Introduction to the Induced Dimension Reduction Method IDR(s)

Orsolya Bosáková

Bachelor's thesis
2026:K10

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Abstract

Finding the solution to a large, nonsymmetric system of linear equations is a challenging problem in mathematics that often requires a lot of computational power. In this thesis, we provide a basis for understanding the Induced Dimension Reduction theorem. We introduce the IDR(s) theorem and the algorithm derived from it, which is a short recurrence method that addresses these challenges in an efficient way. This is a Krylov subspace method, which calculates the residuals in the nested subspaces of decreasing dimension. IDR(s) is an iterative method that computes the true solution in exact arithmetic in at most $N + \frac{N}{s}$ matrix-vector products, where N is the size of the linear system and s is the dimension relative to N. Lastly, we show the performance of the algorithm through some numerical examples. First, we validate our implementation against the academic example of a convection-diffusion problem from the literature. Then, we apply the same principle to two randomly chosen linear systems and finally demonstrate the real-world application to a weather model simulating a Gaussian bubble. Our results illustrate the convergence behavior and efficiency of IDR(s) compared to other well-known iterative methods.

Popular scientific summary

Many modern simulation problems rely on solving a linear system of equations of the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is of size N by N. When talking about weather models or flow problems, these systems are very large, with N often greater than 100 000. Finding an exact solution for systems of this size is impossible within a reasonable timeframe, because direct computation requires more memory and computing power than today's computers have. To overcome this, iterative methods were developed that can acquire an approximate solution to the system.

The most common type of method used for such problems are Krylov subspace methods. (However, these methods are limited by the properties of the matrix $\mathbf{A}$.) As shown by Faber and Manteuffel [6], it is impossible to create a method that works for a general $\mathbf{A}$, which is both memory efficient and optimal. Here, memory efficient means that the method uses short recurrences, storing only the most recent iterations rather than all of them. Optimal means that it finds the best possible mathematical approximation to the solution at every step. This limitation forces a compromise, which is why different approaches have been developed. Most of these are generalizations of the conjugate gradient method. The most prevalent being GMRES, which aims to minimize the distance to the solution. However, in the process, it needs to store large amounts of information, which leads to more computations and memory needs. An alternative is the Bi-CG method, which does not require that much memory but is more computationally expensive.

This leads us to the goal of finding a method, which is reliable and also efficient, additionally it works for nonsymmetric systems of equations. The induced dimension reduction method, or IDR(s), is based on an algorithm created by Sonneveld in 1980 [1] and it is memory efficient, because it uses short recurrences just as Bi-CG, but also reliable, because it is able to compute the exact solution in at most $2N$ steps.

IDR(s) works based on the idea that in order to find the solution, we do not need to save all the information about where we have searched for it, but we can create a "map", in which we systematically search certain areas and "forget" about them if the solution is not in that specific area. In other words, we are forcing each area that we search in, to shrink into a smaller dimension at each step.

In this thesis, we will explain the IDR(s) theorem and algorithm and present some examples of how it can be applied, whilst discussing the efficiency of the method.

Acknowledgments

I would like to thank my supervisor, Philipp Birken, for introducing me to this topic, guiding me through writing of my thesis and assisting me with my questions. I would also like to thank my family and friends for their support.

Contents

Chapter 1

Introduction

In the field of numerical linear algebra, the goal is to develop algorithms that can efficiently compute accurate approximations to solutions $\mathbf{x} \in \mathbb{R}^N$ of large scale problems. We specifically consider the problem of solving a linear system of the form

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

where $\mathbf{A}$ is a nonsymmetric matrix of dimensions N by N and $\mathbf{b}$ is a vector of dimension N.

Considering that a general dense matrix $\mathbf{A}$ has N^2 entries, any solution method will need to combine these with the vector $\mathbf{b}$. Overall, the operations needed to solve a system of this kind will be at least $\mathcal{O}(N^2)$. However, an algorithm that would indeed solve the system in $\mathcal{O}(N^2)$ operations has not been found, but many have been proposed. Whilst some direct approaches, such as the Gaussian elimination provide a solution in approximately $\mathcal{O}(N^3)$ operations, the cubic scaling is computationally expensive for large linear systems.

However, in large scale applications in practice, these big matrices are commonly not dense, but sparse. This means that most of their entries consist of zeros, which can lead to lower computational costs, reducing it from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. Direct solvers, such as the Gaussian elimination, do not exploit this property instead, they actually turn the zero entries into non-zero entries during the elimination process, making the calculations highly inefficient. Consequently, large scale problems use iterative methods instead, which account for the sparsity of the matrix by using sparse matrix-vector products.

Iterative methods are based on an initial solution vector $\mathbf{x}_0$, which generates a sequence of $\mathbf{x}_k$'s, such that they approach the true solution $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$ of the linear equation as k goes to infinity. However, the method is terminated after finitely many iterations, giving us an approximate solution $\mathbf{x}_p$. Iterative methods tend to be used for large systems of linear equations, approximately for systems that have N greater than about a 100 000. The benefit of this approach is that it can give an approximate solution much faster than any of the direct methods [3].

One of the earliest iterative method was the gradient method, which set out to solve the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ by viewing it as a problem of the inner product form

$$Q(\mathbf{x}) = \frac{1}{2}((\mathbf{A}^{-1}\mathbf{b} - \mathbf{x}), \mathbf{A}(\mathbf{A}^{-1}\mathbf{b} - \mathbf{x}))$$

or with our notation

$$Q(\mathbf{x}) = \frac{1}{2}((\mathbf{x}^* - \mathbf{x}), \mathbf{A}(\mathbf{x}^* - \mathbf{x})),$$

and trying to find the solution vector by minimizing the residual $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$. However, the fault of such a method is that the algorithm does not keep track of the directions in which it has searched for the solution, only the value of the residual is kept. This can lead to overlapping search directions and a lengthened process of finding a solution or stagnation.

An alternative is the Conjugate Gradient (CG) method, which creates a sequence of solution vectors $\mathbf{x}_k$, such that each of them is an element of the Krylov subspace $\mathcal{K}_n(\mathbf{A}, \mathbf{r}_0) := span(\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0 \cdots, \mathbf{A}^n\mathbf{r}_0)$, which minimizes the problem $Q(\mathbf{x})$. Additionally, CG uses short recurrence relations meaning that the next iterate can be computed only using information obtained in the previous step [6].

The downside of the CG method is that it is restricted to a symmetric positive definite matrix $\mathbf{A}$. This has inspired the search for an alternative method, that preserves the optimality and efficiency seen in CG, but can be applied to any nonsingular matrix $\mathbf{A}$. In 1984, Faber and Manteuffel proposed a theorem which answers this problem.

Theorem 1.0.1 ([6])**.** *An s-term conjugate gradient method exists for the solution of* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *if and only of either*
(i) the minimal polynomial of $\mathbf{A}$ *has degree* $\leq s$*, or*
(ii)$\mathbf{A}^H$ *is a polynomial of degree* $\leq s - 2$ *in* $\mathbf{A}$*.*

Here, the $\mathbf{A}^H$ refers to the adjoint of $\mathbf{A}$ with respect to some inner product. The specification that $\mathbf{A}^H$ is a polynomial in $\mathbf{A}$ of a certain degree is the same as saying that $\mathbf{A}$ is normal with respect to the same inner product.

The proof is beyond the scope of this paper, but can be found in [6].

This theorem tells us that for a method to satisfy both optimality and effectivity condition, the matrix $\mathbf{A}$ must belong to a very specific class. The optimality condition requires that the algorithm finds the best possible approximation $\mathbf{x}_p$ within the Krylov subspace, usually, by minimizing the residual $\mathbf{r}_p = \mathbf{b} - \mathbf{A}\mathbf{x}_p$. Efficiency refers to the low computational and memory cost of the method. This can be achieved by using short recurrences of length s, meaning the algorithm only needs to store and perform calculations with the last s vectors.

It shows that a s-term CG method can only exist, if and only if the matrix $\mathbf{A}$ is Hermitian, the matrix $\mathbf{A}$ is equal to its conjugate transpose $\overline{\mathbf{A}^T}$. More specifically, for such a method to exist with short recurrence of length s, one of the two strict conditions must hold. Either, the problem has to be small or simple enough for it to collapse before the limit of the recurrence is met, or, as specified above, $\mathbf{A}$ must be normal with respect to some inner product. For most nonsymmetric problems that we encounter in practice, such strict conditions are never actually met. This leads to a trade off, we either give preference to optimality of the minimalization, leading to methods such as GMRES, or to efficiency of the short recurrence, which results in the Bi-CG method [6].

These constraints have led to the development of many other methods, aiming to find an ideal balance between optimality and efficiency, mostly based off the Bi-CG method. However, Sonneveld and van Gijzen have looked for an alternative approach in their paper [1], which was based on an algorithm proposed by Sonneveld in 1980, called the Induced Dimension Reduction (IDR) algorithm. This was generalized to the IDR(s) family, which aims to solve the linear system in the case where $\mathbf{A}$ is nonsymmetric. IDR(s) does not aim to minimize the problem over the entire Krylov subspace, instead it forces successive residuals into a series of nested subspaces of decreasing dimension. This allows the method to move between optimality and efficiency by varying the value of s.

$$\mathbf{r}_k \in \mathcal{G}_j \subset \mathcal{G}_{j-1} \subset \cdots \subset \mathcal{G}_0.$$

IDR(s) uses short recurrences, to obtain the solution in at most $N + \frac{N}{s}$ matrix-vector products in exact arithmetic.

The thesis is structured in the following way. In Chapter 2, we introduce the reader to some basic concepts and definitions, which are necessary to understand the workings of the IDR method. In this chapter, we also include the explanation of the GMRES method to have a comparative background. In Chapter 3, we introduce the IDR theorem, along with an explanation of its proof to serve as theoretical background for the algorithm, followed by the derivation of the algorithm and its expected performance. In Chapter 4, we recreate the first example from the paper [1] for testing and present two additional examples using randomly selected matrices and discuss the performance of the method. Lastly, we include an example from a weather model, about a Gaussian bubble, to show the application of the method to a real world application.

