THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Présentée et soutenue par :
Rafael Lago

le jeudi 13 juin 2013

Titre :

A study on block flexible iterative solvers with applications to Earth imaging problem in geophysics

---

École doctorale et discipline ou spécialité :
ED MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance

Unité de recherche :

Directeur(s) de Thèse :

GRATTON, Serge and VASSEUR, Xavier

Jury :
Hélène Barucq, Referee, Henri Calandra, Member of jury, Luiz Mariano Carvalho, Invited Member, Jocelyne Erhel, Member of jury, Serge Gratton, PhD advisor, Felix Herrmann, Member of jury, Hassane Sadok, Referee, Xavier Vasseur, Senior Researcher

Dissertation for the degree of doctor in Mathematics, Computer Science and Telecommunications (ED MITT)

A study on block flexible iterative solvers with applications to Earth imaging problem in geophysics

Rafael Lago (PhD student, CERFACS and INPT)

Hélène Barucq	Research director, INRIA and University of Pau	France	Referee
Henri Calandra	Senior advisor, TOTAL	France	Member of jury
Luiz Mariano Carvalho	Professor, IME-UERJ	Brazil	Invited Member
Jocelyne Erhel	Professor, INRIA	France	Member of jury
Serge Gratton	Professor, ENSEEIHT and INPT/IRIT	France	PhD advisor
Felix Herrmann	Professor, University of British Columbia	Canada	Member of jury
Hassane Sadok	Professor, ULCO	France	Referee
Xavier Vasseur	Senior researcher, CERFACS	France	PhD co-advisor

September 30, 2013

Thesis Summary

This PhD thesis concerns the development of flexible Krylov subspace iterative solvers for the solution of large sparse linear systems of equations with multiple right-hand sides. Our target application is the solution of the acoustic full waveform inversion problem in geophysics associated with the phenomena of wave propagation through an heterogeneous model simulating the subsurface of Earth. When multiple wave sources are being used, this problem gives raise to large sparse complex non-Hermitian and nonsymmetric linear systems with thousands of right-hand sides. Specially in the three-dimensional case and at high frequencies, this problem is known to be difficult. The purpose of this thesis is to develop a flexible block Krylov iterative method which extends and improves techniques already available in the current literature to the multiple right-hand sides scenario. We exploit the relations between each right-hand side to accelerate the convergence of the overall iterative method. We study both block deflation and single right-hand side subspace recycling techniques obtaining substantial gains in terms of computational time when compared to other strategies published in the literature, on realistic applications performed in a parallel environment.

Résumé de la Thèse

Les travaux de ce doctorat concernent le développement de méthodes itératives pour la résolution de systèmes linéaires creux de grande taille comportant de nombreux seconds membres. L'application visée est la résolution d'un problème inverse en géophysique visant à reconstruire la vitesse de propagation des ondes dans le sous-sol terrestre. Lorsque de nombreuses sources émettrices sont utilisées, ce problème inverse nécessite la résolution de systèmes linéaires complexes non symétriques non hermitiens comportant des milliers de seconds membres. Dans le cas tridimensionnel ces systèmes linéaires sont reconnus comme difficiles à résoudre plus particulièrement lorsque des fréquences élevées sont considérées. Le principal objectif de cette thèse est donc d'étendre les développements existants concernant les méthodes de Krylov par bloc. Nous étudions plus particulièrement les techniques de déflation dans le cas multiples seconds membres et recyclage de sous-espace dans le cas simple second membre. Des gains substantiels sont obtenus en terme de temps de calcul par rapport aux méthodes existantes sur des applications réalistes dans un environnement parallèle distribué.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this thesis we are concerned with the development of iterative solvers for large sparse linear systems with multiple right-hand sides with application to an industrial problem in geophysics and geology, the *acoustic full waveform inversion* [20, 28]. It consists of an optimization problem which targets to generate an approximate model of the velocity of propagation of acoustic waves in the subsurface of Earth. Experimental data is gathered by triggering an acoustic wave source at a certain position in Earth's surface, and as these waves propagate through the subsurface and encounter discontinuities, they are scattered and propagated back. Special tools called geophones (a special type of microphone) record information concerning the waves that were propagated back. This process is repeated for several positions and the data is again recorded several times. The acoustic full waveform inversion then builds and improves a velocity model (supposing that a proper initial guess model is known) of the subsurface iteratively until a reliable approximation is obtained. To determine whether an approximation is reliable or not it is thus necessary to simulate how the waves would have propagated through the velocity model. In this thesis, we are concerned with simulating the wave propagation phenomena given a velocity model using the Helmholtz equation

$$-\Delta u(\boldsymbol{x}) - \left(\frac{2\pi f}{v(\boldsymbol{x})}\right)^2 u(\boldsymbol{x}) = s(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^3$$

where $u(\boldsymbol{x})$ denotes the wave pressure, $v(\boldsymbol{x})$ is the velocity of the propagation of the wave and $s(\boldsymbol{x})$ is the source term, and using *perfectly matched layers* [13] (or PML) as boundary condition in order to simulate an infinite domain. The discretization of the Helmholtz equation with PML using finite difference techniques yields a sparse linear system of the form

$$AX = B,$$

where $A \in \mathbb{C}^{n\times n}$ is a nonsingular, non-Hermitian and non-symmetric matrix, $B \in \mathbb{C}^{n\times p}$ is supposed to be full rank, $X \in \mathbb{C}^{n\times p}$ and p is the number of acoustic wave sources triggered (usually of the order of $\mathcal{O}(10^4)$). The difficulties for solving this problem come when the acoustic wave sources are triggered at a high frequency. In this case, n can be of the order of $\mathcal{O}(10^9)$ meaning that the memory needed for solving this problem with direct solvers might be prohibitively large. In this situation iterative solvers are preferred since they allow the control of the memory used. Also, at high frequencies, the matrix A may present properties that complicate the preconditioning of iterative solvers, culminating in a slow convergence [52]. Recent publications show the interest of using geometric multigrid based techniques with an approximate coarse solution as preconditioner [24, 96], characterizing thus a variable preconditioner and imposing the use of flexible block Krylov subspace methods.

Several approaches have been used in the literature to solve the resulting linear system. Among the direct and hybrid solver solutions we mention [18, 67, 93, 121], but these often require a prohibitively large memory storage which grows proportionally with frequency being used, being thus suitable only for

low and medium frequencies range. Recent publications [3, 137, 138, 139] use the *low-rank approximation* techniques to reduce the memory cost of direct solvers for Helmholtz, being thus to successfully solve three-dimensional problems for high-frequencies. Other techniques as the sweeping preconditioner [47, 97] report low storage cost for solving problems at mid frequencies.

Nevertheless, due to their optimal memory control and scalability, we opt for focusing on Krylov iterative methods. The solution of systems with multiple right-hand sides using Krylov subspace methods has been addressed in the literature. The so called *block Krylov subspace methods* are increasingly popular in many application area in computational science and engineering (e.g. electromagnetic scattering (monostatic radar cross section analysis) [19, 78, 119], lattice quantum chromodynamics [110], model reduction in circuit simulation [53], stochastic finite element with uncertainty restricted to the right-hand side [45], and sensitivity analysis of mechanical systems [12] to name a few). Denoting by $X_0 \in \mathbb{C}^{n\times p}$ the initial guess for the system and by $R_0 = B - AX_0$ the initial block residual associated with such initial guess, a block Krylov space method for solving the p systems is an iterative method that generates approximations $X_m \in \mathbb{C}^{n\times p}$ with $m \in \mathbb{N}$ such that

$$\text{range}\,(X_m - X_0) \subset \mathcal{K}_m^{\square}(A, R_0)$$

where the block Krylov space $\mathcal{K}_m^{\square}(A, R_0)$ (in the unpreconditioned case) is a generalization of the well known Krylov subspace, defined as

$$\mathcal{K}_m^{\square}(A, R_0) = \text{range}\left(\begin{bmatrix} R_0 & AR_0 & \dots & A^{m-1}R_0 \end{bmatrix}\right) \subset \mathbb{C}^n.$$

We refer the reader to [64] for a recent detailed overview on block Krylov subspace methods and note that most of the standard Krylov subspace methods have a block counterpart (see, e.g., block GMRES [134], block BiCGStab [63], block IDR(s) [38] and block QMR [54]). To be effective in terms of computational operations it is recognized that block iterative methods must incorporate a strategy for detecting when a linear combination of the systems has approximately converged [64]. A simple strategy to remove useless information from a block Krylov subspace - called initial deflation - consists in detecting possible linear dependency in the block right-hand side B or in the initial block residual R_0 ([64, §12] and [78, §3.7.2]). When a restarted block Krylov subspace method is used, this block size reduction can be also performed at each initial computation of the block residual, i.e., at the beginning of each cycle [64, Section 14]. In addition Arnoldi deflation [64] may be also considered; it aims at detecting a near rank deficiency occurring in the block Arnoldi procedure to later reduce the current block size. These strategies based on rank-revealing QR-factorizations [21] or singular value decompositions [60] have been notably proposed both in the Hermitian [89, 104] and non-Hermitian cases [2, 9, 32, 54, 81, 92] for block Lanczos methods. They have been shown to be effective with respect to standard block Krylov subspace methods.

While initial deflation or deflation at the beginning of a cycle are nowadays popular, BGMRES based methods incorporating deflation at each iteration have been rarely studied. In [103] Robbé and Sadkane have introduced the notion of inexact breakdown to study block size reduction techniques in block GMRES. Two criteria have been proposed either based on the numerical rank of the generated block Krylov basis (W-criterion) or on the numerical rank of the block residual (R-criterion). Numerical experiments on academic problems of small dimension with a reduced number of right-hand sides illustrated the advantages and drawbacks of each variant versus standard block GMRES. Further numerical experiments can be found in [76]. Another method relying on such a strategy is the Dynamic BGMRES (DBGMRES) [33], which is an extension of block Loose GMRES [10]. However, the combination of block Krylov subspace methods performing deflation at each iteration and variable preconditioning has been rarely addressed in the literature.

As an alternative to block methods, it was proposed in the literature the *subspace recycling* techniques [1, 95], for the case in which the right-hand sides are not all known a priori, but instead, the solution of one system is used to compute the right-hand side of the next linear system. Traditionally augmented methods or methods with deflated restart (e.g. [11, 35, 41, 86, 106]) could retain their augmented subspace or the harmonic Ritz pairs (which approximate the smallest eigenvalue and its respective eigenvector of a matrix;

cf. [85]) from the solution of one linear system to accelerate the convergence of the next linear system, if the proper modifications in the algorithm are performed. In [95] the GCRO-DR, a variant of GMRES-DR [88] is proposed with the ability to recycle harmonic Ritz information. This is also particularly interesting for the case in which multiple left-hand sides situation is being addressed

$$A^{(i)}X^{(i)} = B,$$

for $1 \leq i \leq l$ and some relations between each $A^{(i)}$ hold. In the full waveform inversion scenario, the multiple left-hand sides situation is not all uncommon, as some techniques may generate l different models per iteration, thus requiring the solution of l block linear systems with p right-hand sides.

The main purpose of this thesis is to derive a class of flexible minimal block residual methods that incorporate block size reduction at each iteration. We carefully extend the theory available for single right-hand side case to a unified view in the block case, and we draw important conclusions for the variable preconditioner scenario which to the best of our knowledge is new for both single and multiple right-hand sides scenario (cf. Theorem 2.4.3). Using the theory we developed, we will introduce a method belonging to this class, the *deflated minimal block residual* or DMBR. It consists of a variant of BGMRES using a generalization of the deflation technique depicted in [103] which is able to discard subspaces at the beginning of each iteration. Our proposed variant uses the singular values of the scaled block residual to decide which part of the block Krylov subspace should be built for the current iteration, saving matrix-vector operations and preconditioner applications. Since we are considering flexible preconditioner which are potentially the most expensive part of the iterative procedure, avoiding preconditioner applications can bring a substantial computational gain to the overall method. We compare DMBR with recently proposed flexible block space methods using deflation at the beginning of the cycle only [22, 23] as well as with the method proposed in [103], which is able to deflate at the end of each iteration. We observe the advantage of deflating at the beginning of each iteration in practice in several of our real life application numerical experiments.

Along with that, in the single right-hand side scenario, we extended the GCRO-DR method for the variable preconditioner case, as subspace recycling techniques might be of interest in our geophysical application. We then compared FGMRES-DR [58] method with our new proposed method FGCRO-DR [27], and we demonstrate that for the variable preconditioner case, FGMRES-DR and FGCRO-DR are not algebraically equivalent unless a specific collinearity condition hold for every iteration. We thus show the interest of using subspace recycling techniques with variable preconditioner with numerical experiments.

The outline of this thesis is thus as follows:

- In Chapter 2 we study restarted flexible block methods that satisfy a minimum norm property. We discuss key aspects of minimum norm methods and we extend and generalize concepts as well as properties which are well known for the single right-hand side case. Among the key generalizations, we mention partial convergence (in Definition 2.8.1), partial breakdown (in Definition 2.6.1) and partial stagnation (in Definition 2.9.1), concepts which are not trivially obtained from the single right-hand side case. The definitions and properties discussed in this chapter are going to be extensively used in the Chapter 3 when deflation is discussed. Also in Chapter 2 we demonstrate that the flexible block Arnoldi algorithm (cf. Algorithm 2.5.1) always spans a block Krylov subspace whenever the variable preconditioner holds a rank-preserving condition. This result is new to the best of our knowledge and helps the understanding of flexible Krylov methods both for single and multiple right-hand sides.

- In Chapter 3 we use the development of Chapter 2 to characterize which information we would like to deflate at the beginning of each iteration of the BGMRES algorithm. We show the relation between the block residual and the block Krylov subspace being built in absence of partial stagnation phenomena, and that using the singular values of the scaled block residual we are able to focus on the minimization of the residual associated only with the right-hand sides which did not converge yet, thus avoiding to perform expensive computations to further improve already converged approximate solutions. The final deflation strategy we propose in this chapter is based on [103] but it is not

algebraically equivalent whenever restarting occurs in the algorithm. We compare our proposed algorithm, which we call *deflated minimal block residual* or DMBR, with other methods which are known to be efficient for solving the Helmholtz problem, and we obtain substantial gains in terms of computational time when performing experiments on real life problems.

- In Chapter 4 we introduce in more detail the acoustic full waveform inversion, clarifying some of the recent developments done in several areas to improve its performance. We discuss several issues involving the preconditioning and discretization of the Helmholtz equation and we finally discuss a software implementation using FORTRAN03 and object orientation strategies in order to obtain a modular code which can be easily adapted and modified to conform the newest technologies and techniques for the solution of the full waveform inversion. We use this software to perform numerical experiments of considerable dimension (from $\mathcal{O}(10^8)$ to $\mathcal{O}(10^9)$) at a relevant frequency range (from $5HZ$ to $12Hz$) using no more than 128 cores. We successfully compute, using block Krylov methods, the wavefield for multiple sources given at once using an average of 11.8 Gb of memory per shot for realistic problems at $5Hz$. This numerical experiments reinforces the interest in using deflation techniques for multisource scenario.

- In Chapter 5 we discuss the subspace recycling techniques, more specifically the development and analysis of FGCRO-DR method and how it is related to FGMRES-DR method in the single right-hand side case. We conclude that in spite of the similarities, the methods are not algebraically equivalent and produce different iterates in our numerical experiments. We conclude that an algebraic equivalence could be achieved if a certain collinearity condition holds for every iteration, but that such situation is rare in practice. Numerical experiments allow us to witness the advantages of using subspace recycling techniques when solving sequences of linear systems.

- In Chapter 6 we present the final remarks of this thesis and the plans for future research.

1.1 Notation

Through this entire thesis we denote by e_i the i-th vector of the canonical basis of the appropriate dimension. We write $||.||$ for any unitarily invariant norm. Some important norms used include the Frobenius norm denoted by $||.||_F$, the spectral norm $||.||_2$ when used for a matrix or the well known Euclidean norm when referring to a vector and the psi-norm $||.||_\psi$ defined as

$$||M||_\psi = \max_i ||m_i||_2$$

where m_i is the i-th column of M (this norm can be found, for instance, in [111, 134] and it is unitarily invariant).

We denote the condition number in the $\|.\|_2$ of a nonsingular matrix A by

$$\kappa(A) = \|A\|_2 \left\|A^{-1}\right\|_2$$

and X_* as the exact solution for $AX = B$.

We use $I_k \in \mathbb{C}^{k\times k}$ to denote the identity matrix of dimension k and $0_{i\times j} \in \mathbb{C}^{i\times j}$ the zero rectangular matrix with i rows and j columns. The superscript ^{H} denotes the transpose conjugate operation. Given a vector $d \in \mathbb{C}^k$ with components d_i, $D = \text{diag}(d_1 \quad \dots \quad d_k)$ is the diagonal matrix $D \in \mathbb{C}^{k\times k}$ such that $D_{ii} = d_i$. If $C \in \mathbb{C}^{k\times l}$ we denote the singular values of C by

$$\sigma_1(C) \geq \cdots \geq \sigma_{\min(k,l)}(C) \geq 0.$$

Whenever we mention that a given matrix $V \in \mathbb{C}^{n\times t}$ with $n \geq t$ is *orthonormal*, it means that it has *orthonormal columns* meaning that $V^H V = I_{t\times t}$. When $t = n$ we say that the matrix is *unitary* instead.

Regarding the algorithmic part, we adopt notation similar to those of MATLAB in the presentation. For instance, $U(i,j)$ denotes the U_{ij} entry of matrix U, $U(1:m, 1:j)$ refers to the submatrix made of the first m rows and first j columns of U and $U(:,j)$ corresponds to its jth column.

Whenever we refer to a subspace $\mathcal{V}$, we implicitly suppose that $\mathcal{V}$ is a subset of $\mathcal{H}$, where $\mathcal{H}$ is the Hilbert subspace. The notation $\mathcal{V} \subsetneq \mathcal{W}$ reads "$\mathcal{V}$ is a proper subset of $\mathcal{W}$" (i.e. $\mathcal{V}$ is contained in $\mathcal{W}$ but $\mathcal{V} \neq \mathcal{W}$) and the notation $\mathcal{V} \subset \mathcal{W}$ reads "$\mathcal{V}$ is a subset of $\mathcal{W}$" (i.e. $\mathcal{V}$ is contained in $\mathcal{W}$ and $\mathcal{V} = \mathcal{W}$ may be true). Also, for any matrix $V \in \mathbb{C}^{n\times p}$

$$\operatorname{range}(V), \operatorname{rank}(V), \operatorname{null}(V)$$

denote respectively the range of V, the rank of V and the nullity of V. We also use $\dim(\mathcal{V})$ to denote the dimension of the subspace $\mathcal{V}$.

Let $P \in \mathbb{C}^{a\times c}$ and $Q \in \mathbb{C}^{c\times b}$. Following Matlab standards, we abuse the notation allowing $c = 0$ resulting in operations as

$$M = \begin{bmatrix} N & P \end{bmatrix} = \begin{bmatrix} P & N \end{bmatrix} = N$$

for any $N \in \mathbb{C}^{a\times d}$ for a given positive integer d. We reinforce that in such a situation P is undefined as a mathematical entity, and we are simply defining $\begin{bmatrix} P & N \end{bmatrix} = N$. Similarly, we allow

$$S = \begin{bmatrix} Q \\ T \end{bmatrix} = \begin{bmatrix} T \\ Q \end{bmatrix} = T$$

with $T \in \mathbb{C}^{d\times b}$ for a given positive integer d. In our pseudo-code examples, we write $P = []$ and $Q = []$ without specifying dimensions.

Chapter 2

Introduction to Block Iterative Solvers

2.1 Introduction

In this chapter we address the basis for the development of methods for solving the problem

$$AX = B \tag{2.1.1}$$

where $A \in \mathbb{C}^{n\times n}$ is any nonsingular complex non-Hermitian nonsymmetric matrix, and $X, B \in \mathbb{C}^{n\times p}$, always considering $p \ll n$, and that $\text{rank}\,(B) = p$, and for the sake of simplicity, we consider the zero initial guess (that is, $X_0 = 0$) for the moment.

Although some of the content of this chapter can be found in other publications (as [40, 103, 64, 65, 134]), here we present, reorder and generalize these key concepts aiming at establishing grounds for the more advanced techniques we are going to show in further chapters. We explicitly mention whenever new concepts or proofs are presented. Notably we bring to the attention of the reader the results formalized later in Theorem 2.4.3 and Proposition 2.5.3.

To solve (2.1.1) the most straightforward approach is to separate such system in p linear systems as

$$AXe_i = Be_i, \qquad\qquad i = 1, ..., p$$

where e_i represents the i-th vector of the canonical basis in $\mathbb{C}^p$, and then apply the known preconditioned GMRES [108] method (or any other convenient iterative solver) to each system independently. A technique known as *"subspace recycling"* [36, 95, 37] could be used here to gather information from A (as information concerning the approximation of the invariant subspace, approximate spectral information such as harmonic Ritz pairs [85], etc) when solving $AXe_k = Be_k$ and use this knowledge to accelerate the convergence of the chosen method for solving $AXe_{k+1} = Be_{k+1}$. This strategy is particularly common for the cases in which Be_{k+1} depends on Xe_k, meaning that each linear system has to be solved *in sequence*. In this chapter however, we do not address the situation, and we suppose that all the columns of the right-hand side B are known beforehand.

In such a scenario, we can thus consider a generalization of iterative methods like GMRES for multiple right-hand side (see [64] for a recent detailed overview on block Krylov subspace methods). This generalization is often called in the literature *"block iterative method"*. Among the reasons for considering such a generalization is the possibility of exploiting BLAS-3 operations rather than BLAS-2. From a more theoretical point of view, block iterative solvers span a larger search subspace than its single right-hand side counterpart and thus potentially finds an approximation for (2.1.1) with less computational effort. This behaviour has been shown experimentally in a number of publications [23, 64, 103, 134] and we discuss it with more details through this chapter.

We focus on restarted block methods that satisfy a minimum norm property as introduced in [107, §6.12]. We first discuss key aspects of minimum norm methods and we extend properties which are known in the single right-hand side case to the block case. We propose a generalization of concepts common to

the single right-hand side scenario as convergence (cf. Definition 2.8.1), breakdown (Definition 2.6.1) and stagnation (Definition 2.9.1) to the multiple right-hand side scenario. We present also in this chapter the block Arnoldi (Algorithm 2.5.1) in Section 2.5 and the block GMRES (BGMRES, in Algorithm 2.7.1) in Section 2.7 and we briefly discuss some few differences between GMRES and BGMRES. In Section 2.10 we provide the final remarks for this chapter.

During all the developments of this chapter, except when explicitly mentioned, we consider exact arithmetic.

2.2 Subspaces and Minimum Block Residual

We reproduce the following definitions which can be found, for instance, in [40, (3.1)] for the sake of completeness.

Definition 2.2.1 (Nested Subspaces)**.** Let the sequence of subspace $\mathcal{V}_i \subset \mathbb{C}^n$ be given for $1 \leq i \leq j$. If it holds that

$$\{0\} = \mathcal{V}_0 \subsetneq \mathcal{V}_1 \subsetneq \mathcal{V}_2 ... \tag{2.2.1}$$

we then say that (2.2.1) is a *nested sequence* of . ■

It is important to notice that the sequence of subspaces in Definition 2.2.1 is finite by definition, since $\dim\left(V_{j-1}\right) < \dim(V_j) \leq n$ has to hold for every $j \geq 1$, meaning that once we find a k such that $\mathcal{V}_k = \mathbb{C}^n$, the sequence stops and $\mathcal{V}_{k+1}$ is undefined. Also, the number of elements in this sequence is bounded by n.

We define now our main object of study for this chapter:

Definition 2.2.2 (Minimum block residual family)**.** A *minimum block residual (MBR) method* is an iterative method that, at each step j solves

$$\min_{\text{range}(\bar{X}) \subset \mathcal{Z}_j} \left(\sum_{i=1}^{p} \left\| Be_i - A\bar{X}e_i \right\|_2^2 \right) \tag{2.2.2}$$

for a given sequence of nested subspaces $\mathcal{Z}_j$ (cf. Definition 2.2.1). ■

By definition, methods belonging to the Euclidean minimum block residual family will converge in a finite number of steps (at worst, when $\mathcal{Z}_j = \mathbb{C}^n$) because the subspace $\mathcal{Z}_j$ has to grow with j.

