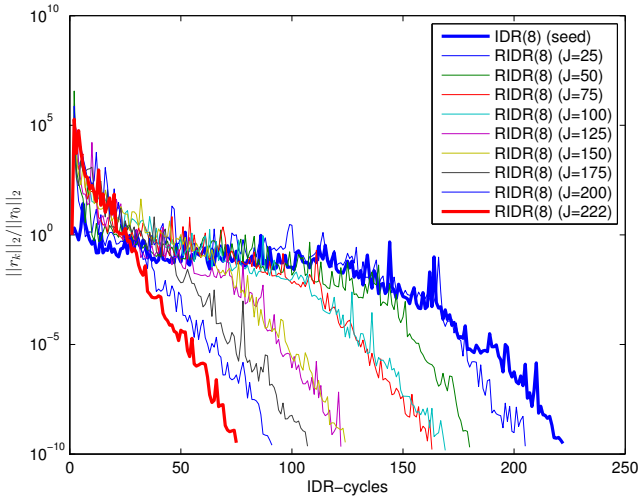


Figure 7.3: *An experiment on* DW2048 *in which the fetching point J is varied. We can see how the convergence of the residual norm speeds up as J increases.*

Figure 7.3 shows that for this problem, we need not worry about the fetching point. Other experiments seems to confirm that this is usually the case; recycling the very last search matrix from the seed run usually yields the best results.

7.2 $\mathcal{M}$-spaces and termination

Recycling merely auxiliary vectors rather than Sonneveld subspaces by not recycling $(\omega)_{k=1}^{J}$, may result in residuals residing in spaces that need not be subspaces of $\mathcal{G}_J$. Therefore, we can not apply the already derived results on termination from Part I to explain the convergence behavior of RIDR(s). In order to justify the performance of RIDR(s), theory has been developed by Neuenhofen, presented in [9], giving theoretical bounds on the number of iterations needed for termination of Miltenbergers approach, and thus the mathematically equivalent RIDR(s). We will summarize the most important aspects in this section.

Definition 7.1. $\mathcal{M}$-spaces
Let $\mathbf{A} \in \mathbb{C}^{N\times N}, (\omega_k)_{k\in\mathbb{N}} \in \mathbb{C}^N \setminus \{0\}$ and let $(\mathcal{Q})_{k\in\mathbb{N}}, (\mathcal{P})_{k\in\mathbb{N}}$ be sequences of subspaces of $\mathbb{C}^N$, with $\forall_k \mathcal{Q}_{k+1} \subseteq \mathcal{Q}_k, \mathcal{P}_k \subseteq \mathcal{P}_{k+1}$, $\mathcal{M}_0 \equiv \mathbb{C}^N$, and recursively

$$\mathcal{M}_{k+1} \equiv (\mathbf{I} - \omega_{k+1}\mathbf{A}) \cdot (\mathcal{M}_k \cap \mathcal{P}_k^{\perp}) + \mathcal{Q}_k.$$

We call the $\mathcal{M}_k$ **$\mathcal{M}$-spaces**, the $\mathcal{Q}_k$ **add spaces**, and the $\mathcal{P}_k$ **test spaces**.

One can readily see similarities between Sonneveld spaces and $\mathcal{M}$-spaces, and the intuition is that we will be using $\mathcal{M}$-spaces as Sonneveld to which we have added some directions. Observe that for $\mathcal{Q}_1 = \{\mathbf{0}\}$, and $\forall_{k\in\mathbb{N}} : \mathbf{P}_k^{\perp} = \mathcal{R}^{\perp}$, the $\mathcal{M}$-spaces generated by these sequences are in fact Sonneveld subspaces. Just like for Sonneveld spaces, one can prove the nestedness of a sequence of $\mathcal{M}$-spaces. We will state a more general result here whilst omitting the proof for it is mainly straight-forward, and can be found in [9].

Theorem 7.1. *Nestedness of $\mathcal{M}$-spaces*
Let $(\mathcal{Q}_k^1)_{k\in\mathbb{N}}, (\mathcal{Q}^2)_{k\in\mathbb{N}}$ and $(\mathbf{P}^1)_{k\in\mathbb{N}}, (\mathbf{P}^2)_{k\in\mathbb{N}}$ be two sequences of add- and test spaces respectively, with $\mathcal{Q}_k^1 \subseteq \mathcal{Q}_k^2$ and $\mathcal{P}_k^2 \subseteq \mathcal{P}_k^1$ for all $k \in \mathbb{N}$. Let $\mathcal{M}_k^i$ be the k-th $\mathcal{M}$-space corresponding to $(\mathcal{Q}^i)_{k\in\mathbb{N}}, (\mathcal{P}^i)_{k\in\mathbb{N}}$, $i = 1, 2$.
Then

$$\forall_{k\in\mathbb{N}} \mathcal{M}_{k+1}^i \subseteq \mathcal{M}_k^i.$$

Moreover, we have

$$\forall_{k\in\mathbb{N}} \mathcal{M}_k^1 \subseteq \mathcal{M}_k^2.$$

With this theorem in place, we may regard the seed run using IDR(s) as using $\mathcal{M}$-spaces, $\mathcal{M}_k^1$, equal to the $\mathcal{G}_k$ by setting $\mathcal{Q}_1^1 = \{\mathbf{0}\}$, and $\forall_{k\in\mathbb{N}} : \mathcal{P}_k^{1\perp} = \mathcal{R}^{\perp}$. Hence, when fetching the auxiliary vectors after J iterations, we have $\mathbf{A}\mathbf{U}_J \in \mathcal{M}_J^1 = \mathcal{G}_J$. Now, consider the following sequence of $\mathcal{M}$-spaces, $(\mathcal{M}^2)_{k\in\mathbb{N}}$, generated by

$$\mathcal{Q}_k^2 = \begin{cases} \{\tilde{\mathbf{r}}_0\} & \text{if} \quad k \leq J \\ \{\mathbf{0}\} & \text{otherwise,} \end{cases}$$

$\forall_{k\in\mathbb{N}} (\mathcal{P}_k^2)^{\perp} = \mathcal{R}^{\perp}$ and $(\omega)_{k=1}^J$ the sequence of relaxation parameters from the seed run. Note that with these conditions, we have by construction of $\mathcal{M}$-spaces

$$\mathbf{A}^i \tilde{\mathbf{r}}_0 \in \mathcal{M}_J^2 \quad \text{with} \quad i = 0, \ldots, J,$$

and by Theorem 7.1, it also holds that

$$\mathcal{G}_J = \mathcal{M}_J^1 \subseteq \mathcal{M}_J^2,$$

and hence

$$\mathbf{A}\mathbf{U}_J \in \mathcal{M}_J^2.$$

Observe furthermore that the construction of $\mathcal{M}$-spaces allow the already derived IDR-algorithms to be used to restrict residuals to the shrinking sequences of $\mathcal{M}$-spaces. In particular, since $\mathbf{A}\mathbf{U}_J, \tilde{\mathbf{r}}_0 \in \mathcal{M}_J^2$, we may assume that the recycling run (RIDR(s)) actually starts its first IDR-cycle in $\mathcal{M}_J^2$, generating a residual $\tilde{\mathbf{r}}_1$ and new auxiliary vectors in $\mathcal{M}_{J+1}^2$, etc. From this point of view, it makes sense to analyze the dimension of the involved $\mathcal{M}$-spaces. For any sequence of $\mathcal{M}$-spaces have:

$$\mathbf{Dim}(\mathcal{M}_{k+1}) \leq \max(0, \mathbf{Dim}(\mathcal{M}_k) - \mathbf{Dim}(\mathcal{P}_k)) - \mathbf{Dim}(\mathcal{Q}_k).$$

With this bound, we can already see that, in general, if for some $k \leq \frac{N}{s}$, $\mathcal{Q}_k = \{\mathbf{0}\}$, we will need an equal amount of IDR-cycles as with our Sonneveld-based IDR-methods in order to arrive at a residual contained in the null-space. But, with analysis of the recycling

run in terms of $\mathcal{M}$-spaces, this also gives us an upper-bound at which the recycling run should terminate; If we fetch $\mathbf{U}_J$ and $\mathbf{AU}_J$, RIDR(s) will start at $\mathcal{M}_J^2$, which, assuming the canonical case, has dimension

$$\mathbf{Dim}(\mathcal{M}_J^2) \leq N - J \cdot s + J = \mathbf{Dim}(\mathcal{G}_J) + J,$$

form which point on, the algorithm will start reducing the dimension with s, every IDR-cycle. Note how, for $s = 1$, this would imply that RIDR(s) actually starts with $\mathbf{Dim}(\mathcal{M}_J^2) \leq N$, and how the effect of the recycling of auxiliary vectors, dimension wise, should increase with if s does.

$\mathcal{M}(s)$**Stab**(ℓ)

Neuenhofen proposes to use IDRStab, rather than IDR(s) for the recycling runs. In the context of the above presented theory the method should then be called $\mathcal{M}$(s)Stab(ℓ). It has the advantages over RIDR(s) on might expect from the higher order minimization. We, however, will mainly use RIDR(s) for our experiments as the difference in performance between $\mathcal{M}(s)$Stab(ℓ) is not ground breaking on the small test matrices used in this thesis, as can be seen in Figure 7.4

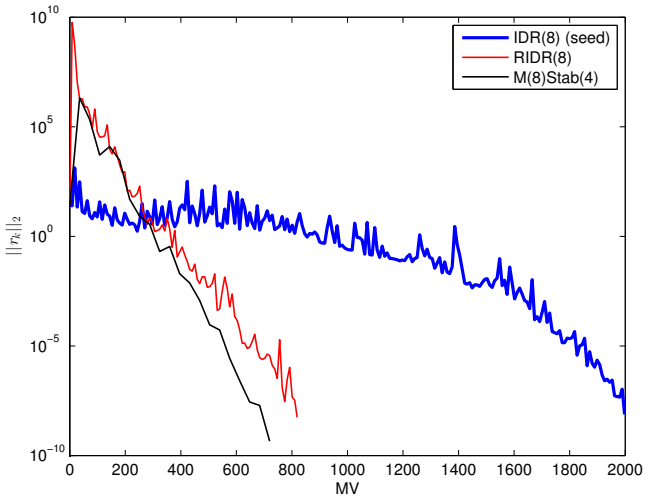


Figure 7.4: *A comparison of RIDR(s) and $\mathcal{M}(s)$Stab(ℓ) on* DW2048. *The advantage in convergence per MV in does not differ much on this test problem.*

7.3 Termination in Practice

Although the above theory in which RIDR(s) is, theoretically, building upon the dimension reduction from the seed run, there are many cases in which other aspects seem to dominate the convergence. In particular, the relaxation parameters seem to have a influential role. As an example, consider the following experimental results in which we compare three restarts on the same problem, each resulting from seed runs using different relaxation parameters. We set $J = 30$, and fetch $\mathbf{U}_{30}^1$ from a regular seed run, in which the parameters are chosen to minimize $\|\mathbf{r}_{k+1}\|_2$ as usual. For comparison, we fetch $\mathbf{U}_{30}^2$ and $\mathbf{U}_{30}^3$ from a seed run in which we have, quite arbitrarily, chosen to set $\omega_k = k$ and $\omega_k = \frac{1}{1000+k}$ respectively. The results are shown in Figure 7.5.

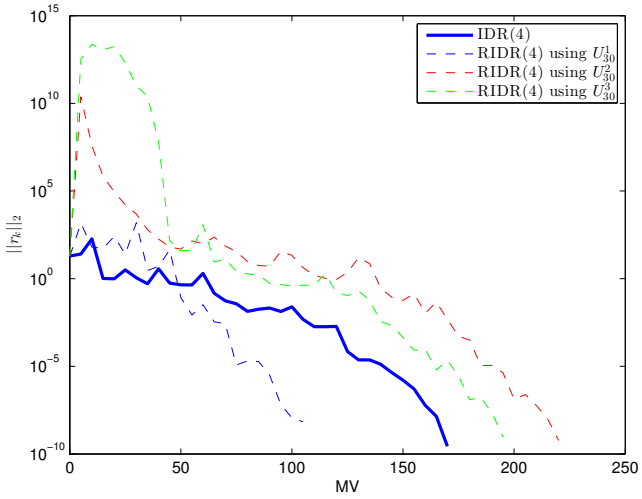


Figure 7.5: *Experiment in which different seeds are used for RIDR(4) on* SHERMAN4. *Clearly, the convergence of residual norm is influences by the choices of the relaxation parameters in the seed run.*

In the runs using $\mathbf{U}_{30}^2$ and $\mathbf{U}_{30}^3$, we can firstly see how the convergence is less prosperous compared to the seed run. Secondly we can distinguish a case of noteworthy growth in the first iteration, which we will study in Chapter 9.