Chapter 2

Preliminaries

2.1 Krylov Subspaces

Definition 2.1.1 ([2])**.** Given a nonsingular $\mathbf{A} \in \mathbb{C}^{N\times N}$ and $\mathbf{z} \neq \mathbf{0}$, $\mathbf{z} \in \mathbb{C}^N$, the n-th **Krylov (sub)space** $\mathcal{K}_n$ generated by $\mathbf{A}$ from $\mathbf{z}$ is

$$\mathcal{K}_n := \mathcal{K}_n(\mathbf{A}, \mathbf{z}) := span(\mathbf{z}, \mathbf{Az}, \cdots, \mathbf{A}^n\mathbf{z}). \tag{2.1.1}$$

This definition gives us a sequence of nested subspaces $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \mathcal{K}_3 \subseteq \ldots$, where the dimension, either increases by one in each step or $\mathcal{K}_{t-1} = \mathcal{K}_t$ for some $t \leq n$.

The more common form of the definition is $\mathcal{K}_n := \mathcal{K}_n(\mathbf{A}, \mathbf{z}) := span(\mathbf{z}, \mathbf{Az}, \cdots, \mathbf{A}^{n-1}\mathbf{z})$. However, in this thesis we chose to use the notation set forth by Sonneveld and von Gijzen in their paper [1], which uses the definition (2.1.1).

The idea is to create a sequence of approximate solutions $\mathbf{x}_n$ of $\mathbf{Ax} = \mathbf{b}$, such that $\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{A}, \mathbf{r}_0)$, where $\mathbf{x}_0$ is some initial approximation of the solution and $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ is the residual. The goal is to ensure that the residuals approach the zero vector or become equal to it as n increases.

Definition 2.1.2 ([3])**.** Given a sparse matrix $\mathbf{A}$ of dimension N by N, we can define a matrix valued polynomial of degree n by

$$p(\mathbf{A}) = a_n\mathbf{A}^n + a_{n-1}\mathbf{A}^{n-1} + \cdots + a_1\mathbf{A} + a_0\mathbf{I},$$

where the a_i's are the coefficients of the polynomial for $i = 0, 1, \ldots, n$ and $\mathbf{I}$ is the identity matrix.

If $\mathbf{A}$ is similar to $\mathbf{B} \in \mathbb{R}^{N\times N}$, we have:

$$\begin{aligned} \mathbf{A} = \mathbf{SBS}^{-1} &\implies \mathbf{A}^n = (\mathbf{SBS}^{-1})^n = \mathbf{SBS}^{-1}\mathbf{SBS}^{-1}\cdots\mathbf{SBS}^{-1} = \mathbf{SB}^n\mathbf{S}^{-1} \\ &\implies p(\mathbf{A}) = p(\mathbf{SBS}^{-1}) = \mathbf{S}p(\mathbf{B})\mathbf{S}^{-1}. \end{aligned}$$

where $\mathbf{S} \in \mathbb{R}^{N\times N}$.

Furthermore, if $\mathbf{A}$ is diagonalizable, we have

$$p(\mathbf{A}) = p(\mathbf{X}\mathbf{\Lambda}\mathbf{X}^{-1}) = \mathbf{X}p(\mathbf{\Lambda})\mathbf{X}^{-1},$$

where, $\mathbf{\Lambda} = diag(\lambda_1, \ldots, \lambda_n)$.
From this follows the next theorem.

Theorem 2.1.3 ([3]). ***(The Cayley-Hamilton Theorem)*** *Let $\chi(\cdot)$ be the characteristic polynomial of a matrix $\mathbf{A}$, then*

$$\chi(\mathbf{A}) = 0.$$

If we assume the matrix $\mathbf{A}$ to be invertible, we have that

$$\chi(\mathbf{A}) = \mathbf{A}^n + a_{n-1}\mathbf{A}^{n-1} + \cdots + a_1\mathbf{A} + a_0\mathbf{I} = 0,$$

which in turn implies that if we divide this equation by the term a_0 and multiply by $\mathbf{A}^{-1}$ from the right hand side, we get

$$\mathbf{A}^{-1} = -\frac{1}{a_0}\mathbf{A}^{n-1} - \frac{a_{n-1}}{a_0}\mathbf{A}^{n-2} - \cdots - \frac{a_1}{a_0}\mathbf{I},$$

which tells us that we can express the matrix inverse as a polynomial of degree $n-1$.

This motivates the Krylov subspace method as follows:

Definition 2.1.4 ([2]). A **Krylov space method for solving a linear system $\mathbf{Ax} = \mathbf{b}$** is an iterative method starting from some initial approximation $\mathbf{x}_0$ and the corresponding residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and generating for all, or at least most n, until it possibly finds the exact solution, iterates $\mathbf{x}_n$, such that

$$\mathbf{x}_n - \mathbf{x}_0 = q_{n-1}(\mathbf{A})\mathbf{r_0} \in \mathcal{K}_n(\mathbf{A}, \mathbf{r}_0)$$

where q_{n-1} is a polynomial of degree $n-1$.

The unusual part of the definition "for all, or at least most n" comes from the fact that it aims to cover the various methods in a general definition. In certain cases, it can happen that $\mathbf{x}_n$ and $\mathbf{r}_n$ may not be defined or the polynomial may be of lower degree.

2.2 Arnoldi Iteration

The Arnoldi iteration is a modified version of the Gram-Schmidt orthogonalization process, used to create an orthogonal basis for a Krylov space with only matrix vector products:

$$\mathcal{K}_n(\mathbf{A}, \mathbf{b}) = span(\mathbf{b}, \mathbf{A}\mathbf{b}, \cdots, \mathbf{A}^n\mathbf{b}) = span(\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_n),$$

where $\mathbf{q}_i^T\mathbf{q}_j = \delta_{ij}$ (Kronecker delta).

Let $\mathbf{Q}_n \in \mathbb{R}^{N\times n}$ be the matrix whose columns are the vectors $\mathbf{q}_i$, for $i = 1, 2, \ldots, n$. The algorithm can be written as

$$\mathbf{A}\mathbf{Q}_n = \mathbf{Q}_{n+1}\widetilde{\mathbf{H}}_n, \tag{2.2.1}$$

where $\widetilde{\mathbf{H}}_n$ is an $(n+1) \times n$ Hessenberg matrix. The individual elements of this matrix can be calculated as

$$h_{i,n} = q_i^T \mathbf{A} q_n,$$

which is called the Arnoldi iteration [3].

The algorithm for the Arnoldi iteration can be written as below.

Algorithm 1 The Arnoldi iteration [3]

b =arbitrary; $q_1 = b/\|b\|$;
for $n = 1, 2, 3, \ldots$ **do**
  $v = Aq_n$;
  for $i = 1$ to n **do**
    $h_{i,n} = q_i^* v$;
    $v = v - h_{i,n} q_i$;
  end for
  $h_{n+1,n} = \|v\|$;
  $q_{n+1} = v/h_{n+1,n}$;
end for

The Arnoldi relation will become important in the next section, when we explain the workings of the GMRES method. By using this relation, GMRES transforms a massive $N \times N$ least-squares minimization problem into a much smaller $(n+1) \times n$ problem that can be solved in significantly less time. Explaining this relationship is important because it shows us the difference between other Krylov subspace methods and IDR(s).

2.3 GMRES Method

The Generalized Minimal Residual method, or GMRES, is a method which uses Krylov subspaces to find an approximate solution to systems of linear equations.

Let $\mathbf{A}$ be a nonsingular matrix of dimensions $N \times N$ and $\mathbf{b}$ be a vector of dimension N resulting in the linear system $\mathbf{Ax} = \mathbf{b}$. The aim of the GMRES method is to find $\mathbf{x}_n \in \mathcal{K}_n$, such that it minimizes the norm of the residual $\mathbf{r}_n = \mathbf{b} - \mathbf{Ax}_n$. In other words,

$$\min_{\mathbf{x} \in \mathcal{K}_n} \|\mathbf{Ax} - \mathbf{b}\|.$$

This minimization is a large scale least squares problem, which can be simplified by using the orthogonal basis generated by the Arnoldi iteration. The columns of $\mathbf{Q}_n$ span the Krylov subspace, allowing us to write the approximate solution $\mathbf{x}_n$ as $\mathbf{x}_n = \mathbf{Q}_n \mathbf{y}$. The problem then becomes

$$\min_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{AQ}_n \mathbf{y} - \mathbf{b}\|.$$

With this we have reduced the residual minimization problem to a least squares problem with the use of the orthogonal basis from the Krylov subspaces. From the Arnoldi iteration we have (2.2.1) and with this we can rewrite our residual minimalization problem as follows

$$\begin{aligned}\min_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{Q}_n\mathbf{y} - \mathbf{b}\| &= \min_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{Q}_{n+1}\widetilde{\mathbf{H}}_n\mathbf{y} - \mathbf{b}\| \\ &= \min_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{Q}_{n+1}^T\mathbf{Q}_{n+1}\widetilde{\mathbf{H}}_n\mathbf{y} - \mathbf{Q}_{n+1}^T\mathbf{b}\| \\ &= \min_{\mathbf{y}\in\mathbb{R}^n} \|\widetilde{\mathbf{H}}_n\mathbf{y} - \mathbf{Q}_{n+1}^T\mathbf{b}\|,\end{aligned}$$

which gives the GMRES problem

$$\min_{\mathbf{y}\in\mathbb{R}^n} \|\widetilde{\mathbf{H}}_n\mathbf{y} - \|\mathbf{b}\|e_1\|,$$

where $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)^T$ is the standard basis vector.

The residual minimization problem can also be stated in polynomial form. Assume that we have some polynomial $p_n \in P_n$, where P_n is the set of polynomials of degree less than or equal to n with $p(0) = 1$. Now, the GMRES iterate $\mathbf{x}_n$ can be expressed as a polynomial expansion in $\mathbf{A}$ acting on the initial vector

$$\mathbf{x}_n = q_n(\mathbf{A})\mathbf{b},$$

where q_n is some polynomial of degree $n - 1$.

The corresponding residual then becomes

$$\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n = (\mathbf{I} - \mathbf{A}q_{n-1}(\mathbf{A}))\mathbf{b} = p_n(\mathbf{A})\mathbf{b},$$

where $p_n(z) = 1 - zq_{n-1}(z)$ and $p_n \in P_n$.