Definition 2.2.2 is in fact an attempt to generalize the concept of *"minimum residual family"* (or MR family) proposed in [40]. Therein, we find a definition analogous to Definition 2.2.2 for the single right-hand side case which considers any norm $\|.\|_N$ in a specific vectorial subspace, i.e. $\|.\|_N$ does not necessarily have to be a norm in the entire $\mathbb{C}^n$. We opt for a simpler approach in Definition 2.2.2, considering only the Euclidean norm. This choice allows us to simplify equation (2.2.2) to

$$\min_{\text{range}(\bar{X}) \subset \mathcal{Z}_j} \left\| B - A\bar{X} \right\|_F . \tag{2.2.3}$$

Here the Frobenius norm is specially suitable because we can consider it as a block vector norm instead of a matrix norm (cf. [64, p.8]). Moreover, if we require $\bar{X}$ to have minimum Frobenius norm, since A is nonsingular, the problem (2.2.3) has an unique solution.

Next we show Definition 2.2.5 and Definition 2.2.3 which are due to [40] and are just reproduced here for the sake of completeness. We highlight that in [40] it is expected that $\dim(\mathcal{Z}_{j-1} - \mathcal{Z}_j) = 1,\ \forall j = 2, ...,$ whereas we expect $\dim(\mathcal{Z}_{j-1} - \mathcal{Z}_j) \geq 1,\ \forall j = 2, ...$ (cf. Definition 2.2.4) due to the block nature of the problem.

Definition 2.2.3 (Correction Subspace)**.** Consider a given MBR method and its sequence of nested subspaces $\mathcal{Z}_j$ for $j = 1, \ldots$. We define the s_j-dimensional subspace $\mathcal{Z}_j \subset \mathbb{C}^n$ as the *"correction*[1] *subspace"*. ■

Definition 2.2.4. We define

$$k_j = \dim\left(\mathcal{Z}_j - \mathcal{Z}_{j-1}\right), k_1 = \dim\left(\mathcal{Z}_1\right), \quad \text{and therefore,} \quad s_j = \sum_{i=1}^{j} k_i,$$

that is, k_j represents how much the correction subspace grew from iteration $j-1$ to iteration j. ■

Definition 2.2.5 (Residual Approximation Subspace)**.** We denote s_j-dimensional subspace $\mathcal{W}_j = A\mathcal{Z}_j$ as *"approximation subspace"*. Since A is nonsingular, it also follows that

$$\mathcal{W}_{j-1} \subset \mathcal{W}_j$$

for every $j > 1$. Since $\{\mathcal{Z}_j\}_j$ is a nested sequence of subspaces, then $\{\mathcal{W}_j\}_j$ is also a sequence of nested subspaces. ■

Let the columns of $\breve{\mathcal{Z}}_j \in \mathbb{C}^{n \times s_j}$ represent any basis for $\mathcal{Z}_j$. Then

$$Y_j = \arg\min_{\bar{Y} \in \mathbb{C}^{s_j \times p}} \left\| B - A\breve{\mathcal{Z}}_j \bar{Y} \right\|_F, \tag{2.2.4}$$

defines an approximate solution as $X_j = \breve{\mathcal{Z}}_j Y_j$. It is a well known result ([60, p.257], for instance) that the solution of (2.2.4) is given by

$$Y_j = (A\breve{\mathcal{Z}}_j)^\dagger B. \tag{2.2.5}$$

We define the *block residual* of such approximation as

$$\begin{aligned} R_j = B - AX_j &= B - A\breve{\mathcal{Z}}_j (A\breve{\mathcal{Z}}_j)^\dagger B, \\ &= (I - A\breve{\mathcal{Z}}_j (A\breve{\mathcal{Z}}_j)^\dagger) B \end{aligned} \tag{2.2.6}$$

which is clearly an orthogonal projection onto $(A\mathcal{Z}_j)^\perp$. If we define $W_j = P_{\mathcal{W}_j} B$ where $\mathcal{W}_j$ is the approximation subspace according to Definition 2.2.5, we see that

$$R_j = B - W_j, \tag{2.2.7}$$

and in such fashion, we can interpret W_j as the *approximation* of B in $\mathcal{W}_j$ subspace. For this reason, we call $\mathcal{W}_j$ *"residual approximation subspace"* or simply *"approximation subspace"*.

The two following results (Property 2.2.6 and Property 2.2.7) are a generalization of results found in [103], using the concepts of Definition 2.2.3 and Definition 2.2.5. Our contribution in these particular statements is that [103] relies on a specific choice of correction subspace $\mathcal{Z}_j$. Having such a result for any nested subspace instead is crucial for our further discussions in Chapter 3.

From equation (2.2.6) and from the fact that B has full rank, we immediately deduce the following property:

Property 2.2.6 (Residual Nullity)**.** *For any MBR method, it holds that*

$$\operatorname{null}\left(R_j\right) = \dim\left(\mathcal{W}_j \cap \operatorname{range}\left(B\right)\right).$$

The following property can be trivially deduced from Property 2.2.6 by just using the nonsingularity of A.

[1] Following the notation in Eiermann and Ernst [40], if we have an initial guess X_0, we try to find a *correction* $C_j \in \mathbb{C}^{n \times p}$ where $\operatorname{range}\left(C_j\right) \subset \mathcal{Z}_j$ such that $X_j = X_0 + C_j$ minimizes the residual.

Property 2.2.7 (Exact Solution Intersection)**.** *For any MBR method, it holds that*

$$\operatorname{null}\left(R_j\right) = \dim\left(\mathcal{Z}_j \cap \operatorname{range}\left(X_*\right)\right),$$

where X_* is the exact solution of $AX = B$.

In other words, whenever the nullity of the residual is different from zero, it means that we have found the exact solution of one or more (or a linear combination of) linear systems inside $\mathcal{Z}_j$. Several strategies that will be studied later are heavily based on this property of minimum block residual methods.

Because we assumed that $\mathcal{Z}_j$ is a nested subspace, every $\breve{\mathcal{Z}}_j$ can be decomposed in a recursion as

$$\breve{\mathcal{Z}}_j = \begin{bmatrix}\breve{\mathcal{Z}}_{j-1} & \breve{Z}_j\end{bmatrix} = \begin{bmatrix}\breve{Z}_1 & \breve{Z}_2 & \dots & \breve{Z}_j\end{bmatrix} \tag{2.2.8}$$

where each $\breve{Z}_i \in \mathbb{C}^{n\times k_i}$, $1 \le i \le j$ is a full rank matrix and k_i is given in Definition 2.2.4.

Next we define some figures useful for describing how we expand the subspace $\mathcal{Z}_j$ to $\mathcal{Z}_{j+1}$.

Definition 2.2.8 (Expansion Subspace)**.** Let $\mathcal{S}_j \subset \mathbb{C}^n$ be a given subspace of dimension $p_j + n_j$ for some $p_j, n_j \in \mathbb{N}$ such that

$$n_j = \dim\left(\mathcal{Z}_j \cap \mathcal{S}_j\right) \quad \text{and naturally} \quad p_j = \dim\left(\mathcal{S}_j\right) - n_j.$$

We expand the subspace $\mathcal{Z}_j$ as

$$\mathcal{Z}_{j+1} = \mathcal{Z}_j + \mathcal{S}_j.$$

■

In Section 2.3 we show that most methods perform an expansion of subspace according to Definition 2.2.8, in the sense that it disposes of a subspace to be *"added"* to $\mathcal{Z}_j$, but nothing guarantees that a direct sum holds.

Remark 2.2.9. In a more advanced scenario, instead of disposing of a subspace $\mathcal{S}_j$ of dimension $p_j + n_j$ to add to $\mathcal{Z}_j$, we dispose of a subspace $\hat{\mathcal{K}}_j$ of dimension $p_j + n_j$ instead, where

$$\begin{aligned}\hat{\mathcal{K}}_j &= \mathcal{S}_j \oplus \mathcal{P}_j \\ \dim\left(\mathcal{S}_j\right) &= k_{j+1} + n_j,\ \dim\left(\mathcal{P}_j\right) = d_{j+1} \\ p_j &= k_{j+1} + d_{j+1}.\end{aligned}$$

These methods use a subspace of dimension k_{j+1} to update $\mathcal{Z}_j$ and neglect a subspace of dimension d_{j+1} (either discarding it or saving it for other purposes). This scenario is going to be discussed in details in Chapter 3. For the purpose of this chapter, we simply assume that $d_j = 0, \forall j \ge 1$, meaning that $k_{j+1} = p_j$ i.e. we use all the subspace available to expand the correction subspace $\mathcal{Z}_j$. ■

2.3 The Block Krylov Subspace

In this section we define the so called *block Krylov subspace*. Altough many authors (cf. [64]) use a matrices subspace for that purpose, we choose here to define it as a subspace of $\mathbb{C}^n$ as in [44] :

Definition 2.3.1 (Block Krylov Subspace)**.** We define a *block Krylov subspace*, as

$$\mathcal{K}_k^{\square}(A, B) = \operatorname{range}\left(\begin{bmatrix}B & AB & \dots & A^{(k-1)}B\end{bmatrix}\right) \in \mathbb{C}^n \tag{2.3.1}$$

This can be rewritten as

$$\mathcal{K}_k^{\square}(A,B) = \sum_{i=1}^{j} \mathcal{K}_k(A, Be_i) \tag{2.3.2}$$

$$= \mathcal{K}_k(A, Be_1) + \mathcal{K}_k(A, Be_2) + ... + \mathcal{K}_k\left(A, Be_p\right), \tag{2.3.3}$$

(cf. [44, (2.12)]) where $\mathcal{K}_k(A, Be_i)$ denotes the k-th Krylov subspace of A with relation to Be_i. ■

Nothing guarantees that a direct sum holds in (2.3.3), therefore these subspaces might have a nonzero intersection - a key property which we explore in Section 3.6 of Chapter 3.

Having in mind what was discussed in the previous section, it is a common practice to set $\mathcal{Z}_j = \mathcal{K}_j^{\square}(A,B)$, giving rise to the so called *block Krylov subspace iterative method*. Examples of block Krylov iterative methods are block GMRES [134] (presenting minimal block residual properties as in Definition 2.2.2), block BiCGStab [63], block IDR(s) [38] (another class of methods, focusing on orthogonal residual properties which we do not address in this thesis) and block QMR [54] (presenting a quasi-minimal residual property, that is, it minimizes the residual only with respect to a given subspace rather than the entire $\mathbb{C}^n$). This strategy is popular because one can ensure that the exact solution lies inside a block Krylov subspace, as the following corollary shows:

Corollary 2.3.2. *Let X_* be the block solution of $AX = B$ and $\ell^{(i)}$ the degree of the minimum polynomial of A with respect to Be_i. Then*

$$\text{range}(X_*) \subset \mathcal{K}_\ell^{\square}(A,B)$$

where $\ell = \max\{\ell^{(i)}\}_{i=1}^{p}$. We call ℓ the block grade *of A with respect to B.*

Proof. It is a well known result that the exact solution of each $AXe_i = Be_i$ lies in $\mathcal{K}_{\ell^{(i)}}(A, Be_i)$ (see [74], for instance). From equation (2.3.3) we have the proof completed. □

We refer to [64, §3] for details on the block grade of a matrix with respect to a given block right-hand side, specially to Lemma 5 (where we find an equivalent definition for the block grade) and Theorem 9 (for an equivalent version of Corollary 2.3.2).

Since each $\ell^{(i)}$ is equal or smaller than[2] n, it is always advantageous to look for a solution in $\mathcal{K}_\ell^{\square}(A,B)$ rather than in $\mathbb{C}^n$, but it is specially interesting for the cases in which $\dim(\mathcal{K}_\ell^{\square}(A,B))$ is much smaller than n. Also the block Krylov subspaces allow us to derive some properties in the minimum block residual scenario, which we discuss along this chapter.

Using the notation in Definition 2.2.8, we imply from the definition of the block Krylov subspace (2.3.1) that for building such a subspace, we set $\mathcal{S}_j = \text{range}(A^jB)$ every iteration $j \geq 1$, with $\mathcal{Z}_1 = \text{range}(B)$. For this specific choice then, according to Definition 2.2.3, for every j we obtain

$$s_j = \dim\left(\mathcal{K}_j^{\square}(A,B)\right)$$

$$n_j = \dim\left(\mathcal{K}_j^{\square}(A,B) \cap \text{range}\left(A^jB\right)\right)$$

$$p_j = k_{j+1} = p - n_j.$$

An important result for the single right-hand side case (i.e. $p = 1$) is that if $\text{range}\left(A^jB\right) \subset \mathcal{K}_j(A,B)$, then[3] $\text{range}(X_*) \subset \mathcal{K}_j(A,B)$, meaning that the exact solution for the system $AX = B$ is already known and that $j \geq \ell$, where ℓ is the block grade of A with respect to B. Following our notation, for $p = 1$,

[2] this result can be found, for instance, in [72, p.142, Theorem 3.3.1]

[3] this can be found, for instance, in [74]

range $\left(A^j B\right) \subset \mathcal{K}_j(A,B)$ implies in $n_j = 1$ and thus $p_j = 0$. This is called in the literature *happy breakdown* or *Arnoldi breakdown*[4].

In the multiple right-hand side scenario (i.e. $p > 1$) an analogous phenomenon is observed. In fact, whenever $p - p_j > 0$, it can be proved that a linear combination of solutions already lies inside $\mathcal{K}_j^{\square}(A,B)$ (cf. [103]). The following proposition formalizes this concept.

Proposition 2.3.3. *At each iteration j of a MBR method using block Krylov subspace as correction subspace, it holds that a linear combination of $p - p_j$ solutions lie inside $\mathcal{K}_j^{\square}(A,B)$.*

Proof. We have that

$$\begin{aligned}\dim\left(\text{range}\,(B) \cap A\mathcal{K}_j^{\square}(A,B)\right) &= \dim\left(\text{range}\,(B)\right) + \dim\left(A\mathcal{K}_j^{\square}(A,B)\right) \\ &\quad - \dim\left(\text{range}\,(B) + A\mathcal{K}_j^{\square}(A,B)\right)\end{aligned}$$

But

$$\text{range}\,(B) + A\mathcal{K}_j^{\square}(A,B) = \mathcal{K}_{j+1}^{\square}(A,B).$$

Moreover from the nonsingularity of A we have that

$$\dim\left(A\mathcal{K}_j^{\square}(A,B)\right) = \dim\left(\mathcal{K}_j^{\square}(A,B)\right)$$

therefore,

$$\begin{aligned}\dim\left(\text{range}\,(B) \cap A\mathcal{K}_j^{\square}(A,B)\right) &= \dim\left(\text{range}\,(B)\right) + \dim\left(\mathcal{K}_j^{\square}(A,B)\right) \\ &\quad - \dim\left(\mathcal{K}_{j+1}^{\square}(A,B)\right) \\ &= p + s_j - s_{j+1} \\ &= p - k_{j+1} = p - p_j.\end{aligned}$$

Again from the nonsingularity of A, we obtain

$$p - p_j = \dim\left(\text{range}\left(A^{-1}B\right) \cap \mathcal{K}_j^{\square}(A,B)\right) = \dim\left(\text{range}\,(X_*) \cap \mathcal{K}_j^{\square}(A,B)\right)$$

where X_* denotes the exact solution of $AX = B$ finalizing the proof. □

Note that for this choice of $\mathcal{Z}_j$, if we always set $\mathcal{S}_j = \text{range}(A^j B)$ we have that $\dim(\mathcal{S}_j) = p$, thus $p - p_j = n_j$ for any j. However, in further discussions we show that practical algorithms based on this idea do not always span a subspace $\mathcal{S}_j$ of dimension p, and thus $n_j \neq p - p_j$ for every j, reason why we choose to present Proposition 2.3.3 (and specially Corollary 2.3.4 at the end of this section) with $p - p_j$ instead.

To clearly understand Proposition 2.3.3, let $\bar{X}_* \in \mathbb{C}^{n\times(p-p_j)}$ represent a basis for the subspace

$$\text{range}\,(X_*) \cap \mathcal{K}_j^{\square}(A,B).$$

Since $\bar{X}_*$ can be written as a linear combination of a basis both of range(X_*) and $\mathcal{K}_j^{\square}(A,B)$, it follows immediately that there are full-rank complex matrices L_1 and L_2 of suitable dimension such that

$$\bar{X}_* = \mathscr{Z}_j L_1 \qquad \text{and} \qquad \bar{X}_* = X_* L_2,$$

[4] the origin of the name is not related to Krylov subspaces but to the well known Arnoldi algorithm (which can be found in [5] or [107, §6.12], for instance) which generates a stable basis for the Krylov subspace. We study the block version of this algorithm and the occurrence of breakdowns in more details in Section 2.5.

where $\mathscr{Z}_j \in \mathbb{C}^{n\times s_j}$ is a basis to $\mathcal{K}_j^{\square}(A,B)$, and consequently

$$X_* L_2 = \mathscr{Z}_j L_1,$$

showing that one can write a part of X_* as a linear combination of $\mathscr{Z}_j$.

Although an analogous of the following corollary can also be found in [103, Corolary 1], we reproduce it here for the sake of completeness. It is a direct implication of Proposition 2.3.3 and Property 2.2.7.

Corollary 2.3.4. *At each iteration j of a MBR method using block Krylov subspace as correction subspace, it holds that*

$$\operatorname{null}\left(R_j\right) = p - p_j.$$

2.4 Preconditioning the Correction Subspace

A practice even more common than choosing $\mathcal{Z}_j$ as $\mathcal{K}_j^{\square}(A,B)$ is to choose a *"preconditioned block Krylov subspace"* instead. It consists in choosing a nonsingular *preconditioning matrix* $\mathscr{M} \in \mathbb{C}^{n\times n}$ such that $A\mathscr{M}$ presents better numerical properties, to then solve the system

$$A\mathscr{M}\mathscr{M}^{-1}X = A\mathscr{M}T = B \tag{2.4.1}$$

instead. Once T is known, we take $X = \mathscr{M}T$ to retrieve the solution of the original system. For solving (2.4.1) one then uses

$$\mathcal{Z}_j = \mathscr{M} \times \operatorname{span}\left\{\begin{bmatrix} B & A\mathscr{M}B & \dots & (A\mathscr{M})^{(j-1)}B\end{bmatrix}\right\} = \mathscr{M}\mathcal{K}_j^{\square}(A\mathscr{M},B) \tag{2.4.2}$$

as correction subspace[5].

An even more general concept is the use of *nonlinear variable preconditioners*. It consists in applying a (nonlinear) preconditioning operator which depends on the iteration. For a given iteration j and matrix $V \in \mathbb{C}^{n\times p}$, we represent the application of the preconditioner on V with

$$Z = \mathcal{M}_j(V) \tag{2.4.3}$$

where $Z \in \mathbb{C}^{n\times p}$ is the *preconditioned* V . Also, for the sake of simplicity, we always consider that the operator $\mathcal{M}_j(.)$ is rank-preserving for every j (that is, $\operatorname{rank}(V) = \operatorname{rank}(Z)$) and that it satisfies the following definition.

Definition 2.4.1 (Rank-Preserving Variable Nonlinear Preconditioner)**.** Suppose we dispose of a matrix $\mathscr{V}_j = \begin{bmatrix} V_1 & V_2 & \dots & V_j\end{bmatrix}$ where each $V_i \in \mathbb{C}^{n\times k_i}$ for some sequence of values of k_i with $1 \le i \le j$, and that we obtain $\mathscr{Z}_j = \begin{bmatrix}\mathcal{M}_1(V_1) & \mathcal{M}_2(V_2) & \dots & \mathcal{M}_j(V_j)\end{bmatrix}$ by applying the sequence of (possibly) nonlinear operators $\mathcal{M}_i$, $1 \le i \le j$. If $\operatorname{rank}\left(\mathscr{Z}_j\right) = \operatorname{rank}\left(\mathscr{V}_j\right)$, we say that the sequence of preconditioner operator $\mathcal{M}_i$, $1 \le i \le j$ is rank-preserving.

We then establish the following lemma.

Lemma 2.4.2 (Equivalent Preconditioner Matrix)**.** *Given $\mathscr{V}_j$ a full rank matrix, for any sequence of rank-preserving preconditioner operators $\mathcal{M}_i$, $1 \le i \le j$, there is at least one nonsingular matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$ such that*

$$\mathscr{Z}_j = \mathscr{M}_j\mathscr{V}_j \tag{2.4.4}$$

holds.

[5] because $\operatorname{range}(T) \subset \mathcal{K}_\ell^{\square}(A\mathscr{M},B)$ according to Corollary 2.3.2, we know that $\operatorname{range}(X) \subset \mathscr{M}\mathcal{K}_\ell^{\square}(A\mathscr{M},B)$; thus $\mathscr{M}\mathcal{K}_\ell^{\square}(A\mathscr{M},B)$ is a more suitable correction subspace than $\mathcal{K}_\ell^{\square}(A\mathscr{M},B)$

Proof. One example of such a matrix is

$$\mathscr{M}_j = \begin{bmatrix} \mathscr{Z}_j & \mathscr{Z}_j^{\perp} \end{bmatrix} \begin{bmatrix} \mathscr{V}_j & \mathscr{V}_j^{\perp} \end{bmatrix}^{-1}$$

where $\mathscr{Z}_j^{\perp}$ (respectively $\mathscr{V}_j^{\perp}$) is the orthogonal complement of $\mathscr{Z}_j$ (respectively $\mathscr{V}_j$) in $\mathbb{C}^n$. □

Lemma 2.4.2 also implies that there is a matrix M_i such that

$$Z_i = M_i V_i$$

for every $1 \leq i \leq j$. We prefer the notation on (2.4.4) over the one in (2.4.3) throughout this thesis, but they are equivalent.

Both flexible and fixed preconditioners are relevant and widely applied, and the choice between either is problem dependent. Perhaps the most well known class of fixed preconditioners are the incomplete factorizations [109, Chapter 10]. Concerning variable preconditioners we can exemplify inner-outer iteration methods (see [113]). Quoting [113] about variable preconditioners, *"one can also consider preconditioners which might improve using information from previous iterations"* (see [8, 48, 75]). Because the variable preconditioner also covers fixed preconditioner and unpreconditioned case, in this whole thesis we consider that a variable preconditioner is being used unless otherwise noted, and we always suppose that the matrices $M_i \in \mathbb{C}^{n\times n}$, $1 \leq i \leq j$ as well as $\mathscr{M}_j$ were properly chosen.

The block Krylov correction subspace using a variable preconditioner is thus introduced as follows

$$\mathcal{Z}_j = \operatorname{span}\left\{ \begin{bmatrix} M_1 B & M_2 A M_1 B & \dots & M_j A \dots A M_2 A M_1 B \end{bmatrix} \right\}, \tag{2.4.5}$$

for a given sequence of nonsingular preconditioning matrices $M_i \in \mathbb{C}^{n\times n}$. It is not trivial to write (2.4.5) as a (block) Krylov subspace, but the following theorem shows that if the sequence of preconditioning operators M_i is rank-preserving this is always possible.