Chapter 8

Recalculating $\mathbf{A}\tilde{\mathbf{U}}_0$

The IDR(s) methods as derived in Part I rely on matrices of the form $\mathbf{AU}_k$. However, in finite precision arithmetic, we may only hope that

$$\boxed{\mathbf{A}} * \boxed{\mathbf{U}_k} \approx \boxed{\mathbf{AU}_k}$$

with some error as small as possible. If the values of $\boxed{\mathbf{A}} * \boxed{\mathbf{U}_k}$ and $\boxed{\mathbf{AU}_k}$ differ much, the values of $\tilde{\mathbf{r}}_1$ and $\tilde{\mathbf{b}} - \mathbf{A}\tilde{\mathbf{x}}_1$ will too, for the simple reason that $\tilde{\mathbf{r}}_1$ is calculated using $\mathbf{AU}_J$ only whereas $\tilde{\mathbf{x}}_1$ is calculated by also using $\mathbf{U}_J$. If the deficiency between $\boxed{\mathbf{AU}_k}$ and $\boxed{\mathbf{A}} * \boxed{\mathbf{U}_k}$ introduces a **residual gap**, defined as the difference between the recursively computed residual and the true residual, of some order 10^m between $\tilde{\mathbf{r}}_1$ and $\tilde{\mathbf{x}}_1$, we do not expect this initial error to get corrected during the process, as the recursive residual updates are independent of $\mathbf{x}_k$. Although it seems advisable in general to recalculate $\boxed{\mathbf{A}\tilde{\mathbf{U}}_0} \leftarrow\!\shortmid \boxed{\mathbf{A}} * \boxed{\mathbf{U}_J}$, experiments show that somehow the **residual gap norm** in the first iterations might shrink during the process, which is surprising, as it is common to assume that this gap is unstructured. Interestingly, this effect seems to increase with increasing s, hinting on it being an IDR(s)-specific phenomenon.

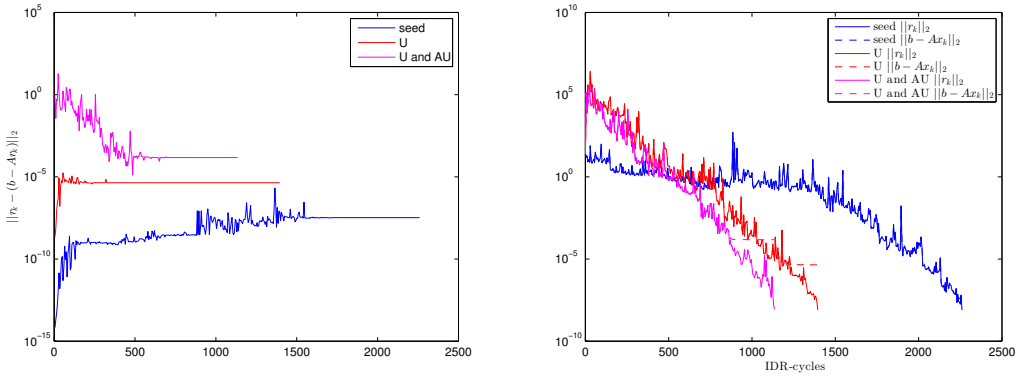


Figure 8.1: *Comparison of recycling only $\mathbf{U}_J$ vs both $\mathbf{U}_J$ and $\mathbf{AU}_J$. On the left the history of the residual graphs is shown. They correspond nicely to the stagnation levels of the true residuals shown together with the recursively computed residuals in the plot on the right.*

To show the immediate influence of the deficiency between $\boxed{\mathbf{U}_J}$ and $\boxed{\mathbf{AU}_J}$, we give the following derivation showing the structure of the first residual gap, $\mathbf{g}_1$. Denote the residuals, σ and approximate solution of the run recycling both $\mathbf{U}_J$ and $\mathbf{AU}_J$ by $\hat{\mathbf{r}}_k, \hat{\sigma}$ and $\hat{\mathbf{x}}_k$ respectively. We have

$$\hat{\mathbf{r}}_1 = \hat{\mathbf{r}}_0 + \hat{N}_0\hat{\mathbf{r}}_0$$

with

$$\hat{N}_0 = -\boxed{\mathbf{AU}_J}\hat{\sigma}^{-1}\mathbf{R}^* - \omega_{k+1}\mathbf{A} + \omega_{k+1}\mathbf{A}\boxed{\mathbf{AU}_J}\hat{\sigma}^{-1}\mathbf{R}^*$$

and similarly

$$\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_0 + \hat{M}_0\hat{\mathbf{r}}_0$$

with

$$\hat{M}_0 = \boxed{\mathbf{U}_J}\hat{\sigma}^{-1}\mathbf{R}^* + \omega_{k+}\mathbf{I} - \omega_{k+1}\boxed{\mathbf{AU}_J}\hat{\sigma}^{-1}\mathbf{R}^*,$$

such that we have for the first residual gap,

$$\mathbf{g}_1 \equiv \hat{\mathbf{r}}_1 - (\tilde{\mathbf{b}} - \mathbf{A}\hat{x}_1) = \tilde{\mathbf{b}} + \hat{N}_0\tilde{\mathbf{b}} - (\tilde{\mathbf{b}} - \mathbf{A}(\hat{\mathbf{x}}_0 + \hat{M}_0\tilde{\mathbf{b}})),$$

which equals, using that $\hat{\mathbf{x}}_0 = \mathbf{0}, \hat{\mathbf{r}}_0 = \tilde{\mathbf{b}}$,

$$\mathbf{g}_1 = \tilde{\mathbf{b}} + \hat{N}_0\tilde{\mathbf{b}} - (\tilde{\mathbf{b}} - \mathbf{A}(\hat{x}_0 + \hat{M}_0\tilde{\mathbf{b}})) = (\hat{N}_0 + \mathbf{A}\hat{M}_0)\tilde{\mathbf{b}}.$$

Hence, if we define $E \equiv \boxed{\mathbf{A}} * \boxed{\mathbf{U}_J} - \boxed{\mathbf{AU}_J}$

$$\mathbf{g}_1 = E\hat{\sigma}^{-1}\mathbf{R}^*\tilde{\mathbf{b}}.$$

Figure 8.2 shows the result of an experiment in which several runs of RIDR(s) versus the variant also recycling $\mathbf{AU}_J$ for several s. Per variant, the runs shared the initial columns of $\mathbf{U}_J$, $\mathbf{AU}_J$ and $\mathbf{R}$, but used only the first s columns of 12 in total, to keep the error-matrix E as constant as possible to make a fair comparison. We can see how in normal RIDR(s), the residual gap increases with the iteration number. For the runs recycling $\mathbf{AU}_J$ however, we see the residual gap decreasing before stagnation. In particular, the effect seems to be more apparent for higher values of s. For $s = 1$ the curve is increasing, as one would normally expect, although these results are chosen because they seem to be a good representation, a decreasing residual gap for $s = 1$ has also been witnessed.

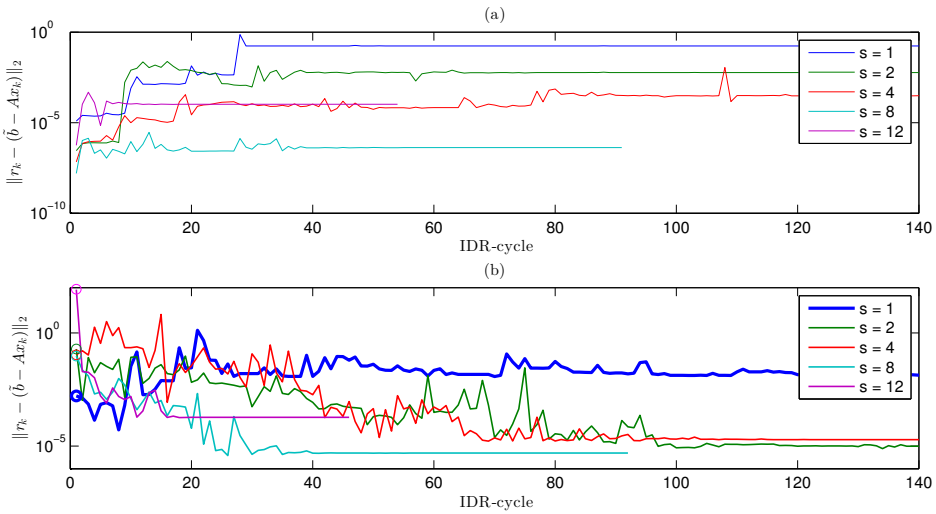


Figure 8.2: *Comparison of recycling only* $\mathbf{U}_J$ *vs both* $\mathbf{U}_J$ *and* $\mathbf{AU}_J$ *for different values of* s*. Plot a) shows the results for only recycling* $\mathbf{U}_J$*, b) for* $\mathbf{AU}_J$*. Note that the residual gap in plot b) is decreasing before is stagnates for most values of* s*.*

Chapter 9

Residual Growth in RIDR(s)

When recycling auxiliary vectors, we will usually have that $\|\tilde{\mathbf{r}}_1\|_2 \approx 10^m \cdot \|\tilde{\mathbf{r}}_0\|_2$ for some m rather large, even for small input problems, leading to a loss of accuracy as mentioned in Chapter 5. By inspection of involved quantities, we usually find that in those cases, $\left\|\mathbf{R}^*\mathbf{A}\tilde{\mathbf{U}}_0\right\|_2 = \|\tilde{\sigma}\|_2 \approx 10^{-m}$. This implies that, roughly, $\mathbf{A}\tilde{\mathbf{U}}_0 \perp \mathbf{R}$. To justify the occurrence of this growth we conjecture the following.

9.1 Growth and Relaxation Parameters

Let $J \in \mathbb{N}$ be some iteration number at which we have fetched $\mathbf{U}_J$ from the seed system. Assuming that we are working in exact arithmetic for the moment, this would imply that $\mathbf{A}\mathbf{U}_J \in \mathcal{G}_J$. Hence, using the definition of Sonneveld subspaces, we can write

$$\mathbf{A}\mathbf{U}_J = P_J(\mathbf{A})\mathbf{W} \quad \text{with} \quad \mathbf{W} \perp \mathcal{K}_J(\mathbf{A}^*, \mathbf{R}).$$

Let $q \in \{1, \ldots, s\}$, and let $\mathbf{w}_q \equiv \mathbf{W}e_q$. Now note that, since $\mathbf{w}_q \perp \mathcal{K}_J(\mathbf{A}^*, \mathbf{R})$ we have

$$\forall_{i<(J-1)} \mathbf{R}^*\mathbf{A}\mathbf{w}_q = \mathbf{0}.$$

Hence, if we write $P_J(\mathbf{A}) = \sum_{i=0}^{J} \alpha_i \mathbf{A}^i$, with coefficients $\alpha_i \in \mathbb{C}$, we yield

$$\mathbf{R}^* P_J(\mathbf{A})\mathbf{w}_q = \sum_{i=0}^{J} \alpha_i \mathbf{R}^*\mathbf{A}^i\mathbf{w}_q = \alpha_J \mathbf{R}^*\mathbf{A}^J\mathbf{w}_q.$$

First, observe that we have

$$\alpha_J = \prod_{i=1}^{J} \omega_i.$$

and second that if we, for simplicity, assume that

$$\left\{\mathbf{y}_i \;\middle|\; i = 1, \ldots, N, \mathbf{y}_i \in \mathbb{C}^N, \|\mathbf{y}_i\|_2 = 1, \mathbf{A}\mathbf{y}_i = \lambda_i \mathbf{y}_i, |\lambda_i| \geq |\lambda_{i+1}|\right\}$$

forms a basis of eigenvectors of $\mathbf{A}$ spanning $\mathbb{C}^N$, we can write

$$\mathbf{w}_q = \sum_{i=1}^{N} \beta_i \mathbf{y}_i \quad \text{with coefficients} \quad \beta_i \in \mathbb{C},$$

we, for a sufficiently high value of J, have, assuming $\beta_1 \neq 0$

$$\mathbf{A}^J \mathbf{w}_q \approx \beta_1 \lambda_1^J \mathbf{y}_1,$$

as repeated multiplication of $\mathbf{A}$ with any vector will ultimately result in a vector pointing in the direction of the eigenvector component present with largest absolute eigenvalue. Now, when we assume that $\forall_{i \leq J} |\omega_i| < \frac{1}{|\lambda_1|}$, we see that $\|\mathbf{R}^* P_J(A)\mathbf{w}_q\|_2$ will be very small for all $1 \leq q \leq s$. Hence, the norm of $\mathbf{R}^*\mathbf{A}\mathbf{U}_J$ we will be very small, thus the norm of σ^{-1} will be very large.