The objective of GMRES can be expressed as finding the coefficients of the polynomial $p_n \in P_n$ that minimize the norm of this residual vector. In other words, as expressed in [3],

$$\min_{p_n\in P_n} \|p_n(\mathbf{A})\mathbf{b}\|.$$

The explanation of GMRES and its polynomial minimization property serves as a contrast to the IDR(s) method. In the case of GMRES, it explicitly searches for and calculates the polynomial that minimizes the residual norm over the entire Krylov subspace. It uses long recurrences, resulting in larger memory and computational costs, which grow with each iteration. In the next chapter, we will explain how IDR(s) differs from this approach.

Chapter 3

The IDR Method

3.1 The IDR Theorem

The IDR method is based on the IDR theorem proposed by Wesseling and Sonneveld in 1980 [1]. To understand the theorem, we first need to define the concept of an invariant subspace of a matrix.

Definition 3.1.1. A subspace $\mathcal{T}$ is called an invariant subspace of the matrix $\mathbf{A}$ if multiplying any vector $\mathbf{t} \in \mathcal{T}$ by $\mathbf{A}$ results in $\mathbf{At} \in \mathcal{T}$. This can be denoted as $\mathbf{A}\mathcal{T} \subseteq \mathcal{T}$.

With this said, we can state the IDR theorem.

Theorem 3.1.2 ([1])**.** *Let $\mathbf{A}$ be any matrix in $\mathbb{C}^{N\times N}$, let $\mathbf{v_0}$ be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov space $\mathcal{K}^N(\mathbf{A}, \mathbf{v_0})$. Let $\mathcal{S}$ denote any (proper) subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of $\mathbf{A}$ and define the sequence $\mathcal{G}_j$, $j = 1, 2, \ldots,$ as*

$$\mathcal{G}_j = (\mathbf{I} - \omega_j \mathbf{A})(\mathcal{G}_{j-1} \cap \mathcal{S}), \tag{3.1.1}$$

where the ω_j's are nonzero scalar. Then the following hold:

(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, $\forall\ j > 0$.

(ii) $\mathcal{G}_j = \{\mathbf{0}\}$ *for some* $j \leq N$.

The IDR theorem assumes that two specific spaces ($\mathcal{S}$ and $\mathcal{G}_0$) do not share a nontrivial invariant subspace of $\mathbf{A}$. In other words, they do not share any other invariant subspace of $\mathbf{A}$ than the zero vector space $\{\mathbf{0}\}$. This condition is necessary to ensure that the dimensions of the subspaces keep decreasing and we do not end up with stagnation.

This theorem allows us to create a nested sequence of subspaces $\mathcal{G}_j$ where each new subspace is contained within the previous one. Compared to other Krylov subspace methods, instead of working with expanded spaces, this theorem ensures that the dimensions of the subspaces decrease steadily with an increasing value of j. Under the conditions stated in the theorem, the dimensions will decrease until the final subspace is eventually equal to the zero vector space $\{\mathbf{0}\}$.

The following proof is based on the proof published in [1]:

Proof. We divide the proof into two main parts for the properties (i) and (ii).

Proof of property (i):

We use mathematical induction to first show that $\mathcal{G}_j$ is indeed a subset of $\mathcal{G}_{j-1}$ for all $j > 0$.

Base case: We need to show that $\mathcal{G}_1 \subset \mathcal{G}_0$. By definition in (3.1.1) the first subspace is given by

$$\mathcal{G}_1 = (\mathbf{I} - \omega_1 \mathbf{A})(\mathcal{G}_0 \cap \mathcal{S}).$$

Since the intersection fulfills $(\mathcal{G}_0 \cap \mathcal{S}) \subset \mathcal{G}_0$, it follows that:

$$\mathcal{G}_1 \subset (\mathbf{I} - \omega_1 \mathbf{A})\mathcal{G}_0.$$

By definition $\mathcal{G}_0 = \mathcal{K}^N(\mathbf{A}, \mathbf{v}_0)$ is a Krylov subspace. By the Cayley-Hamilton Theorem, if we multiply the full Krylov subspace $\mathcal{K}^N$ by $\mathbf{A}$, then it maps back into itself. This means that $\mathbf{A}\mathcal{G}_0 \subset \mathcal{G}_0$. Therefore, for any linear combination:

$$\mathcal{G}_1 \subset (\mathbf{I} - \omega_1 \mathbf{A})\mathcal{G}_0 \subset \mathcal{G}_0 - \omega_1 \mathbf{A}\mathcal{G}_0 \subset \mathcal{G}_0.$$

We have shown that $\mathcal{G}_1 \subset \mathcal{G}_0$.

Induction hypothesis: We assume that (i) holds for some j, meaning that $\mathcal{G}_j \subset \mathcal{G}_{j-1}$.

Induction Step: We need to show that $\mathcal{G}_{j+1} \subset \mathcal{G}_j$.

Let $\mathbf{x}$ be some arbitrary vector in $\mathcal{G}_{j+1}$. By definition, there exists a vector $\mathbf{y} \in (\mathcal{G}_j \cap \mathcal{S})$, such that

$$\mathbf{x} = (\mathbf{I} - \omega_{j+1} \mathbf{A})\mathbf{y}.$$

Because $\mathbf{y}$ is an element of the intersection of $\mathcal{G}_j$ and $\mathcal{S}$, we get that $\mathbf{y} \in \mathcal{G}_j$. Applying the induction hypothesis ($\mathcal{G}_j \subset \mathcal{G}_{j-1}$), we can see that $\mathbf{y}$ is also an element of the space $\mathcal{G}_{j-1}$. Thus, $\mathbf{y} \in (\mathcal{G}_{j-1} \cap \mathcal{S})$

Next, we apply $(\mathbf{I} - \omega_j \mathbf{A})$ to the vector $\mathbf{y}$. Since, $\mathbf{y} \in (\mathcal{G}_{j-1} \cap \mathcal{S})$, the vector $(\mathbf{I} - \omega_j \mathbf{A})\mathbf{y}$ lies in the space $\mathcal{G}_j$ by (3.1.1):

$$\mathbf{y} - \omega_j \mathbf{A}\mathbf{y} \in \mathcal{G}_j.$$

Now, we would like to show that $\mathbf{A}\mathbf{y} \in \mathcal{G}_j$, to do this we can subtract $\mathbf{y} - \omega_j \mathbf{A}\mathbf{y}$ from $\mathbf{y}$:

$$\begin{aligned} \mathbf{y} - (\mathbf{y} - \omega_j \mathbf{A}\mathbf{y}) &\in \mathcal{G}_j, \\ \implies \omega_j \mathbf{A}\mathbf{y} &\in \mathcal{G}_j, \\ \implies \mathbf{A}\mathbf{y} &\in \mathcal{G}_j. \end{aligned}$$

Both $\mathbf{y} \in \mathcal{G}_j$ and $\mathbf{A}\mathbf{y} \in \mathcal{G}_j$, meaning that any linear combination of them will also be an element of $\mathcal{G}_j$. We can rewrite the vector $\mathbf{x}$ according to this, such that

$$\mathbf{x} = (\mathbf{I} - \omega_{j+1} \mathbf{A})\mathbf{y} \in \mathcal{G}_j.$$

Consequently, $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, which completes our proof for property (i).

Proof of property (ii):

After proving that the subspaces are nested $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, we now need to look at the dimension of these spaces. For each step j, there are two cases:

Case 1: The space $\mathcal{G}_{j+1}$ is a proper subspace of $\mathcal{G}_j$. In this case, the dimension strictly decreases:

$$\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j).$$

Case 2: The space $\mathcal{G}_{j+1}$ is equal to $\mathcal{G}_j$. This equality would mean that

$$\mathcal{G}_j = (\mathbf{I} - \omega_{j+1}\mathbf{A})(\mathcal{G}_j \cap \mathcal{S}). \tag{3.1.2}$$

For this equality to hold, $\mathcal{G}_j \subset \mathcal{S}$. If this was not the case, then $\dim(\mathcal{G}_j \cap \mathcal{S}) < \dim(\mathcal{G}_j)$, which would mean that $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$ and we are in Case 1. Therefore, have that $\mathcal{G}_j \cap \mathcal{S} = \mathcal{G}_j$. Substituting this back into (3.1.2) gives

$$\mathcal{G}_j = (\mathbf{I} - \omega_{j+1}\mathbf{A})\mathcal{G}_j.$$

Let $\mathbf{z} \in \mathcal{G}_j$, then there exists some arbitrary vector $\mathbf{w} \in \mathcal{G}_j$, such that $\mathbf{z} = \mathbf{w} - \omega_{j+1}\mathbf{A}\mathbf{w}$. Rearranging this equation gives us

$$\mathbf{A}\mathbf{w} = \frac{1}{\omega_{j+1}}(\mathbf{w} - \mathbf{z}).$$

Both vectors $\mathbf{z}$ and $\mathbf{w}$ belong to $\mathcal{G}_j$, thus also their difference belongs to $\mathcal{G}_j$. This shows that multiplying any vector in $\mathcal{G}_j$ by the matrix $\mathbf{A}$ keeps the results inside of $\mathcal{G}_j$. From this, we can see that $\mathcal{G}_j$ is an invariant subspace of $\mathbf{A}$.

We have shown that in Case 2, $\mathcal{G}_j$ is an invariant subspace of $\mathbf{A}$ and is also a subspace of $\mathcal{S}$. From property (i), we know that $\mathcal{G}_j \subset \mathcal{G}_0$.

Thus, $\mathcal{G}_j$ is an invariant subspace that is contained in both $\mathcal{G}_0$ and $\mathcal{S}$. However, the IDR theorem states that $\mathcal{G}_0$ and $\mathcal{S}$ do not share a nontrivial invariant subspace, meaning the only space that they can share is the zero vector space $\{\mathbf{0}\}$. Therefore, $\mathcal{G}_j = \{\mathbf{0}\}$.