Theorem 2.4.3. *Define the s_j-dimensional subspaces*

$$\mathcal{Z}_j = \operatorname{span}\left\{ \begin{bmatrix} M_1 B & M_2 A M_1 B & \dots & M_j A \dots A M_2 A M_1 B \end{bmatrix} \right\},$$

where the sequence of variable preconditioners $M_i \in \mathbb{C}^{n\times n}$, $1 \leq i \leq j$ is rank-preserving (see Definition 2.4.1). There is always a nonsingular matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$ such that

$$\mathcal{Z}_j = \mathscr{M}_j \mathcal{K}_j^{\square}(A\mathscr{M}_j, B).$$

Proof. For the theorem to be proved, there should be a matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$ such that the subspace

$$\begin{aligned} \mathcal{Z}_j &= \mathscr{M}_j \mathcal{K}_j^{\square}(A\mathscr{M}_j, B) \\ &= \mathscr{M}_j \times \operatorname{span}\left\{ \begin{bmatrix} B & A\mathscr{M}_j B & \dots & (A\mathscr{M}_j)^{j-1} B \end{bmatrix} \right\} \end{aligned}$$

equals (2.4.5). One possible case in which this is true is when there is a matrix $\mathscr{M}_j$ such that all the equalities

$$\begin{aligned} &\mathscr{M}_j B = M_1 B \\ &\mathscr{M}_j A \mathscr{M}_j B = M_2 A M_1 B \\ &\mathscr{M}_j (A\mathscr{M}_j)^2 B = M_3 A M_2 A M_1 B \\ &\vdots \\ &\mathscr{M}_j (A\mathscr{M}_j)^{j-1} B = M_j A \dots M_3 A M_2 A M_1 B \end{aligned} \tag{2.4.6}$$

hold. However, substituting the first equality in the subsequent, we can rewrite the requirement as

$$\begin{aligned}
&\mathscr{M}_j B = M_1 B \\
&\mathscr{M}_j A M_1 B = M_2 A M_1 B \\
&\mathscr{M}_j A \mathscr{M}_j A M_1 B = M_3 A M_2 A M_1 B \\
&\vdots \\
&\mathscr{M}_j (A\mathscr{M}_j)^{j-2} A M_1 B = M_j A \dots M_3 A M_2 A M_1 B.
\end{aligned}$$

Substituting the second equality on the subsequent, and doing so recursively implies that this requirement can be written as

$$\begin{aligned}
&\mathscr{M}_j B = M_1 B \\
&\mathscr{M}_j A M_1 B = M_2 A M_1 B \\
&\mathscr{M}_j A M_2 A M_1 B = M_3 A M_2 A M_1 B \\
&\vdots \\
&\mathscr{M}_j A \dots M_3 A M_2 A M_1 B = M_j A \dots M_3 A M_2 A M_1 B.
\end{aligned}$$

Denoting $\breve{S}_k = A M_{k-1} \dots A M_1 B$, for $1 < k < j$ and $\breve{S}_1 = B$ we finally have that

$$\begin{aligned}
\mathscr{M}_j \breve{S}_1 &= M_1 \breve{S}_1 \\
\mathscr{M}_j \breve{S}_2 &= M_2 \breve{S}_2 \\
\mathscr{M}_j \breve{S}_3 &= M_3 \breve{S}_3 \\
&\vdots \\
\mathscr{M}_j \breve{S}_j &= M_j \breve{S}_j.
\end{aligned}$$

This means that we are looking for a nonsingular matrix $\mathscr{M}_j$ such that

$$\mathscr{M}_j \begin{bmatrix} \breve{S}_1 & \breve{S}_2 & \dots & \breve{S}_j \end{bmatrix} = \begin{bmatrix} M_1 \breve{S}_1 & M_2 \breve{S}_2 & \dots & M_j \breve{S}_j \end{bmatrix}. \tag{2.4.7}$$

Because we assumed that the sequence of variable preconditioners $M_i \in \mathbb{C}^{n \times n}$ is rank-preserving,

$$\operatorname{rank}\left(\begin{bmatrix} \breve{S}_1 & \breve{S}_2 & \dots & \breve{S}_j \end{bmatrix} \right) = \operatorname{rank}\left(\begin{bmatrix} M_1 \breve{S}_1 & M_2 \breve{S}_2 & \dots & M_j \breve{S}_j \end{bmatrix} \right)$$

allowing us to use Lemma 2.4.2 to show that there is always a nonsingular matrix $\mathscr{M}_j$ such that (2.4.7) holds, proving the theorem. □

The following corollary is another way to state Theorem 2.4.3.

Corollary 2.4.4. *Define the s_j-dimensional subspaces*

$$\mathcal{Z}_j = \operatorname{span}\left\{ \begin{bmatrix} M_1 B & M_2 A M_1 B & ..., M_j A ... A M_2 A M_1 B \end{bmatrix} \right\},$$

where the sequence of variable preconditioners $M_i \in \mathbb{C}^{n \times n}$ is rank-preserving (see Definition 2.4.1). There is always a nonsingular matrix $\mathscr{T}_j \in \mathbb{C}^{n \times n}$ such that

$$\mathcal{Z}_j = \mathcal{K}_j^{\square}(\mathscr{T}_j A, Z_1).$$

where $Z_1 = M_1 B$.

Proof. Analogous to Theorem 2.4.3. □

To the best of our knowledge Theorem 2.4.3 (followed by Corollary 2.4.4) is the first demonstration on how to describe subspaces of the form (2.4.5) as a (block) Krylov subspace for both single and multiple right-hand side case, being one of the main theoretical contributions of this chapter of the thesis.

Notice that Corollary 2.4.4 clarifies what was stated in [114, p.27] for the single right-hand side case: "there may not exist any Krylov subspace containing $\mathcal{Z}_j$". In fact, the proof of Theorem 2.4.3 relies on the rank-preserving assumption of the variable preconditioner, and in a general case, we do not guarantee that $\mathcal{Z}_j$ can be written as a block Krylov subspace. Although theoretically speaking this assumption is rather strong, in practice, a rank-deterioration rarely happens due to the preconditioner application, thus justifying our assumption.

Also, it is not trivial to define the concept of block grade (cf. Corollary 2.3.2) for a flexibly preconditioned (block) Krylov subspace because the matrix $\mathscr{M}_j$ changes every time we expand the subspace $\mathcal{Z}_j$. i.e

$$\mathcal{Z}_j = \mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B) \quad \text{but} \quad \mathcal{Z}_{j+1} \not\subset \mathscr{M}_j\mathcal{K}_{j+1}^{\square}(A\mathscr{M}_j, B).$$

However, as long as the subspace $\mathcal{Z}_j$ grows with j, the respective MBR method is convergent (cf. [114, p.27]). In practice, when using a variable preconditioner, we expect to build a variable preconditioned block Krylov subspace such that

$$\operatorname{range}(X_*) \subset \mathscr{M}_k\mathcal{K}_k^{\square}(A\mathscr{M}_k, B) \subset \mathcal{K}_\ell^{\square}(A, B)$$

where $\dim(\mathscr{M}_k\mathcal{K}_k^{\square}(A\mathscr{M}_k, B)) < \dim(\mathcal{K}_\ell^{\square}(A, B))$ but such inequality holds only if the variable preconditioning matrices M_i were properly chosen, and such a choice is highly problem dependent.

We rewrite Proposition 2.3.3 for the flexibly preconditioned case.

Proposition 2.4.5. *At each iteration j of a MBR method using*

$$\mathcal{Z}_j = \operatorname{span}\left\{\begin{bmatrix} M_1B & M_2AM_1B & \dots & M_jA\dots AM_2AM_1B \end{bmatrix}\right\}, \tag{2.4.8}$$

as correction subspace for a given sequence of rank-preserving variable preconditioning nonsingular matrices $M_i \in \mathbb{C}^{n\times n}$, it holds that a linear combination of $p - p_j$ solutions lies inside $\mathcal{Z}_j$.

Proof. Thanks to Theorem 2.4.3 we know that $\mathcal{Z}_j = \mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)$ for some nonsingular matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$. Thus

$$\begin{aligned}\dim\left(\operatorname{range}(B) \cap A\mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right) &= \dim\left(\operatorname{range}(B)\right) + \dim\left(A\mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right) \\ &\quad - \dim\left(\operatorname{range}(B) + A\mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right)\end{aligned}$$

But

$$\operatorname{range}(B) + A\mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B) = \mathcal{K}_{j+1}^{\square}(A\mathscr{M}_j, B).$$

Moreover from the nonsingularity of $A\mathscr{M}_j$ we have that

$$\dim\left(A\mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right) = \dim\left(\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right)$$

therefore,

$$\begin{aligned}\dim\left(\operatorname{range}(B) \cap A\mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right) &= \dim\left(\operatorname{range}(B)\right) + \dim\left(\mathcal{K}_j^{\square}(A\mathscr{M}_j, B)\right) \\ &\quad - \dim\left(\mathcal{K}_{j+1}^{\square}(A\mathscr{M}_j, B)\right) \\ &= p + s_j - \dim\left(\mathcal{K}_{j+1}^{\square}(A\mathscr{M}_j, B)\right).\end{aligned}$$

To find out the dimension of $\mathcal{K}^{\square}_{j+1}(A\mathscr{M}_j, B)$ we use once again the rank-preserving assumption of the variable preconditioner. It holds that

$$\mathcal{Z}_{j+1} = \text{span}\left\{\begin{bmatrix} M_1 B & M_2 A M_1 B & \dots & M_{j+1} A M_j \dots A M_1 B \end{bmatrix}\right\}$$

and

$$\begin{aligned}\mathcal{K}^{\square}_{j+1}(A\mathscr{M}_j, B) &= \text{range}\,(B) + A\mathscr{M}_j \mathcal{K}^{\square}_j(A\mathscr{M}_j, B) = \text{range}\,(B) + A\mathcal{Z}_j \\ &= \text{span}\left\{\begin{bmatrix} B & AM_1 B & AM_2 AM_1 B & \dots & AM_j \dots AM_1 B \end{bmatrix}\right\}.\end{aligned}$$

Because of the rank-preserving assumption on the variable preconditioner, we conclude that $\dim(\mathcal{Z}_{j+1}) = \dim(\mathcal{K}^{\square}_{j+1}(A\mathscr{M}_j, B))$, and as so

$$\begin{aligned}\dim\left(\text{range}\,(B) \cap A\mathscr{M}_j \mathcal{K}^{\square}_j(A\mathscr{M}_j, B)\right) &= p + s_j - \dim\left(\mathcal{K}^{\square}_{j+1}(A\mathscr{M}_j, B)\right) \\ &= p + s_j - s_{j+1} \\ &= p - p_j.\end{aligned}$$

Again from the nonsingularity of A, we obtain

$$p - p_j = \dim\left(\text{range}\left(A^{-1}B\right) \cap \mathscr{M}_j \mathcal{K}^{\square}_j(A\mathscr{M}_j, B)\right) = \dim\left(\text{range}\,(X_*) \cap \mathcal{Z}_j\right)$$

where X_* denotes the exact solution of $AX = B$, finalizing the proof. □

Although the unpreconditioned case shown in Proposition 2.3.3 can be found in other publications (cf. [103]), Proposition 2.4.5 is new to the best of our knowledge. This result will be used in Section 3.6.

For a detailed study over variable preconditioners for iterative solvers, we recommend the reading of [107, §9.4], [113, 105, 27, 23].

2.5 The Block Arnoldi Algorithm

As mentioned previously, looking for an approximate solution inside the (preconditioned) block Krylov subspace is a common strategy. However, the (block) Krylov basis can be very ill-conditioned if it is built naïvely according to its definition (2.3.3) and it is desirable to construct an orthonormal basis to $\mathcal{K}^{\square}_j(A, B)$ (or $\mathcal{K}^{\square}_j(A\mathscr{M}_j, B)$) for stability reasons. We refer to [78] for a deep study on the conditioning of Gram-Schmidt-based algorithms for generating orthonormal bases and its stability when considering finite precision arithmetic.

We consider that the reader is familiar with variable preconditioners and preconditioned Arnoldi algorithm (as in [107, p.256]). We introduce now the block flexible Arnoldi method (Algorithm 2.5.1) and the block flexible Arnoldi iteration (Algorithm 2.5.2), which are commonly used not only for generating a stable orthonormal basis for a block Krylov subspace, but also in a number of applications such as solving eigenvalues problems (see [79, 122] for instance).

Remark 2.5.1. Algorithm 2.5.2 is presented such that it will remove linear dependent columns of S (if any; cf. line 5 of Algorithm 2.5.2) ensuring thus that V_{j+1} has full rank: a rank-revealing QR (RRQR)[17] algorithm would be used to determine both the deficiency n_j and the decomposition $S\Pi_c = QT$ (with Π_c designing a column permutation matrix). In the literature, removing the linear dependent columns of S is called *"Arnoldi deflation"* [64].

As discussed later, a deficiency of S characterizes a breakdown in the block Arnoldi procedure. We will show in Section 3.6 that this behaviour is rare in practice because it means that a linear combination of $p - p_j$ exact solutions has been found. Thus it is more realistic to consider that the relations $n_j = 0$ and $p_j = p$ do hold for every iteration j. Consequently a standard QR decomposition based on modified Gram-Schmidt is then used instead. ■

Algorithm 2.5.1: *Block flexible Arnoldi*

1 Compute the QR decomposition $B = V_1\Lambda_0$ obtaining $n_0 = \text{null}\,(B) = 0$, $p_0 = \text{rank}\,(B) = p$, $V_1 \in \mathbb{C}^{n\times p_0}$ and $\Lambda_0 \in \mathbb{C}^{p_0\times p}$;
2 Define $s_0 = 0$, and $\mathscr{V}_1 = V_1$;
3 **for** $j = 1, \ldots$ **do**
4 | Apply one block flexible Arnoldi iteration (Algorithm 2.5.2)
5 **end for**

Algorithm 2.5.2: *Block flexible Arnoldi iteration: completion of* $\mathscr{Z}_j \in \mathbb{C}^{n\times s_j}$, $\mathscr{V}_{j+1} \in \mathbb{C}^{n\times(s_j+p_j)}$, $\mathscr{H}_j \in \mathbb{C}^{(s_j+p_j)\times s_j}$ *with* $V_i, Z_i \in \mathbb{C}^{n\times p_{i-1}}$ *for* $1 \le i \le j$, *such that* $(\mathscr{V}_{j+1})^H\mathscr{V}_{j+1} = I$

1 $Z_j = M_jV_j$;
2 $S = AZ_j$;
3 $H_j = \mathscr{V}_j^H S$, where $H_j \in \mathbb{C}^{(s_{j-1}+p_{j-1})\times p_{j-1}}$;
4 $S = S - \mathscr{V}_jH_j$;
5 Compute the QR decomposition $S = V_{j+1}H_{j+1,j}$ obtaining $n_j = \text{null}\,(S)$, $p_j = p_{j-1} - n_j$, $V_{j+1} \in \mathbb{C}^{n\times p_j}$ and $H_{j+1,j} \in \mathbb{C}^{p_j\times p_{j-1}}$;
6 Define $s_j = s_{j-1} + p_{j-1}$, $k_{j+1} = p_j$;
7 Define $\mathscr{Z}_j = \begin{bmatrix} Z_1 & \ldots & Z_j \end{bmatrix}$, $\mathscr{V}_{j+1} = \begin{bmatrix} V_1 & \ldots & V_{j+1} \end{bmatrix}$;
8 Define $\mathscr{H}_j = \begin{bmatrix} \mathscr{H}_{j-1} & H_j \\ 0_{p_j\times s_{j-1}} & H_{j+1,j} \end{bmatrix}$, or $\mathscr{H}_1 = \begin{bmatrix} H_1 \\ H_{2,1} \end{bmatrix}$ if $j = 1$;

Remark 2.5.2. Steps 3 and 4 of Algorithm 2.5.2 amount for the orthogonalization of the basis, which may lack of stability if not performed properly. Indeed in Algorithm 2.5.2 we just present a naïve perspective for the sake of clearness, and an advanced method is suggested for such orthogonalization when implementing this method in practice. Examples cover CGS2 (Classical Gram-Schmidt with reorthogonalization), or BMGS (block Modified Gram-Schmidt) or Ruhe's variant of BMGS [104]. We refer to [59] and [78, Chapter 1] for a deep study on the stability of these methods. ■

Proposition 2.5.3. *After* j *iterations of Algorithm 2.5.1, it holds that*

$$\begin{aligned} \text{range}\left(\mathscr{V}_j\right) &= \mathcal{K}_j^{\square}(A\mathscr{M}_j, B) \\ \text{range}\left(\mathscr{Z}_j\right) &= \mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B) \end{aligned} \tag{2.5.1}$$

for some nonsingular matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$ *representing the action of the variable preconditioner up to iteration* j. *Moreover,* $\mathscr{V}_j \in \mathbb{C}^{n\times s_j}$ *is a full rank orthonormal matrix, and* $\mathscr{Z}_j \in \mathbb{C}^{n\times s_j}$ *has full rank.*

Proof. It is easy to infer from Algorithm 2.5.2 that

$$V_jH_{j,j-1} = (I - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)AM_{j-1}V_{j-1} \tag{2.5.2}$$

for every $j \ge 2$, and because $V_jH_{j,j-1}$ arises from an economic QR decomposition, it always holds that

$$\text{range}(V_jH_{j,j-1}) = \text{range}(V_j).$$

for every $j \ge 2$. From line 1 of Algorithm 2.5.1 we find that $\text{range}\,(V_1) = \text{range}\,(B)$.

We prove then that

$$\text{range}\left(V_j\right) = \text{range}\left(AM_{j-1}\ldots AM_1B\right) - \text{range}\left(\mathscr{V}_{j-1}\right) \tag{2.5.3}$$

for every $j \ge 2$. From (2.5.2), we find that

$$V_2H_{2,1} = (I - \mathscr{V}_1\mathscr{V}_1^H)AM_1V_1.$$

and thus

$$\begin{aligned}\text{range}\,(V_2) &= \text{range}\,\left(V_2 H_{2,1}\right) = \text{range}\,\left((I - \mathscr{V}_1\mathscr{V}_1^H)AM_1V_1\right)\\ &= \text{range}\,\left((I - \mathscr{V}_1\mathscr{V}_1^H)AM_1B\right)\\ &= \text{range}\,(AM_1B) - \text{range}\,(\mathscr{V}_1)\end{aligned}$$

Assuming it is correct for $j-1$, we find out that

$$\begin{aligned}\text{range}\,\left(V_j\right) &= \text{range}\,\left(V_j H_{j,j-1}\right) = \text{range}\,\left((I - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)AM_{j-1}V_{j-1}\right)\\ &= \text{range}\,\left((I - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)AM_{j-1}...AM_1B\right)\\ &= \text{range}\,\left(AM_{j-1}...AM_1B\right) - \text{range}\,\left(\mathscr{V}_{j-1}\right)\end{aligned}$$

proving (2.5.3) by induction. Using this knowledge for every j show us that

$$\begin{aligned}\text{range}\,\left(\mathscr{V}_j\right) &= \text{range}\,\left(\begin{bmatrix}V_1 & V_2 & \dots & V_j\end{bmatrix}\right)\\ &= \text{range}\,(V_1) + \text{range}\,(V_2) + \ldots + \text{range}\,\left(V_j\right)\\ &= \text{range}\,(B) + \text{range}\,(AM_1B) + \text{range}\,(AM_2AM_1B)\\ &+ \ldots + \text{range}\,\left(AM_{j-1}...AM_1B\right)\\ &= \text{span}\left\{\begin{bmatrix}B & AM_1B & \dots & AM_{j-1}...AM_1B\end{bmatrix}\right\}.\end{aligned}$$

In the very same fashion, noticing that $Z_j = M_jV_j$ by definition (and the rank-preserving assumption of the variable preconditioner; see Definition 2.4.1), we obtain that

$$\text{range}\,\left(\mathscr{Z}_j\right) = \text{span}\left\{\begin{bmatrix}M_1B & M_2AM_1B & \dots & M_jAM_{j-2}...AM_1B\end{bmatrix}\right\}.$$

From Theorem 2.4.3 we know that there is always a nonsingular matrix $\mathscr{M}_j$ such that

$$\begin{aligned}\text{range}\,\left(\mathscr{Z}_j\right) &= \text{span}\left\{\begin{bmatrix}M_1B & M_2AM_1B & \dots & M_jAM_{j-2}...AM_1B\end{bmatrix}\right\}\\ &= \mathscr{M}_j\mathcal{K}_j^{\square}(A\mathscr{M}_j, B).\end{aligned}$$

To show that range $\left(\mathscr{V}_j\right) = \mathcal{K}_j^{\square}(A\mathscr{M}_j, B)$ we use the proof of Theorem 2.4.3. To satisfy both equalities in (2.5.1) at once, a possibility is to find a nonsingular matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$ such that all the equalities in (2.4.6) hold as well as

$$\begin{aligned}&A\mathscr{M}_jB = AM_1B\\ &A\mathscr{M}_jA\mathscr{M}_jB = AM_2AM_1B\\ &A\mathscr{M}_j(A\mathscr{M}_j)^2B = AM_3AM_2AM_1B\\ &\vdots\\ &A\mathscr{M}_j(A\mathscr{M}_j)^{j-2}B = AM_{j-1}A\ldots M_3AM_2AM_1B.\end{aligned} \tag{2.5.4}$$

Using the nonsingularity of A, and multiplying from the left every equation in (2.5.4) by A^{-1} we verify that all the conditions in (2.5.4) are already contained in (2.4.6), and thus, any nonsingular matrix $\mathscr{M}_j \in \mathbb{C}^{n\times n}$ satisfying (2.4.6) also satisfies (2.5.4).

To finalize the proof, we highlight that $\mathscr{V}_j$ is orthonormal and full rank by construction, and that $\mathscr{Z}_j$ has full rank because of the assumption of a rank-preserving variable preconditioner. □

To the best of our knowledge, Proposition 2.5.3 is the first proof that the block flexible Arnoldi algorithm indeed generates a basis for a block Krylov subspace, being this one of the contributions of this thesis. Even considering $p = 1$, we are unaware of such a demonstration in the flexible case, although for a fixed preconditioner (or unpreconditioned case) this result is well-known.

Remark 2.5.4. We recall that we represent the application of the flexible preconditioner $\mathcal{M}_j(.)$ on V_j by M_jV_j, that is

$$\mathcal{M}_j(V_j) = M_jV_j. \tag{2.5.5}$$

However, in a general scenario

$$\mathcal{M}_j(B) \neq M_jB. \tag{2.5.6}$$

In Proposition 2.5.3 we just clarify that there always exists a linear operator $\mathscr{M}_j$ such that the referred subspaces are Krylov subspaces. ■

Even though $\mathscr{V}_j$ is an orthonormal basis to the block Krylov subspace, $\mathscr{Z}_j$ is not an orthonormal basis to the correction subspace proposed in Section 2.4 though it is considered a reliable and stable basis [107, 112]. We also note that applying the block Arnoldi algorithm is not equivalent to apply p times the Arnoldi algorithm, because the later would generate p orthonormal basis to p different subspaces but these bases need not to be orthonormal among each other. There is an extra computation effort whenever we prefer the block methods. However, block methods can greatly improve the convergence by using information from all subspaces simultaneously. There are also computational gains whenever we are considering a massively parallel computation environment, but we detail this in the end of Chapter 3.

2.6 Breakdown in Block Arnoldi

In this subsection we define the generalization of the concept of Arnoldi breakdown, or happy breakdown for block Arnoldi algorithm. We postpone some more generic proofs and concepts to Section 3.6 in Chapter 3, and we present only some basic concepts here.

In line 5, if S has full rank (that is, $n_j = 0$ and $p_j = p_{j-1}$), then $H_{j+1,j}$ will be square and nonsingular. In such case, we will have $V_{j+1} = SH_{j+1,j}^{-1}$ which resembles the traditional Arnoldi algorithm for the single right-hand side case, where $H_{j+1,j}$ would be in fact $||S||_2$.

Again considering $p = 1$, the so called "*happy breakdown*" or "*Arnoldi breakdown*" is a situation in which the Arnoldi algorithm can not proceed its execution because range $(S) \subset$ range $\left(\mathscr{V}_j\right)$. Since steps 3 and 4 of Algorithm 2.5.2 can be summarized as

$$S = (I - \mathscr{V}_j\mathscr{V}_j^H)S,$$

at the beginning of step 5 of Algorithm 2.5.2 we will have $S = 0$. In the traditional Arnoldi algorithm for single right-hand side, step 5 of Algorithm 2.5.2 is replaced by $V_{j+1} = SH_{j+1,j}^{-1}$, and thus, it generates a division by zero reason why this phenomena is called "breakdown". This is, however, a "happy" breakdown in the sense that, as mentioned previously (see Proposition 2.4.5), whenever range $(S) \subset$ range$(\mathcal{K}_j^{\square}(A, B)) =$ range$(\mathscr{V}_j)$ we can ensure that the solution X_* lies inside $\mathscr{Z}_j$.

For the multiple right-hand side scenario, we have a similar concept. It is worthy to note that because the QR decomposition always exists, Algorithm 2.5.2 never interrupts its execution, yet we still call such phenomena "breakdown".

Definition 2.6.1 (Partial Breakdown)**.** At the j-th iteration of Algorithm 2.5.1, we say that n_j *partial Arnoldi breakdowns* (or simply *partial breakdowns*) have been detected. Whenever $n_j = p_{j-1}$, we say that a *full Arnoldi breakdown* (or simply *full breakdown*) has been detected.