From this rough analysis we can see that the balance between the values of $(\omega)_{k \in \mathbb{N}}$ and the eigenvalues of $\mathbf{A}$ will ultimately govern the growth in the first iteration of the recycling run. A natural question that arises would then be why do we not have this problem in every iteration of the seed run? The answer in this particular theory would be that the problematic factor α_J is exactly cancelled out when the k-th Π_1 is applied to a vector that is an element of $\mathcal{G}_k$, which is the case in IDR(s): We have, in iteration k,

$$\mathbf{r}'_k = \Pi_1 \mathbf{r}_k = r_k - \mathbf{A}\mathbf{U}_k \sigma^{-1} \mathbf{R}^* \mathbf{r}_k.$$

Write

$$\mathbf{A}\mathbf{U}_k = P_J(\mathbf{A})\mathbf{W}, \quad \mathbf{r}_k = P_J(\mathbf{A})\mathbf{v}_k \quad \text{with} \quad \mathbf{W}, \mathbf{v}_k \perp \mathcal{K}_k(\mathbf{A}^*, \mathbf{R}).$$

Now, by the same reasoning as above we have, with $P_J(\mathbf{A}) = \sum_{i=0}^{k} \alpha_i \mathbf{A}^i$,

$$\mathbf{R}^* \mathbf{r}_k = \alpha_k \mathbf{R}^* \mathbf{A}^k \mathbf{v}_k.$$

Note also that we have

$$\sigma = \alpha_k \mathbf{R}^* \mathbf{A}^k \mathbf{W},$$

hence

$$\sigma^{-1} = \alpha_k^{-1} (\mathbf{R}^* \mathbf{A}^k \mathbf{W})^{-1}.$$

Therefore, we can derive that

$$\mathbf{r}'_k = P_J(\mathbf{A})\mathbf{v}_k - P_J(\mathbf{A})\mathbf{W}(\alpha_k^{-1}(\mathbf{R}^*\mathbf{A}^k\mathbf{W})^{-1})\alpha_k \mathbf{R}^*\mathbf{A}^k\mathbf{v}_k =$$

$$P_J(\mathbf{A})\mathbf{v}_k - P_J(\mathbf{A})\mathbf{W}(\mathbf{R}^*\mathbf{A}\mathbf{W})^{-1}\mathbf{R}^*\mathbf{A}^k\mathbf{v}_k.$$

Ultimately yielding, if you will,

$$\mathbf{r}'_k = P_J(\mathbf{A})(\mathbf{I} - \mathbf{W}(\mathbf{R}^*\mathbf{A}^k\mathbf{W})^{-1}\mathbf{R}^*\mathbf{A}^k)\mathbf{v}_k.$$

From the but last expression we can see that the mechanism causing the growth in the recycling run, no longer plays its role. Furthermore, in terms of norm-wise growth, we may hope that the higher powers of $\mathbf{A}$ are compensated for by higher powers of $\mathbf{A}^{-1}$.

9.2 Experimental verification

In order to verify the above theory, we present below the results of two experiments. Both have the same set-up, but use a different problem as input. We will be inspecting the 2-norms of three quantities, after J iterations. That is (using the theory presented above):

1) $\sigma^{-1} = (\mathbf{R}^*\mathbf{A}\mathbf{U}_J)^{-1} = (\alpha_J\mathbf{R}^*\mathbf{A}^J\mathbf{W})^{-1} = \frac{1}{\alpha_J}(\mathbf{R}^*\mathbf{A}^J\mathbf{W})^{-1}$. We expect that the factor $\frac{1}{\alpha_J}$ plays an important role in the norm of this quantity, as well as in $\|\tilde{\mathbf{r}}_1\|_2$.

2) $\tilde{\mathbf{r}}_1$, the first residual (after $\tilde{\mathbf{r}}_0$) of the recycling run.

3) $\mathbf{R}^*\hat{r}_J = \alpha_J\mathbf{R}^*\mathbf{A}^J\hat{v}_k$, with $\hat{\mathbf{r}}_J \equiv \frac{\mathbf{r}_J}{\|\mathbf{r}_J\|_2}$.

Now, in the first experiment, we will perform IDR(s) with the following inputs

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & & & & \\ & 1 & 1 & & 0 & \\ & & \ddots & \ddots & & \\ & 0 & & & & 1 \\ & & & & & 1 \end{pmatrix} \in \mathbb{C}^{20\times 20}.$$

We choose $\mathbf{b}$ and $\tilde{\mathbf{b}}$ random but normalized, $\mathbf{x}_0 = \mathbf{0}$. Note that as $\|\hat{\mathbf{r}}_J\|_2 = \|\tilde{\mathbf{r}}_0\|_2 = 1$, we may hope to see a clear distinction between the effect of $\hat{\mathbf{r}}_J \in \mathcal{G}_J$ and $\tilde{\mathbf{r}}_0 \notin \mathcal{G}_J$ We set $s = 1$ and $\mathbf{R} = \frac{\vec{\mathbf{1}}}{\|\vec{\mathbf{1}}\|_2}$. This last choice is made to ensure that left-multiplication with $\mathbf{R}^*$ does not annihilate values in the quantities to be measured, as could be the case with a randomly chosen $\mathbf{R}$. Now, for $J = 5$, we perform J iterations of IDR(s) in which we do **not** choose ω_k to be minimizing the norm of $\mathbf{r}_{k+1}$, but rather set them to fixed values, and fetch $\mathbf{U}_J$. We let $(\omega_k)_{k=1}^J$ vary from 10^2 down to 10^{-3} with step size -0.2 in the exponent, resulting in values for α_J varying from 10^{10} down to 10^{-15}.
Now note that the eigenvalues of $\mathbf{A}$ are all equal to 1, which makes the terms containing higher powers of $\mathbf{A}$ a bit more controllable, norm-wise. This, in conjunction with $\mathbf{b}$ and $\tilde{\mathbf{b}}$ (and hence $\tilde{\mathbf{r}}$)) being vectors of unity, makes that we may expect the following for the quantities to be measured:

1) $\left\|\sigma^{-1}\right\|_2 \approx \alpha_J$.

2) $\|\tilde{\mathbf{r}}_1\|_2 \approx \alpha_J$.

3) $\|\mathbf{R}^*\hat{r}_J\|_2 \approx \frac{1}{\alpha_J}$.

Figure 9.1 summarizes the results and verifies the expectations for the cases in which $|\omega_k| < |\lambda_1| = 1$, as the graph of $\left\|\sigma^{-1}\right\|_2$ and $\|\tilde{\mathbf{r}}_1\|_2$ nicely coincide for these values for ω_k, and the graph of $\|\mathbf{R}^*\hat{\mathbf{r}}_5\|_2$ showing the inverse movement as expected.

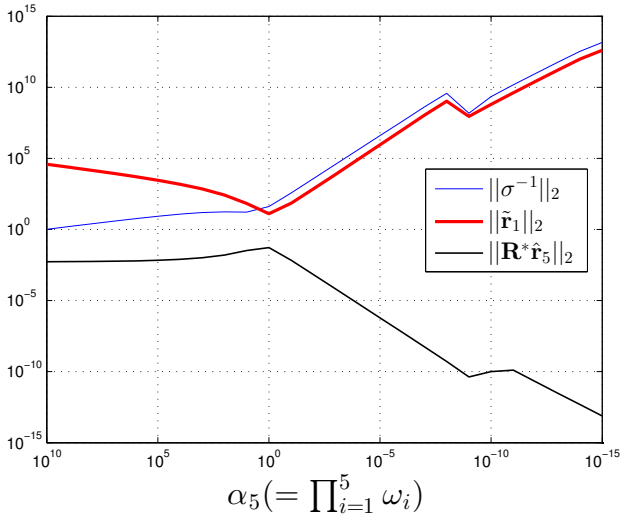


Figure 9.1: *The measured history of the quantities from a run on a* 20×20 *bi-diagonal matrix. We can observe that using small values for the relaxation parameters indeed causes growth in the first residual in the recycling run as predicted.*

Repeating the same experiment for SHERMAN4 shows the same pattern, as shown in Figure 9.2

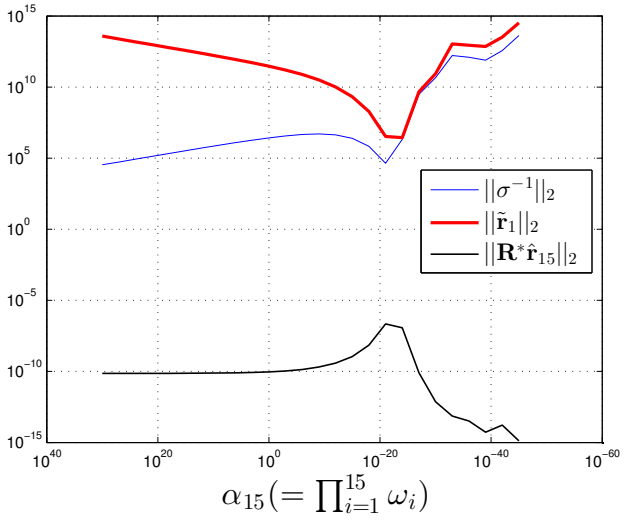


Figure 9.2: *The same experiment as shown in Figure 9.1, but using* SHERMAN4 *as a test problem. Again, notice the growth for smaller values for the relaxation parameters.*

Note that both graphs also show growth of $\|\tilde{\mathbf{r}}_1\|_2$ for large values of α_J, and thus ω_i. This can be justified by realizing that the algorithm uses multiple multiplications with

$(\mathbf{I} - \omega_{k+1}\mathbf{A})$ to construct $\tilde{\mathbf{r}}_{k+1}$. In particular, we have that

$$\mathbf{U}_J = (\mathbf{I} - \omega_k \mathbf{A})\mathbf{V} \quad \text{with } \mathbf{V} \text{ orthonormal},$$

hence $\|\mathbf{V}\|_2 = 1$. Now, numerical inspection of the spectrum of SHERMAN4 yields us that $\Lambda(\mathbf{A}) \subset [1, 67]$, hence we may expect that $\|w_i \mathbf{A}\|_2 \approx \omega_i \cdot 67$. Now focusing on the $\tilde{\mathbf{r}}_1$;

$$\tilde{\mathbf{r}}_1 = (\mathbf{I} - \tilde{\omega}_1 \mathbf{A})(\tilde{\mathbf{r}}_0 - \mathbf{A}(\mathbf{I} - \omega_J \mathbf{A})\mathbf{V}\sigma^{-1}\mathbf{R}^* \tilde{\mathbf{r}}_0),$$

and noting that we have used $\tilde{\omega}_1 = \omega_J$, simply calculating norms of the individual terms yield us results that are indeed of the same order as shown in the graphs. Hence, choosing ω_k very large, blows up the residual norms, as there is a lack of compensation from σ^{-1}. Finally, note that the effect on SHERMAN4 is most evident for $\alpha_J < 10^{-20}$, meaning that the values for ω_i used were smaller than $\sqrt[15]{10^{-20}} \approx \frac{1}{22}$. This value could be explained by the fact that its inverse is lies in $[1, 67] \subset \mathbb{R}$, and therefore it could very well be that it corresponds to a combination of eigenvalues of $\mathbf{A}$ that correspond to the eigenvectors making up $\mathbf{W}$, similar to our analysis using the eigenvectors with largest (absolute) eigenvalue.
We conclude this chapter with a second experiment in which we compare two recycled runs on the same bi-diagonal matrix of previous experiments., each recycling a different search matrix; a regular one, and one resulting from a run with fixed, and constant values for ω_k, that we will call the *special* seed. We can observe that the run recycling the special search matrix does not exhibit growth in the first iteration. Of course, this example is rather clinical, and this result does not easily apply to larger matrices with more daunting spectra, which would make the results much harder to interpret correctly.