At every step j, the dimension of the subspace is either reduced or equal to 0. Because $\dim(\mathcal{G}_0) \leq N$, the dimension can be decreased by at most N steps. Thus, there is some index $j \leq N$, where the subspace $\mathcal{G}_j$ is equal to $\{\mathbf{0}\}$.

□

3.2 The IDR Algorithm

3.2.1 Derivation

Our goal is to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{C}^{N\times N}$ and $\mathbf{b} \in \mathbb{C}^N$. Let the iterates of the Krylov method have residuals $\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n$, which are elements of the space generated by $\mathcal{K}^n(\mathbf{A}, \mathbf{r}_0)$. As before, $\mathbf{x}_0$ denotes the initial guess of the solution. From the definition of Krylov subspaces we know that we can write $\mathbf{r}_n$ as product of a polynomial of $\mathbf{A}$ and $\mathbf{r}_0$, $\mathbf{r}_n = \Phi_n(\mathbf{A})\mathbf{r}_0$, where the polynomial Φ_n is of exactly degree n, meaning it is an element of the space $\mathbb{P}^n/\mathbb{P}^{n-1}$. This provides us with a recursion formula for the residuals, which also enables us to derive a recursion formula for iterations $\mathbf{x}_n$. If we assume that this is possible for the first n terms, then we can define it for the $(n+1)$-th iterate. We have:

$$-\mathbf{b} + \mathbf{A}\mathbf{x}_{n+1} + \mathbf{b} - \mathbf{A}\mathbf{x}_n = \mathbf{A}(\mathbf{x}_{n+1} - \mathbf{x}_n) = -(\mathbf{r}_{n+1} - \mathbf{r}_n). \tag{3.2.1}$$

For conciseness, we use the difference operator defined as $\Delta \mathbf{u}_k = \mathbf{u}_{k+1} - \mathbf{u}_k$ and rewrite (3.2.1) as follows:

$$\mathbf{A}\Delta\mathbf{x}_n = -\Delta\mathbf{r}_n = (\Phi_n(\mathbf{A}) - \Phi_{n+1}(\mathbf{A}))\mathbf{r}_0. \tag{3.2.2}$$

Now, we want to express $\mathbf{x}_{n+1}$ from (3.2.2) in the form of a recurrence relation. This is always possible to do, if the difference of $\Phi_{n+1}(\tau)$ and $\Phi_n(\tau)$ is divisible by the variable τ. Because of this, when we evaluate the difference between any two of these polynomials at zero, we always get $1 - 1 = 0$. Since zero is a root of this difference, it is guaranteed by the polynomial remainder theorem that the expression is strictly divisible by τ. Let us assume that the following holds:

$$\Phi_{n+1}(\tau) - \Phi_n(\tau) = \tau\Psi_n(\tau), \tag{3.2.3}$$

where $\tau\Psi_n(\tau)$ is also a polynomial of the space $\mathbb{P}^n/\mathbb{P}^{n-1}$.

Applying (3.2.3) to the matrix $\mathbf{A}$ and multiplying with the initial residual vector $\mathbf{r}_0$ gives us:

$$(\Phi_n(\mathbf{A}) - \Phi_{n+1}(\mathbf{A}))\mathbf{r}_0 = -\mathbf{A}\Psi_n(\mathbf{A})\mathbf{r}_0. \tag{3.2.4}$$

Substituting (3.2.4) into (3.2.2), we get

$$-\Delta\mathbf{r}_n = \mathbf{r}_n - \mathbf{r}_{n+1} = -\mathbf{A}\Psi_n(\mathbf{A})\mathbf{r}_0, \tag{3.2.5}$$

$$\mathbf{A}\Delta\mathbf{x}_n = \mathbf{A}(\mathbf{x}_{n+1} - \mathbf{x}_n) = -\mathbf{A}\Psi_n(\mathbf{A})\mathbf{r}_0. \tag{3.2.6}$$

From (3.2.5) we can see that we can express $\mathbf{x}_{n+1}$ and $\mathbf{r}_{n+1}$ as follows

$$\begin{aligned}\mathbf{r}_{n+1} &= \mathbf{r}_n + \mathbf{A}\Psi_n(\mathbf{A})\mathbf{r}_0,\\ \mathbf{x}_{n+1} &= \mathbf{x}_n - \Psi_n(\mathbf{A})\mathbf{r}_0.\end{aligned}$$

We know that $\mathbf{r}_n = \Phi_n(A)\mathbf{r}_0$ is an element of the space $\mathcal{K}^n(\mathbf{A}, \mathbf{r}_0)$. Assume that there exists a vector $\mathbf{v}_n$ in $\mathcal{K}^n(\mathbf{A}, \mathbf{r}_0)/\mathcal{K}^{n-1}(\mathbf{A}, \mathbf{r}_0)$, so that every vector $\mathbf{w}_n$ in $\mathcal{K}^n$ can be written as

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \alpha\mathbf{v}_n$$

where $\mathbf{w}_{n-1} \in \mathcal{K}^{n-1}$ and α is a scalar.

We also know that $\Delta\mathbf{r}_n = \mathbf{A}\Psi_n(\mathbf{A})\mathbf{r}_0$, which implies that $\Delta\mathbf{r}_n$ is an element of $\mathcal{K}^{n+1}$ or, in another way, it is such that $\Delta\mathbf{r}_n$ is an element of $\mathbf{A}\mathcal{K}^n$. Then we need to realize that the span of the n-th Krylov space can be written using the residuals, so $\mathcal{K}^n = span(\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_n)$. To this end, rewrite the residuals as follows,

$$\mathbf{r}_1 = \mathbf{r}_0 + \Delta\mathbf{r}_0, \tag{3.2.7}$$

$$\mathbf{r}_2 = \mathbf{r}_1 + \Delta\mathbf{r}_1 = \mathbf{r}_0 + \Delta\mathbf{r}_0 + \Delta\mathbf{r}_1, \tag{3.2.8}$$

$$\vdots \tag{3.2.9}$$

$$\mathbf{r}_n = \mathbf{r}_0 + \sum_{i=0}^{n-1} \Delta\mathbf{r}_i. \tag{3.2.10}$$

From (3.2.7) we can see that $\mathcal{K}^n = span(\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_n) = span(\Delta\mathbf{r}_0, \Delta\mathbf{r}_1, \ldots, \Delta\mathbf{r}_{n-1})$.
Consequently, we can now express $\mathbf{r}_{n+1}$ as

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{A}\Psi_n(\mathbf{A})\mathbf{r}_0 = \mathbf{r}_n - \alpha\mathbf{A}\mathbf{v}_n + \mathbf{w}_{n-1} = \mathbf{r}_n - \alpha\mathbf{A}\mathbf{v}_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\mathbf{r}_{n-l}.$$

where the sum corresponds to the linear combination of previous residual differences.

Similarly, we can obtain the expression for the iterations, simply by using $\mathbf{A}\Delta\mathbf{x}_n = -\Delta\mathbf{r}_n$,

$$-\mathbf{A}\Delta\mathbf{x}_n = -\alpha\mathbf{A}\mathbf{v}_n + \sum_{l=1}^{\hat{l}} \gamma_l \mathbf{A}\Delta\mathbf{x}_{n-l}$$

by applying the inverse mapping of $\mathbf{A}$ and expanding $\Delta\mathbf{x}_n$, the final form becomes

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha\mathbf{v}_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\mathbf{x}_{n-l}$$

The integer $\hat{l}$ denotes how deep the recursion is. It can happen that $\hat{l}$ is equal to n, the dimension of the Krylov space, in which case we have a long recurrence, meaning that the work and memory requirements increase with n. Conversely, if $\hat{l}$ is a small fixed integer, then we have a short recurrence, which is more beneficial with respect to the amount of work and memory required.

We now apply the IDR theorem to the residuals $\mathbf{r}_n$, which become an element of the subspaces $\mathcal{G}_j$, where j is nondecreasing, whereas n is increasing. Under the assumptions of the theorem, the linear system will be solved in at most N dimension reduction steps. In IDR(s), the algorithm generates a sequence of residuals where a group of consecutive residuals shares the same subspace index j. Specifically, for $\mathcal{S}$ of dimension s, we generate $s+1$ consecutive residuals $\mathbf{r}_n, \mathbf{r}_{n+1}, \ldots, \mathbf{r}_{n+s}$ that all lie within the current subspace $\mathcal{G}_j$.

A residual $\mathbf{r}_{n+1}$ belongs to the space $\mathcal{G}_{j+1}$ if

$$\mathbf{r}_{n+1} = (\mathbf{I} - \omega_{j+1}\mathbf{A})\mathbf{v}_n,$$

where $\mathbf{v}_n$ is a vector in the intersection of $\mathcal{G}_j$ and $\mathcal{S}$.

If we assume that we can choose the vector $\mathbf{v}_n$ as

$$\mathbf{v}_n = \mathbf{r}_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\mathbf{r}_{n-l}, \tag{3.2.11}$$

then we can transform the expression for $\mathbf{r}_{n+1}$ to the following

$$\mathbf{r}_{n+1} = \mathbf{r}_n - \omega_{j+1}\mathbf{A}\mathbf{v}_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\mathbf{r}_{n-l},$$

which is our new Krylov solver recursion equation.

To calculate the coefficients γ_l and ω_{j+1}, we require $\mathbf{v}_n \in \mathcal{G}_j \cap \mathcal{S}$ as defined before. Here, s denotes a small fixed integer representing the dimension of the space $\mathcal{S}$. We introduce a matrix $\mathbf{P} \in \mathbb{C}^{N\times s}$, whose s linearly independent columns form a basis that we can project our vectors onto. Meaning, that $\mathcal{S}$ is the nullspace of the transpose of some matrix $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_s)$.

$$\mathcal{S} = \mathcal{N}(\mathbf{P}^H).$$

We know that $\mathbf{v}_n$ is an element of the space $\mathcal{S}$, which means it is also an element of the left nullspace of $\mathbf{P}$. In other words,

$$\mathbf{P}^H\mathbf{v}_n = \mathbf{0}. \tag{3.2.12}$$

If we substitute (3.2.11) into (3.2.12), the resulting equation is

$$\mathbf{P}^H(\mathbf{r}_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\mathbf{r}_{n-l}) = \mathbf{0}, \tag{3.2.13}$$

an $s \times \hat{l}$ linear system for the $\hat{l}$ number of coefficients of γ_l. This system is uniquely solvable if $\hat{l} = s$. Calculating the first vector in the space $\mathcal{G}_{j+1}$ needs $s+1$ vectors in the previous space $\mathcal{G}_j$. We can expect the residuals $\mathbf{r}_n$ to be in the space $\mathcal{G}_{j+1}$ for all $n \leq (j+1)(s+1)$.