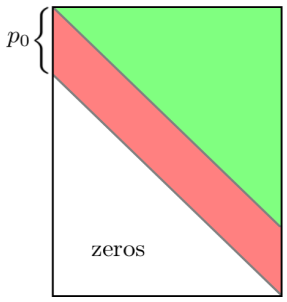


Figure 2.1: *Representation of a block lower Hessenberg matrix with a band of p elements in its lower off-diagonal.*

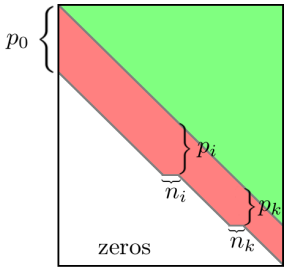


Figure 2.2: *Representation of $\mathscr{H}_j$ generated by Algorithm 2.5.1 after the occurrence of a partial breakdowns at iteration i and anther in iteration k.*

If we use Algorithm 2.5.1 for $p = 1$ we see that the concept of of breakdown as stated in Definition 2.6.1 also applies to the single right-hand side case, except that any breakdown is a full breakdown. Also, likewise in the single right-hand side case, Algorithm 2.5.1 generates a decomposition which we formalize with the following definition.

Definition 2.6.2 (Block Arnoldi Decomposition)**.** Considering the notation in Algorithm 2.5.1 and Algorithm 2.5.2, we refer to

$$A\mathscr{Z}_j = \mathscr{V}_{j+1}\mathscr{H}_j \tag{2.6.1}$$

as the *block Arnoldi relation* or *block Arnoldi decomposition.* ■

Here we note that the matrix $\mathscr{H}_j$ is not an upper Hessenberg matrix as in the ordinary Arnoldi decomposition. In case of no breakdown occurrence, $\mathscr{H}_j$ is block Hessenberg, that is, if no breakdown occurred until iteration j, there is a band of width p of nonzeros below the main diagonal of $\mathscr{H}_j$ (see Figure 2.1). Whenever a breakdown occurs in the block Arnoldi algorithm, the size of the diagonal block decreases. Figure 2.2 shows a graphical representation of the $\mathscr{H}_j$ generated by Algorithm 2.5.2 supposing that one breakdown occurred at iteration i and at iteration k.

2.7 Block GMRES

In this section we quickly present the Block GMRES (BGMRES) method first introduced by Vital in [134]. Analogously to GMRES, BGMRES uses the block Arnoldi method (Algorithm 2.5.2) to generate a block Arnoldi decomposition (Definition 2.6.2). The advantage of using a block Krylov subspace in this case goes beyond ensuring that a solution will be found in a finite number of steps. In fact, the cost for solving the problem (2.2.2) can be greatly decreased thanks to the block Arnoldi relation (2.6.1) .

Defining[6]

$$\Lambda_j = \begin{bmatrix} \Lambda_{j-1} \\ 0_{p_j \times p} \end{bmatrix} \in \mathbb{C}^{(s_j+p_j)\times p}, \qquad \forall j \geq 1,$$

we can write (2.2.2) as

$$\begin{aligned} &\min_{\text{range}(X)\subset\mathcal{Z}_j} \|B - AX\|_F = \min_{Y\in\mathbb{C}^{s_j\times p}} \left\|B - A\mathscr{Z}_j Y\right\|_F \\ &= \min_{Y\in\mathbb{C}^{s_j\times p}} \left\|\mathscr{V}_{j+1}\Lambda_j - \mathscr{V}_{j+1}\mathscr{H}_j Y\right\|_F = \min_{Y\in\mathbb{C}^{s_j\times p}} \left\|\Lambda_j - \mathscr{H}_j Y\right\|_F \end{aligned}$$

[6]The matrix $\Lambda_0 \in \mathbb{C}^{p_0\times p}$ is obtained from the first line of Algorithm 2.5.2

whose solution is given by

$$Y_j = \mathscr{H}_j^{\dagger} \Lambda_j. \tag{2.7.1}$$

Since $\mathscr{H}_j \in \mathbb{C}^{(s_j+p_j)\times s_j}$ and $(A\mathscr{Z}_j) \in \mathbb{C}^{n\times s_j}$, as long as $(s_j + p_j) < n$ it is more advantageous to solve (2.7.1) than (2.2.5). Finally we set the approximate solution as

$$X_j = X_0 + \mathscr{Z}_j Y_j.$$

Naturally BGMRES falls into Definition 2.2.2, and shares all the properties mentioned in Section 2.2. For the sake of completion we present in Algorithm 2.7.1 a pseudocode for restarted block flexible GMRES (BFGMRES).

Algorithm 2.7.1: *Restarted block flexible GMRES (BFGMRES)*

Choose an initial guess $X_0 \in \mathbb{C}^{n\times p}$, a restart parameter m and define a convergence criterion and its scaling matrix;
for *cycle* $= 1, \ldots, m$ **do**
  Compute the initial true block residual $R_0 = B - AX_0$;
  Compute the QR decomposition $R_0 = V_1\Lambda_0$ obtaining $n_0 = \text{null}\,(R_0)$, $p_0 = p - n_0$, $V_1 \in \mathbb{C}^{n\times p_0}$ and $\Lambda_0 \in \mathbb{C}^{p_0\times p}$;
  Define $\mathscr{V}_1 = V_1$, $s_0 = 0$ and $s_1 = p_0$;
  for $j = 1, \ldots, m$ **do**
    Completion of $\mathscr{V}_{j+1}$, $\mathscr{Z}_j$ *and* $\mathscr{H}_j$*:* Apply Algorithm 2.5.2 to obtain

$$A\mathscr{Z}_j = \mathscr{V}_{j+1}\,\mathscr{H}_j \quad \text{with} \quad \mathscr{V}_{j+1} = \left[V_1, V_2, \ldots, V_{j+1}\right], \tag{2.7.2}$$

    s_j and p_j, with $\mathscr{Z}_j \in \mathbb{C}^{n\times s_j}$, $\mathscr{V}_{j+1} \in \mathbb{C}^{n\times(s_j+p_j)}$ and $\mathscr{H}_j \in \mathbb{C}^{(s_j+p_j)\times s_j}$;
    Set $\Lambda_j \in \mathbb{C}^{(s_j+p_j)\times p}$ as $\Lambda_j = \begin{bmatrix}\Lambda_{j-1}\\ 0_{p\times p}\end{bmatrix}$;
    Set $Y_j \in \mathbb{C}^{s_j\times p}$ as the unique minimum Frobenius norm solution of the problem

$$\min_{Y\in\mathbb{C}^{s_j\times p}} \left\|\Lambda_j - \mathscr{H}_j Y\right\|_F \tag{2.7.3}$$

    if *full convergence detected (see Definition 2.8.1)* **then** break;
    ;
  end for
  $X_0 = X_0 + \mathscr{Z}_m Y_m$;
end for

2.8 Convergence Criteria in MBR Methods

In this section we discuss a generalization to the multiple right-hand side scenario of common stopping criteria known for the single right-hand side. In Definition 2.8.1 we propose the *partial convergence*, a key concept which we explore in later sections and chapters.

The intention of MBR methods in most cases is not to find an exact solution for the problem $AX = B$, but only an approximation according to a chosen criterion, usually relying on the norm of the *scaled residual*, defined as

$$R_j D_j,$$

where $D_j \in \mathbb{C}^{p\times p}$ is a nonsingular "*scaling matrix*". The choice for D_j is application dependent, but common choices are the *relative residual criterion*

$$D_j^R = \begin{bmatrix} \frac{1}{\|R_0 e_1\|_2} & & & \\ & \frac{1}{\|R_0 e_2\|_2} & & \\ & & \ddots & \\ & & & \frac{1}{\|R_0 e_p\|_2} \end{bmatrix},$$

the *backward error*

$$D_j^B = \begin{bmatrix} \frac{1}{\|A\|_2\|X_j e_1\|_2 + \|R_0 e_1\|_2} & & \\ & \ddots & \\ & & \frac{1}{\|A\|_2\|X_j e_p\|_2 + \|R_0 e_p\|_2} \end{bmatrix},$$

or the *backward error with respect to A*

$$D_j^A = \begin{bmatrix} \frac{1}{\|A\|_2\|X_j e_1\|_2} & & \\ & \ddots & \\ & & \frac{1}{\|A\|_2\|X_j e_p\|_2} \end{bmatrix}.$$

The relative residual criterion is constant over j, whereas the backward error and the backward error with respect to A varies with j. For the sake of generality, we always consider a generic matrix D_j which may or may not vary at each iteration j.

The goal of the algorithm is then to find a solution X_j such that

$$\|(B - AX_j)D_j\|_F = \|R_j D_j\|_F \leq \varepsilon \tag{2.8.1}$$

for a given threshold $\varepsilon \geq 0$ (notice that if we set $\varepsilon = 0$ we are looking for $X_j = X_*$). Another possibility rather than (2.8.1) is to look for an approximation of the solution X_j such that

$$\|(B - AX_j)D_j\|_\psi = \|R_j D_j\|_\psi \leq \varepsilon \tag{2.8.2}$$

but since satisfying (2.8.1) implies satisfying (2.8.2), for the moment we choose the former approach for the sake of simplicity.

As mentioned earlier, during the execution of Algorithm 2.7.1 (or any MBR method), in some situations it may be known that we have found the exact solution for a linear combination of right-hand sides, namely when R_j is rank deficient. However, we are interested in finding approximations for the solution of a linear combination of right-hand sides rather than the exact solution itself. In fact, according to equation (2.2.2) in Definition 2.2.2, if $\|Be_i - A\bar{X}e_i\|_2$ is large for $i = k$ and small for all $i \neq k$, the Frobenius norm of R_j will be at least as large as $\|Be_k - A\bar{X}e_k\|_2$, hiding the information that some of the linear systems might have already converged.

Another possibility is to compute the norm of each individual column of the block residual and then use the scaled residual for checking convergence individually for each one of them. However, again it doesn't give any information about the linear combination of the solution. The relative block residual

may happen to have two linear dependent columns (therefore R_j is rank deficient) and nothing guarantees that the norm of those individual columns will be small.

For this reason we propose a "*near-rank deficiency*" of the block residual, based on a threshold ε according to the following definition.

Definition 2.8.1 (Partial Convergence)**.** If at the end of the j-th iteration of any MBR method there is a full rank orthonormal matrix $W \in \mathbb{C}^{p\times t}$ such that

$$\left\|R_j D_j W\right\|_F \leq \sqrt{\frac{t}{p}}\,\varepsilon$$

holds, then we say that t *partial convergences* have been detected at iteration j. If $t = p$ (thus $||R_j D_j W||_F = ||R_j D_j||_F \leq \varepsilon$) we say that a *full convergence* has been detected at iteration j. ■

2.9 Stagnation in BGMRES

In this section we propose a new generalization of the concept of stagnation for multiple right-hand side. This concept will be used later on section Section 3.4.

When $p = 1$, it is commonly said that we have a stagnation in iteration j whenever

$$R_j = R_{j-1}.$$

This phenomenon implies a series of other conclusions for the single right-hand side case that are not trivially extended to the multiple right-hand side case. To the best of our knowledge, a generalization of this concept and these properties for multiple right-hand side has not been already addressed.

We propose in Definition 2.9.1 a new definition of stagnation for multiple right-hand side scenario, and we dedicate the remaining of this section showing the equivalence of this definition with the single right-hand side case, and showing that some common behaviours associated with the single right-hand side stagnation are also analogous for this definition of stagnation.

Definition 2.9.1 (Partial Stagnation)**.** Define

$$t_j = p_j - \operatorname{rank}\left((I - \mathscr{V}_j\mathscr{V}_j^H)R_j\right)$$

Whenever $t_j > 0$ at the end of the j-th iteration of BFGMRES (Algorithm 2.7.1), we say that t_j *partial stagnations* have been detected at iteration j. If $t_j = p_j$ then we say that a *full stagnation* has been detected instead. ■

With Proposition 2.9.3 and Proposition 2.9.4 we show some consequences of the occurrence of partial stagnation according to Definition 2.9.1.

Lemma 2.9.2. *Consider the j-th iteration of Algorithm 2.7.1 and define*

$$\left(\Lambda_j - \mathscr{H}_j Y_j\right) = \begin{bmatrix}\hat{R}_j^s \\ \hat{R}_j^p\end{bmatrix} \quad \text{and} \quad \left(\Lambda_{j-1} - \mathscr{H}_{j-1} Y_{j-1}\right) = \hat{R}_{j-1}$$

with $\hat{R}_j^s \in \mathbb{C}^{s_j\times p}$, $\hat{R}_j^p \in \mathbb{C}^{p\times p}$, $\hat{R}_{j-1} \in \mathbb{C}^{s_j\times p}$, and assume that $n_i = 0, 1 \leq i \leq j$. If $\operatorname{null}\left(\hat{R}_j^p\right) = t$, then there exists an orthonormal matrix $L_1 \in \mathbb{C}^{p\times t}$ such that

$$\hat{R}_j^s L_1 = \hat{R}_{j-1} L_1.$$

Proof. We write

$$\begin{bmatrix} \hat{R}_j^s \\ \hat{R}_j^p \end{bmatrix} = \begin{bmatrix} \Lambda_{j-1} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathscr{H}_{j-1} & H_j \\ 0 & H_{(j+1,j)} \end{bmatrix} \begin{bmatrix} Y_j^s \\ Y_j^p \end{bmatrix}$$

with $Y_j^T = [(Y_j^s)^T \quad (Y_j^p)^T]$, $Y_j^s \in \mathbb{C}^{s_j - p \times p}$, $Y_j^p \in \mathbb{C}^{p \times p}$. Then

$$\begin{bmatrix} \hat{R}_j^s \\ \hat{R}_j^p \end{bmatrix} = \begin{bmatrix} \Lambda_{j-1} - \mathscr{H}_{j-1} Y_j^s \\ 0_{p \times p} \end{bmatrix} - \begin{bmatrix} H_j Y_j^p \\ H_{(j+1,j)} Y_j^p \end{bmatrix}$$

and thus

$$\hat{R}_j^p = -H_{(j+1,j)} Y_j^p.$$

If $\operatorname{null}\left(\hat{R}_j^p\right) = t$ then there is an orthonormal matrix $L_1 \in \mathbb{C}^{p \times t}$ such that

$$\hat{R}_j^p L_1 = -H_{(j+1,j)} Y_j^p L_1 = 0.$$

Because we assumed $n_i = 0, 1 \leq i \leq j$, we have that $H_{(j+1,j)}$ is nonsingular, so that

$$H_{(j+1,j)} Y_j^p L_1 = 0 \iff Y_j^p L_1 = 0$$

and consequently

$$\begin{bmatrix} \hat{R}_j^s \\ \hat{R}_j^p \end{bmatrix} L_1 = \begin{bmatrix} \Lambda_{j-1} - \mathscr{H}_{j-1} Y_j^s \\ 0_{p \times p} \end{bmatrix} L_1.$$

Noticing that Y_{j-1} solves the problem

$$\min_{Y \in \mathbb{C}^{s_j - p \times p}} \left\| \Lambda_{j-1} - \mathscr{H}_{j-1} Y \right\|_F$$

and the solution for this minimization is always unique, we deduce that $Y_j^s L_1 = Y_{j-1} L_1$ and

$$\hat{R}_j^s L_1 = \left(\Lambda_{j-1} - \mathscr{H}_{j-1} Y_{j-1} \right) L_1 = \hat{R}_{j-1} L_1$$

finalizing the proof. □

Proposition 2.9.3. *Suppose that each $n_i = 0$ for $1 \leq i \leq j$. Then, for every iteration j of Algorithm 2.7.1 it holds that*

$$\dim\left(\operatorname{range}\left(R_j\right) \cap \operatorname{range}\left(R_{j-1}\right)\right) = t_j \tag{2.9.1}$$

where t_j is given in Definition 2.9.1.

Proof. Following the notation from Lemma 2.9.2, the following always holds

$$\begin{aligned} (I_n - \mathscr{V}_j \mathscr{V}_j^H) R_j &= (I_n - \mathscr{V}_j \mathscr{V}_j^H) \begin{bmatrix} \mathscr{V}_j & V_{j+1} \end{bmatrix} \begin{bmatrix} \hat{R}_j^{s_j} \\ \hat{R}_j^p \end{bmatrix} \\ &= (I_n - \mathscr{V}_j \mathscr{V}_j^H) \mathscr{V}_j \hat{R}_j^{s_j} + (I_n - \mathscr{V}_j \mathscr{V}_j^H) V_{j+1} \hat{R}_j^p \\ &= V_{j+1} \hat{R}_j^p, \end{aligned} \tag{2.9.2}$$

and therefore

$$t_j = \operatorname{null}\left((I_n - \mathscr{V}_j \mathscr{V}_j^H) R_j \right) = \operatorname{null}\left(V_{j+1} \hat{R}_j^p \right) = \operatorname{null}\left(\hat{R}_j^p \right).$$

From Lemma 2.9.2 the proposition is proved. □

The immediate consequence of Lemma 2.9.2 and Proposition 2.9.3 is that a stagnation implies that the residual R_j is linear dependent with the residual R_{j-1}, that is, a linear combination of residuals did not change from iteration $j-1$ to iteration j, establishing the connection between the multiple right-hand side and the single right-hand side case.

The next proposition shows a result which is going to be particular useful in Section 3.4 later on.

Proposition 2.9.4. *For every iteration j of Algorithm 2.7.1, it holds that*

$$\text{range}\left(V_{j+1}\right) \supseteq \text{range}\left((I_n - \mathscr{V}_j\mathscr{V}_j^H)R_j\right). \tag{2.9.3}$$

If no partial stagnation (see Definition 2.9.1) has occurred, then

$$\text{range}\left(V_{j+1}\right) = \text{range}\left((I_n - \mathscr{V}_j\mathscr{V}_j^H)R_j\right). \tag{2.9.4}$$

Proof. From (2.9.2) in the demonstration of Proposition 2.9.3 we already know that (2.9.3) holds.

If no partial stagnation occurs, then we have that

$$\text{rank}\left((I_n - \mathscr{V}_j\mathscr{V}_j^H)R_j\right) = p_j$$

and since $\text{rank}\left(V_{j+1}\right) = p_j$ by definition, we prove (2.9.4). □

2.10 Conclusions

In this chapter we have studied a generalization of the well known GMRES algorithm for multiple right-hand side scenario, the block GMRES. We have developed a theoretical basis related to the subspaces being spanned by BGMRES in the presence of a variable preconditioner. The result found in Theorem 2.4.3 is new to the best of our knowledge even in the single right-hand side case. Some other new properties have been demonstrated (e.g. Proposition 2.5.3 which is a consequence of Theorem 2.4.3).

We also determined a generalization of a set of concepts which are common in the single right-hand side scenario, but not globally formalized for the multiple right hands side scenario (as Definition 2.6.1, Definition 2.8.1 and Definition 2.9.1). Among those, we remark the definition of partial stagnation (cf. Definition 2.9.1) which is not trivially deduced from the single right-hand side case. The importance of these definitions will come clear as we advance in a more complex scenario, in Chapter 3.

Chapter 3

Deflation

3.1 Introduction

In the previous chapter we have studied Block GMRES (BGMRES) due to Vital [134] for solving the problem $AX = B$, with $A \in \mathbb{C}^{n\times n}$ nonsingular, $B, X \in \mathbb{C}^{n\times p}$ where $n \gg p$ and rank $(B) = p$. We extended concepts common in the single right-hand side scenario for the block scenario, as *partial convergence* (when a linear combination of approximate solutions is found rather than the approximate solution of the entire block system) and the *partial breakdown* (which is basically a linear combination of happy breakdowns). BGMRES has ever since been improved based on the assumption that a subspace of the correction subspace can be discarded, a process called *deflation* [64, 78, 103]. It is recognized that to be effective in terms of computational operations, block iterative methods must incorporate a deflation strategy [64], most notably when a partial convergence is detected.

We briefly summarize now the most common deflation techniques available in the literature. In [23, 64, 78, 96] it is proposed the *initial deflation*. It consists in performing a block size reduction of the (scaled) initial residual R_0 relying on its (near) rank deficiency. BFGMRESD [23, 78, 96] computes

$$\begin{array}{cc} R_0 = QT & \text{(QR decomposition)} \\ TD_0 = U_+\Sigma_+W_+^H + U_-\Sigma_-W_-^H + & \text{(singular value decomposition)} \end{array} \tag{3.1.1}$$

where all the singular values larger or equal than a threshold ε^d lie in Σ_+ and those smaller lie in Σ_-, and D_0 is the nonsingular scaling matrix. This gives raise to the low rank approximation of R_0 as $\mathbb{C}^{n\times k_1} \ni \tilde{R}_0 = QU_+\Sigma_+W_+^H$. The algorithm then proceeds with one cycle of nonsingular BGMRES, minimizing $\tilde{R}_0 - A\tilde{X}$ rather than $R_0 - AX$. The cost per iteration is thus reduced since every V_j has k_1 columns instead of p. At the end of the said cycle, some manipulations are performed in order to retrieve the approximate solution for the original problem. A truncated variant called BFGMREST [23, 78, 96] allows the user to set a maximum number of columns allowed in $\tilde{R}_0$ thus *truncating* the block initial residual even if the singular values of the residual are not smaller than ε^d. This strategy aims at reducing the memory requirements of the method when many right-hand sides are considered at once. Furthermore, both BFGMRESD and BFGMREST are proposed for variable preconditioner scenario.

The BlMResDefl [64] uses a very similar technique, but relying on a rank-revealing QR decomposition to obtain the decomposition

$$R_0 = \begin{bmatrix} V_1 & V_1^\Delta \end{bmatrix} \begin{bmatrix} \Lambda_0 \\ \Lambda_0^\Delta \end{bmatrix} \Pi_c^H$$

(cf. [64, (12.1)]), where Π_c is a permutation matrix responsible for reordering the columns of R_0 such that the elements in the diagonal of $[(\Lambda_0)^T \quad (\Lambda_0^\Delta)^T]^T$ are given in nonincreasing order. BlMResDefl then sets the new initial residual to $V_1\Lambda_0$ and proceeds executing one cycle of BGMRES algorithm, as BFGMRESD.

The BlMResDefl [64] proposes not only initial deflation techniques, but also the so called *"Arnoldi deflation"*, already discussed in Remark 2.5.1. It consists of determining which columns of S in line 5 of Algorithm 2.5.2 are linear dependent to ensure that V_{j+1} is not rank deficient. In case of linear dependency, these columns are removed from V_{j+1} (characterizing thus the deflation), and the required manipulations are performed over $\mathscr{H}_j$. This deflation is reported in [64] to never become active in practical numerical experiments.

In [103] a deflation technique similar to *Arnoldi deflation* is proposed for BGMRES-W, therein called *inexact (Arnoldi) breakdown*. Supposing that no deflation was performed, BGMRES-W deflates whenever S is near rank deficient, basing this choice on the singular values of $H_{j+1,j}$ and a threshold ε^d. The deflation incurs some modifications in the block Arnoldi iteration, and the subsequent deflation relies on a submatrix of $\mathscr{H}_j$ instead of $H_{j+1,j}$. We refer the reader to [103] for more details. Nevertheless, as the authors observe, BGMRES-W tends to deflate only at the end of the convergence history, and this observation is confirmed in [76].

In the same publication [103], BGMRES-R is proposed as an alternative to BGMRES-W relying on the singular values of the (scaled) residual R_j every iteration j. Similarly to BFGMRESD, it computes a decomposition as (3.1.1) for R_j, and using a submatrix of U_+ to choose which columns of V_j are going to be carried out for the next block Arnoldi iteration, *postponing* the remaining ones. The postponed columns are used in the block Arnoldi algorithm for orthogonalization purposes only. A FOM variant of BGMRES-W and BGMRES-R can be found in [102], and further numerical experiments in [76].

Besides the aforementioned methods, strategies based on rank-revealing QR-factorizations [21] or singular value decomposition [60] have been notably proposed both in the Hermitian [89, 104] and non-Hermitian cases [2, 9, 32, 54, 81, 92] for block Lanczos methods. They have been shown to be effective with respect to standard block Krylov subspace methods, but since we are focusing on iterative methods showing a minimal residual property for non-Hermitian problems, we do not focus on the study of these methods.