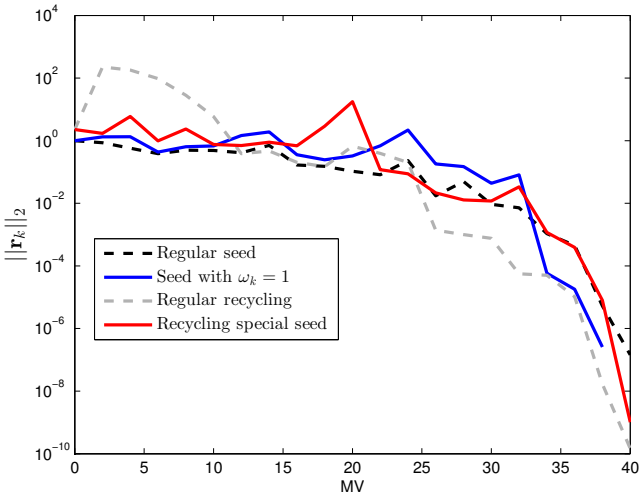


Figure 9.3: *The theory in action on a 20x20 bi-diagonal matrix. We can witness the lack of growth in the first iteration of the run recycling the special seed, which can be explained by having $\alpha_J = 1$. We may for this reason expect a more accurate result from the special recycling run, even if the convergence takes equally many IDR cycles.*

Chapter 10

The IDR Projection Theorem

The theory on termination developed by Neuenhofen (Section 7.2) gives us theoretical upper-bounds, but does not provide much insight into the mechanisms and numerical aspects the make RIDR(s) fail in some cases and show convergence much faster than predicted in other cases. In this chapter we will try to apply theory on spectra of involved matrices to explain the prosperous convergence results, leading to perhaps an plausible theory, that, unfortunately, seems to be contradicted by numerical experiments.

10.1 Theory on Spectra

In a paper by Collignon, Sleijpen and Van Gijzen ([1]) theory is presented in which an IDR-cycle is interpreted as a single step of what is called Richardson iteration on a pre-conditioned system. Richardson iteration is a simple recursive scheme resulting in a Krylov subspace-method, that is summarized by the following equations.

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \mathbf{A}\mathbf{r}_k = (\mathbf{I} - \mathbf{A})\mathbf{r}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{r}_k.$$

We can see that when all eigenvalues of $\mathbf{I} - \mathbf{A}$ are strictly smaller than 1 in absolute value, the residual norm will gradually shrink to 0, at which point the method has found the exact solution $\mathbf{x}$. Since we may not expect from a problem in general that $\mathbf{I} - \mathbf{A}$ meets this requirement, a common technique[1] known as **deflation** is used which consists of **pre-conditioning** the problem with a matrix $\mathbf{P}$ such that $\mathbf{PA}$ does have a spectrum with the desired property as much as possible. Now, pre-conditioning is a field of research on its own that is beyond the scope of this thesis, but we will summarize the aspects that are important for this chapter. Restricting ourselves to this particular situation, we will regard an IDR-cycle as a Richardson step applied to the system

$$\mathbf{PAx} = \mathbf{Pb},$$

in which $\mathbf{P}$ is a **deflation** type pre-conditioner, meaning that (part of) its purpose is to remove certain unwanted eigenvalues from the spectrum of $\mathbf{A}$. Usually, in the context of Krylov subspace methods, one tries to cluster the eigenvalues of $\mathbf{A}$ away from zero, as

[1] which is not exclusive to Richardson iteration but is commonly used within the context of Krylov subspace methods

small eigenvalues hamper the convergence in CG-type methods. We will first show that the structure of a Richardson step applied to a deflated system is indeed equivalent to an IDR-cycle, before shifting our attention to the spectrum of this deflated $\mathbf{A}$.

IDR-cycle as Richardson step

First, we re-write some quantities to make the Richardson structure more apparent. Define

$$\mathbf{Q}_k \equiv \mathbf{U}_k(\mathbf{R}^*\mathbf{A}\mathbf{U}_k)^{-1}\mathbf{R}^*, \quad \text{and} \quad \mathbf{P}_k \equiv \omega_{k+1}\Pi_1 + \mathbf{Q}_k,$$

we then have

$$\mathbf{r}_{k+1} = (\mathbf{I} - \omega_{k+1}\mathbf{A})\Pi_1\mathbf{r}_k = \mathbf{r}_k - \mathbf{A}\mathbf{Q}_k\mathbf{r}_k - \omega_{k+1}\mathbf{A}\mathbf{r}_k + \omega_{k+1}\mathbf{A}^2\mathbf{Q}_k\mathbf{r}_k = \mathbf{r}_k - \mathbf{A}\mathbf{P}_k\mathbf{r}_k,$$

and

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{P}_k(\mathbf{b} - \mathbf{A}\mathbf{x}_k).$$

Without elaborating too much on the details here (they are all explained in [1]), $\mathbf{P}_k$ can be seen as a variable deflation-type pre-conditioner. Variable, in the sense that $\mathbf{P}_k$ is dependent on the IDR-cycle or iteration number k. Interestingly, it turns out that, in IDR(s), we can influence the spectrum of $\mathbf{P}_k\mathbf{A}$ during the iterative process. In particular, it turns out that $\Lambda(\mathbf{P}_k\mathbf{A})$ is related to $\Lambda(\mathbf{P}_{k+1}\mathbf{A})$. We will continue by stating the theorem that is central in this chapter, followed by its implications and numerical experiments based on which we will conjecture the positive influence of these mechanisms in the case of RIDR(s).

Spectral aspects

To ease notation, define for the remainder of this chapter

$$\mu_i \equiv \frac{1}{\omega_i},$$

for the sequence of relaxation parameters $(\omega)_{k\in\mathbb{N}_+}$ used in IDR(s). We start by restating a theorem which can be found in [1].

Theorem 10.1. ***The IDR Projection Theorem***
Let $k \in \mathbb{N}$, then for some $n \times s$-matrix $\mathbf{W}\perp\mathcal{K}_k(\mathbf{A}^*, \mathbf{R})$ and $\tilde{P}_k(\xi) \equiv \prod_{i=1}^{k}(\mu_i - \xi)$, we have

$$\Lambda(\Pi_1\mathbf{A}) = \big\{0\big\} \cup \big\{\lambda \;\big|\; \tilde{P}_k(\lambda) = 0\big\} \cup \big\{\lambda \;\big|\; \mathbf{Det}(\mathbf{R}^*(\mathbf{A} - \lambda\mathbf{I})^{-1}\mathbf{W}) = 0\big\}.$$

In particular, the zero eigenvalue, and the zeros of $\tilde{P}_k$ have geometric multiplicity s.

Proof. (*We will prove the part of the IDR-theorem that we will use in the remainder of this chapter. That is: We omit the prove of $\big\{\lambda \;\big|\; \mathbf{Det}(\mathbf{R}^*(\mathbf{A} - \lambda\mathbf{I})^{-1}\mathbf{W}) = 0\big\} \subset \Lambda(\mathbf{P}_k\mathbf{A})$ (It can be found in [1]), and prove only that $\big\{0\big\} \cup \big\{\lambda \;\big|\; \tilde{P}_k(\lambda) = 0\big\} \subset \Lambda(\Pi_1\mathbf{A})$, and that the elements in these subsets have geometric multiplicity s.*)
Firstly, by definition of Π_1 we have

$$\Pi_1\mathbf{A}\mathbf{U}_k = \mathbf{A}\mathbf{U}_k - \mathbf{A}\mathbf{U}_k(\mathbf{R}^*\mathbf{A}\mathbf{U}_k)^{-1}\mathbf{R}^*\mathbf{A}\mathbf{U}_k = \mathbf{A}\mathbf{U}_k - \mathbf{A}\mathbf{U}_k = \mathbf{O},$$

hence the columns of $\mathbf{U}_k$ form s independent eigenvectors of $\Pi_1\mathbf{A}$ with eigenvalue 0. Note that $\mathbf{AU}_k \in \mathcal{G}_k$ implies that for any iteration k we have, by the definition of Sonneveld subspaces,

$$\mathbf{AU}_k = P_k\mathbf{W} \quad \text{with} \quad \mathbf{W}\perp\mathcal{K}_k(\mathbf{A}^*,\mathbf{R}),$$

Through re-scaling with a factor $\left(\prod_{i=1}^{k}\frac{1}{\omega_i}\right)$, can thus also state that

$$\mathbf{AU}_k = \tilde{P}_k\tilde{\mathbf{W}} \quad \text{with} \quad \tilde{\mathbf{W}}\perp\mathcal{K}_k(\mathbf{A}^*,\mathbf{R}),$$

and hence, for any $\nu \in \{\mu_1,\ldots,\mu_k\}$ (the zeros of $\tilde{P}$),

$$\mathbf{AU}_k = (\nu\mathbf{I}-\mathbf{A})\mathbf{W}' \quad \text{with} \quad \mathbf{W}'\perp\mathcal{R}^{\perp},$$

and

$$\mathbf{W}' \equiv (\nu\mathbf{I}-\mathbf{A})^{-1}\tilde{P}_k\tilde{\mathbf{W}}.$$

(Note that linear factors $(\mu_i\mathbf{I}-\mathbf{A})$ commute with each other.)
We now have

$$(\nu\mathbf{I}-\Pi_k\mathbf{A})\mathbf{W}' = (\nu\mathbf{I}-\mathbf{A})\mathbf{W}' + \mathbf{AQ}_k\mathbf{AW}' =$$
$$(\nu\mathbf{I}-\mathbf{A})\mathbf{W}' + \mathbf{AU}_k(\mathbf{R}^*\mathbf{AU}_k)^{-1}\mathbf{R}^*\mathbf{AW}' =$$
$$(\nu\mathbf{I}-\mathbf{A})\mathbf{W}'(\mathbf{I}+(\mathbf{R}^*\mathbf{AU}_k)^{-1}\mathbf{R}^*\mathbf{AW}').$$

Next, note that $\mathbf{R}^*\mathbf{AU}_k = \mathbf{R}^*(\nu\mathbf{I}-\mathbf{A})\mathbf{W}' = \nu\mathbf{R}^*\mathbf{W}' - \mathbf{R}^*\mathbf{AW}'$, and since $\mathbf{W}'\perp\mathcal{R}^{\perp}$, this equals $-\mathbf{R}^*\mathbf{AW}'$. Hence

$$(\nu\mathbf{I}-\Pi_k\mathbf{A})\mathbf{W}' = (\nu\mathbf{I}-\mathbf{A})\mathbf{W}'(\mathbf{I}-(\mathbf{R}^*\mathbf{AW}')^{-1}\mathbf{R}^*\mathbf{AW}') = \mathbf{O}.$$

We may conclude that $\mathbf{W}'$ is in the kernel of $(\nu\mathbf{I}-\Pi_1\mathbf{A})$, hence the columns of $\mathbf{W}'$ are eigenvectors with eigenvalue ν. Therefore ν is an eigenvalue of $\Pi_1\mathbf{A}$ with geometric multiplicity s.

□

Summarizing, IDR Projection theorem relates all the chosen ω_k (and hence μ_k) to the eigenvalues of $\Pi_1\mathbf{A}$. In particular, we see that there is a build up of clusters of size s of eigenvalues in $\Lambda(\Pi_1\mathbf{A})$ as more IDR-cycles are performed. The reason why we might be interested in these eigenvalues starts with the following lemma.

Lemma 10.1. *For all IDR-cycles we have*

$$\Lambda(\Pi_1\mathbf{A}) = \{0\} \cup \{\lambda_{s+1},\ldots,\lambda_N\}$$
$$\Rightarrow$$
$$\Lambda((\Pi_1+\mathbf{Q}_k)\mathbf{A}) = \{1\} \cup \{\lambda_{s+1},\ldots,\lambda_N\}$$

where the eigenvalues 0 and 1 both have multiplicity s.