We define the following matrices

$$\Delta\mathbf{R}_n = (\Delta\mathbf{r}_{n-1}, \Delta\mathbf{r}_{n-2}, \ldots, \Delta\mathbf{r}_{n-s}),$$
$$\Delta\mathbf{X}_n = (\Delta\mathbf{x}_{n-1}, \Delta\mathbf{x}_{n-2}, \ldots, \Delta\mathbf{x}_{n-s}).$$

With the use of these matrices, we can establish the algorithm for the computation of $\mathbf{r}_{n+1} \in \mathcal{G}_{j+1}$.

Let $\mathbf{c} \in \mathbb{C}^s$, then we can rewrite (3.2.12) and (3.2.13) as

$$\begin{aligned} \mathbf{P}^H\mathbf{v}_n &= \mathbf{P}^H\mathbf{r}_n - \mathbf{P}^H \sum_{l=1}^{\hat{l}} \gamma_l \Delta\mathbf{r}_{n-l}, \\ \iff \mathbf{P}^H\mathbf{v}_n &= \mathbf{P}^H\mathbf{r}_n - \mathbf{P}^H\Delta\mathbf{R}_n\mathbf{c}, \\ \iff \mathbf{v}_n &= \mathbf{r}_n - \Delta\mathbf{R}_n\mathbf{c}. \end{aligned}$$

From this we conclude that

$$\mathbf{v}_n = \mathbf{r}_n - \Delta\mathbf{R}_n\mathbf{c},$$
$$\mathbf{r}_{n+1} = \mathbf{v}_n - \omega_{j+1}\mathbf{A}\mathbf{v}_n.$$

Iterating gives us new residuals $\mathbf{r}_{n+2}, \mathbf{r}_{n+3}, \ldots$ in $\mathcal{G}_{j+1}$ in each iteration, because $\mathcal{G}_{j+1} \subset \mathcal{G}_j$. If we have reached computing $s+1$ residuals in the space $\mathcal{G}_{j+1}$, we expect the $s+2$-nd residual to be in the space $\mathcal{G}_{j+2}$.

To be able to compute this recurrence, we need a suitable choice of ω_{j+1}. This value can be chosen freely, however an appropriate choice would be the value that minimizes the norm of the corresponding residual $\mathbf{r}_{n+1}$. However, once we choose this value for the calculation of the first residual $\mathbf{r}_{n+1}$ in $\mathcal{G}_{j+1}$, we must choose the same value for all following calculations of residuals in the space $\mathcal{G}_{j+1}$.

When calculating the updates for the residuals $\mathbf{r}_{n+1}$, the same should be done for the solution vector $\mathbf{x}_{n+1}$. Of course, to be able to do any of this, we must start the process with some initial values and updates to the solutions, which all must be generated by a standard Krylov subspace method.

As concluded in the paper [1], the operation count of this algorithm for $(s+1)$ IDR(s) steps is $(s+1)$ matrix vector products, s^2+s+2 inner products and $2s^2+\frac{7}{2}s+\frac{5}{2}$ vector updates. Scaling and simple addition of vectors have been counted as half an operation each.

3.2.2 Performance

The IDR theorem tells us that the dimension will be reduced, however, it does not specify how much. Each full dimension reduction step from $\mathcal{G}_{j-1}$ to $\mathcal{G}_j$ needs $(s+1)$ matrix-vector products, which possibly results in approximately $(s+1)N$ matrix-vector products for the entire finite process. The convergence is much faster in practice however, in this section we will introduce a prediction for how the algorithm behaves for a finite termination. The following extended IDR theorem specifies the rate of dimension reduction of the algorithm.

Theorem 3.2.1. [1] *Let* $\mathbf{A}$ *be any matrix in* $\mathbb{C}^{N\times N}$*, let* $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_s \in \mathbb{C}^N$ *be linearly independent, let* $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_s]$*, let* $\mathcal{G}_0 = \mathcal{K}^N(\mathbf{A}, \mathbf{r}_0)$ *be the full Krylov space corresponding to* $\mathbf{A}$ *and the vector* $\mathbf{r}_0$*, and let the sequence of spaces* $\{\mathcal{G}_j, j = 1, 2, ...\}$ *be defined by*

$$\mathcal{G}_j = (\mathbf{I} - \omega_j\mathbf{A})(\mathcal{G}_{j-1} \cap \mathcal{N}(\mathbf{P}^H)), \tag{3.2.14}$$

where ω_j *are nonzero numbers, such that* $\mathbf{I} - \omega_j\mathbf{A}$ *is nonsingular. Let* $\dim(\mathcal{G}_j) = d_j$*; then the sequence* $\{d_j, j = 0, 1, 2, ...\}$ *is monotonically nonincreasing and satisfies*

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

The following proof is based on the proof provided in [1].

Proof. Let us denote the intersection of $\mathcal{G}_{j-1}$ (the previous Krylov subspace) and $\mathcal{N}(\mathbf{P}^H)$ (the nullspace of the matrix $\mathbf{P}^H$) by $\mathcal{U} := \mathcal{G}_{j-1} \cap \mathcal{N}(\mathbf{P}^H)$. Let the columns of a matrix $\mathbf{G}_{j-1}$ form a basis for the subspace $\mathcal{G}_{j-1}$. Consequently, any vector $\mathbf{x} \in \mathcal{G}_{j-1}$,can be written as

$$\mathbf{x} = \mathbf{G}_{j-1}\mathbf{c},$$

Algorithm 2 The IDR(s) algorithm

Require: $\mathbf{A} \in \mathbb{C}^{N\times N}; \mathbf{P} \in \mathbb{C}^{N\times s}; TOL \in (0,1); MAXIT > 0$
Ensure: $\mathbf{x}_n$ such that $\|\mathbf{b} - \mathbf{A}\mathbf{x}_n\| \leq TOL$
  {*Initialization*}
  Calculate $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$;
  {*Apply s minimum norm steps, to build enough vectors in* $\mathcal{G}_0$}
  for $n = 0$ to $s - 1$ **do**
    $v = \mathbf{A}\mathbf{r}_n$; $\omega = (\mathbf{v}^H\mathbf{r}_n)/(\mathbf{v}^H\mathbf{v})$;
    $\Delta\mathbf{x}_n = \omega\mathbf{r}_n$; $\Delta\mathbf{r}_n = -\omega\mathbf{v}$;
    $\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta\mathbf{r}_n$; $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$;
  end for
  $\Delta\mathbf{R}_{n+1} = (\Delta\mathbf{r}_n \cdots \Delta\mathbf{r}_0)$; $\Delta\mathbf{X}_{n+1} = (\Delta\mathbf{x}_n \cdots \mathbf{x}_0)$;
  {*Building* $\mathcal{G}_j$ *spaces for* $j = 1, 2, 3, \ldots$}
  $n = s$
  {*Loop over* $\mathcal{G}_j$ *spaces*}
  while $\|\mathbf{r}_n\| > TOL$ **and** $n < MAXIT$ **do**
    {*Loop over* $\mathcal{G}_j$ space}
    for $k = 0$ to s **do**
      Solve $\mathbf{c}$ from $\mathbf{P}^H\Delta\mathbf{R}_n\mathbf{c} = \mathbf{P}^H\mathbf{r}_n$
      $\mathbf{v} = \mathbf{r}_n - \Delta\mathbf{R}_n\mathbf{c}$;
      if $k = 0$ **then**
        {*Entering* $\mathcal{G}_{j+1}$}
        $\mathbf{t} = \mathbf{A}\mathbf{v}$;
        $\omega = (\mathbf{t}^H\mathbf{v})/(\mathbf{t}^H\mathbf{t})$;
        $\Delta\mathbf{r}_n = -\Delta\mathbf{R}_n\mathbf{c} - \omega\mathbf{t}$;
        $\Delta\mathbf{x}_n = -\Delta\mathbf{X}_n\mathbf{c} + \omega\mathbf{v}$;
      else
        {*Subsequent vectors in* $\mathcal{G}_{j+1}$}
        $\Delta\mathbf{x}_n = -\Delta\mathbf{X}_n\mathbf{c} + \omega\mathbf{v}$;
        $\Delta\mathbf{r}_n = -\mathbf{A}\Delta\mathbf{x}_n$;
      end if
      $\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta\mathbf{r}_n$;
      $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$;
      $n = n + 1$;
      $\Delta\mathbf{R}_n = (\Delta\mathbf{r}_{n-1} \cdots \Delta\mathbf{r}_{n-s})$;
      $\Delta\mathbf{X}_n = (\Delta\mathbf{x}_{n-1} \cdots \mathbf{x}_{n-s})$;
    end for
  end while

for some vector $\mathbf{c}$.