Variable preconditioning is often required when solving large linear systems. This is notably the case when inexact solutions of the preconditioning system using, e.g., nonlinear smoothers in multigrid [96] or approximate interior solvers in domain decomposition methods [127, Section 4.3] are considered. The combination of block methods performing deflation at each iteration and variable preconditioning has been rarely addressed in the literature, although the combination of initial deflation with variable preconditioning has been already explored in [96, 22, 23]. Thus the main purpose of this chapter is to derive a class of flexible minimal block residual methods for non-Hermitian problems that incorporate deflation at each iteration.

This chapter is organized as follows. In Section 3.2 we propose a generalization of the concepts established in Chapter 2 aiming at a method able to judiciously choose which (block) Krylov directions are interesting for expanding the correction subspace $\mathcal{Z}_j$ every iteration j, to then present and re-interpret a block iterative procedure firstly introduced in [103] which is able to build an orthonormal basis for the chosen directions only (the *"deflated block Arnoldi"*). In Section 3.3 a general framework for deflated block Krylov subspace methods is presented. We show that the resulting method (named *"deflated minimal block residual"* method or DMBR for short) always minimizes the Frobenius norm of the block residual and that the singular values of the scaled block residual are always nonincreasing. In Section 3.4 we propose a criterion for choosing which directions to take into account when expanding $\mathcal{Z}_j$ at iteration j, which is mostly based on Section 2.9 as well as [103] and [78]. Then, in Section 3.6 we show that DMBR using the proposed criterion never breaks down, thus guaranteeing convergence (considering a large enough restart size) along with other properties. In Section 3.5 we use DMBR as a framework to describe existing algorithms as BGMRES, BGMRES-R and BFGMRESD and in which situations these algorithms could be considered as equivalent to DMBR. Then in Section 3.9 we demonstrate the effectiveness of DMBR on three academic illustrations and one real life application, showing that in practical cases, none of these methods are algebraically or numerically equivalent to DMBR. These conclusions are later extended with further experiments in Section 4.6. Finally we draw some conclusions in Section 3.10.

3.2 Deflated Block Arnoldi

In this section we present a generalization of the block Arnoldi method (cf. Algorithm 2.5.2 in Section 2.5), which we call here *"deflated block Arnoldi"*, whose focus it to take into account Remark 2.2.9 to then generate a basis of a subspace of smaller dimension.

The deflated block Arnoldi was firstly proposed in [103] although the authors do not define a specific name for the method. The main idea is to redefine line 5 of Algorithm 2.5.1 to

$$S = V_{j+1}H_{j+1,j} + Q_j \quad \text{with} \quad \text{range}\left(Q_j\right) \perp \text{range}\left(\mathscr{V}_{j+1}\right) \tag{3.2.1}$$

where Q_j is chosen such that it takes into account the numerical rank in the QR factorization. It proceeds orthogonalizing every Q_i against $\mathscr{V}_{j+1}$, obtaining $\tilde{\mathscr{Q}}_j = (I - \mathscr{V}_{j+1}\mathscr{V}_{j+1}^H)[Q_1 \quad \dots \quad Q_j]$ and generating what is called in [103] the *inexact block Arnoldi* relation

$$A\mathscr{Z}_j = \mathscr{V}_{j+1}\underline{\mathscr{L}}_j + \tilde{\mathscr{Q}}_j$$

where $\underline{\mathscr{L}}_j = \mathscr{V}_{j+1}^H A\mathscr{Z}_j$ (cf. [103, (26),(28)]). To obtain such a relation a series of other modifications is required in the original block Arnoldi algorithm. We refer to [103, §5] and to Subsection 3.5.1 for more details on BGMRES-R algorithm.

We propose in this section a reformulation of the deflated block Arnoldi to fit the discussion of Chapter 2, specifically Section 2.2 (specially Remark 2.2.9) and Section 2.9. Also, in contrast with [103] in our redefinition the criterion for splitting (3.2.1) is generalized and decoupled from the orthonormalization procedure itself. We address possible criterion later in Section 3.4 and Section 3.7.

Letting $K_{j-1} \in \mathbb{C}^{n\times p_{j-1}}$ denote a full rank matrix whose range contains all the latest p_{j-1} Krylov directions available at the end of iteration $(j-1)$ (with $1 \leq p_{j-1} \leq p$), the most expensive part of Algorithm 2.5.2 at the j-th iteration - when n is large - lies in the p_{j-1} applications of the preconditioner (see line 1) and the subsequent p_{j-1} matrix-vector products (see line 2).

As discussed later in Section 3.4, subspaces of range $\left(K_{j-1}\right)$ may no longer be needed for ensuring convergence along the iterative procedure. The goal of deflated block Arnoldi is thus to exploit this knowledge spanning a smaller correction subspace, avoiding preconditioner application and matrix-vector products over the uninteresting directions, as well as reducing the orthogonalization cost for the next iteration. Suppose that we are able to judiciously decompose range $\left(K_{j-1}\right)$ into:

$$\begin{aligned} \text{range}\left(K_{j-1}\right) &= \text{range}\left(V_j\right) \oplus \text{range}\left(P_{j-1}\right), \\ \begin{bmatrix} V_j & P_{j-1} \end{bmatrix}^H \begin{bmatrix} V_j & P_{j-1} \end{bmatrix} &= I_{p_{j-1}}, \end{aligned} \tag{3.2.2}$$

where $V_j \in \mathbb{C}^{n\times k_j}$, $P_{j-1} \in \mathbb{C}^{n\times d_j}$ with $k_j + d_j = p_{j-1}$. For the sake of generality, having in mind the notation established in Section 1.1, we define

$$\begin{bmatrix} V_i & P_{i-1} \end{bmatrix} = V_i, \quad \text{whenever} \quad d_i = 0, \tag{3.2.3}$$

since P_{i-1} is undefined in such a case, and we consider that d_0 is always equal to zero.

At iteration j we consider the k_j Krylov directions contained in range(V_j), while leaving aside (or deflating) d_j directions contained in range(P_{j-1}), performing matrix-vector products and preconditioner applications *only* over the chosen k_j directions of V_j. We show the j-th iteration of deflated block Arnoldi method using a variable preconditioner in Algorithm 3.2.1.

As in standard block Arnoldi (cf. Algorithm 2.5.2), Algorithm 3.2.1 orthonormalizes AZ_j against $\mathscr{V}_j$ but additionally against P_{j-1} also (line 4 and 5 of Algorithm 3.2.1). Here also Remark 2.5.2 is applicable: a naïve orthonormalization algorithm is presented in Algorithm 3.2.1, but a version of block Arnoldi due to Ruhe [104] or block Householder orthonormalization [8, 123] could be used as well. Notice also that whenever $d_j = 0$, using (3.2.3), we find that one iteration of Algorithm 3.2.1 is equivalent to one iteration of Algorithm 2.5.1, having $k_j = p_{j-1}$, $\hat{\mathscr{H}}_j = \mathscr{H}_j$ and $\hat{\mathscr{V}}_{j+1} = \mathscr{V}_{j+1}$.

In Proposition 3.2.1 we show the flexible Arnoldi relation that is obtained when using the deflated block Arnoldi procedure shown in Algorithm 3.2.1.

Algorithm 3.2.1: *Deflated block flexible Arnoldi iteration: completion of* $\mathscr{Z}_j \in \mathbb{C}^{n\times s_j}$, $\hat{\mathscr{V}}_{j+1} \in \mathbb{C}^{n\times(s_j+p_j)}$, $\hat{\mathscr{H}}_j \in \mathbb{C}^{(s_j+p_j)\times s_j}$ *with* $V_i, Z_i \in \mathbb{C}^{n\times k_i}$ *for* $1 \le i \le j$, *such that* $(\hat{\mathscr{V}}_{j+1})^H \hat{\mathscr{V}}_{j+1} = I$

1 Receives $\begin{bmatrix} V_j & P_{j-1}\end{bmatrix} \in \mathbb{C}^{n\times p_{j-1}}$ with $V_j \in \mathbb{C}^{n\times k_j}$ and $p_{j-1} = k_j + d_j$ where $\begin{bmatrix} V_1 & \dots & V_j & P_{j-1}\end{bmatrix}$ is orthonormal;
2 $Z_j = M_j V_j$;
3 $S = AZ_j$;
4 $H_j = \begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix}^H S$, where $H_j \in \mathbb{C}^{(s_{j-1}+p_{j-1})\times k_j}$;
5 $S = S - \begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix} H_j$;
6 Compute the QR decomposition $S = \hat{V}_{j+1} H_{j+1,j}$ obtaining $n_j = \operatorname{null}(S)$, $\hat{V}_{j+1} \in \mathbb{C}^{n\times(k_j-n_j)}$ and $H_{j+1,j} \in \mathbb{C}^{(k_j-n_j)\times k_j}$;
7 Define $s_j = s_{j-1} + k_j$ and $p_j = p_{j-1} - n_j$;
8 Define $\mathscr{Z}_j = \begin{bmatrix} Z_1 & \dots & Z_j\end{bmatrix}$, $\hat{\mathscr{V}}_{j+1} = \begin{bmatrix} V_1 & \dots & P_{j-1} & \hat{V}_{j+1}\end{bmatrix}$;
9 Define $\hat{\mathscr{H}}_j = \begin{bmatrix} \mathscr{H}_{j-1} & H_j \\ 0_{(k_j-n_j)\times s_{j-1}} & H_{j+1,j}\end{bmatrix}$, or $\hat{\mathscr{H}}_1 = \begin{bmatrix} H_1 \\ H_{2,1}\end{bmatrix}$ for $j = 1$;

Proposition 3.2.1. *With the notation of Algorithm 3.2.1, given* $\mathscr{Z}_{j-1} \in \mathbb{C}^{n\times s_{j-1}}$,$\mathscr{V}_j \in \mathbb{C}^{n\times s_j}$, $\mathscr{H}_{j-1} \in \mathbb{C}^{(s_{j-1}+p_{j-1})\times s_{j-1}}$, $\begin{bmatrix} V_j & P_{j-1}\end{bmatrix} \in \mathbb{C}^{n\times p_{j-1}}$ *and* $V_j \in \mathbb{C}^{n\times k_j}$, *satisfying*

$$A\mathscr{Z}_{j-1} = \begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix} \mathscr{H}_{j-1}$$

with $s_j = s_{j-1} + k_j$, $p_{j-1} = k_j + d_j$ *and* $[\mathscr{V}_j \quad P_{j-1}]^H [\mathscr{V}_j \quad P_{j-1}] = I_{(s_j+d_j)}$, *after applying Algorithm 3.2.1 we obtain the block flexible Arnoldi relation*

$$A\mathscr{Z}_j = \hat{\mathscr{V}}_{j+1} \hat{\mathscr{H}}_j. \tag{3.2.4}$$

where $\hat{\mathscr{V}}_{j+1}$ *is orthonormal.*

Proof. We can summarize one iteration of Algorithm 3.2.1 as

$$\hat{V}_{j+1} H_{j+1,j} = \left(I - \begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix} \begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix}^H\right) AZ_j.$$

Taking H_j according to line 4 of Algorithm 3.2.1, the equation above lead us to

$$AZ_j = \hat{\mathscr{V}}_{j+1} \begin{bmatrix} H_j \\ H_{j+1,j}\end{bmatrix}$$

and this together with the line 1 of Algorithm 3.2.1 yields

$$\begin{bmatrix} A\mathscr{Z}_{j-1} & AZ_j\end{bmatrix} = \left[\begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix} \mathscr{H}_{j-1} \quad \hat{\mathscr{V}}_{j+1} \begin{bmatrix} H_j \\ H_{j+1,j}\end{bmatrix}\right]$$

$$A\begin{bmatrix} \mathscr{Z}_{j-1} & Z_j\end{bmatrix} = \begin{bmatrix} \mathscr{V}_j & P_{j-1} & \hat{V}_{j+1}\end{bmatrix} \left[\begin{array}{c|c} \mathscr{H}_{j-1} & H_j \\ \hline 0_{(k_j-n_j)\times s_{j-1}} & H_{j+1,j}\end{array}\right]$$

$$A\mathscr{Z}_j = \hat{\mathscr{V}}_{j+1} \hat{\mathscr{H}}_j.$$

The orthonormality of $\hat{\mathscr{V}}_{j+1}$ comes from the fact that we considered $\begin{bmatrix} \mathscr{V}_j & P_{j-1}\end{bmatrix}$ already orthonormal and that $\hat{V}_{j+1}$ was orthonormalized using Classical Gram-Schmidt. □

Notice that (3.2.4) is identical to the standard block flexible Arnoldi relation (cf. Definition 2.6.2) except that $\mathscr{V}_{j+1}$ and $\mathscr{H}_j$ are replaced respectively by $\hat{\mathscr{V}}_{j+1}$ and $\hat{\mathscr{H}}_j$. Also, Algorithm 3.2.1 does not address the choice of P_{j-1} or how to define respectively $\mathscr{V}_{j+1}$ and $\mathscr{H}_j$ from $\hat{\mathscr{V}}_{j+1}$ and $\hat{\mathscr{H}}_j$ for its next iteration, which is intimately related to the splitting we mentioned in (3.2.2).

3.3 Deflated Minimal Block Residual

We propose in this section the *"deflated minimal block residual"* method (hence, DMBR), as BGMRES-R [103], uses the deflated block Arnoldi iteration (cf. Algorithm 2.5.2), to build an orthonormal basis for the approximation subspace. We postpone to Section 3.5 a major comparison between DMBR and BGMRES-R, but briefly mention here that the main difference is that DMBR is able to deflate at the beginning of each iteration, whereas BGMRES-R can deflates only at the end of the iteration. For doing so, DMBR performs an extra step while deflating. This difference between the methods gives raise to a considerably different behaviour as we discuss in Section 3.5 and as we show in practice in Section 3.9.

Also in this section we introduce in our framework a *unitary deflation matrix* $\mathscr{F}_j$ whose dimension is yet to be defined. The purpose of this matrix is to represent the action of reorganizing or recombining the columns of a matrix according to a criteria to be established. For instance, letting K be any matrix of proper dimensions, K is said to be *undeflated* whereas $K\mathscr{F}_j$ is said to be *deflated* in the sense that the first (or last) columns of $K\mathscr{F}_j$ satisfy the chosen criteria (to be discussed in Section 3.4), allowing the method itself to delete these last columns of $K\mathscr{F}_j$ or to reuse them for a different purpose. Similarly, $\mathscr{F}_j^H$ can also be used to reorganize and recombine the rows of a matrix.

We introduce in Algorithm 3.3.1 a pseudocode for DMBR following the mentioned framework. Therein the deflation is performed on $\hat{\mathscr{V}}_{j+1}$ and reflected on the rows of $\hat{\mathscr{H}}_j$ (cf. line 9-10 of Algorithm 3.3.1) as in BGMRES-R, but in addition, the rows of $\hat{\Lambda}_j$ are deflated, being this the reason why DMBR is able to deflate at the beginning of each iteration. Also, as in BGMRES-R, because $\mathscr{F}_j$ is unitary its application do not *delete* the direction which are not chosen, but simply *postpone* them. We remark that in [103] no unitary deflation matrix as $\mathscr{F}_j$ is presented for BGMRES-R because the formulation of the methods are different, but as we discuss later in Section 3.5, under certain conditions, both methods are algebraically equivalent.

After applying the unitary deflation operator $\mathscr{F}_j$ in line 9 of Algorithm 3.3.1 during iteration j, we obtain

$$A\mathscr{Z}_{j-1} = \hat{\mathscr{V}}_j\mathscr{F}_j\mathscr{F}_j^H\hat{\mathscr{H}}_{j-1} = \begin{bmatrix}\mathscr{V}_j & P_{j-1}\end{bmatrix}\mathscr{H}_{j-1} \tag{3.3.3}$$

which is precisely the relation we need for the j-th iteration of Algorithm 3.2.1 to generate the decomposition (3.2.4). Also, thanks to the fact that $\mathscr{F}_j$ is unitary, we guarantee that $[\mathscr{V}_j \quad P_{j-1}]$ is still orthonormal.

We show now that regardless of the choice of $\mathscr{F}_j$ and k_j, DMBR minimizes the block residual over $\mathscr{Z}_j$.

Lemma 3.3.1. *In Algorithm 3.3.1, at the end of the j-th iteration for any $1 \leq j \leq m$ and any chosen unitary matrix $\mathscr{F}_j \in \mathbb{C}^{(s_j+d_j)\times(s_j+d_j)}$, we have*

$$R_0 = \hat{\mathscr{V}}_{j+1}\hat{\Lambda}_j.$$

Proof. For $j = 0$ it follows directly from line 4 of Algorithm 3.3.1 that the lemma is correct. Assuming it is true for $j = k-1$

$$\hat{\mathscr{V}}_k\hat{\Lambda}_{k-1} = R_0,$$

recalling that every $\mathscr{F}_k$ is unitary, from Algorithm 3.3.1 we obtain

$$\begin{aligned}
\hat{\mathscr{V}}_{k+1}\hat{\Lambda}_k &= \begin{bmatrix}\mathscr{V}_k & P_{k-1} & \hat{V}_{k+1}\end{bmatrix}\begin{bmatrix}\Lambda_k \\ 0_{(k_k-n_k)\times p}\end{bmatrix} && (3.3.4)\\
&= \begin{bmatrix}\mathscr{V}_k & P_{k-1}\end{bmatrix}\Lambda_k && (3.3.5)\\
&= \hat{\mathscr{V}}_k\mathscr{F}_k\mathscr{F}_k^H\Lambda_{k-1} && (3.3.6)\\
&= R_0 && (3.3.7)
\end{aligned}$$

proving by induction. □

Algorithm 3.3.1: *Restarted Flexible DMBR*

Choose an initial guess $X_0 \in \mathbb{C}^{n\times p}$, a restart parameter m and define a convergence criterion and its scaling matrix;
for *cycle* $= 1, \ldots, m$ **do**
  Compute the initial block residual $R_0 = B - AX_0$;
  $\hat{\mathscr{V}}_1 \hat{\Lambda}_0 = R_0$ (thin QR decomposition) and determine $p_0 = \operatorname{rank}(R_0)$, with $\hat{\mathscr{V}}_1 \in \mathbb{C}^{n\times p_0}$ and $\hat{\Lambda}_0 \in \mathbb{C}^{p_0\times p}$;
  Define $s_0 = 0$ and $\mathscr{H}_0 Y_0 = 0_{p_0\times p}$;
  for $j = 1, \ldots, m$ **do**
    Choose the scalar $1 \le k_j \le p_{j-1}$, set $s_j = s_{j-1} + k_j$ and $d_j = p_{j-1} - k_j$;
    Choose the *unitary deflation operator* $\mathscr{F}_j \in \mathbb{C}^{(s_j+d_j)\times(s_j+d_j)}$;
    Update $\begin{bmatrix}\mathscr{V}_j & P_{j-1}\end{bmatrix} = \hat{\mathscr{V}}_j \mathscr{F}_j$, with $\mathscr{V}_j \in \mathbb{C}^{n\times s_j}$ as the first s_j columns of $\hat{\mathscr{V}}_j \mathscr{F}_j$;
    Update $\Lambda_j = \mathscr{F}_j^H \hat{\Lambda}_{j-1}$ with $\Lambda_j \in \mathbb{C}^{s_j\times p}$ and also $\mathscr{H}_{j-1} = \mathscr{F}_j^H \hat{\mathscr{H}}_{j-1}$ with $\mathscr{H}_{j-1} \in \mathbb{C}^{s_j\times s_j}$ (if $j > 1$);
    Completion of $\hat{\mathscr{V}}_{j+1}$, $\mathscr{Z}_j$ *and* $\hat{\mathscr{H}}_j$*:* Apply Algorithm 3.2.1 to obtain

$$A\mathscr{Z}_j = \hat{\mathscr{V}}_{j+1}\hat{\mathscr{H}}_j \quad \text{with} \quad \hat{\mathscr{V}}_{j+1} = \left[V_1, \ldots, V_j, P_{j-1}, \hat{V}_{j+1}\right] \tag{3.3.1}$$

    as well as the constants p_j and n_j;
    Set $\hat{\Lambda}_j \in \mathbb{C}^{(s_j+p_j)\times p}$ as $\hat{\Lambda}_j = \begin{bmatrix}\Lambda_j \\ 0_{(k_j-n_j)\times p}\end{bmatrix}$;
    Set $Y_j \in \mathbb{C}^{s_j\times p}$ as the unique minimal Frobenius norm solution to the problem

$$\min_{Y\in\mathbb{C}^{s_j\times p}} \left\|\hat{\Lambda}_j - \hat{\mathscr{H}}_j Y\right\|_F \tag{3.3.2}$$

    if *full convergence detected (see Definition 2.8.1)* **then** break;
    ;
  end for
  $X_0 = X_0 + \mathscr{Z}_m Y_m$;
end for

Thanks to Lemma 3.3.1 we can write the following proposition.

Corollary 3.3.2 (Block Residual Minimization)**.** *Consider that j iterations of Algorithm 3.3.1 have been carried out in a given cycle for any chosen sequence of unitary matrices $\mathscr{F}_i$ and $1 \le k_i \le p_{i-1}$, with $1 \le i \le j$. Then $\mathscr{Z}_j Y_j$ solves the problem*

$$\min_{\operatorname{range}(Z)\subset\operatorname{range}\left(\mathscr{Z}_j\right)} \|R_0 - AZ\|_F . \tag{3.3.8}$$

Proof. We rewrite the problem (3.3.8) as

$$\min_{Y\in\mathbb{C}^{s_j\times p}} \left\|R_0 - A\mathscr{Z}_j Y\right\|_F .$$

Using Lemma 3.3.1 and (3.3.1) we obtain

$$\begin{aligned}\left\|R_0 - A\mathscr{Z}_j Y\right\|_F &= \left\|\hat{\mathscr{V}}_{j+1}\hat{\Lambda}_j - \hat{\mathscr{V}}_{j+1}\hat{\mathscr{H}}_j Y\right\|_F \\ &= \left\|\hat{\Lambda}_j - \hat{\mathscr{H}}_j Y\right\|_F .\end{aligned}$$

Since Y_j is defined as the unique minimal Frobenius norm solution for this problem (see (3.3.2) in Algorithm 3.3.1) and since such solution always exists, the corollary is proven. □

Remarking that $R_j = B - AX_j = B - A(X_0 + \mathscr{Z}_j Y_j) = R_0 - A\mathscr{Z}_j Y_j$, Corollary 3.3.2 shows that DMBR method is in fact minimizing the block residual R_j over the entire range $\left(\mathscr{Z}_j\right) + \text{range}\left(X_0\right)$ subspace, regardless of the choice of $\mathscr{F}_j$ and k_j. The following corollary follows directly from Corollary 3.3.2.

Corollary 3.3.3. *Let R_j be the block residual generated at the j-th iteration of Algorithm 3.3.1 in a given cycle. Then, $\left\|R_j\right\|_F \leq \left\|R_{j-1}\right\|_F$.*

We can guarantee not only that the Frobenius norm is monotonically decreasing, but also the singular values of the block residual as we show next.