Proof. Note that for $\mathbf{U}_k e_i$ with $i = 1, \ldots, s$, we have $(\Pi_1 + \mathbf{Q}_k)\mathbf{A}\mathbf{U}_k = \mathbf{U}_k$, and, as mentioned in the proof of theorem 10.1, $\Pi_1\mathbf{A}\mathbf{U}_k = \mathbf{O}$. Hence the eigenvectors of $\Pi_1\mathbf{A}$ corresponding to zero are the same eigenvectors of $(\Pi_k + \mathbf{Q}_k)\mathbf{A}$ that correspond to 1. For $i = s+1, \ldots, N$, let $(\mathbf{v}_i, \lambda_i)$ be an eigenpair such that $\Pi_1\mathbf{A}\Pi_0\mathbf{v}_i = \lambda_i\Pi_0\mathbf{v}_i$. (Recall that $\Pi_1\mathbf{A} = (\Pi_1)^2\mathbf{A} = \Pi_1\mathbf{A}\Pi_0$.) Then, since

$$\mathbf{Q}_k\mathbf{A}\Pi_0 = \mathbf{O},$$

we have

$$(\Pi_1 + \mathbf{Q}_k)\mathbf{A}\Pi_0\mathbf{v}_i = \Pi_1\mathbf{A}\Pi_0\mathbf{v}_i + \mathbf{Q}_k\mathbf{A}\Pi_0\mathbf{v}_i = \lambda_i\Pi_0\mathbf{v}_i.$$

Hence, excluding the zeros that are replaced by ones, the eigenvalues of $(\Pi_k + \mathbf{Q}_k)\mathbf{A}$ are the same as the eigenvalues of $\Pi_1\mathbf{A}$ with eigenvectors $\Pi_k\mathbf{v}_i$. □

Having this lemma in place, we can describe the spectrum of $\mathbf{P}_k\mathbf{A}$, which is of direct influence in the 'Richardson step'

$$\mathbf{r}_{k+1} = (\mathbf{I} - \mathbf{P}_k\mathbf{A})\mathbf{r}_k.$$

Corollary 10.1. ***Spectrum of*** $\mathbf{P}_k\mathbf{A}$
For $k \in \mathbb{N}$ we have

$$\Lambda(\mathbf{P}_k\mathbf{A}) = \big\{1\big\} \cup \big\{\omega_{k+1} \cdot \mu_i \;\big|\; i = 1, \ldots, k\big\} \cup \big\{\omega_{k+1} \cdot \lambda \;\big|\; \mathbf{Det}(\mathbf{R}^*(\mathbf{A} - \lambda\mathbf{I})^{-1}\mathbf{W}) = 0\big\}.$$

Proof. Let $k \in \mathbb{N}$ and let$(\lambda, \mathbf{v})$ be an eigenpair of $\Pi_k\mathbf{A}$. Then

$$\mathbf{P}_k\mathbf{A}\mathbf{v} = (\omega_{k+1}\Pi_k + \mathbf{Q}_k)\mathbf{A}\mathbf{v} = ((\omega_{k+1}+1-1)\Pi_k + \mathbf{Q}_k)\mathbf{A}\mathbf{v} = \omega_{k+1}\Pi_k\mathbf{A}\mathbf{v} + (\Pi_k + \mathbf{Q}_k)\mathbf{A}\mathbf{v} - \Pi_k\mathbf{A}\mathbf{v}.$$

Now by lemma 10.1 and theorem 10.1 it follows that if $\lambda = 0$, $\mathbf{P}_k\mathbf{A}\mathbf{v} = \mathbf{v}$, and if $\lambda \neq 0$, $\mathbf{P}_k\mathbf{A}\mathbf{v} = \omega_{k+1}\lambda\mathbf{v}$. □

We may thus conclude that, in IDR, the spectrum of $\mathbf{P}_k\mathbf{A}\mathbf{v}$ is related to the sequence ω. Note that the corollary actually states that the eigenvalues of $\mathbf{P}_k\mathbf{A}$ will get increasingly clustered during the process. Since the above mentioned spectra are fully governed by the auxiliary vectors $\mathbf{U}_k$, the coefficient matrix $\mathbf{A}$, and the test vectors $\mathbf{R}$, we may suspect these results to be related to the convergence of a run of RIDR(s). By this we mean, while we do not explicitly reuse $(\omega)_{i=1}^{J}$, the sequence already had its spectral impact on $\tilde{\mathbf{P}}_1\mathbf{A}$ and hence $\tilde{\mathbf{P}}_k\mathbf{A}$, for any k. We will see that we can not simply extend the proof of the IDR projection theorem to apply to RIDR(s) in a sense that the spectrum corresponding to the seed run will be retained, although experimental results seem to suggest exactly that. After that, we will try to understand why this clustering might be beneficial.

10.2 The IDR Projection and RIDR(s)

Note that in the proof of the IDR Projection Theorem, we have repeatedly used that $\mathbf{A}\mathbf{U}_k \in \mathcal{G}_k$ by stating that $\mathbf{A}\mathbf{U}_k = Q(A)\mathbf{W}$ for some polynomial Q and some $n \times s$-matrix $\mathbf{W}$ with some orthogonality properties. Now note that when performing a restart, we need not have $\tilde{\mathbf{r}}_0 \in \mathcal{G}_J$. That is, the first residual of the restarted run, does not need to be an element of the Sonneveld space from which we have fetched our auxiliary vectors. Now, using Lemma 5.1, it follows that if we try to express $\mathbf{A}\tilde{\mathbf{U}}_1$ in terms of $\mathcal{G}_J$, we are forced to include the space spanned by $\tilde{\mathbf{r}}_0$, as was exactly the use of the $\mathcal{M}$-spaces. This eliminates the possibility

to express $\mathbf{A}\tilde{\mathbf{U}}_1$ as some polynomial of the form $\tilde{P}_1 P_J$ with P_J the stabilization polynomial of the seed-run up to the J-th iteration.
It is unfortunate that we can not (and have not succeeded in) extend(ing) the scope of the IDR projection theorem to a run of RIDR(s), as we will see in experiments that there is a clear indication that much of the spectral structure of $\mathbf{P}_J\mathbf{A}$ is preserved in the beginning phase of the recycling run, which could possibly be related to the speed-up in convergence.

Numerical results hinting on preservation of spectral structure in RIDR(s)

Although we hare lacking in a theoretical ground to expect $\Lambda(\tilde{\mathbf{P}}\mathbf{A})$ to be related to ω (the relaxation parameters from the seed run). We can, by performing experiments and numerically inspecting $\Lambda(\tilde{\mathbf{P}}\mathbf{A})$, see why the IDR Projection Theorem seems like a nice starting point in explaining the, sometimes impressive, convergence behaviour of RIDR(s). Below the results of two similar experiments are shown, in which we compare the spectra at different iterations in a seed-run with a corresponding restarted run on the same problem but with a different right-hand side, $\tilde{\mathbf{b}}$.
The experiments shown were set-up as follows. For some coefficient matrix $\mathbf{A}$, two orthogonal right-hand side vectors , $\mathbf{b}$ and $\tilde{\mathbf{b}}$, were generated. These right-hand side vectors are chosen to be orthogonal to each other, to emphasize that the vectors may have as little as possible to do with each other. We choose test vectors random and orthonormalize $\mathbf{R}$ as we always do in standard IDR(s). Then, first a regular run of IDR up to some iteration J is done, and for several $i \in \mathbb{N}$ we fetch $\mathbf{U}_k$ for $k = 1, \ldots, i, J-i, \ldots, J$. With these $\mathbf{U}_k$, we construct the corresponding Π_1 and $\mathbf{P}_k\mathbf{A}$, of which Matlab's '**eig**' function is used to obtain approximations to all the eigenvalues of $\mathbf{P}_k\mathbf{A}$. We define E_k to be the array of eigenvalues returned by $\mathbf{eig}(\mathbf{P}_k\mathbf{A})$. This gives us an overview of the spectra in both the starting and ending fase of the seed run. Next, we perform a restarted run using $\tilde{\mathbf{U}}_0 = \mathbf{U}_J$, and for the same i, fetch the $\tilde{\mathbf{U}}_k$ for $k = 1, \ldots, i$. We construct $\tilde{\mathbf{P}}_k\mathbf{A}$ and obtain arrays $\tilde{E}_k$ containing approximations to its eigenvalues. Next, to account for the effects of working in finite arithmetic, we set some *cluster criterion* $\kappa \in \mathbb{R}$ that will act as the maximum Euclidean distance that two eigenvalues of the same operator may have in order to regard them as being in the same cluster. Using this criterion, we let an algorithm loop over E_k and $\tilde{E}_k$ for each k and construct a list of clusters and their size by letting an eigenvalue that is not already in a cluster (i.e. has Euclidean distance greater than κ to all already known cluster centers) be a new cluster of size 1. And, if the eigenvalue has Euclidean distance to a cluster smaller than κ, the size of this cluster is incremented by 1.

Experiment 1: A diagonal

The following table summarizes the results for the experiment performed on a 35×35 diagonal matrix $\mathbf{A} = \mathbf{diag}([0.1 : 0.1 : 2 \quad 3 : 17])$[2] that we have used before. The small size should keep round-off error to a minimum enhancing the visibility of the effect. Here $\kappa = 0.001, s = 6, i = 3$, and $J = 7$ are used, hence we should see the formation of clusters of size 6. Note that we can clearly see the build-up of clusters in the first 7 iterations, and how the structure of the spectrum of $\mathbf{P}_J\mathbf{A}$ is still visible in the spectrum of $\tilde{\mathbf{P}}_k\mathbf{A}$ for $k = 1, 2, 3$.

[2]Matlab notation for the diagonal matrix with diagonal elements 0.1 up to 2 with step size 0.1, followed by the values 3 up to 17 with step size 1

Table 10.1: *Part of the spectral history of the seed and recycling run on a 35×35 diagonal matrix. In the columns showing the eigenvalue clusters of $\tilde{\mathbf{P}}_k\mathbf{A}$, the seemingly retained clustered structure of the spectrum of $\mathbf{P}_J\mathbf{A}$ are encircled.*

Eig($\mathbf{P}_1\mathbf{A}$)	Size	Eig($\mathbf{P}_2\mathbf{A}$)	Size	Eig($\mathbf{P}_3\mathbf{A}$)	Size
$1.000 + 0.000i$	**6**	$1.000 - 0.000i$	**6**	$1.000 + 0.000i$	**6**
$0.009 + 0.000i$	1	$2.492 + 0.000i$	**6**	$1.228 + 0.000i$	**6**
$0.019 - 0.003i$	1	$0.025 + 0.000i$	1	$3.061 - 0.000i$	**6**
$0.019 + 0.003i$	1	$0.054 - 0.010i$	1	$0.057 + 0.000i$	1
$0.037 - 0.002i$	1	$0.054 + 0.010i$	1	$0.096 - 0.009i$	1
$0.037 + 0.002i$	1	$0.082 + 0.000i$	1	$0.096 + 0.009i$	1
$0.046 + 0.000i$	1	$0.101 + 0.000i$	1	$0.126 + 0.000i$	1
$0.053 + 0.000i$	1	$0.118 + 0.000i$	1	$0.164 + 0.000i$	1
$0.067 + 0.000i$	1	$0.134 + 0.000i$	1	$0.204 - 0.011i$	1
$0.072 + 0.000i$	1	$0.165 + 0.000i$	1	$0.204 + 0.011i$	1
$0.079 + 0.000i$	1	$0.179 + 0.000i$	1	$0.249 + 0.000i$	1
$0.086 + 0.000i$	1	$0.198 + 0.000i$	1	$0.258 + 0.000i$	1
$0.093 + 0.000i$	1	$0.213 + 0.000i$	1	$0.257 - 0.088i$	1
$0.075 - 0.064i$	1	$0.233 + 0.000i$	1	$0.257 + 0.088i$	1
$0.075 + 0.064i$	1	$0.263 + 0.000i$	1	$0.306 + 0.000i$	1
$0.106 + 0.000i$	1	$0.271 + 0.000i$	1	$0.323 + 0.000i$	1
$0.110 + 0.000i$	1	$0.296 + 0.000i$	1	$0.360 - 0.005i$	1
$0.120 + 0.000i$	1	$0.324 + 0.000i$	1	$0.360 + 0.005i$	1
$0.131 - 0.009i$	1	$0.366 + 0.000i$	1	$0.757 + 0.000i$	1
$0.131 + 0.009i$	1	$0.672 + 0.000i$	1	$1.968 + 0.000i$	1
$0.228 + 0.000i$	1	$0.853 + 0.000i$	1	-	-
$0.336 + 0.000i$	1	$1.190 + 0.000i$	1	-	-
$0.422 + 0.000i$	1	$1.633 + 0.000i$	1	-	-
$0.571 - 0.059i$	1	$2.078 + 0.000i$	1	-	-
$0.571 + 0.059i$	1	$2.257 + 0.000i$	1	-	-
$0.652 + 0.000i$	1	-	-	-	-
$0.837 + 0.000i$	1	-	-	-	-
$0.886 + 0.000i$	1	-	-	-	-
$1.006 + 0.000i$	1	-	-	-	-
$2.022 + 0.000i$	1	-	-	-	-