Similarly, if we consider $\mathbf{x} \in \mathcal{U}$, meaning $\mathbf{x} \in \mathcal{G}_{j-1}$ and $\mathbf{x} \in \mathcal{N}(\mathbf{P}^H)$, we can write it as $\mathbf{x} = \mathbf{G}_{j-1}\mathbf{c}$, where $\mathbf{c}$ has an extra restriction, $\mathbf{P}^H\mathbf{G}_{j-1}\mathbf{c} = 0$. Therefore, $\mathbf{c} \in \mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1})$ and we can write $\mathcal{U}$ as follows:

$$\mathcal{U} = \mathbf{G}_{j-1}(\mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1})). \tag{3.2.15}$$

Combining (3.2.14) with (3.2.15) gives us an alternative definition for the current Krylov subspace, namely

$$\mathcal{G}_j = (\mathbf{I} - \omega_j\mathbf{A})\mathbf{G}_{j-1}(\mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1})).$$

Within the theorem we have required $\mathbf{I} - \omega_j\mathbf{A}$ to be nonsingular, which provides us with the useful property that it is of full rank. Consequently, the dimension of $\mathcal{G}_j$ is equal to the dimension of $\mathcal{U}$.

$$d_j = \dim(\mathcal{G}_j) = \dim(\mathcal{U}) = \dim(\mathbf{G}_{j-1}(\mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1}))).$$

Considering $\mathbf{G}_{j-1}$ is a basis matrix, its columns are linearly independent. So, the mapping $\mathbf{G}_{j-1}$ is injective, thus

$$d_j = \dim(\mathbf{G}_{j-1}(\mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1}))) = \dim(\mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1})). \tag{3.2.16}$$

By definition, the matrix $\mathbf{P}^H$ is of size $s \times N$ and the matrix $\mathbf{G}_{j-1}$ is $N \times d_{j-1}$. Then their product $\mathbf{P}^H\mathbf{G}_{j-1}$ is a matrix of size $s \times d_{j-1}$. By the Rank-nullity theorem, we can rewrite d_{j-1} as

$$d_{j-1} = \dim(\mathcal{N}(\mathbf{P}^H\mathbf{G}_{j-1})) + \operatorname{rank}(\mathbf{P}^H\mathbf{G}_{j-1}).$$

Using (3.2.16), we get

$$d_j = d_{j-1} - \operatorname{rank}(\mathbf{P}^H\mathbf{G}_{j-1}). \tag{3.2.17}$$

Conversely, we can also apply the Rank-nullity theorem to the conjugate transpose of our matrix, $(\mathbf{P}^H\mathbf{G}_{j-1})^H = \mathbf{G}_{j-1}^H\mathbf{P}$. The matrix $\mathbf{G}_{j-1}^H\mathbf{P}$ has dimensions $d_{j-1} \times s$, meaning it maps from a domain of dimension s. The rank of any matrix is bounded by its smallest dimension, satisfying:

$$\operatorname{rank}(\mathbf{P}^H\mathbf{G}_{j-1}) \leq \min(s, d_{j-1}).$$

Taking the conjugate transpose keeps the rank of a matrix. Then, applying the Rank-Nullity theorem to this $d_{j-1} \times s$ system gives us:

$$\dim(\mathcal{N}(\mathbf{G}_{j-1}^H\mathbf{P})) + \operatorname{rank}(\mathbf{P}^H\mathbf{G}_{j-1}) = s. \iff \operatorname{rank}(\mathbf{P}^H\mathbf{G}_{j-1}) = s - \dim(\mathcal{N}(\mathbf{G}_{j-1}^H\mathbf{P})).$$

Let us denote $\dim(\mathcal{N}(\mathbf{G}_{j-1}^H\mathbf{P})) \in [0, s]$ by l and we end up with

$$d_j = d_{j-1} - s + l,$$
$$d_{j-1} - d_j = s - l,$$

which shows that indeed $0 \leq d_{j-1} - d_j \leq s$ as desired.

For the second part of the inequality we assume that there exists a non-zero vector $\mathbf{v} \in \mathcal{N}(\mathbf{G}_{j-1}^H \mathbf{P})$, then we know that $\mathbf{G}_{j-1}^H \mathbf{P}\mathbf{v} = 0$, which implies that $\mathbf{P}\mathbf{v} \in \mathcal{N}(\mathbf{G}_{j-1}^H)$, also meaning that $\mathbf{P}\mathbf{v}$ is orthogonal to the space $\mathcal{G}_{j-1}$. By the theorem, $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, then indeed the subspace has to also be orthogonal to $\mathbf{P}\mathbf{v}$ and the vector $\mathbf{v} \in \mathcal{N}(\mathbf{G}_j^H \mathbf{P})$. Consequently, $\mathcal{N}(\mathbf{G}_{j-1}^H \mathbf{P}) \subset \mathcal{N}(\mathbf{G}_j^H \mathbf{P})$, which tells us that the $\dim(\mathcal{N}(\mathbf{G}_{j-1}^H \mathbf{P})) \leq \dim(\mathcal{N}(\mathbf{G}_j^H \mathbf{P}))$. Using the same notation as before, we can denote this as

$$d_{j+1} = d_j - s + l',$$

where $l' = \dim(\mathcal{N}(\mathbf{G}_j^H \mathbf{P})) \geq l$. By moving the term around, we get that

$$d_j - d_{j+1} = s - l'$$

In conclusion, we have shown that $0 \leq d_j - d_{j+1} \leq d_{j-1} - d_j \leq s$, as required.

□

This proof establishes that the dimension reduction step, $d_{j-1} - d_j$, is between 0 and s. However, this reduction is only equal to zero if the previous subspace satisfies $\mathcal{G}_{j-1} \subseteq \mathcal{N}(\mathbf{P}^H)$, which represents a highly improbable linear relationship in practice. Practically, the reduction value at each step is equal to the maximal value s. In 3.2.17 we can see that the dimension reduction value is equal to that of the $\text{rank}(\mathbf{P}^H \mathbf{G}_{j-1})$. We know that the columns of the matrix $\mathbf{G}_{j-1}$ are independent, because it is a basis matrix for the space $\mathcal{G}_{j-1}$. Similarly, the columns of the matrix $\mathbf{P}$ are linearly independent, due to how it was defined. Given this, $\text{rank}(\mathbf{P}^H \mathbf{G}_{j-1}) < s$, meaning that there exists a nonzero linear combination of the rows of the matrix that equals to the zero vector. In other words, $\mathbf{p}^H \mathbf{G}_{j-1} = \mathbf{0}^H$, where $\mathbf{p}$ is some nonzero vector, that satisfies $\mathbf{p} = \mathbf{P}\mathbf{c}$. If we were in the situation that $d_{j-1} > s$, then finding a suitably valued $\mathbf{p}$ that would fulfill the above mentioned requirements would be considerably improbable, given that the only free parameter is the vector $\mathbf{c}$.

However, the case that the dimension reduction value is indeed equal to s in all cases is not proven, because that would require the matrices $\mathbf{G}_{j-1}$ and $\mathbf{P}$ to be constructed independently of each other. When the dimension reduction value is indeed s, it is referred to as the generic case, otherwise it is called the nongeneric case.

Corollary 3.2.2 ([1])**.** *In the generic case IDR(s) requires at most $N + \frac{N}{s}$ matrix-vector multiplications to compute the exact solution in exact arithmetic.*

Proof. In the general case, we assume the dimension reduction step to be maximal, which means that

$$d_{j-1} - d_j = s$$

where $d_j = \dim(\mathcal{G}_j)$ and s is the dimension reduction step as previously.

Since $\mathcal{G}_0 = K^N(\mathbf{A}, \mathbf{r}_0)$, then $\dim(\mathcal{G}_0) = d_0 = N$. Each dimension reduction step reduces the subspace dimension by s, then the maximum number of reduction steps $j_{\max}$, will reduce it to $d_{j_{\max}} = 0$. This is given by

$$j_{\max} = \frac{d_0}{s} = \frac{N}{s}.$$

We count the total number of matrix-vector products. The algorithm starts with an initial residual from which are calculated s vectors for the subspace $\mathcal{G}_0$ and then $s + 1$ vectors are needed for each additional subspace. Resulting in

$$s + j_{\max}(s + 1) = s + \frac{N}{s}(s + 1) = s + N + \frac{N}{s}$$

matrix-vector products.

Given that $N \gg s$, the initial s value becomes negligible. Thus, the total number of matrix vector products that is required to compute the exact solution becomes at most

$$N + \frac{N}{s}.$$

□

The corollary establishes an upper bound for the total number of matrix-vector products required to find the exact solution with IDR(s). However, it is important to note that, in practical applications, this upper bound is rarely met. While the bound guarantees that the solver will not stagnate indefinitely, the algorithm typically converges well before reaching $N + \frac{N}{s}$ steps. Furthermore, the parameter s determines the computational and memory complexity of the algorithm. Increasing s lowers the upper bound, but also requires more memory and more computational power. In practice, the optimal choice for s is between 4 and 8 depending on the needs of the linear system.

Chapter 4

Implementation

The implementation is based on the method provided in [1]. It is done in Python 3.13 and the standard Python function from the library SciPy has been used for the methods GMRES and Bi-CGSTAB.

We have applied the algorithm to three different linear systems as examples. The first example is taken from [1] and used to confirm the correctness of the implementation. However, the rest of the examples presented in the paper were deemed too complex for this thesis. Consequently, we have randomly chosen two matrices to present the effectivity of the method and compare it to GMRES and Bi-CGSTAB.

Due to the complexity of the additional examples included in [1], we chose to present two different examples. We apply the same algorithm as in the example for the convection-diffusion problem, with only one major difference. Previously, for the matrix P, we have exported the randomized matrix from Matlab, to recreate the results seen in [1] and test our implementation in Python. However, for these examples we have used the randomization algorithm that is provided in Python, namely `random.rand` from the `numpy` library, to produce a matrix of the required size.

Lastly, we include an example of a weather model simulation for a Gaussian bubble. The same algorithm has been used as for the previous two examples with minor adjustments to the code to be compatible with the model. The example presents a possibility for real life application of the IDR(s) algorithm and discusses the performance values.

4.1 Convection-Diffusion Problem

The one-dimensional convection-diffusion problem consists of the finite difference discretizations of the equation

$$-\frac{d^2u}{dx^2} + w\frac{du}{dx} = 0, \quad x \in (0,1),$$

with $u(0) = u(1) = 1$ and the choice of the convection parameter w is exactly $\frac{wh}{2} = 0.5$, with the mesh size $h = \frac{1}{61}$, meaning there are 60 grid points, not counting the boundary points. Furthermore, for both the convection and diffusion term central differences have been used:

$$\frac{du}{dx} \approx \frac{u_{i+1} - u_{i-1}}{2h},$$
$$\frac{d^2u}{dx^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2},$$

for $i = 1, 2, \ldots, 60$.