Proposition 3.3.4. *Let R_j be the residual at the end of the j-th iteration of Algorithm 3.3.1 in a given cycle, and suppose that the sequence of preconditioners $M_i \in \mathbb{C}^{n\times n}$ is rank-preserving. It holds that*

$$\sigma_i(R_j) \leq \sigma_i(R_{j-1}), \quad 1 \leq i \leq p.$$

Proof. Due to the minimization property (see Corollary 3.3.2) we can write each R_i as

$$\begin{aligned}
R_i &= \hat{\mathscr{V}}_{i+1}(I_{s_i} - \hat{\mathscr{H}}_i\hat{\mathscr{H}}_i^\dagger)\hat{\Lambda}_i \\
&= \hat{\mathscr{V}}_{i+1}\hat{\Lambda}_i - \hat{\mathscr{V}}_{i+1}\hat{\mathscr{H}}_i\hat{\mathscr{H}}_i^\dagger\hat{\mathscr{V}}_{i+1}^H\hat{\mathscr{V}}_{i+1}\hat{\Lambda}_i \\
&= R_0 - \hat{\mathscr{V}}_{i+1}\hat{\mathscr{H}}_i\hat{\mathscr{H}}_i^\dagger\hat{\mathscr{V}}_{i+1}^H R_0 \\
&= (I_n - \hat{\mathscr{V}}_{i+1}\hat{\mathscr{H}}_i(\hat{\mathscr{V}}_{i+1}\hat{\mathscr{H}}_i)^\dagger)R_0 \\
&= (I_n - A\mathscr{Z}_i(A\mathscr{Z}_i)^\dagger)R_0
\end{aligned}$$

Consider that we dispose of an orthonormal basis $\mathscr{W}_j$ to range $\left(A\mathscr{Z}_j\right)$ where $\mathscr{W}_j$ can be defined recursively as $\mathscr{W}_j = [W_{j-1} \quad W_j], \mathscr{W}_1 = W_1$. Then using the idempotence property of projectors we obtain

$$\begin{aligned}
R_i &= (I_n - \mathscr{W}_i\mathscr{W}_i^H)(I_n - \mathscr{W}_i\mathscr{W}_i^H)R_0 \\
&= (I_n - \mathscr{W}_i\mathscr{W}_i^H)(I_n - \mathscr{W}_{i-1}\mathscr{W}_{i-1}^H - W_iW_i^H)R_0 \\
&= (I_n - \mathscr{W}_i\mathscr{W}_i^H)(R_{i-1} - W_iW_i^H R_0) \\
&= (I_n - \mathscr{W}_i\mathscr{W}_i^H)R_{i-1}
\end{aligned}$$

From [71, Theorem 3.3.16], the proof is finished. □

In fact, since Proposition 3.3.4 relies only on the orthogonality with respect to the updated basis, it is in fact applicable to any block method presenting minimal residual properties, which comprises BGMRES, DMBR, BGMRES-R among others. It is not applicable to BFGMRESD or BFGMREST however, because these methods rely on a low rank approximation of the initial residual R_0 which is computed at the beginning of each cycle, characterizing thus a different scenario.

Let $R_j^{(k)}$ denote the true block residual obtained at the end of j-th iteration of the k-th cycle of DMBR, where $1 \leq j \leq m$ (that is, the restart size is m). Because $R_0^{(k+1)} = R_m^{(k)}$, we can ensure also that $\sigma_i(R_1^{(k+1)}) \leq \sigma_i(R_m^{(k)})$ meaning that DMBR ensures the monotonically nonincreasing behaviour of the singular values of the true block residual, not only along the iterations, but also along the cycles regardless of the choice of $\mathscr{F}_j$ and k_j, a property also present in BGMRES-R method. This property can be easily extended to the Frobenius norm or the Euclidean norm of the true block residual, and could be extended to any unitarily invariant norm, see [23, Section 3.3] for a similar proof related to block flexible Krylov subspace methods with deflation performed at restart only.

The nonincreasing behaviour of the singular values will appear as particularly important when determining the unitary deflation operator in Section 3.4. Also, although we can not guarantee that for an arbitrary choice of D_j the singular values of R_jD_j will be also monotonically decreasing, for a fixed $D_j = D$ we trivially obtain that Proposition 3.3.4 also holds for R_jD.

3.4 Choosing the Unitary Deflation Operator

In this section we discuss in detail a criterion for choosing the unitary deflation operator $\mathscr{F}_j$, consisting of a reformulation of the so called *R-criterion* proposed in [103, §5] for BGMRES-R, adapted to fit the discussion in Definition 2.2.8 in Section 2.2 and our previous discussion on the decomposition Equation 3.2.2. We stress that in [103, §5] there is no mention of $\mathscr{F}_j$ introduced in Section 3.3, and that we use this framework to be able to deduce different possibilities for deflation in block iterative solvers based on BGMRES.

Generalizing Proposition 2.9.4 to DMBR and in view of (3.2.2), we obtain that

$$\text{range}\left(K_j\right) = \text{range}\left((I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}\right) \quad \forall j > 1$$

with $\text{range}\left(K_1\right) = \text{range}\left(R_0\right)$. The BFGMRES method uses the whole $\text{range}\left(K_j\right)$ to expand the correction subspace (that is, it aims at setting $\text{range}\left(Z_j\right) = AM_j\text{range}\left(K_j\right)$, reminding that $\text{range}\left(K_j\right) = \text{range}\left(V_j\right)$ in this setting when no partial stagnation occurs, cf. Proposition 2.9.4) for iteration j, but as mentioned in Section 3.2 we want to be able to judiciously split $\text{range}\left(K_j\right)$.

Recalling that D_{j-1} is the chosen scaling matrix for iteration $(j-1)$ (cf. Section 2.8), consider that we are at the beginning j-th iteration of Algorithm 3.3.1 and that $d_j > 0$ was already chosen, and that we dispose of the following thin singular value decomposition [60, p.72 §2.5.4]

$$\left(\hat{\Lambda}_{j-1} - \hat{\mathscr{H}}_{j-1}Y_{j-1}\right)D_{j-1} = \tag{3.4.1}$$

$$\begin{bmatrix}U_+ & U_-\end{bmatrix}\begin{bmatrix}\Sigma_+ & 0 & 0_{k_j\times(p-p_{j-1})}\\ 0 & \Sigma_- & 0_{d_j\times(p-p_{j-1})}\end{bmatrix}\begin{bmatrix}W_+^H\\ W_-^H\\ W_0^H\end{bmatrix} = \tag{3.4.2}$$

$$U_+\Sigma_+W_+^H + U_-\Sigma_-W_-^H \tag{3.4.3}$$

with $U_+ \in \mathbb{C}^{(s_j+d_j)\times k_j}$, $U_- \in \mathbb{C}^{(s_j+d_j)\times d_j}$, $\Sigma_+ \in \mathbb{C}^{k_j\times k_j}$,$\Sigma_- \in \mathbb{C}^{d_j\times d_j}$, $W_+ \in \mathbb{C}^{p\times k_j}$, $W_0 \in \mathbb{C}^{p\times(p-p_{j-1})}$ and $W_- \in \mathbb{C}^{p\times d_j}$ where $p_{j-1} = k_j + d_j$, and the singular values are given in nonincreasing order. Knowing that this gives us the thin singular decomposition of the scaled block residual as

$$R_{j-1}D_{j-1} = \hat{\mathscr{V}_j}\left(U_+\Sigma_+W_+^H + U_-\Sigma_-W_-^H\right),$$

and denoting $R_{j-1}^+ = R_{j-1}D_{j-1}W_+$ and $R_{j-1}^- = R_{j-1}D_{j-1}W_-$, we then conclude that

$$\begin{aligned}\left\|R_{j-1}^+\right\|_F &= \left\|R_{j-1}D_{j-1}W_+\right\|_F = \|\Sigma_+\|_F\\ \left\|R_{j-1}^-\right\|_F &= \left\|R_{j-1}D_{j-1}W_-\right\|_F = \|\Sigma_-\|_F\\ \left\|R_{j-1}D_{j-1}W_0\right\|_F &= 0\end{aligned}$$

If $||\Sigma_-||_F < \sqrt{(p-k_j)/p}\,\varepsilon$ where ε is the convergence tolerance, then we have found $(p-k_j)$ partial convergence according to Definition 2.8.1. Namely, $X_{j-1}[W_- \quad W_0]$ is an approximate solution of $AV = B[W_- \quad W_0]$.

In such a case, we no longer need to expand the correction subspace $\mathcal{Z}_{j-1} = \text{range}\left(\mathscr{Z}_{j-1}\right)$ targeting further minimization of $||B[W_- \quad W_0] - AX||_F$ (which is associated to R_{j-1}^- and the nullspace of the residual) but only to further minimize $||BW_+ - AX||_F$ (which is related to R_{j-1}^+).

Inspired by Proposition 2.9.4, the criterion proposed in [103] for choosing $\mathscr{F}_j$ consists thus in splitting

range $\left(K_j\right)$ as

$$\begin{aligned}\operatorname{range}\left(V_j\right) &= \operatorname{range}\left((I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}^+\right),\\ \operatorname{range}\left(P_{j-1}\right) &= \operatorname{range}\left((I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}^-\right),\\ \operatorname{range}\left(K_j\right) &= \operatorname{range}\left((I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}\right) = \operatorname{range}\left(V_j\right) \oplus \operatorname{range}\left(P_{j-1}\right).\end{aligned}$$

If we further split the matrices

$$\begin{bmatrix} U_+ & U_- \end{bmatrix} = \begin{bmatrix} U_{k_1}^+ & U_{k_1}^- \\ U_{k_2}^+ & U_{k_2}^- \\ \vdots & \vdots \\ U_{k_{j-1}}^+ & U_{k_{j-1}}^- \\ U_{p_{j-1}}^+ & U_{p_{j-1}}^- \end{bmatrix},$$

with $U_{k_i}^+ \in \mathbb{C}^{k_i \times k_j}$,$U_{k_i}^- \in \mathbb{C}^{k_i \times d_j}$ for $1 \leq i \leq (j-1)$, $U_{p_{j-1}}^+ \in \mathbb{C}^{p_{j-1} \times k_j}$ and $U_{p_{j-1}}^- \in \mathbb{C}^{p_{j-1} \times d_j}$, then

$$\begin{aligned}R_{j-1}^+ = \hat{\mathscr{V}_j} U_+ \Sigma_+ &= \left(\sum_{i=1}^{(j-1)} V_i U_{k_i}^+ + \begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} U_{p_{j-1}}^+\right)\Sigma_+\\ R_{j-1}^- = \hat{\mathscr{V}_j} U_- \Sigma_- &= \left(\sum_{i=1}^{(j-1)} V_i U_{k_i}^- + \begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} U_{p_{j-1}}^-\right)\Sigma_-.\end{aligned}$$

This directly drives us to the conclusion that

$$(I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}^+ = \begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} U_{p_{j-1}}^+ \Sigma_+ \tag{3.4.4}$$

$$(I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}^- = \begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} U_{p_{j-1}}^- \Sigma_- \tag{3.4.5}$$

and thus

$$\begin{aligned}\operatorname{range}\left(V_j\right) &= \operatorname{range}\left(\begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} U_{p_{j-1}}^+\right)\\ \operatorname{range}\left(P_{j-1}\right) &= \operatorname{range}\left(\begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} U_{p_{j-1}}^-\right).\end{aligned}$$

Let

$$F_j T = \begin{bmatrix} U_{p_{j-1}}^+ & U_{p_{j-1}}^- \end{bmatrix}$$

be a full QR decomposition of $[U_{p_{j-1}}^+ \quad U_{p_{j-1}}^-] \in \mathbb{C}^{p_{j-1} \times p_{j-1}}$, with $F_j \in \mathbb{C}^{p_{j-1} \times p_{j-1}}$ and $T \in \mathbb{C}^{p_{j-1} \times p_{j-1}}$. One possible orthonormal basis of range $\left(V_j\right)$ (or range $\left(P_{j-1}\right)$) can be obtained by simply setting V_j (or P_{j-1}) as the first k_j (or last d_j) columns of $[P_{j-2} \quad \hat{V}_j]F_j$, i.e.

$$\begin{bmatrix} V_j & P_{j-1} \end{bmatrix} = \begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} F_j. \tag{3.4.6}$$

With this particular choice of V_j and P_{j-2}, a natural choice for $\mathscr{F}_j$ such that line 9 of Algorithm 3.3.1 holds is

$$\mathscr{F}_j = \begin{bmatrix} I_{s_{j-1}} & 0 \\ 0 & F_j \end{bmatrix}$$

yielding

$$\underbrace{\begin{bmatrix}\mathscr{V}_{j-1} & P_{j-2} & \hat{V}_j\end{bmatrix}}_{\hat{\mathscr{V}}_j}\underbrace{\begin{bmatrix}I_{s_{j-1}} & 0\\ 0 & F_j\end{bmatrix}}_{\mathscr{F}_j} = \begin{bmatrix}\mathscr{V}_{j-1} & \begin{bmatrix}P_{j-2} & \hat{V}_j\end{bmatrix}F_j\end{bmatrix},$$

$$= \begin{bmatrix}\mathscr{V}_{j-1} & V_j & P_{j-1}\end{bmatrix}$$

Naturally, this discussion is valid only for the case in which $d_j > 0$, otherwise no deflation is needed and setting $\mathscr{F}_j = I_{s_j}$ suffices. We finally formalize this criterion for determining $\mathscr{F}_j$ and choosing k_j in Algorithm 3.4.1.

Algorithm 3.4.1: *Choosing $\mathscr{F}_j$ - Largest Singular Values of $R_{j-1}D_{j-1}$*

1 Receive $R_{j-1}D_{j-1} = \begin{bmatrix}\mathscr{V}_j & P_{j-1}\end{bmatrix}(\hat{\Lambda}_{j-1} - \hat{\mathscr{H}}_{j-1}Y_{j-1})D_{j-1}$ and the parameters $1 \leq k_{max} \leq p_{j-1}$ and ε^d;
2 $U\Sigma W^H = (\hat{\Lambda}_{j-1} - \hat{\mathscr{H}}_{j-1}Y_{j-1})D_{j-1}$ (Thin SVD) with $U \in \mathbb{C}^{(s_j+d_j)\times p_{j-1}}$, $\Sigma \in \mathbb{C}^{p_{j-1}\times p}$ and $W \in \mathbb{C}^{p\times p}$;
3 Choose $\tilde{k}_j$ such that $\sigma_l(R_{j-1}D_{j-1}) \geq \sqrt{1/p}\,\varepsilon^d$ for all $1 \leq l \leq \tilde{k}_j$;
4 $U_{p_{j-1}} = U(s_{j-1}+1 : s_{j-1}+p_{j-1}, 1 : p_{j-1})$, with $U_{p_{j-1}} \in \mathbb{C}^{p_{j-1}\times p_{j-1}}$ (that is, the p_{j-1} last rows of U);
5 Define $F_jT = U_{p_{j-1}}$, (full QR Decomposition) with $F_j \in \mathbb{C}^{p_{j-1}\times p_{j-1}}$;
6 Set $k_j = \min(\tilde{k}_j, k_{max})$;
7 Define $\mathscr{F}_j = \begin{bmatrix}I_{s_{j-1}} & 0\\ 0 & F_j\end{bmatrix} \in \mathbb{C}^{(s_j+d_j)\times(s_j+d_j)}$;

Remark 3.4.1. Even if no deflation has occurred until iteration $(j-1)$ (that is, $k_i = p$ for $1 \leq i \leq (j-1)$), Algorithm 3.4.1 builds a matrix $\mathscr{F}_j$ which is different from the identity in a general scenario. In such a case, we can then force $F_j = I_{p_j}$ for every iteration until a deflation happens, meaning that we are neglecting the application of $\mathscr{F}_j$ in line 9 and line 10 since the respective subspaces remain the same. This change aims at saving computational times. ■

Remark 3.4.2. One disadvantage of Algorithm 3.4.1 is that when a partial stagnation occurs (cf. Definition 2.9.1), then the block residual R_{j-1} lacks of information concerning the last directions, that is

$$\text{range}\left(K_j\right) \supset \text{range}\left((I_n - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}\right).$$

When a partial stagnation occurs, then $U_{p_{j-1}}$ is rank deficient in line 5 of Algorithm 3.4.1. Because Algorithm 3.4.1 builds the F_j matrix out of a full QR decomposition, we still can guarantee that

$$\text{range}\left(K_j\right) = \text{range}\left(\begin{bmatrix}P_{j-2} & \hat{V}_j\end{bmatrix}\right) = \text{range}\left(\begin{bmatrix}V_j & P_{j-1}\end{bmatrix}\right).$$

(see also Proposition 3.4.3 ahead) even if a stagnation occurs, but we can not guarantee that the whole subspace of range $\left(K_j\right)$ associated with R_{j-1}^+ (respectively R_{j-1}^-) lies in range(V_j) (respectively range(P_{j-1})). We remark that a stagnation as in Definition 2.9.1 is a rare phenomenon[1] and that it has not been observed in our numerical experiments. Algorithm 3.4.1 never breakdowns, even in case of a full stagnation. ■

The following discussion (Proposition 3.4.3 and Proposition 3.4.4) aims at showing that the subspace $\dim(\mathcal{Z}_j)$ is monotonically increasing (cf. Corollary 3.4.5). The goal of such a proof is to satisfy the condition previously established in Definition 2.2.1 for nested subspaces.

[1] a more common case would be a *near stagnation*, where the rank of $(I - \mathscr{V}_{j-1}\mathscr{V}_{j-1}^H)R_{j-1}$ is computed using a threshold ε^t. We do not address this situation in this thesis

Proposition 3.4.3. *After j iterations of Algorithm 3.3.1 with Algorithm 3.4.1 for choosing each $\mathscr{F}_i$, $1 \leq i \leq j$, it holds that*

$$\operatorname{rank}\left(\begin{bmatrix} V_j & P_{j-1} \end{bmatrix}\right) = \operatorname{rank}\left(\begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix}\right) = p_{j-1},$$

where $\begin{bmatrix} P_{j-2} & \hat{V}_j \end{bmatrix} = \hat{V}_j$ *if* $d_{j-1} = 0$ *or if* $j = 1$.

Proof. Each $\hat{V}_i$ has full rank and orthogonal to P_{i-2} (if defined) by construction. To show that each P_i has full rank whenever they are defined, we use a simple induction: P_0 has full rank by construction. If P_{i-1} has full rank, then $\begin{bmatrix} P_{i-1} & \hat{V}_{i+1} \end{bmatrix}$ has full rank, and since F_i is unitary, $\begin{bmatrix} V_{i+1} & P_i \end{bmatrix}$ will be full rank, completing the proof. □

Proposition 3.4.4. *After j iterations of Algorithm 3.3.1 with Algorithm 3.4.1 for choosing each $\mathscr{F}_i$, $1 \leq i \leq j$, it holds that*

$$\operatorname{rank}(\hat{\mathscr{V}}_{j+1}) = \operatorname{rank}(\begin{bmatrix} \mathscr{V}_{j+1} & P_j \end{bmatrix}) = s_j + p_j.$$

If additionally the sequence of variable preconditioners M_i, $1 \leq i \leq j$ is rank-preserving (cf. Definition 2.4.1), we can ensure that

$$\operatorname{rank}\left(\hat{\mathscr{H}}_j\right) = \operatorname{rank}\left(\mathscr{H}_j\right) = s_j.$$

Proof. Proposition 3.4.3 already states that $\operatorname{rank}(V_i) = p_i$ for every $1 \leq i \leq j$. Noting that each V_i is orthogonal to $\mathscr{V}_j$ by construction with $1 \leq i \leq j$, we find out that $\operatorname{rank}(\mathscr{V}_j) = \sum_{i=1}^{j} k_i = s_j$. This together with Proposition 3.4.3 proves that $\operatorname{rank}(\hat{\mathscr{V}}_{j+1}) = s_j + p_j$.

Supposing that the sequence of variable preconditioning operators M_i, $1 \leq i \leq j$ is rank-preserving, we find that $\operatorname{rank}(\hat{\mathscr{V}}_{j+1}\hat{\mathscr{H}}_j) = \operatorname{rank}(A\mathscr{Z}_j) = s_j$. Since $\hat{\mathscr{V}}_{j+1}$ has full rank, $\operatorname{rank}(\hat{\mathscr{H}}_j) = s_j$ and the proposition is proven. □

Corollary 3.4.5. *Assume that j iterations of DMBR (Algorithm 3.3.1) with Algorithm 3.4.1 have been performed in a given cycle, and that a full convergence has not been detected until iteration j. If the sequence of variable preconditioners M_i, $1 \leq i \leq j$ is rank-preserving, then $\{\mathcal{Z}_i\}_{i=1}^{j}$ is a sequence of nested subspaces, where each $\mathcal{Z}_i = \operatorname{range}(\mathscr{Z}_i)$, $1 \leq i \leq j$.*

Proof. This corollary is an implication of Proposition 3.4.3, Proposition 3.4.4 and the rank-preserving assumption on the variable preconditioner. Because $\operatorname{rank}(\mathscr{Z}_j) = \dim(\mathcal{Z}_j) = s_j$ and also because $\mathcal{Z}_{i-1} \subset \mathcal{Z}_i$ $1 < i \leq j$ by construction, we just have to show that $s_{i-1} < s_i$ $1 < i \leq j$. But $s_i = s_{i-1} + k_i$, and $k_i \geq 1$ (cf. line 6). □

Thanks to Corollary 3.4.5 we conclude that DMBR with Algorithm 3.4.1 spans a nested correction subspace (as in Definition 2.2.1) and thus it can indeed be classified as a method belonging to the MBR family of methods (cf. Definition 2.2.2). Before proceeding to the next section we establish one additional relation concerning the correction subspace constructed by DMBR with Algorithm 3.4.1.

Proposition 3.4.6 (Block Krylov Subspace Membership)**.** *Consider that we have performed j iterations of Algorithm 3.3.1 with Algorithm 3.4.1 for choosing each $\mathscr{F}_i$ in a given cycle, and that each $M_i = M$, $1 \leq i \leq j$, where M is an nonsingular square matrix. It holds that*

$$\operatorname{range}\left(\hat{\mathscr{V}}_{j+1}\right) \subset \mathcal{K}_{j+1}^{\square}(AM^{-1}, R_0).$$

Proof. We have range $\left(\hat{\mathscr{V}}_1\right) \subset \mathcal{K}_1^{\square}(AM^{-1}, R_0) = \text{range}\,(R_0)$. Supposing that range $\left(\hat{\mathscr{V}}_j\right) \subset \mathcal{K}_j^{\square}(AM^{-1}, R_0)$ we deduce that

$$\begin{aligned}\text{range}\left(\hat{V}_{j+1}\right) &\subset \text{range}\left(AM_j^{-1}V_j\right) \subset AM^{-1}\text{range}\left(\begin{bmatrix}V_j & P_{j-1}\end{bmatrix}\right)\\ &= AM^{-1}\text{range}\left(\begin{bmatrix}P_{j-2} & \hat{V}_j\end{bmatrix} F_{j-1}\right) \subset AM^{-1}\mathcal{K}_j^{\square}(AM^{-1}, R_0)\\ &\subset \mathcal{K}_{j+1}^{\square}(AM^{-1}, R_0),\end{aligned}$$

proving the proposition. □

Proposition 3.4.6 states that in the case of a fixed preconditioner case, DMBR spans a subspace contained in a block Krylov subspace. However, in the flexible case, we do not guarantee that such a property holds, or any statement analogous to Proposition 2.5.3. For instance, knowing that

$$\begin{aligned}\text{range}\,(R_0) &= \text{range}\,(V_1) \oplus \text{range}\,(P_0)\\ \text{range}\left(\hat{V}_2\right) &= \text{range}\left((I - \begin{bmatrix}V_1 & P_0\end{bmatrix}\begin{bmatrix}V_1 & P_0\end{bmatrix}^H)AM_1V_1\right)\end{aligned}$$

it is clear that range $\left(\hat{\mathscr{V}}_2\right) \subseteq \text{span}\left\{\begin{bmatrix}R_0 & AM_1R_0\end{bmatrix}\right\}$. But if Algorithm 3.4.1 sets

$$V_2 = P_0 \qquad \text{and} \qquad P_1 = \hat{V}_2,$$

then

$$\text{range}\left(\hat{\mathscr{V}}_3\right) \subseteq \text{span}\left\{\begin{bmatrix}R_0 & AM_1R_0 & AM_2R_0\end{bmatrix}\right\}.$$

We could explore the existence of a matrix $\hat{M}_2 \in \mathbb{C}^{n\times n}$ such that

$$\text{range}\left(\hat{\mathscr{V}}_3\right) \subseteq \text{span}\left\{\begin{bmatrix}R_0 & A\hat{M}_2R_0\end{bmatrix}\right\}.$$

instead, allowing the characterization of $\mathcal{Z}_j$ as a subspace of a Krylov subspace, but this issue is beyond the scope of this thesis, and it is subject for future studies.

This characterization of the correction subspace spanned by DMBR explains why its behaviour differs from the behaviour of other similar methods, as BFGMRESD and BGMRES-R as well as BGMRES. In fact, each one of these methods spans a different correction subspace, and after the first cycle there is no guarantee that there is an intersection between the correction subspace spanned by each method, explaining thus why their convergence behaviour tend to be significantly different when compared with equivalent parameters. We explore with more details the difference between these methods in Section 3.5.