Eig($\mathbf{P}_5\mathbf{A}$)	Size	Eig($\mathbf{P}_6\mathbf{A}$)	Size	Eig($\mathbf{P}_7\mathbf{A}$)	Size
$1.000 + 0.000i$	**6**	$1.000 + 0.000i$	**6**	$1.000 + 0.000i$	**6**
$0.504 - 0.000i$	**6**	$0.592 + 0.000i$	**6**	$0.918 + 0.000i$	**6**
$0.620 + 0.000i$	**6**	$5.138 + 0.000i$	**6**	$0.512 + 0.000i$	**5**
$0.086 + 0.000i$	**6**	$3.476 + 0.000i$	**5**	$4.444 + 0.000i$	**5**
$1.544 + 0.000i$	**5**	$4.269 + 0.000i$	**5**	$3.006 + 0.000i$	**4**
$0.083 + 0.000i$	1	$10.640 + 0.000i$	**4**	$3.692 + 0.000i$	**4**
$0.110 - 0.014i$	1	$0.831 + 0.000i$	1	$9.202 + 0.000i$	**3**
$0.110 + 0.014i$	1	$1.177 + 0.000i$	1	$0.500 + 0.000i$	1
$0.126 + 0.000i$	1	$4.739 + 0.000i$	1	$1.345 + 0.000i$	1
$0.163 + 0.000i$	1	-	-	-	-
$-0.601 + 0.000i$	1	-	-	-	-

Eig($\tilde{\mathbf{P}}_1\mathbf{A}$)	Size	Eig($\tilde{\mathbf{P}}_2\mathbf{A}$)	Size	Eig($\tilde{\mathbf{P}}_3\mathbf{A}$)	Size
$1.000 + 0.000i$	**6**	$1.000 - 0.000i$	**6**	$1.000 + 0.000i$	**6**
$0.210 + 0.000i$	(6)	$2.331 - 0.000i$	**6**	$0.593 + 0.000i$	**6**
$0.108 + 0.000i$	(5)	$0.495 + 0.000i$	(5)	$0.161 + 0.000i$	**6**
$0.182 + 0.000i$	(5)	$0.255 + 0.000i$	(5)	$0.126 + 0.000i$	(4)
$1.253 + 0.000i$	(4)	$0.428 + 0.000i$	(4)	$0.065 + 0.000i$	(4)
$0.632 + 0.000i$	(3)	$2.952 + 0.000i$	(3)	$0.109 + 0.000i$	(3)
$0.776 + 0.000i$	(3)	$1.489 + 0.000i$	(3)	$0.751 + 0.000i$	(3)
$1.934 + 0.000i$	2	$1.829 + 0.000i$	2	$0.085 + 0.000i$	2
$0.189 + 0.000i$	1	$0.260 + 0.000i$	1	$0.120 - 0.012i$	1

Experiment 2: A is a 225×225 matrix from a discretized pde problem

To show the effect for a slightly more realistic matrix, we repeat the same experiment on a 225×225 matrix called PDE225 originating from a discretized partial differential equation. For the experiment we use $i = 3$ and $\kappa = 0.0001$ as in the previous one but we choose $s = 8, J = 9$.
In the tables we can clearly see how the clustered structure of $\Lambda(\mathbf{P}_J\mathbf{A})$ is initially preserved into $\Lambda(\tilde{\mathbf{P}}_1\mathbf{A})$ and gradually broken off to form a similar structure during the first J IDR-cycles of the experiments.

Table 10.2: *We repeat the experiment on a more realistic problem, and can observe the same effect.*

Eig($\mathbf{P}_1\mathbf{A}$)	Size	Eig($\mathbf{P}_2\mathbf{A}$)	Size	Eig($\mathbf{P}_3\mathbf{A}$)	Size
$1.000 + 0.000i$	**8**	$1.000 + 0.000i$	**8**	$1.000 + 0.000i$	**8**
$0.098 + 0.000i$	1	$0.849 - 0.000i$	**8**	$1.091 + 0.000i$	**8**
$0.057 - 0.083i$	1	$-0.025 + 0.000i$	1	$1.285 + 0.000i$	**8**
$0.057 + 0.083i$	1	$0.087 + 0.000i$	1	$0.108 + 0.000i$	1
$0.150 + 0.000i$	1	$0.134 + 0.000i$	1	$0.164 + 0.000i$	1
$0.084 - 0.187i$	1	$0.183 - 0.005i$	1	$0.230 - 0.007i$	1
$0.084 + 0.187i$	1	$0.183 + 0.005i$	1	$0.230 + 0.007i$	1
$0.207 - 0.008i$	1	$0.212 + 0.000i$	1	$0.272 + 0.000i$	1
$0.207 + 0.008i$	1	$0.143 - 0.159i$	1	$0.280 - 0.059i$	1
$0.249 + 0.000i$	1	$0.143 + 0.159i$	1	$0.280 + 0.059i$	1
$0.257 - 0.053i$	1	$0.218 - 0.048i$	1	$0.168 - 0.245i$	1
$0.257 + 0.053i$	1	$0.218 + 0.048i$	1	$0.168 + 0.245i$	1
$0.284 + 0.000i$	1	$-0.027 - 0.227i$	1	$0.299 + 0.000i$	1
$0.288 - 0.043i$	1	$-0.027 + 0.227i$	1	$0.294 - 0.126i$	1
$0.288 + 0.043i$	1	$0.237 + 0.000i$	1	$0.294 + 0.126i$	1

Eig($\mathbf{P}_7\mathbf{A}$)	Size	Eig($\mathbf{P}_8\mathbf{A}$)	Size	Eig($\mathbf{P}_9\mathbf{A}$)	Size
$1.000 + 0.000i$	**8**	$1.000 + 0.000i$	**8**	$1.000 - 0.000i$	**8**
$0.986 + 0.000i$	**8**	$0.908 + 0.000i$	**8**	$1.513 + 0.000i$	**8**
$0.867 + 0.000i$	**8**	$1.033 - 0.000i$	**8**	$1.374 + 0.000i$	**7**
$1.409 + 0.000i$	**8**	$1.047 + 0.000i$	**8**	$1.563 + 0.000i$	**7**
$1.152 + 0.000i$	**7**	$1.476 + 0.000i$	**8**	$1.585 + 0.000i$	**7**
$1.292 + 0.000i$	**7**	$1.738 + 0.000i$	**8**	$2.234 - 0.000i$	**7**
$1.659 + 0.000i$	**7**	$1.206 - 0.000i$	**7**	$2.630 - 0.000i$	**7**
$0.293 + 0.000i$	1	$1.352 + 0.000i$	**7**	$1.825 + 0.000i$	**6**
$0.349 - 0.008i$	1	$0.203 + 0.000i$	1	$2.047 + 0.000i$	**6**
$0.349 + 0.008i$	1	$0.240 + 0.000i$	1	$0.314 + 0.000i$	1
$0.358 - 0.077i$	1	$0.364 + 0.000i$	1	$0.533 - 0.126i$	1
$0.358 + 0.077i$	1	$0.375 + 0.000i$	1	$0.533 + 0.126i$	1
$0.373 - 0.169i$	1	$0.372 - 0.090i$	1	$0.554 + 0.000i$	1
$0.373 + 0.169i$	1	$0.372 + 0.090i$	1	$0.604 - 0.102i$	1
$0.411 - 0.028i$	1	$0.437 - 0.026i$	1	$0.604 + 0.102i$	1

Eig($\tilde{\mathbf{P}}_1\mathbf{A}$)	Size	Eig($\tilde{\mathbf{P}}_2\mathbf{A}$)	Size	Eig($\tilde{\mathbf{P}}_3\mathbf{A}$)	Size
$1.000 + 0.000i$	**8**	$1.000 + 0.000i$	**8**	$1.000 + 0.000i$	**8**
$0.691 + 0.000i$	(8)	$1.540 - 0.000i$	**8**	$1.250 + 0.000i$	**8**
$1.124 + 0.000i$	(8)	$0.737 - 0.000i$	(7)	$1.831 + 0.000i$	**8**
$0.503 - 0.001i$	(7)	$1.012 + 0.000i$	(7)	$0.876 + 0.000i$	(7)
$1.000 + 0.001i$	(7)	$1.645 + 0.000i$	(7)	$1.203 + 0.000i$	(7)
$1.030 - 0.000i$	(7)	$1.344 + 0.000i$	(6)	$1.955 + 0.000i$	(7)
$1.323 + 0.000i$	(7)	$1.937 - 0.000i$	(6)	$1.599 - 0.000i$	(6)
$0.762 - 0.000i$	(6)	$1.115 - 0.000i$	(5)	$1.792 - 0.000i$	(6)
$0.786 + 0.000i$	(6)	$1.151 + 0.000i$	(5)	$2.303 + 0.000i$	(6)
$0.797 - 0.000i$	(6)	$1.167 + 0.000i$	(5)	$1.325 + 0.000i$	(5)
$0.919 + 0.000i$	(6)	$1.508 + 0.000i$	(5)	$1.369 + 0.000i$	(4)
$0.166 + 0.000i$	1	$0.189 - 0.059i$	1	$1.388 + 0.000i$	4
$0.000 - 0.182i$	1	$0.189 + 0.059i$	1	$0.209 + 0.000i$	1
$0.000 + 0.182i$	1	$0.059 - 0.226i$	1	$0.459 - 0.161i$	1
$0.279 + 0.000i$	1	$0.059 + 0.226i$	1	$0.459 + 0.161i$	1

Why convergence might benefit from spectral clustering of $\mathbf{P}_k\mathbf{A}$

Let us recall the single step of Richardson iteration that was proven to be equivalent to the $k-th$ IDR-cycle:

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \mathbf{A}\mathbf{P}_k\mathbf{r}_k,$$

and assume that $\mathbf{A}\mathbf{P}_k$ admits a basis of eigenvectors. Note that in the k-th iteration, we may assume without loss of generalization that $\Lambda(\mathbf{A}\mathbf{P}_k)$ consists of m clusters of size at least 1, valued $\hat{\lambda}_1, \ldots, \hat{\lambda}_m$, and define s_i to be the geometric multiplicity of eigenvalue $\hat{\lambda}_i$. We also may assume the eigenvectors of $\mathbf{A}\mathbf{P}_k$ that form a basis to be ordered in sets $\nu_i \equiv \{\mathbf{v}_1^i, \ldots, \mathbf{v}_{s_i}^i\}$ such that $\forall_{q\leq s_i}\mathbf{A}\mathbf{P}_k\mathbf{v}_q^i = \hat{\lambda}_i\mathbf{v}_q^i$. By our assumption of the existence of a basis of eigenvectors, we may write

$$\mathbf{r}_k = \sum_{i=1}^{m}\Big(\sum_{q}^{s_i}\alpha_q^i\mathbf{v}_q^i\Big), \quad \text{with scalars} \quad \forall_{i\leq m}\forall_{q\leq s_i}\alpha_q^i \in \mathbb{C}.$$

Now, since in our implementations of IDR, we choose ω_k such that

$$\omega_k = \arg\min_{\omega}\|(\mathbf{I}-\omega\mathbf{A})\Pi_1\mathbf{r}_k\|_2 = \arg\min_{\omega}\|\mathbf{r}_k - \mathbf{A}\mathbf{P}_k\mathbf{r}_k\|_2\,,$$

from which it can be seen that, for any $i \leq m$, ω_k can be chosen such that the entire component $\sum_{q=1}^{s_i}\alpha_q^i\mathbf{v}_q^i$ is removed from $\mathbf{r}_k$, by setting $\omega_k = \frac{1}{\hat{\lambda}_i}$. Now since this will also introduce new components in $\mathbf{r}_{k+1}$ corresponding to all other sets of eigenvectors, this will most often not be the case. It should, however, be noted that the minimization of $\|\mathbf{r}_{k+1}\|_2$ will be hampered by outliers in $\Lambda(\mathbf{A}\mathbf{P}_k)$ that have a large absolute value. I.e., when attempting to remove as many components from $\mathbf{r}_k$ as possible, ω_k must be correcting for the components that might be introduced by the eigenvectors corresponding to these outliers. Since by the IDR Projection Theorem we know that the spectrum of $\Pi_1 + \mathbf{Q}_k$ will get increasingly clustered with every IDR-cycle, it will be beneficial for the spectrum of $\mathbf{A}\mathbf{P}_k$ to select ω_k that on one hand acts as a minimizer (recall that a small value for ω_k will reduce the absolute value of the eigenvalues of $\mathbf{A}\mathbf{P}_k$, and at the same time should minimize $\|\mathbf{r}_{k+1}\|_2$), and on the other hand should not be too small so that it does not introduces eigenvalue of *large* ($\mu_{k+1} = \frac{1}{\omega_{k+1}}$) absolute value and of geometric multiplicity s that might hamper the minimization in the future.
Since the ω_k are chosen to be locally optimal, we conjecture that the increased clustering of $\Lambda(\mathbf{A}\mathbf{P}_k)$ per IDR-cycle gradually removes outliers, increasing the effectiveness of the polynomial step. In order to substantiate this conjecture, we make the following observation; Assume that for some iteration number k, we have $r_k \in \mathbb{C}$ such that

$$r_k = \inf\big\{r \;\big|\; r \in \mathbb{C} \wedge \forall_{\lambda\in\Lambda(\mathbf{P}_k\mathbf{A})}\,\|\lambda\|_2 \leq r\big\}.$$

That is, we know the radius of the smallest complex disk in which the spectrum of $\mathbf{P}_k\mathbf{A}$ is fully contained. Then, any choice for $\omega_k > r$ will be likely to be locally sub-optimal, as it will always leave components in $\mathbf{r}_{k+1}$ of the eigenvectors corresponding to μ_i with $i \leq k$. This prevents very small eigenvalues, μ_{k+1}, from entering the spectrum of $\mathbf{P}_{k+1}\mathbf{A}$. On the other hand, choosing ω_{k+1} very small will not be very effective, unless very small eigenvalues are *already* in $\Lambda(\mathbf{P}_k\mathbf{A})$. Therefore, it seems reasonable to assume that $r_{k+1} \leq r_k$, which in conjunction with some lower bound, would imply that $\Lambda(\mathbf{P}_k\mathbf{A})$ not only becomes increasingly made up out of clusters, but the clusters themselves will be increasingly grouped

together. Note that this also would imply that $\mathcal{C}(\mathbf{P}_{k+1}\mathbf{A}) \leq \mathcal{C}(\mathbf{P}_k\mathbf{A})$. In particular, when performing a recycling run, the already clustered spectrum makes for a faster decreasing residual norm, as it is more likely that outliers have already been removed in the seed-run, and the freedom in choosing $\tilde{\omega}_k$ can be exploited more effectively. Note also that increasing s would, according to this theory, be likely to lead to a increased effect, justifying results presented in chapter 7.