We discretize the differential equation the following way:

$$\begin{aligned} -\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\right) + w\left(\frac{u_{i+1} - u_{i-1}}{2h}\right) &= 0, \\ -u_{i+1} + 2u_i - u_{i-1} + \frac{wh}{2}(u_{i+1} - u_{i-1}) &= 0, \\ -u_{i+1} + 2u_i - u_{i-1} + 0.5u_{i+1} - 0.5u_{i-1} &= 0, \\ -0.5u_{i+1} + 2u_i - 1.5u_{i-1} &= 0. \end{aligned}$$

We can use the boundary conditions to get

$$\begin{aligned} -0.5u_2 + 2u_1 - 1.5u_0 = u(0) = 1 \Rightarrow -0.5u_2 + 2u_1 = 1.5, \\ -0.5u_{k+1} + 2u_k - 1.5u_{k-1} = u(1) = 1 \Rightarrow 2u_k - 1.5u_{k-1} = 0.5. \end{aligned}$$

After solving this, we arrive at the system that looks as follows

$$\begin{bmatrix} 2u_1 & -0.5u_2 & 0 & \cdots & 0 \\ -1.5u_0 & 2u_2 & -0.5u_3 & \ddots & \vdots \\ 0 & -1.5u_1 & 2u_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -0.5u_k \\ 0 & \cdots & 0 & -1.5u_{k-1} & 2u_k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} 1.5 \\ 0 \\ \vdots \\ 0 \\ 0.5 \end{bmatrix}.$$

To reproduce the results in [1], we choose the same parameters. The initial guess is the nullvector and the $\mathbf{P}$ matrix is defined as the orthogonalization of $s - 1$ random vectors, obtained by the standard orthogonalization method in SciPy, with the first column being the initial residual. IDR(s) has been computed for $s = 1, 2, 4, 6$.

We expect the system to terminate in 120, 90, 75 and 70 matrix vector products, respectively, for a system of 60 equations, according to 3.2.2 . The plot below shows the number of matrix vector products as a function of the scaled residual norm shown on a logarithmic scale. The GMRES and Bi-CGSTAB methods have also been included for comparison, and these should terminate in 60 and 120 matrix vector products, respectively [1].

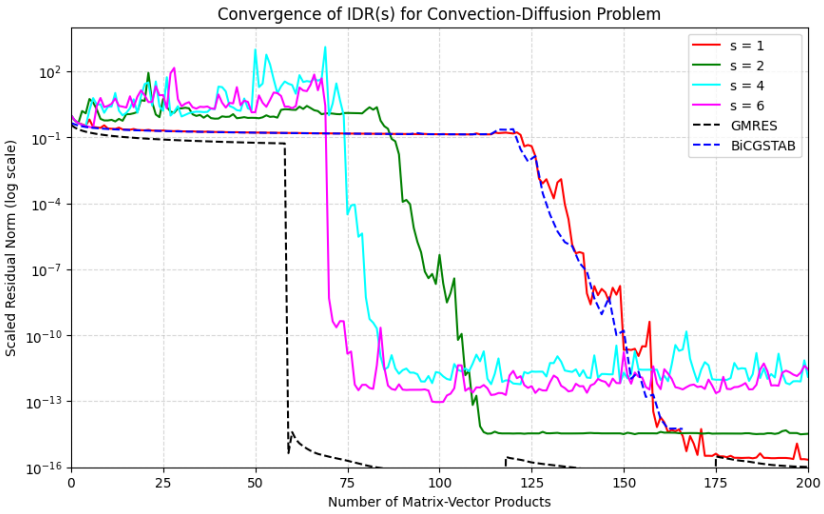

Figure 4.1: Termination of IDR(s), Bi-CGSTAB and GMRES

In Figure 4.1 we can see that the residual count drops below 10^{-8} at the expected values of matrix vector products. Visually, this behavior closely agrees with the graph presented in [1]. We can observe that IDR(1) and Bi-CGSTAB are almost identical in their decrease and norms stagnate at a similar level, as expected. Due to the differences in the implementation of the convection-diffusion problem we can see some differences in the noise that is generated after stagnation at each s, but the level at which they stagnated is as expected. As [1] explains, this level is related to the values of the residual norms in the initial iteration. We can observe that IDR(4) and IDR(6) have a slightly higher level at which they stagnate, compared to the other two s values.

4.2 Driven Cavity

We applied the IDR(s) method to the matrix E05R0000 with the provided corresponding right-hand side [4]. It is a real nonsymmetric matrix of size 236×236, with 5856 entries containing information about linear systems from modeling two-dimensional fluid flow in a driven cavity. The structure of the matrix can be seen in the Figure 4.2.

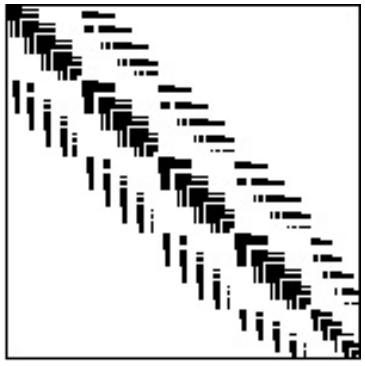

Figure 4.2: Structure of E05R0000

We can see the convergence of the solution in Figure 4.3. GMRES has the earliest termination at approximately 70 matrix-vector products and the result of IDR(1) is closely aligned with Bi-CGSTAB. However, neither of them reach termination within 1500 matrix-vector products. What is more interesting, is that we can very clearly see a gradual improvement with an increase in the dimension reduction value, with IDR(6) stagnating at around 500 matrix-vector products.

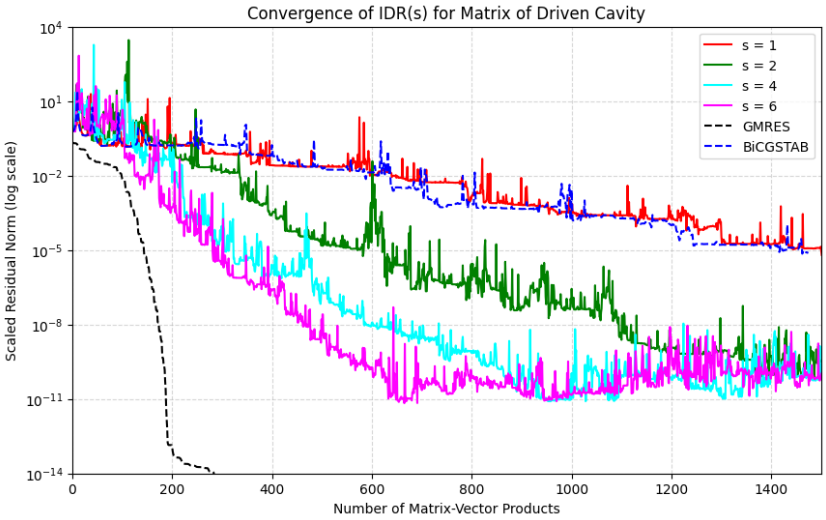

Figure 4.3: Convergence of the Driven Cavity problem

4.3 Oil Reservoir

For the second example, we have applied the IDR(s) method to the matrix SHERMAN1 with the corresponding right-hand side [5]. It is a real nonsymmteric matrix of size 1000×1000, with 3750 entries containing information about linear systems from three-dimensional oil reservoir modeling programs. The structure of the matrix can be seen in the figure 4.4.

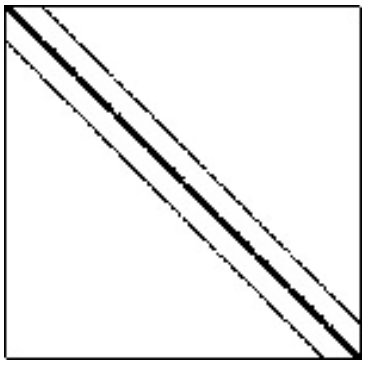

Figure 4.4: Structure of SHERMAN1

We can see the convergence of the solution in the figure 4.5. Surprisingly the differences between our IDR(s) results compared to GMRES or Bi-CGSTAB are not very distinct, espe-

cially in the beginning. As expected, GMRES provides the best result in terms of residuals, however it has a quite high count of matrix-vector products. Also, Bi-CGSTAB and IDR(1) seem to overlap mostly, which is to be expected. Around 800 matrix-vector products we can see a slight dive below GMRES for IDR(4) and IDR(6), however they stagnate at a higher level, even higher than IDR(1) or Bi-CGSTAB, which suggest that in this case higher dimension reduction actually results in a more unstable algorithm.

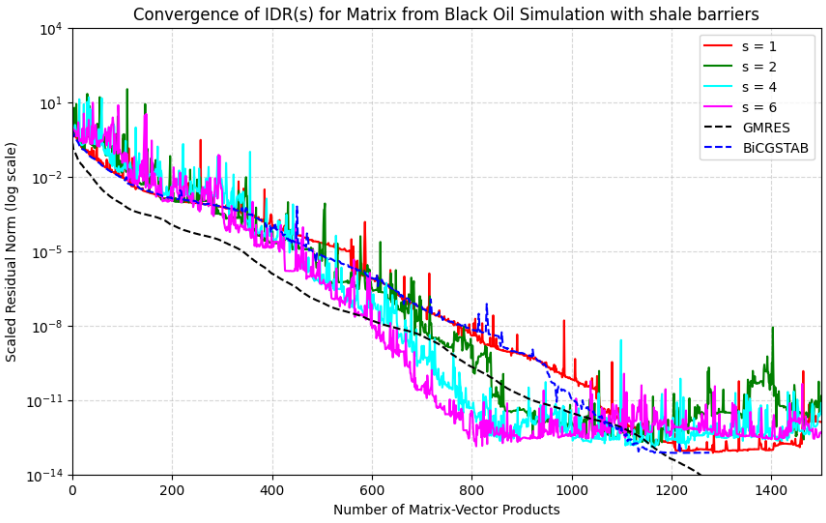

Figure 4.5: Convergence of the Oil Reservoir problem

4.4 Gaussian Bubble

In our last example, we show how IDR(s) can be used in more practical situations. The algorithm was integrated into a weather simulation model, WxFactory [7]. As a test case, we have chosen a Gaussian bubble simulation to present the workings of the IDR(s) method. The Gaussian bubble test is a standard fluid dynamics test case, to assess the solver's capabilities.