Lastly, the following corollary guarantees that if a fixed scaling matrix D is used, then the value of k_j is monotonically nonincreasing, not only along the iterations but also along the cycles.

Corollary 3.4.7. *For every iteration $j > 1$ of DMBR (Algorithm 3.3.1) with Algorithm 3.4.1, if $D_i = D$ for every $1 \leq i \leq j$ for a given square matrix $D \in \mathbb{C}^{p\times p}$, it holds that $k_j \geq k_{j-1}$. This property also holds along the cycles of DMBR.*

Proof. This corollary is an implication of the choice of each k_j in Algorithm 3.4.1 and Proposition 3.3.4. □

Despite of the similarities between DMBR and BGMRES-R, the later cannot guarantee that Corollary 3.4.7 holds along the cycles, but only within one cycle. That happens because BGMRES-R deflates only in the end of each iteration, and as so it cannot choose $k_1 < p_0$. We discuss this and other differences between these methods in Subsection 3.5.1, and we analyse the numerical experiments in Section 3.9.

3.5 Connections With Existing Methods

In this section we discuss the similarities and disparities between DMBR and other methods published in the literature. We show that most of these methods can be deduced from Algorithm 3.3.1 by simply choosing $\mathscr{F}_j$ properly. In Subsection 3.5.1 we discuss the connections with BGMRES-R [103].

3.5.1 Connections with BGMRES-R

We now consider a comparison between DMBR and BGMRES-R [103]. For the sake of generality we consider a variant with variable preconditioner even though in [103] the algorithm is considered in the case of fixed preconditioner only. Recalling our previous discussion in the beginning of Section 3.2, BGMRES-R generates the *inexact (block) Arnoldi relation*

$$A\mathscr{Z}_j = \mathscr{V}_{j+1}\underline{\mathscr{L}}_j + \tilde{\mathscr{Q}}_j,$$

where $\underline{\mathscr{L}}_j = \mathscr{V}_{j+1}^H A\mathscr{Z}_j$. Instead of storing $\tilde{\mathscr{Q}}_j \in \mathbb{C}^{n\times s_j}$ however, BGMRES-R uses a clever algebraic manipulation storing the QR decomposition

$$\tilde{\mathscr{Q}}_j = P_j G_j,$$

where $P_j \in \mathbb{C}^{n\times d_{j+1}}$ and $G_j \in \mathbb{C}^{d_{j+1}\times s_j}$. Using our notation, without considering major algebraic details, we state that

$$\mathscr{H}_j = \begin{bmatrix} \underline{\mathscr{L}}_j \\ G_j \end{bmatrix}.$$

Moreover, BGMRES-R computes a $\mathscr{F}_j$ matrix according to Algorithm 3.4.1, and deflates $\hat{\mathscr{V}}_{j+1}$ and $\hat{\mathscr{H}}_j$ at the end of every iteration. We show in Algorithm 3.5.3 a simplified pseudocode for both methods, highlighting the differences.

Notice that in BGMRES-R, because there is no deflation at the beginning of the first iteration, it holds that

$$\mathscr{F}_2^H\hat{\Lambda}_1 = \begin{bmatrix} \hat{\Lambda}_0 \\ F_2^H 0_{(p_0-n_1)\times p} \end{bmatrix} = \begin{bmatrix} \hat{\Lambda}_0 \\ 0_{p_1\times p} \end{bmatrix},$$

and thus, for every j we have that

$$\mathscr{F}_{j+1}^H\hat{\Lambda}_j = \hat{\Lambda}_j,$$

that is, it is unnecessary to deflate $\hat{\Lambda}_j$ in BGMRES-R. Both methods are thus algebraically equivalent for every cycle in which DMBR chooses $k_1 = p_0$, and not equivalent otherwise. The lack of deflation at the beginning of the first iteration of BGMRES-R brings some crucial drawbacks for BGMRES-R, as we highlight next:

- *When the deflations happen early in the convergence history.* According to our numerical experiments, this behaviour seems to be *common among the tested problems.* The value of k_j tends to quickly decrease in the first cycles, having k_m small (often equal to one) at the end of the cycle (cf. Figure 3.1). Considering the extreme case in which $k_m = 1$, BGMRES-R then performs in the following cycle $p_0 + m - 1$ matrix vector and preconditioner applications, whereas DMBR performs only m. Also, notice that in such a case $[\mathscr{V}_j \quad P_{j+1}]$ has $2p_0+j$ columns in BGMRES-R and $j+p_0-1$ columns in DMBR, meaning that BGMRES-R has a more expensive orthogonalization than DMBR.
- *When the restart size m is small.* Since in BGMRES-R $k_1 = p_0$, a small restart makes the aforementioned behaviour more evident.

Algorithm 3.5.3: *Comparison between DMBR and BGMRES-R*

Choose X_0, m and a convergence criterion with its scaling matrix;
for *cycle* $= 1, \dots, m$ **do**
  Compute $R_0 = B - AX_0$;
  $\hat{\mathscr{V}}_1 \hat{\Lambda}_0 = R_0$ (thin QR decomposition);

DMBR	***BGMRES-R***
	Define $k_1 = p_0$ and $\mathscr{V}_1 = \hat{\mathscr{V}}_1$;
for $j = 1, \dots, m$ **do**	**for** $j = 1, \dots, m$ **do**
Choose k_j and $\mathscr{F}_j$;	
Deflate: $\begin{bmatrix}\mathscr{V}_j & P_{j-1}\end{bmatrix} = \hat{\mathscr{V}}_j \mathscr{F}_j$;	
Deflate: $\Lambda_j = \mathscr{F}_j^H \hat{\Lambda}_{j-1}$;	
Deflate: $\mathscr{H}_{j-1} = \mathscr{F}_j^H \hat{\mathscr{H}}_{j-1}$;	
Apply Algorithm 3.2.1 obtaining $A\mathscr{Z}_j = \hat{\mathscr{V}}_{j+1}\hat{\mathscr{H}}_j$;	Apply Algorithm 3.2.1 obtaining $A\mathscr{Z}_j = \hat{\mathscr{V}}_{j+1}\hat{\mathscr{H}}_j$;
Set $\hat{\Lambda}_j = \begin{bmatrix}\Lambda_j \\ 0_{(k_j - n_j)\times p}\end{bmatrix}$;	Set $\hat{\Lambda}_j = \begin{bmatrix}\hat{\Lambda}_{j-1} \\ 0_{(k_j - n_j)\times p}\end{bmatrix}$;
Solve: $\min \left\| \hat{\Lambda}_j - \hat{\mathscr{H}}_j Y \right\|_F$;	Solve: $\min \left\| \hat{\Lambda}_j - \hat{\mathscr{H}}_j Y \right\|_F$;
if *full convergence detected* **then** break;	**if** *full convergence detected* **then** break;
;	;
	Choose k_{j+1} and $\mathscr{F}_{j+1}$;
	Deflate: $\begin{bmatrix}\mathscr{V}_{j+1} & P_j\end{bmatrix} = \hat{\mathscr{V}}_{j+1}\mathscr{F}_{j+1}$;
	Deflate: $\mathscr{H}_j = \mathscr{F}_{j+1}^H \hat{\mathscr{H}}_j$;
end for	**end for**

  Update $R_0 = B - AX_0$;
  $X_0 = X_0 + \mathscr{Z}_m Y_m$;
end for

- *When the number of right-hand sides p_0 is large.* Same as above.

One of the main novelties of DMBR over BGMRES-R is hence the deflation of $\hat{\Lambda}_j$, which allows the deflation steps to take place in the beginning of the iteration while still minimizing the norm of the true residual $R_0 - AX_j$. Other novelties we propose are the truncation (that is, setting $k_{max} < p_0$ in Algorithm 3.4.1) and the flexible preconditioner, as well as decoupling the deflation strategy from the method itself (that is, allowing any unitary $\mathscr{F}_j$ to be chosen). We refer to Section 3.9 for more details on the numerical experiments and the practical difference between the behaviour of the two methods.

3.5.2 Connections with BFGMRESD

We now consider now a comparison between DMBR and BFGMRESD (as well as BFGMREST) [23, 78, 96]. Recalling what was mentioned in the beginning of this chapter, BFGMRESD performs a block size reduction of the (scaled) initial residual R_0 relying on its (near) rank deficiency. Rewriting (3.1.1) with our notation, BFGMRESD computes

$$R_0 = \hat{\mathscr{V}}_1 \hat{\Lambda}_0 \qquad \text{(QR decomposition)} \tag{3.5.1}$$

$$\hat{\Lambda}_0 D_0 = U_+ \Sigma_+ W_+^H + U_- \Sigma_- W_-^H \qquad \text{(singular values decomposition)} \tag{3.5.2}$$

where (3.5.2) has exactly the same dimensions as (3.4.1)-(3.4.3) (assuming $\mathscr{H}_0 Y_0 = 0_{k_1 \times p}$) and where D_0 is the chosen nonsingular scaling matrix. Denoting the structures of BFGMRESD with a $^\#$ superscript, it then proceeds setting

$$\Lambda_1^\# = \Sigma_+ W_+^H$$
$$\mathscr{V}_1^\# = \hat{\mathscr{V}}_1 U_+.$$

Notice however that during the first iteration of DMBR using Algorithm 3.4.1 we have that $\mathscr{F}_1 = F_1 = \begin{bmatrix} U_+ & U_- \end{bmatrix}$ and thus

$$\Lambda_1 = \begin{bmatrix} U_+ & U_- \end{bmatrix}^H \hat{\Lambda}_0 = \begin{bmatrix} \Sigma_+ W_+^H \\ \Sigma_- W_-^H \end{bmatrix} D_0^{-1}$$
$$\begin{bmatrix} \mathscr{V}_1 & P_0 \end{bmatrix} = \hat{\mathscr{V}}_1 \begin{bmatrix} U_+ & U_- \end{bmatrix},$$

that is, $\mathscr{V}_1 = \mathscr{V}_1^{\#}$ and the first k_1 rows of Λ_1 are equal to $\Lambda_1^{\#} D_0^{-1}$. Therefore, apart from the scaling, we can rewrite BFGMRESD using DMBR framework: P_0 is discarded as well as the last d_1 rows of Λ_1, and p_0 is defined as k_1 and the deflation happens during the first iteration (or right before the first iteration). We show in Algorithm 3.5.6 a comparison between both methods.

Algorithm 3.5.6: *Comparison between DMBR and BFGMRESD*

Choose X_0, m and a convergence criterion with its scaling matrix;
for *cycle* $= 1, \ldots, m$ **do**
  Compute $R_0 = B - AX_0$;
  $\hat{\mathscr{V}}_1 \hat{\Lambda}_0 = R_0$ (thin QR decomposition);

DMBR	***BFGMRESD***
	Choose k_1 and $\mathscr{F}_1$;
	Deflate: $\begin{bmatrix} \mathscr{V}_1 & P_0 \end{bmatrix} = \hat{\mathscr{V}}_1 \mathscr{F}_1$;
	Deflate: $\Lambda_1 = \mathscr{F}_1^H \hat{\Lambda}_0$;
	Discard P_0 and last d_1 rows of Λ_1, and set $p_0 = k_1$;
for $j = 1, \ldots, m$ **do**	**for** $j = 1, \ldots, m$ **do**
Choose k_j and $\mathscr{F}_j$;	
Deflate: $\begin{bmatrix} \mathscr{V}_j & P_{j-1} \end{bmatrix} = \hat{\mathscr{V}}_j \mathscr{F}_j$;	
Deflate: $\Lambda_j = \mathscr{F}_j^H \hat{\Lambda}_{j-1}$;	
Deflate: $\mathscr{H}_{j-1} = \mathscr{F}_j^H \hat{\mathscr{H}}_{j-1}$;	
Apply Algorithm 3.2.1 obtaining $A\mathscr{Z}_j = \hat{\mathscr{V}}_{j+1} \hat{\mathscr{H}}_j$;	Apply Algorithm 2.5.1 obtaining $A\mathscr{Z}_j = \mathscr{V}_{j+1} \mathscr{H}_j$;
Set $\hat{\Lambda}_j = \begin{bmatrix} \Lambda_j \\ 0_{(k_j - n_j) \times p} \end{bmatrix}$;	Set $\Lambda_j = \begin{bmatrix} \Lambda_j \\ 0_{(k_j - n_j) \times p} \end{bmatrix}$;
Solve: $\min \left\| \hat{\Lambda}_j - \hat{\mathscr{H}}_j Y \right\|_F$;	Solve: $\min \left\| \Lambda_j - \mathscr{H}_j Y \right\|_F$;
if *full convergence detected* **then** break;	**if** *full convergence detected* **then** break;
;	;
end for	**end for**

  Update $R_0 = B - AX_0$;
  $X_0 = X_0 + \mathscr{Z}_m Y_m$;
end for

Thus one cycle of DMBR is algebraically equivalent to BFGMRESD only when no deflation occurs (in which case both algorithms are equivalent to BFGMRES). This happens because BFGMRESD generates a different correction subspace, since it discards P_0 and does not orthogonalize $\mathscr{V}_2$ against it.

Another remark is that the truncation of $\Lambda_1^{\#}$ means in other words that the least squares problem minimizes only (a linear combination of) k_1 columns of the true block residual R_j (namely those which did not converge yet) neglect the remaining ones, which is what we are aiming for in a deflated scenario. This is however not true in the case of BFGMREST, which consists of BFGMRESD where $k_{max} < p_0$. BFGMREST truncates $\Lambda_1^{\#}$ even if the singular values of the scaled true residual are not small, meaning that BFGMREST potentially neglect (a linear combination of) columns of R_j which did not converge yet.

As demonstrated in Section 3.4, this is not the case for DMBR, which always minimizes the norm of the whole true block residual R_j every iteration, regardless of the chosen k_j or k_{max}. This behaviour is one

of the main contributions of DMBR over BFGMREST for the case of truncated scenario, and has shown to provide a considerable computational gain in our numerical experiments. Naturally, another feature present in DMBR and not in BFGMRESD or BFGMREST is the possibility of deflation every iteration.

3.6 Breakdown in DMBR

In this section we continue the study of breakdowns initially mentioned in Section 2.6. We focus on the possibility of occurrence of a breakdown during the execution of DMBR (Algorithm 3.3.1) with Algorithm 3.4.1. We show that the study of breakdowns in DMBR is relevant mainly from the theoretical point of view, since they are associated with the computation of the exact solution (or a linear combination of exact solutions), and the method tends to present a full convergence before the occurrence of any partial breakdown in most practical cases. Because DMBR can be perceived as a generalization of Block GMRES presented in Section 2.7, this section covers both cases.

The concept of (partial) breakdown we use for the deflated scenario is identical to the one used for the undeflated scenario (cf. Definition 2.6.1 in Section 2.6), but we reproduce it here for convenience.

Definition 3.6.1 (Partial Breakdown)**.** At the j-th iteration of Algorithm 3.2.1, we say that n_j *partial Arnoldi breakdowns* (or simply *partial breakdowns*) have been detected. Whenever $n_j = p_{j-1}$, we say that a *full Arnoldi breakdown* (or simply *full breakdown*) has been detected.

The following theorem aims at showing that any breakdown in DMBR with Algorithm 3.4.1 is indeed a "beneficial" breakdown.

Theorem 3.6.2. *After j iterations of DMBR (Algorithm 3.3.1) with Algorithm 3.4.1 if the sequence of variable preconditioners M_i, $1 \leq i \leq j$ is rank-preserving (cf. Definition 2.4.1), then the exact solution of at least $p - p_j$ linear combinations of systems is already known.*

Proof. Due to the assumption of rank preservation of $\mathscr{Z}_j$, we know from Proposition 3.4.4 that $\operatorname{rank}(\hat{\mathscr{H}}_j) = s_j$. Using Lemma 3.3.1 and remarking that $Y_j = \hat{\mathscr{H}}_j^{\dagger}\hat{\Lambda}_j$, we find that

$$\begin{aligned} R_j &= \hat{\mathscr{V}}_{j+1}\left(\hat{\Lambda}_j - \hat{\mathscr{H}}_j Y_j\right) \\ &= \hat{\mathscr{V}}_{j+1}\left(I_{(s_j+p_j)} - \hat{\mathscr{H}}_j\hat{\mathscr{H}}_j^{\dagger}\right)\hat{\Lambda}_j \end{aligned}$$

Thanks to Proposition 3.4.4, $\hat{\mathscr{V}}_j$ has full rank and we thus conclude that

$$\operatorname{rank}\left(R_j\right) = \operatorname{rank}\left(\left(I_{(s_j+p_j)} - \hat{\mathscr{H}}_j\hat{\mathscr{H}}_j^{\dagger}\right)\hat{\Lambda}_j\right).$$

It is known that $(I_{(s_j+p_j)} - \hat{\mathscr{H}}_j\hat{\mathscr{H}}_j^{\dagger})$ is the orthogonal projector onto the orthogonal complement of $\hat{\mathscr{H}}_j$, whose dimension is p_j. In such a case, the projection of the p columns of $\hat{\Lambda}_j$ onto such subspace could not possibly span a subspace of dimensions larger than p_j, and we can ensure that

$$\operatorname{rank}\left(R_j\right) = \operatorname{rank}\left(\left(I_{(s_j+p_j)} - \hat{\mathscr{H}}_j\hat{\mathscr{H}}_j^{\dagger}\right)\hat{\Lambda}_j\right) \leq p_j,$$

proving that $\operatorname{null}(R_j) \geq p - p_j$. From Property 2.2.7 the proof is completed. □

Corollary 3.6.3. *For every iteration j of Algorithm 3.3.1 with Algorithm 3.4.1 for choosing each $\mathscr{F}_i$, $1 \leq i \leq j$, it holds that* $\operatorname{rank}\left(R_j\right) \leq p_j$.

Corollary 3.6.4. *If a full breakdown has been found in j iterations of Algorithm 3.3.1 with Algorithm 3.4.1 for choosing each $\mathscr{F}_i$, $1 \leq i \leq j$, then* $\operatorname{range}\left(X_*\right) \subset \mathcal{Z}_j$.

Theorem 3.6.2 shows that any occurrence of a breakdown can be considered to be the analogous of the *"happy breakdown"* for the single right-hand side case. Also it links the occurrence of a partial breakdown with the rank deficiency of the block residual R_j.

Remark 3.6.5. Note here that partial breakdowns do not stop the execution of the algorithms, and all the previous properties still hold. Only the full breakdown forces the algorithm to stop since it does not exist $1 \leq k_j \leq p_{j-1} = 0$ in line 7 of Algorithm 3.3.1.

We have shown that DMBR with Algorithm 3.4.1 will always converge to the exact solution if the restart size is large enough. However, in practice we are not looking for the exact solution (nor can we afford a restart size as large as n or ℓ in most cases) but for a full partial convergence (cf. Definition 2.8.1). Unless we are considering ε being close to the machine precision, the partial breakdown phenomena is rather uncommon since the algorithm tends to find an approximation satisfying Definition 2.8.1 before the block true residual shows an exact rank deficiency.

3.7 Alternative $\mathscr{F}_j$ for large p

The objective of this section is to propose a computationally cheaper alternative criterion for choosing $\mathscr{F}_j$ instead of Algorithm 3.4.1, which preserves some of the previously discussed properties of DMBR. The criterion we propose in this section is purely academic and we were unable to find a real life application in which the use of such a criterion is advised, reason why we briefly discuss this criterion exclusively in this section.

When the assumption $n \gg p$ does not hold, using Algorithm 3.4.1 may be prohibitive due to the singular value decomposition involving $(\hat{\Lambda}_j - \hat{\mathscr{H}_j} Y_j)$, which has dimension $(s_j + p) \times p$. It is possible to choose an inferior but cheaper deflation criterion, only assuming that it is possible to perform the QR decomposition in line 4 of Algorithm 3.3.1 (which may be as well prohibitive for large values of p). For the sake of simplicity, in this section we always consider that $p_j = p$.

Suppose that $F_i = I_p$, $\forall i < j-1$ (that is, no deflation has been performed so far) in DMBR. Consider that we dispose of a permutation matrix $T_{j-1} \in \mathbb{C}^{p\times p}$ such that the scalars

$$\tau_{j-1}^i = \left\| R_{j-1} D_{j-1} T_{j-1} e^i \right\|_2 \tag{3.7.1}$$

$$= \left\| (\hat{\Lambda}_{j-1} - \hat{\mathscr{H}_{j-1}} Y_{j-1}) D_{j-1} T_{j-1} e^i \right\|_2, i = 1, ..., p \tag{3.7.2}$$

are given in nonincreasing order, that is, $\tau_{j-1}^1 \geq \tau_{j-1}^2 \geq ... \geq \tau_{j-1}^p$. Defining $k_j \leq p$ such that $\tau_{j-1}^{k_j} < \sqrt{1/p}\,\varepsilon$ and $d_j = p - k_j$, we split $T_{j-1} = \begin{bmatrix} T_{j-1}^+ & T_{j-1}^- \end{bmatrix}$, $T_{j-1}^+ \in \mathbb{C}^{p\times k_j}$, $T_{j-1}^+ \in \mathbb{C}^{p\times d_j}$ and we conclude that

$$\left\| R_{j-1} D_{j-1} T_{j-1}^- \right\|_F = \sqrt{\sum_{i=k_j}^{p} \left(\tau_{j-1}^i\right)^2} \leq \sqrt{d_j/p}\,\varepsilon,$$

characterizing a partial convergence (cf. Definition 2.8.1) and thus we can deflate the information associated with $R_{j-1} D_{j-1} T_{j-1}^-$. Since we supposed $p_i = p, F_i = I_p$, $\forall i < j$, we can ensure that $\hat{V}_j e^i$ is the direction associated with $R_{j-1} e^i$ and thus the directions we want to keep are $\hat{V}_j T_{j-1}^+$ and the ones we want to postpone are $\hat{V}_j T_{j-1}^-$ giving us the relation

$$\begin{bmatrix} V_j & P_{j-1} \end{bmatrix} = \hat{V}_j T_{j-1}$$

showing thus that

$$F_j = T_{j-1} \quad \text{and} \quad \mathscr{F}_j = \begin{bmatrix} I_{s_{j-1}} & 0 \\ 0 & F_j \end{bmatrix}$$

is the suitable deflation matrix for iteration j according to this criterion.

For subsequent iterations, we have to retrieve the original ordering instead. Splitting $F_j = \begin{bmatrix} F_j^+ & F_j^- \end{bmatrix}$, $F_j^+ \in \mathbb{C}^{p \times k_j}$, $F_j^+ \in \mathbb{C}^{p \times d_j}$, we know that P_{j-1} was reordered in iteration j and corresponds to $\hat{V}_j F_j^-$, whereas $\hat{V}_{j+1}$ corresponds to the update of $\hat{V}_j F_j^+$. Supposing that the QR decomposition in line 6 of Algorithm 3.2.1 does not reorder any information in $\hat{V}_{j+1}$ (which is possible if using a Gram-Schmidt-based QR algorithm without pivoting, for instance) we find out that the matrix with the correct ordering is

$$\begin{bmatrix} P_{j-1} & \hat{V}_{j+1} \end{bmatrix} \bar{T}_j F_j^H \quad \text{with} \quad \bar{T}_j = \begin{bmatrix} 0 & I_{d_j} \\ I_{k_j} & 0 \end{bmatrix}$$

where $\bar{T}_j$ is a matrix responsible for swapping the position of P_{j-1} and $\hat{V}_{j+1}$ without modifying the ordering of the columns of each matrix.

Writing the residual R_j as

$$R_j = \hat{\mathscr{V}}_{j+1} \begin{bmatrix} I_{s_j} & 0 \\ 0 & \bar{T}_j F_j^H \end{bmatrix} \begin{bmatrix} I_{s_j} & 0 \\ 0 & F_j \bar{T}_j^H \end{bmatrix} (\hat{\Lambda}_j - \hat{\mathscr{H}}_j Y_j)$$

we want to find the matrix T_j such that the scalars

$$\tau_j^i = \left\| R_j D_j T_j e^i \right\|_2, \tag{3.7.3}$$

$$= \left\| (\hat{\Lambda}_j - \hat{\mathscr{H}}_j Y_j) D_j T_j e^i \right\|_2 \tag{3.7.4}$$

for $i = 1, ..., p$ are given in nonincreasing order. Having such a matrix, we set

$$F_{j+1} = \bar{T}_j F_j^H T_j \qquad \text{and} \qquad \mathscr{F}_{j+1} = \begin{bmatrix} I_{s_j} & 0 \\ 0 & F_{j+1} \end{bmatrix}$$

as our deflation matrix. A pseudocode for performing such a choice is depicted in Algorithm 3.7.1.