Experimental falsification

We can simply check the validity of the above by comparing norm ratios $\frac{\mathbf{r}'_k}{\mathbf{r}_{k+1}}$ and $\frac{\tilde{\mathbf{r}}'_k}{\tilde{\mathbf{r}}_{k+1}}$ per iteration between the seed run and the recycling run. Figure 10.1 below shows a typical result. We can witness that there is no improvement to be seen in this ratio in the recycling run, suggesting that the benefits of recycling are to be found in the projections Π_1, as this implies that, just like in regular IDR(s), the gross of the norm-reduction occurs when constructing $\mathbf{r}'_k$.

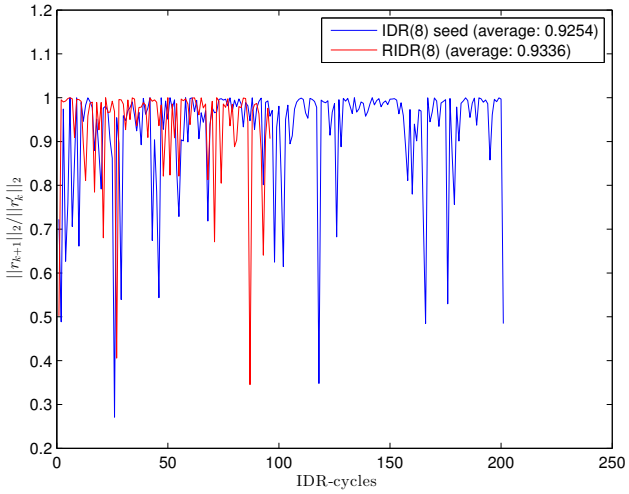


Figure 10.1: *An numerical experiment on* DW2048*: we do not see a significant lower value for the ratio* $\frac{\tilde{\mathbf{r}}_{k+1}}{\tilde{\mathbf{r}}'_k}$*. In fact, in this case, we see the contrary of what we might have hoped.*

In Figure 10.1 we can firstly see no average decrease of $\left\| \frac{\mathbf{r}_{k+1}}{\mathbf{r}'_k} \right\|_2$ as the iteration number increases. Secondly, we can see that the average ratio in the recycling run is in this case greater (and in most cases not significantly lower) than the average ratio in the seed run. As DW2048 is a typical matrix on which using RIDR(s) has a large positive effect, this result could be taken to be a serious counter-example to the above sketched theory.

Chapter 11

Discussion and Conclusions

Before drawing conclusions, we should elaborate somewhat on aspects that might have influenced the results shown in this thesis. Starting off with the theoretical results from Chapters 9 and 10, I will also comment on the experimental results from Chapter 7 and the algorithms derived in Part I.

11.1 Discussion

Growth

it should firstly be noted that the theory on residual norm growth in IDR may only justify the behavior for linear systems that are 'nice'. That is, when dealing with truly large matrices having a spectrum containing very diverse eigenvalues, the theory can only be part of the story in finite arithmetic. In trying to counter the growth in experiments using, for example, SHERMAN5, using all ones for the relaxation parameters, the seed may not even converge, and large errors caused by round-off always tend to make theory aimed at exact arithmetic less applicable.

Applicability of IDR Projection Theorem

In Chapter 10 we have seen an attempt at justifying the success of recycling auxiliary vectors which should also hint on the convergence of IDR methods in general. Besides from the theory not seeming to hold, one could argue that a lot of extra work is to be done if it is supposed to lead to theory with any predictive power. That is, we should for example have knowledge of the *entire* spectrum of $\mathbf{P}_k\mathbf{A}$, for usually, since the seed run is supposed to terminate for $k \ll N$ (this is what makes Krylov subspace methods successfully), only a fraction of $\Lambda(\mathbf{P}_k\mathbf{A})$ will consist out of the mentioned clusters. This would also imply the need for much greater understanding of the relation with the auxiliary vectors. It should be noted that a theory like this will be much more complicated than the known bounds for CG based on the spectrum of $\mathbf{A}$.

Experimental results using RIDR(s)

In this thesis results of spectacular speed-up when using RIDR(s) or similar method are presented. It should be emphasized though that there is a vast amount of input problems for which recycling does not yield such nice results. Although many 'well behaved' matrices have

shown to be very well suited for these methods, one might want (or not) to experiment with different problems found in the Matrix Market [6], for many of them will make RIDR(s) fail. It could be argued though that the matrices from Matrix Market are meant to be pathological examples of linear systems, and that one should focus on more practical examples that are known to work well with Krylov subspace-like methods.

The implementations used

As stated before, the used implementations (which can be found in Appendix A) are by no means the most stable, or efficient, but are mainly written for educational purposes and to have the flexibility/easy of editing to be suited for the experiments done for this thesis. Apart from, for example IDRStab being very slow (CPU-time wise, not MV-wise, these short comings may have had serious influence on experimental results when it comes to stability. Again, in particular our implementation of IDRStab gets unstable for $\ell > 4$, which would be totally unacceptable in any setting other than this one.

11.2 Conclusions

In this work we have made plausible the effectiveness of recycling auxiliary vectors in IDR-methods, and that it is actually superior to explicitly recycling *more* information like relaxation parameters. In fact, we have shown that the relaxation parameters will influence the recycling run, without being recycled. We have elaborated on the issue of residual growth in RIDR(s) and presented a plausible theory on the cause thereof, giving a concrete starting point of how to counter this problem in order to increase the numerical stability of the method. Substantiating the possibilities, an experiment has been presented in which the growth was countered by altering the seed run. It should be noted that once a stable seed run has been done with favourable relaxation paramters, recycling its auxiliary vectors in order to solve a larger sequence of linear systems get increasingly efficient. We have presented experimental results hinting on a (perhaps only partially) structured deficiency between the quantities $\boxed{\mathbf{A}} * \boxed{\mathbf{U}_k}$ and $\boxed{\mathbf{A}\mathbf{U}_k}$, which on one hand might lead to insight in the robustness of IDR-methods, and on the other hand may spark further improvements of IDR-methods in general. These results are, to my best knowledge, new, and should certainly be investigated further. Using IDR-specific spectral theory an attempt was made to justify further the excellent convergence properties of IDR(s) and IDRStab, which at the same time would naturally extend to the recycling variant, RIDR(s). The approach turned out to be not consistent with the outcome experiments, but has the advantage of being perhaps more of a problem-specific *bottom-up* approach to termination as opposed to the more common perspectives on the matter for IDR-methods (based on dimension reduction of Sonneveld- and/or $\mathcal{M}$-spaces) and most Krylov subspace-methods (based on the Cayley-Hamilton theorem), which both tend to be overly pessimistic for many practical situations.

Recommendations concerning further research

Concerning further research it would be advisable to prioritize the gaining of more theoretical insight in the role of both search matrices and relaxation parameters, and in particular their effect on the iteration process as a whole rather than a single step. They may very well be the key to the development of memory-efficient short-recurrence methods exhibiting convergence behaviour similar or better than GMRES. I believe the IDR-projection theorem

might still lead to such insights, but different approaches of how to apply the theorem should be thoroughly explored. It should be emphasized that the choice of $\mathbf{AU}_0$ being a Arnoldi basis of $\mathbf{A}\mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$ is still the only structured choice yielding fine results known, without any concrete justifying theory other than that is makes IDR(s) a Krylov subspace method. I believe IDR methods does not need this Krylov subspace framework, or, at least not in this way. (IDR(s) constructs Krylov like structures, as can be seen in Lemma 5.1.)
We have seen the possibilities of introducing new frameworks in terms of search spaces and the like (Krylov subspaces, Sonneveld subspaces, $\mathcal{M}$-spaces). Searching for more refined structures that apply to existing methods like the ones discussed in this work might as well lead to a better understanding of the underlying mechanisms of convergence, and perhaps a proper extension of the IDR-theorem to the setting of RIDR(s). In particular, the sequences of *shrinking* spaces that we have seen in this thesis have the advantage of enabling theories on termination other than the classical Cayley-Hamilton and Krylov subspace approach, which has clear short comings in predictive power in most practical situations.

That was it!

Thank you very much for taking the time to read my thesis.

Bibliography

[1] T. Collignon, G.L.G. Sleijpen, and M.B. van Gijzen. Interpreting IDR(s) as a deflation method. *Reports of the Department of Applied Mathematical Analysis, 10-21*, 2010.

[2] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM Journal on Numerical Analysis*, 21(2):352–362, 1984.

[3] Roger Fletcher. Conjugate gradient methods for indefinite systems. *Numerical analysis*, pages 73–89, 1976.

[4] Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952.

[5] N.J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[6] Matrix Market. Available on www at url http://math. nist. gov. *Matrix-Market*, 2007.

[7] M. Miltenberger. *Die IDR(s)-Methode zur Lösung von parametrisierten Gleichungssystemen*. PhD thesis, Diploma Thesis, TU Berlin, 2009.

[8] M.P. Neuenhofen. Short-recurrence and-storage recycling of large krylov-subspaces for sequences of linear systems with changing right-hand-sides. *arXiv preprint arXiv:1512.05101*, 2015.

[9] M.P. Neuenhofen. M(s)stab(l): A generalization of IDR(s)stab(L) for sequences of linear systems. *arXiv preprint arXiv:1604.06043*, 2016.

[10] Lewis Fry Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210:307–357, 1911.

[11] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[12] V. Simoncini and D.B. Szyld. Interpreting IDR as a Petrov-Galerkin method. *SIAM Journal on Scientific Computing*, 32(4):1898–1912, 2010.

[13] Gerard LG 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.

[14] Gerard LG Sleijpen and Henk A Van der Vorst. Maintaining convergence properties of bicgstab methods in finite precision arithmetic. *Numerical algorithms*, 10(2):203–223, 1995.

[15] Gerard LG Sleijpen, Henk A Van der Vorst, and Diederik R Fokkema. Bicgstab (l) and other hybrid bi-cg methods. *Numerical Algorithms*, 7(1):75–109, 1994.

[16] Gerard LG Sleijpen and Martin B Van Gijzen. Exploiting bicgstab(ℓ) strategies to induce dimension reduction. *SIAM journal on scientific computing*, 32(5):2687–2709, 2010.

[17] G.L.G. Sleijpen, P. Sonneveld, and M.B. van Gijzen. Bi-CGSTAB as an induced dimension reduction method. *Applied Numerical Mathematics*, 60(11):1100–1114, 2010.

[18] G.L.G. Sleijpen and M.B. van Gijzen. Numerical linear algebra - course material, chapter 1. *http://www.staff.science.uu.nl/ sleij101/Opgaven/NumLinAlg/Assignments/Theory1.pdf*.

[19] G.L.G. Sleijpen and M.B. van Gijzen. Numerical linear algebra - course material, chapter 11. *http://www.staff.science.uu.nl/ sleij101/Opgaven/NumLinAlg/Assignments/Theory11.pdf*.