More specifically, we simulate the nonlinear two-dimensional Euler equations over a physical domain of size 1000m $\times$ 1500m on a Cartesian grid. The spatial domain is discretized using a discontinuous Galerkin spectral element method arranged into a 20×20 element mesh, using 5 solution nodal points. For time integration, the system uses a second-order semi-implicit Rosenbrock scheme, which needs to solve a large, nonsymmetric linear system at each time step. The simulation moves with a very small time step size of $\Delta t = 0.01$ seconds up to a final time of $t_{\text{end}} = 1100$ seconds, resulting in 110 000 time integration steps. To handle these steps, the tolerance for the linear solver was adjusted to 10^{-10}. The dimension reduction was fixed to $s = 6$, running without any preconditioning.

The evolution of the two-dimensional Gaussian bubble is presented in Figure 4.6.

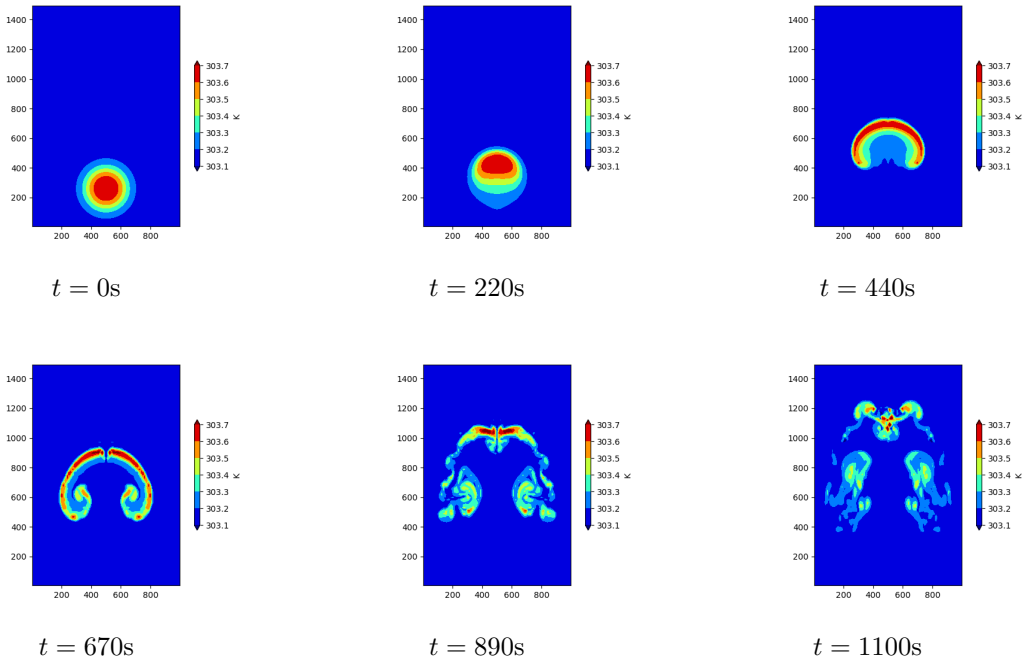

Figure 4.6: Gaussian bubble evolution across selected simulation time (t).

At $t = 0$s, we can see that the bubble is constant and in equilibrium. The first visible changes occur around $t = 220$s. At $t = 440$s, we can see a change in shape, as a result of stress along the outer margins, the lower part of the bubble begins to roll inward. Transforming the circle into a dome shape. Gradually, the bubble transforms into more intricate symmetric shapes until it reaches the end of the simulation at $t = 1100$s. Furthermore, the vertical symmetry is preserved through all the steps. Meaning that the unpreconditioned IDR(6) solver provides an accurate solution, capable of maintaining stability in highly dynamic equations.

Table 4.1: IDR(6) Performance Metrics for the Gaussian Bubble.

Performance Metric	**Value**
Total Timesteps	110 001
Linear Residual Tolerance	10^{-10}
Average IDR(6) Iterations per Step	3.01
Maximum IDR(6) Iterations per Step	18
Total Matrix-Vector Products (Approx.)	2 986 635

The metrics are presented in Table 4.1. Over the duration of 110 001 timesteps, the algorithm successfully converged to the tolerance of 10^{-10} without stagnation in any step. The average iteration per step was very small and the maximum iteration was 18, which tells us that on average the algorithm converged very fast and even for the more computationally complex steps, it did not require a large number of iterations. Overall, the simulation required 2 986 635 matrix-vector products. We can compare this to the theoretical upper bound from 3.2.2. We can calculate N from the element mesh, nodal points and the number of Euler equations $N = (20 \times 5)^2 \times 4 = 40\,000$. Resulting in

$$\begin{aligned} \text{MVP} &\leq \left(N + \frac{N}{s}\right) \times \text{timesteps}, \\ 2\ 986\ 635 &\leq \left(40\ 000 + \frac{40\ 000}{6}\right) \times 110\ 001, \\ 2\ 986\ 635 &\leq 5\ 133\ 380\ 000. \end{aligned}$$

The upper bound is much larger than our total number of matrix-vector products, meaning that IDR(6) terminates much quicker than theoretically expected. This large contrast shows that IDR(6) exploits the invariant subspaces of the fluid operator, converging much faster than the conservative theoretical upper bounds would predict.

Chapter 5

Conclusion

In this thesis, we looked at the numerical properties, implementation, and performance of the Induced Dimension Reduction method for solving large nonsymmetric linear systems. Testing the solver through different cases, starting with the standard convection-diffusion equations, moving on to matrices, and finally testing it on a real weather model, we showcased the benefits of the IDR(s) algorithm.

Our study shows that IDR(s) successfully handles a major issue found in standard Krylov subspace methods. As explained by the Faber-Manteuffel theorem, it is impossible for a solver to have both short-recurrence memory and optimal convergence for general nonsymmetric matrices. While GMRES gives optimal results but uses too much memory over time, methods like BiCGSTAB use less memory but show a highly fluctuating convergence. IDR(s) is a flexible solution method, that can be calibrated. By changing the parameter s, we can choose a fitting ratio between fast convergence and memory requirements for the problem at hand.

The practical implementation of the unpreconditioned IDR(6) method within WxFactory gave us a clear view of its robustness. Solving the nonlinear 2D Euler equations for a Gaussian bubble across 110 001 timesteps generated very complex and changing systems. IDR(6) showed great physical accuracy, keeping the vertical symmetry intact without missing out on the solution details. In terms of performance numbers, the solver was very stable, hitting the 10^{-10} tolerance with an average of only 3.01 iterations per step and a maximum of 18 iterations. The total number of matrix-vector products was also much lower than the theoretical upper bound, meaning that the solver works very efficiently in practical situations.

Overall, the results show that IDR(s) is a good choice for modern scientific computing. It combines the low memory benefits of short-recurrence methods with the stable convergence needed for heavy simulations. The smooth performance and reliable stability shown in our fluid dynamics test prove that IDR(s) is a reliable solution algorithm that can be used in modern weather and fluid simulation frameworks.

Appendix A

Python Implementation

```
import numpy as np
import scipy as sc

def IDR(A, b, s, x0, tol=1e-16, max_it=2000):
    x = x0
    n = len(b)

    #intialization steps
    r = b-A*x
    normr = np.linalg.norm(r)
    tolr = tol*np.linalg.norm(b)

    #creating the list for residuals
    res_list = []
    res_list.append(normr)

    #initial guess is close enough to the solution
    if normr <= tolr:
        iter_c=0
        return res_list

    P = np.random.rand(n, s)
    P[:,0] = r
    P = sc.linalg.orth(P).T

    M = np.zeros((s, s))
    dR = np.zeros((n, s))
    dX = np.zeros((n, s))

    #building G_0
    for i in range(s):
        v = A@r
        omega = np.dot(v,r)/np.dot(v,v)
        dX[:,i] = omega*r
        dR[:,i] = -omega*v
        r = r+dR[:,i]
        x = x+dX[:,i]
        normr = np.linalg.norm(r)
        res_list.append(normr)
        M[:,i] = P @ dR[:, i]
```

```
41
42      #building G_j
43      iter_c = s
44      oldest = 0
45      m = P@r
46
47      while normr>tolr and iter_c<max_it:
48          for k in range(s+1):
49              c = np.linalg.solve(M,m)
50              q = -dR@c
51              v = r+q
52              if k==0:
53                  t = (A @ v).flatten() #matvec
54                  omega = (t @ v) / (t @ t)
55                  dR[:,oldest] = q-omega*t
56                  dX[:,oldest] = -dX@c+omega*v
57              else:
58                  dX[:,oldest] = -dX@c+omega*v
59                  dR[:,oldest] = -A@dX[:,oldest]
60
61              r += dR[:,oldest]
62              x += dX[:,oldest]
63              iter_c += 1
64
65              normr = np.linalg.norm(r)
66              res_list.append(normr)
67
68              dm = P@dR[:,oldest]
69              M[:,oldest] = dm
70              m += dm
71
72              oldest = (oldest+1)%s
73
74      return res_list
```

Bibliography

[1] Sonneveld, P. and van Gijzen, M.B. (2009), *IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations*, SIAM Journal on Scientific Computing, **31(2)**, pp. 1035–1062.

[2] Gutknecht, M.H. (2007) *A brief introduction to Krylov space methods for solving linear systems', Frontiers of Computational Science* pp. 53–62.

[3] Trefethen, L. N. and Bau, D. (1997) *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia.

[4] *Matrix E05R0000.* (n.d.). `https://math.nist.gov/MatrixMarket/data/SPARSKIT/drivcav/e05r0000.html`, Last accessed: 02-05-26.

[5] *Matrix SHERMAN1.* (n.d.). `https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/sherman/sherman1.html`, Last accessed: 02-05-26

[6] Faber, V. and Manteuffel, T. (1984)*Necessary and sufficient conditions for the existence of a Conjugate Gradient method*, SIAM Journal on Scientific Computing, **21(2)**, pp. 352–362.

[7] Gaudreault, S., Charron, M., Dallerit, V., Tokman, M. (2022) *High-order numerical solutions to the shallow-water equations on the rotated cubed-sphere grid*, Journal of Computational Physics, **449**, 110792.

Bachelor's Theses in Mathematical Sciences 2026:K10
ISSN 1654-6229

LUNFNA-4070-2026

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/