[20] P. Sonneveld. On the convergence behaviour of IDR(s). *Reports of the Department of Applied Mathematical Analysis, 10-08*, 2010.

[21] P. Sonneveld and M.B. van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM Journal on Scientific Computing*, 31(2):1035–1062, 2008.

[22] Peter Sonneveld. Cgs, a fast lanczos-type solver for nonsymmetric linear systems. *SIAM journal on scientific and statistical computing*, 10(1):36–52, 1989.

[23] H.A. van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13. Cambridge University Press, 2003.

[24] 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.

[25] M.B. van Gijzen and P. Sonneveld. Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):5, 2011.

[26] P. Wesseling and P. Sonneveld. Numerical experiments with a multiple grid and a preconditioned lanczos type method. *Approximation methods for Navier-Stokes problems*, pages 543–562, 1980.

Appendix A

Matlab Code

IDR(s)

```
1
2  %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
3  %
4  %       Implementation of IDR(s)
5  %       Usage:
6  %       [x,rlog,mvlog,tlog,t,U,W] = idr(A,b,x,R,tol,maxit,U,calc_true)
7  %       x:                      approx. solution (output)
8          x:                      initial x               (input)
9  %       rlog:           residual norm history
10 %       mvlog:          total MV usage per iteration
11 %       tlog:           true residual norm history (set calc_true > 0)
12 %       t:                      elapsed time
13 %       U:                      last set of aux. vectors
14 %       W:                      sequence of relaxation parameters
15 %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
16
17 function [x,rlog,mvlog,tlog,t,U,AU,W] = idr(A,b,x,R,tol,maxit,U,calc_true)
18         % initialization
19         [N,s]    = size(R);
20         AU               = A*U;
21         r                = b - A*x;
22         rlog     = norm(r);
23         tlog     = rlog;
24         mvlog    = [0];
25         W                = [];
26         k                = 0;
27         t                = -1;
28         if ~calc_true
29                 tic;
30         end
31         % main loop
32         while (rlog(end) > tol && k < maxit)
33                 % ***Dimension reduction step***
34                 % Calculation of rp, xp
35                 sig                     = R'*AU;
36                 alpha           = sig\(R'*r);
37
38                 rp                      = r - AU*alpha;
39                 Arp             = A*rp;
40                 xp                      = x + U*alpha;
```

```
41
42                  % Calculation of Ve1 (first column)
43                  beta            = sig\(R'*Arp);
44                  V(:,1)          = rp - U*beta;
45                  AV(:,1)         = Arp - AU*beta;
46                  AAV(:,1)        = A*AV(:,1);
47                  % Call orthonormalizing subroutine
48                  [V(:,1),AV(:,1),AAV(:,1)]       = orth(V,AV,AAV,1);
49
50                  % Calculation of rest of V, AV, AAV
51                  for q = 2:s
52                          beta                    = sig\(R'*AAV(:,q-1));
53                          V(:,q)                  = AV(:,q-1) - U*beta;
54                          AV(:,q)                 = AAV(:,q-1) - AU*beta;
55                          AAV(:,q)                = A*AV(:,q);
56
57                          % Call orthonormalizing subroutine
58                          [V(:,q),AV(:,q),AAV(:,q)] = orth(V,AV,AAV,q);
59
60                  end
61                  % ***Polynomial step***
62                  % Calculation of omega
63                  w                       = Arp'*rp/(Arp'*Arp);
64                  W                       = [W; w];
65
66                  % Update to next r,x (now using reliable updates!)
67                  x                       = xp + w*rp;
68                  r                       = rp - w*Arp;
69
70                  rlog            = [rlog; norm(r)];
71
72                  % Update to U, AU
73                  U                       = V - w*AV;
74                  AU                      = AV - w*AAV;
75                  % Book keeping
76                  k                       = k+1;
77                  mvlog           = [mvlog k*(s+1)];
78                  if (calc_true)
79                          tlog            = [tlog norm(b-A*x)];
80                  end
81          end
82          if (~calc_true)
83                  t = toc;
84          end;
85  end
86
87  function [Vq,AVq,AAVq] = orth(V,AV,AAV,q)
88
89                  if (q>1)
90                          % Orthogonalization (of V) (Modified Gram-Schmidt)
91                          mu                      = V(:,1:q-1)'*V(:,q);
92                          Vq                      = V(:,q) - V(:,1:q-1)*mu;
93                          AVq                     = AV(:,q) - AV(:,1:q-1)*mu;
94                          AAVq            = AAV(:,q) - AAV(:,1:q-1)*mu;
95                          % Normalization (of V)
96                          nV                      = norm(Vq);
97                          Vq                      = Vq/nV;
98                          AVq                     = AVq/nV;
99                          AAVq            = AAVq/nV;
100                 else
```

```
101                             nV                      = norm(V(:,1));
102                             Vq                      = V(:,1)/nV;
103                             AVq                     = AV(:,1)/nV;
104                             AAVq            = AAV(:,1)/nV;
105                     end
106
107 end
```

IDR(s)Stab(ℓ)

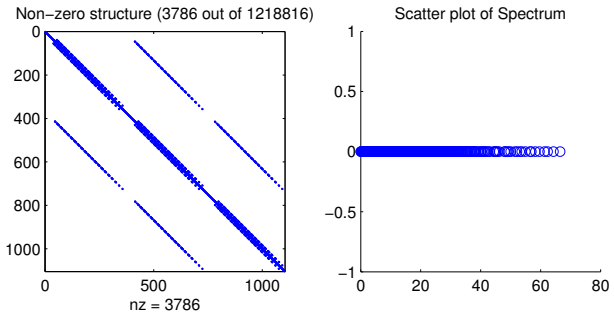
```
%
% Implementation of IDR(s)Stab(el)
% Usage: [x,rlog,mvlog,xlog,t,U,AU] = idrstab(A,b,x,R,tol,maxit,U,ell,
    calc_error)
%
function [x,rlog,mvlog,xlog,t,U,AU] = idrstab(A,b,x,R,tol,maxit,U,ell,
    calc_error)

    global n; %Globalize n for usage with indexing function J

    %initialization
    k               = 0;
    mvs             = 0;
    [n, s]  = size(R);
    r               = [b - A*x];
    rlog    = [norm(r)];
    xlog    = [rlog(1)];
    mvlog   = [0];
    U               = [U;A*U];
    t               = -1;
    if (~calc_error)
        tic;
    end

    while ((rlog(end) > tol) && (k < maxit))
        xp              = x;
        %Dimension Reduction Step
        for j   = 1:ell
            % Part I
            sig             = R'*U(J(j),:);
            alpha   = sig\(R'*r(J(j-1)));

            rp              = r - U(n+1:end,:)*alpha;
            rp              = [rp;A*rp(J(j-1))];
            xp              = xp + U(J(0),:)*alpha;

            % Part II
            beta    = sig\(R'*rp(J(j)));
            V               = rp - U*beta;
            V               = [V(:,1);A*V(J(j),1)];
            V               = orth(V,1,j);
            for q   = 2:s
                V(:,q)                  = zeros((j+2)*n,1);
                beta                    = sig\(R'*V(J(j+1),q
                    -1));
                V(1:(j*n+n),q)  = V((n+1):end,q-1) - U*beta;
                V(:,q)                  = [V(1:j*n+n,q);A*V(J
                    (j),q)];
                V                                       = orth(V,q,j)
                    ;
            end
            if (j<ell)
                U       = V;
                r       = rp;
            else
                U       = [V(J(0),:);V(J(1),:)];
                r       = rp(J(0));
            end
```

```
54                  end
55
56                  % Polynomial Step
57                  B               = zeros(n,ell);
58                  for i   = 1:ell
59                          B(:,i)  = rp(J(i));
60                  end
61
62                  gamma = (B'*B)\(B'*rp(J(0)));
63
64                  r                           = r - B*gamma;
65                  x                           = xp;
66                  for     i                   =1:ell
67                          x                           = x + gamma(i)*rp(J(i-1));
68                          U(J(0),:)           = U(J(0),:) - gamma(i)*V(J(i),:);
69                          U(J(1),:)           = U(J(1),:) - gamma(i)*V(J(i+1),:);
70                  end
71                  k               = k+1;
72                  rlog    = [rlog; norm(r)];
73                  if (calc_error)
74                          xlog    = [xlog;norm(b-A*x)];
75                  end
76                  mvlog   = [mvlog; mvlog(end)+ell*s+ell];
77          end
78          if (~calc_error)
79                  t = toc;
80          end
81          AU = U(J(1),:);
82          U  = U(J(0),:);
83  end
84
85  % Indexing function
86  function I = J(i)
87          global n;
88          I = [n*i+1:n*i+n];
89  end
90
91  % Orthonormalizing subroutine (Modified Gram-Schmidt)
92  function V = orth(V,q,j)
93          global n;
94          for     i=1:q
95                  if (i==q)
96                          V(:,q) = V(:,q)/norm(V(J(j),q));
97                  else
98                          mu              = V(J(j),i)'*V(J(j),q);
99                          V(:,q)  = V(:,q) - mu*V(:,i);
100                 end
101         end
102 end
```

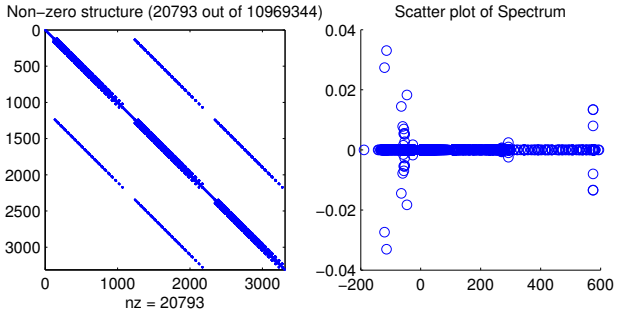
Appendix B

Glossary of test matrices

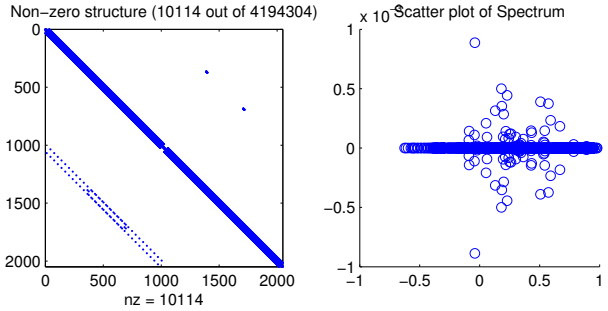
sherman4, $N = 1104$**,** $\mathcal{C}(\mathbf{A}) \approx 2.3 \cdot 10^3$**, Source: MatrixMarket, [6]**



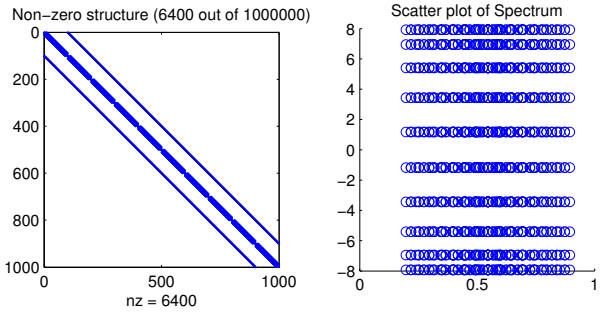
sherman5, $N = 3312$**,** $\mathcal{C}(\mathbf{A}) \approx 1.9 \cdot 10^5$**, Source: MatrixMarket, [6]**



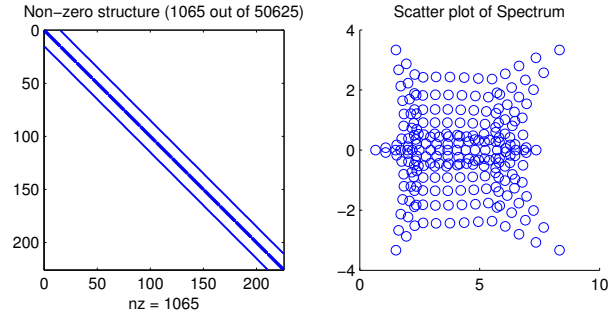
dw2048, $N = 1000$**,** $\mathcal{C}(\mathbf{A}) \approx 2.1 \cdot 10^3$



meier01, $N = 1000$**,** $\mathcal{C}(\mathbf{A}) \approx 7.4$



pde225, $N = 225$**,** $\mathcal{C}(\mathbf{A}) \approx 39.1$



pde900, $N = 900$**,**$\mathcal{C}(\mathbf{A}) \approx 292.9$