Algorithm 3.7.1: *Choosing $\mathscr{F}_j$ - Largest Singular Values of $R_{j-1} D_{j-1}$*

1 Receive $R_{j-1} D_{j-1} = \begin{bmatrix} \mathscr{V}_j & P_{j-1} \end{bmatrix} (\hat{\Lambda}_{j-1} - \hat{\mathscr{H}}_{j-1} Y_{j-1}) D_{j-1}$ and the parameters $1 \leq k_{max} \leq p_{j-1}$ and ε^d;
2 Obtain the permutation matrix T_{j-1} such that the sequence $\{||(\hat{\Lambda}_{j-1} - \hat{\mathscr{H}}_{j-1} Y_{j-1}) D_{j-1} T_{j-1} e^i||_2\}_{i=1}^p$ is given in nonincreasing order;
3 Choose $\tilde{k}_j$ such that $||(\hat{\Lambda}_{j-1} - \hat{\mathscr{H}}_{j-1} Y_{j-1}) D_{j-1} T_{j-1} e^l||_2 \geq \varepsilon^d$ for all $1 \leq l \leq \tilde{k}_j$;
4 Set $k_j = \min(\tilde{k}_j, k_{max})$;
5 Set $\bar{T}_j = \begin{bmatrix} 0 & I_{d_j} \\ I_{k_j} & 0 \end{bmatrix}$ with $k_0 = p$ and $d_0 = 0$;
6 $F_j = \bar{T}_j F_{j-1}^H T_j$;
7 Define $\mathscr{F}_j = \begin{bmatrix} I_{s_{j-1}} & 0 \\ 0 & F_j \end{bmatrix} \in \mathbb{C}^{(s_j + d_j) \times (s_j + d_j)}$;

We remark that each τ_j^i can be found by simply ordering the values of T_j, which can be done in $p \log p$ operations. Also each F_j can be stored as a vector of dimension p, because T_j, F_{j-1} and $\bar{T}_j$ are all permutation matrices. Also, during iteration j we need only F_{j-1}, T_j and the τ_j^i which means that an extra storage of $3p$ is needed when compared to BGMRES. Notice that in this case the application of $\mathscr{F}_{j+1}$ does not require any extra operation: it is simply a reordering of columns.

Although this strategy requires nearly no extra cost for computing $\mathscr{F}_j$ even for values of p close to n, it tends to perform very close to BGMRES in practice, and whenever it is possible, Algorithm 3.4.1 is preferred instead. Therefore we do not focus our attention on Algorithm 3.7.1 but on Algorithm 3.4.1 instead.

3.8 Computational Cost and Memory Requirements

We discuss now the computation cost for applying one cycle of DMBR, including the costs associated with the deflated block Arnoldi iteration (cf. Algorithm 3.2.1) and computing the unitary deflation operator (we always suppose here that Algorithm 3.4.1 is used).

We summarize in Table 3.1 the costs occurring during a given cycle of DMBR(m)[2], excluding matrix-vector products and preconditioning operations which are problem dependent. We have included the costs proportional to both the size of the original problem n and the maximal number of right-hand sides p, assuming a QR factorization based on modified Gram-Schmidt and a Golub-Reinsch SVD[3]; see, e.g, [60, Section 5.4.5] and [70, Appendix C] for further details on operation counts. The total cost of a given cycle is then found to grow as $C_1np^2 + C_2p^3 + C_3np$ and we note that this cost is always nonincreasing along convergence due to block size reduction. Compared to methods including deflation at restart only, additional operations are related to the computations of $\mathscr{F}_1$, Λ_1, $\mathscr{F}_{j+1}$, $\begin{bmatrix}\mathscr{V}_{j+1} & P_j\end{bmatrix}$, Λ_{j+1} and $\mathscr{H}_j$, operations that behave respectively as p^3 and np^2. The computation of $\begin{bmatrix}\mathscr{V}_{j+1} & P_j\end{bmatrix}$ is in practice the most expensive one in a given iteration of DMBR(m). Concerning the truncated variant, the computational cost of a cycle will be reduced only when $\tilde{k}_j > k_{max}$ since the upper bound on k_{j+1} will be then active. This situation occurs at the beginning of the convergence due to the nonincreasing behaviour of the singular values of the block residual.

Step	Computational cost
QR factorization of R_0	$2np^2 + np$
Computation of $\mathscr{F}_1$	$4p_0p^2 + 8p^3 + 2p_0^3$
Computation of $\begin{bmatrix}\mathscr{V}_1 & P_0\end{bmatrix}$	$2np_0^2$
Computation of Λ_1	$2p_0^2p$
Block Arnoldi procedure[1]	C_j
Computation of Y_j	$2(s_j + p_j)s_j^2 + ps_j^2$
Computation of $\hat{\Lambda}_j - \hat{\mathscr{H}}_j Y_j$	$(s_j + p_j)p + 2(s_j + p_j)s_jp$
Computation of $\mathscr{F}_{j+1}$	$4(s_j + p_j)p^2 + 8p^3 + 2p_j^3$
Computation of $\begin{bmatrix}\mathscr{V}_{j+1} & P_j\end{bmatrix}$	$2np_j^2$
Computation of Λ_{j+1}	$2p_j^2p$
Computation of $\mathscr{H}_j$	$2p_j^2p$
Computation of X_m	$np + 2ns_mp + s_mp$

Table 3.1: *Computational cost of a cycle of DMBR(m) (Algorithm 3.3.1). This excludes the cost of matrix-vector operations and preconditioning operations.*

[2] that is, DMBR with restart parameter equal to m

[3] The Golub-Reinsch SVD decomposition $R = U\Sigma V^H$ with $R \in \mathbb{C}^{m\times n}$ requires $4mn^2 + 8n^3$ operations when only Σ and V have to be computed.

[1] Algorithm 3.2.1: the block Arnoldi method based on modified Gram-Schmidt requires $\sum_{j=1}^{m}\sum_{i=1}^{j}(4nk_ik_j + nk_j + 4nd_jk_j)$ operations (line 4 to 5) plus $\sum_{j=1}^{m} 2nk_j^2$ operations for the QR decomposition of S (line 6). Thus $C_j = \sum_{j=1}^{m}(\sum_{i=1}^{j}(4nk_ik_j + nk_j + 4nd_jk_j) + 2nk_j^2)$.

We do not include in Table 3.1 the cost associated with computing and updating the scaling matrix D_j or applying it to the residual R_j, as it is dependent of the scaling strategy being used.

Concerning storage proportional to the problem size n, DMBR(m) requires R_m, X_0, X_m, $\mathscr{V}_{m+1}$ and $\mathscr{Z}_m$ respectively leading to a memory requirement of $2ns_m + np_m + 3np$ at the end of a given cycle. Since s_m varies from cycle to cycle an upper bound of the memory requirement can be given as $n(2m+1)p_0+3np$ when p_0 linear systems have to be considered at the beginning of a given cycle. We note that the storage is monotonically decreasing along convergence, a feature than can be for instance exploited if dynamic memory allocation is used.

3.9 Numerical Experiments

In this section we investigate the numerical behaviour of block flexible Krylov subspace methods including deflation at each iteration on different problems. We start with three academic illustrations, where two of them (namely in Subsection 3.9.1 and Subsection 3.9.2) are meant to reproduce results found previously in [23], and then we extend the numerical experiments to the forward problem associated with wave propagation phenomena where the multiple right-hand side situation frequently occurs. It consists of a challenging realistic application in geophysics, which we describe in more detail in Chapter 4.

Nevertheless, in wave propagation applications normally variable preconditioners are used. Multigrid techniques using Krylov iterative methods both as smoother and coarse grid correction have been used in the current literature and have reported very good performance in massively parallel environment [22, 23, 96]. We discuss in depth the preconditioning issue for this problem in Section 4.4, but since the wave propagation our main application in this thesis, we investigate here the behaviour of the block Krylov methods specifically when using variable preconditioners.

Except for the illustration in Subsection 3.9.2, all the right-hand sides correspond to canonical vectors. Thus the block right-hand side $B \in \mathbb{C}^{n\times p}$ is extremely sparse (only one nonzero element per column) and the initial block residual corresponds to a full rank matrix. This also has connections with our geophysical application described in Chapter 4, where the right-hand sides are often sparse.

We compare both BFGMRES-R(m) and DMBR(m) with other preconditioned iterative methods based on flexible BGMRES(m) for the solution of these problems with a zero initial guess (X_0) and a small value of the restart parameter m when using a variable preconditioner. The choice of small values of m here is suitable since we are considering mostly *inner-outer* methods [112]. It is known that for this kind of method, the dimension of the Krylov subspace being built is in fact a combination of the outer and inner subspace.

We use the same stopping criterion for all experiments also. We attempt to find an approximate solution satisfying $||R_j D_j||_\psi \leq \varepsilon$, where the scaling matrix is always set as

$$D_j^{-1} = diag(||Be^1||_2, ||Be^2||_2, ..., ||Be^p||_2)$$

for every j, for all the experiments (thus D_j is a fixed scaling matrix).

Our goal is to analyse the performance of block Krylov subspace methods including deflation at each iteration versus classical methods discussed in Section 3.5. A primary concern will be to evaluate if DMBR(m) can be efficient when solving problems with multiple right-hand sides both in terms of preconditioner applications and total computational cost.

3.9.1 Poisson Problem

In this experiment we use a Matlab [82] implementation of the referred methods and we attempt to reproduce the results for BFGMRESD in [96] (see Table 2.6 in [96]). It consists of the two-dimensional Poisson problem

$$-\Delta u = g$$

with Dirichlet boundary conditions, discretized with a second-order finite differences scheme for a vertex-centred grid, with a mesh-grid equal to 1/128. The coefficient matrix was taken using Matlab's routine *gallery('poisson',128)*.

In all numerical experiments, the convergence and deflation threshold are set as $\varepsilon = \varepsilon^d = 10^{-6}$. We are interested in analysing the behaviour of the deflation as the number of right-hand sides increases. We start with 5 right-hand sides and proceed by doubling the number of right-hand sides until 160 right-hand sides. All right-hand sides are canonical vectors in this test.

We show in the Tables 3.2 to 3.4 the number of iterations (It), the number of matrix-vector products on a single vector (MVP) and number of preconditioner applications on a single vector (Pr) required for various restarted block flexible Krylov subspace methods performing no deflation (BFGMRES(m)), deflation at the beginning of cycle only (BFGMRESD(m)) and deflation at each iteration (BFGMRES-R(m) and DMBR(m)). We note that all selected methods solve the minimization problem over a subspace of similar maximal dimension (mp). Since the algorithm we propose aims at saving matrix-vector product and preconditioner application, for this numerical experiment we only focus on this factor (cf. Remark 3.9.1 for some comments on the orthogonalization cost).

Remark 3.9.1. During iteration j, DMBR orthogonalizes k_j vectors against $s_j + d_j$ vectors, whereas BFGMRESD orthogonalizes k_1 vectors against $j \times k_1$. Since $k_j \leq k_1$ and $s_j \leq j \times k_1$ (both due to Corollary 3.4.7), DMBR may actually perform less orthogonalizations per iterations than BFGMRESD.

Therefore, we highlight that although we do not detail the orthogonalization cost for this illustration, the number of orthogonalization steps performed by DMBR is equal or smaller than those performed by BGMRES and BGMRES-R, and in some cases also smaller than those performed by BFGMRESD depending on how early in the cycle the deflation takes place. In Subsection 3.9.3 and Subsection 3.9.4 we show numerical experiments addressing the total computational time, including matrix-vector products, preconditioner applications and orthogonalization. ■

For the first experiment, in Table 3.2 we use 5 cycles of BGMRES(5) as variable preconditioner, meaning that each preconditioning application involves 25 matrix vector products. We provide in the ρ column the following ratio:

$$\rho(method) = \frac{MVP(method(m)) + 25 \times Pr(method(m))}{MVP(DMBR(m)) + 25 \times Pr(DMBR(m))}. \tag{3.9.1}$$

which scales the number of matrix-vector product operations performed with respect to the DMBR method. A value of ρ greater than one indicates that the given block subspace method performs more matrix-vector products than DMBR. We set the restart parameter $m = 5$ and $k_{max} = p$. The second experiment, in Table 3.3, we increase the number of iterations per cycle, but reduce the quality of the preconditioner in order to observe the behaviour along several cycles. We use 3 cycles of BGMRES(3) as variable preconditioner (thus, each preconditioning application involves 9 matrix vector products), restart parameter $m = 5$ and $k_{max} = p$, and we update the ratio to

$$\hat{\rho}(method) = \frac{MVP(method(m)) + 9 \times Pr(method(m))}{MVP(DMBR(m)) + 9 \times Pr(DMBR(m))}. \tag{3.9.2}$$

instead of $\rho(method)$. The third experiment, shown in Table 3.4, considers a more limited memory setting. We set use again 5 cycles of BGMRES(5) as variable preconditioner, but we set $k_{max} = 20$ while maintaining the other parameters unchanged. We start with $p = 40$ and proceed by adding 20 right-hand sides until it reaches the 160 limit. Since BGMRES-R cannot limit the memory in such fashion (p_1 is always equal to p in BGMRES-R), we let it aside in this test, and we compare only DMBR with BFMGREST, using $\rho(method)$ from (3.9.1).

Table 3.2 reveals that for this particular problem, DMBR performance is clearly superior to BGMRES-R and BFGMRESD, and that the gap between DMBR and the other methods increases with the number of right-hand sides. Notice that, since the restart size is small, BFGMRESD converges performing less matrix-vector products and preconditioning applications than BFGMRES-R. This behaviour is expected since

Poisson equation - $Grid : 128 \times 128$								
$m = 5$, no truncation ($k_{max} = p$)								
	$p = 5$				$p = 10$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMRES	18	115	90	2.75	19	240	190	4.08
BFGMRES-R	22	77	47	1.45	24	139	79	1.72
BFGMRESD	22	72	42	1.30	23	133	73	1.60
DMBR	23	62	32	1	25	105	45	1
	$p = 20$				$p = 40$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMRES	16	420	320	4.46	15	760	600	5.13
BFGMRES-R	25	260	140	1.98	26	578	298	2.60
BFGMRESD	27	272	132	1.88	31	566	246	2.17
DMBR	26	208	68	1	27	389	109	1
	$p = 80$				$p = 160$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMRES	14	1440	1120	5.65	11	2400	1760	5.35
BFGMRES-R	26	1127	567	2.92	27	2199	1079	3.35
BFGMRESD	29	1019	459	2.38	28	1983	863	2.70
DMBR	28	742	182	1	28	1417	297	1

Table 3.2: *Poisson equation discretized with $h = 1/128$ with 5 cycles of BGMRES(5) as variable preconditioner, restart size 5 and a number of right-hand sides given at once ranging from $p = 5$ to $p = 160$. It denotes the number of iterations, MVP the number of matrix-vector applications on a single vector, Pr the number of preconditioner applications on a single vector and ρ a scaled measure of efficiency in terms of number of matrix-vector products performed both by the method and its preconditioner.*

Poisson equation - $Grid : 128 \times 128$								
$m = 15$, no truncation ($k_{max} = p$)								
	$p = 5$				$p = 10$			
	It	MVP	Pr	$\hat{\rho}$	It	MVP	Pr	$\hat{\rho}$
BFGMRES	49	275	245	2.99	52	590	520	4.67
BFGMRES-R	55	132	97	1.21	60	221	151	1.39
BFGMRESD	52	137	102	1.27	67	247	167	1.54
DMBR	57	115	80	1	60	177	107	1
	$p = 20$				$p = 40$			
	It	MVP	Pr	$\hat{\rho}$	It	MVP	Pr	$\hat{\rho}$
BFGMRES	48	1080	960	5.59	38	1720	1520	5.63
BFGMRES-R	65	429	269	1.62	67	791	471	1.82
BFGMRESD	72	472	292	1.76	73	883	523	2.02
DMBR	68	321	161	1	70	568	248	1
	$p = 80$				$p = 160$			
	It	MVP	Pr	$\hat{\rho}$	It	MVP	Pr	$\hat{\rho}$
	It	MVP	PC	Ratio	It	MVP	PC	Ratio
BFGMRES	29	2640	2320	5.23	27	4960	4320	6.02
BFGMRES-R	65	1498	858	2.02	69	2844	1564	2.29
BFGMRESD	70	1610	970	2.27	75	3235	1795	2.63
DMBR	67	1039	399	1	69	1907	627	1

Table 3.3: *Poisson equation discretized with $h = 1/128$ with 3 cycles of BGMRES(3) as variable preconditioner, restart size 15 and a number of right-hand sides given at once ranging from $p = 5$ to $p = 160$. It denotes the number of iterations, MVP the number of matrix-vector applications on a single vector, Pr the number of preconditioner applications on a single vector and $\hat{\rho}$ a scaled measure of efficiency in terms of number of matrix-vector products performed both by the method and its preconditioner.*

Poisson equation - $Grid : 128 \times 128$								
$m = 5$, truncation with $k_{max} = 20$								
	$p = 40$				$p = 80$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMREST	26	531	251	2.06	27	1066	506	1.98
DMBR	27	397	117	1	26	806	246	1
	$p = 120$				$p = 160$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMREST	40	1880	800	1.94	52	2952	1032	1.80
DMBR	31	1359	399	1	32	1846	566	1

Table 3.4: *Poisson equation discretized with $h = 1/128$ with 5 cycles of BGMRES(5) as variable preconditioner, with a number of right-hand sides given at once ranging from $p = 40$ to $p = 160$ and using truncation ($k_{max} = 20$). It denotes the number of iterations, MVP the number of matrix-vector applications on a single vector, Pr the number of preconditioner applications on a single vector and ρ a scaled measure of efficiency in terms of number of matrix-vector products performed both by the method and its preconditioner.*

BFGMRESD is supposed to benefit from small restart sizes. In Table 3.3, as expected, we observe that BFGMRES-R benefits from the larger restart size and performs considerably less matrix-vector products and preconditioner applications than BFGMRESD. However, DMBR method is still the cheapest in terms of matrix-vector and preconditioner applications. In the more memory constrained test, in Table 3.4, we see that once again the reduction on the number of matrix-vector products of DMBR over BFGMREST is considerable, but this time the difference between the methods does not seem to increase with the number of right-hand sides p.

3.9.2 Convection-Diffusion Problem

We continue Matlab experiments taking into account the matrix-vector products and preconditioner applications cost only. In this subsection we test the behaviour of the deflated methods for a non-Hermitian problem, the convection-diffusion equation given by

$$\begin{cases} -\Delta u + cu_x + du_y = g & \text{in } \Omega, \\ u = 1 & \text{on } \partial\Omega \end{cases}$$

where $\Omega =]0,1[^2$ is the interior of the domain and $\partial\Omega = (0,1)^2$ is the boundary. We take $c = d = 256$ for our experiments, and discretize the convection-diffusion equation using finite differences in a 5-point Cartesian stencil, once again with mesh size equal to $1/128$. We generate a random exact solution satisfying the boundary condition to then obtain the right-hand sides. This experiment is meant to reproduce the results in [96], Table 2.8.

Our setting is very similar to the one in Subsection 3.9.1: first we use 5 cycles of BGMRES(5) as variable preconditioner, $\varepsilon = \varepsilon^d = 10^{-6}$ with restart parameter $m = 5$ and $k_{max} = p$ in Table 3.5. Then in Table 3.6 we use 3 cycles of BGMRES(3) but increase the restart size to $m = 15$. Finally, in Table 3.7 we consider the limited memory setting with $k_{max} = 20$ while using 5 cycles of BGMRES(5) as variable preconditioner and $m = 5$. We also reproduce in the column ρ the same ratio previously mentioned in (3.9.1) and in the column $\hat{\rho}$ the ratio in (3.9.2), and although we do not address orthogonalization costs, Remark 3.9.1 is still valid.

In Table 3.7 we see that DMBR method is always able to save matrix-vector product computations by performing deflation every iteration. Observe however that in contrast with Table 3.5 where the number of iterations of each method remains almost unchanged as p increases, in Table 3.7, DMBR performs considerably less iterations than BFMGREST. This speedup has indeed a reason: due to Corollary 3.3.2, we guarantee that DMBR minimizes the entire residual every iteration (regardless of the value of k_j), whereas BFGMREST chooses just a subset of the residual do minimize every cycle (as discussed in Section 3.5). We consider that this is indeed a critical feature of DMBR mainly if p is considerably large (of the order of hundreds).

Also, comparing Table 3.5 with Table 3.6 we see that once again BFGMRES-R profits from long restart sizes whereas BFGMRESD profits from small restart size, but DMBR performs well in both cases. Also, for $p = 160$ in Table 3.6 only one cycle is performed (all methods converge in 14 iteration) and that BFGMRESD is equivalent to BGMRES and DMBR is equivalent to BFGMRES-R. This behaviour is expected as we explained in Section 3.5.

3.9.3 Complex-valued advection diffusion reaction problem

For this more realistic Matlab experiment, we consider a complex-valued partial differential equation of advection diffusion reaction type in two-dimensions defined on a square domain $\Omega = [0,1]^2$. Recently Haber and MacLachlan [66] have proposed a continuous transformation of the acoustic wave equation based on the Rytov decomposition that requires the solution of a partial differential equation that is more amenable to efficient numerical methods than the original indefinite Helmholtz equation. This equation

Convection-diffusion equation with $c = d = 256$ - $Grid : 128 \times 128$								
$m = 5$, no truncation ($k_{max} = p$)								
	$p = 5$				$p = 10$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMRES	30	185	150	1.45	29	360	290	1.57
BFGMRES-R	34	156	116	1.12	32	293	213	1.16
BFGMREST	33	153	113	1.10	30	265	195	1.06
DMBR	32	143	103	1	31	263	183	1
	$p = 20$				$p = 40$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMRES	28	700	560	1.85	20	1000	800	1.74
BFGMRES-R	30	493	353	1.17	27	867	587	1.29
BFGMREST	29	474	334	1.11	30	795	515	1.13
DMBR	29	441	301	1	28	734	454	1
	$p = 80$				$p = 160$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMRES	16	1680	1280	1.75	11	2400	1760	1.60
BFGMRES-R	22	1428	948	1.30	22	2539	1579	1.45
BFGMREST	27	1407	847	1.17	22	2397	1437	1.32
DMBR	23	1204	724	1	23	2041	1081	1

Table 3.5: *Convection-diffusion equation, with $h = 1/128$ and $c = d = 256$, with 5 cycles of BGMRES(5) as variable preconditioner, restart size 5 and a number of right-hand sides given at once ranging from $p = 5$ to $p = 160$. It denotes the number of iterations, MVP the number of matrix-vector applications on a single vector, Pr the number of preconditioner applications on a single vector and ρ a scaled measure of efficiency in terms of number of matrix-vector products performed both by the method and its preconditioner.*

Convection-diffusion equation with $c = d = 256$ - $Grid : 128 \times 128$								
$m = 15$, no truncation ($k_{max} = p$)								
	$p = 5$				$p = 10$			
	It	MVP	Pr	$\hat{\rho}$	It	MVP	Pr	$\hat{\rho}$
BFGMRES	90	485	450	1.38	89	960	890	1.61
BFGMRES-R	108	390	345	1.06	92	659	579	1.05
BFGMRESD	102	382	342	1.05	102	692	612	1.11
DMBR	103	365	325	1	92	631	551	1
	$p = 20$				$p = 40$			
	It	MVP	Pr	$\hat{\rho}$	It	MVP	Pr	$\hat{\rho}$
BFGMRES	72	1560	1440	1.65	45	1960	1800	1.52
BFGMRES-R	104	1105	945	1.09	62	1519	1279	1.09
BFGMRESD	98	1158	998	1.15	69	1554	1314	1.12
DMBR	97	1028	868	1	65	1416	1176	1
	It	MVP	Pr	$\hat{\rho}$	It	MVP	Pr	$\hat{\rho}$
	It	MVP	PC	Ratio	It	MVP	PC	Ratio
BFGMRES	26	2320	2080	1.64	14	2560	2240	1.24
BFGMRES-R	45	1699	1379	1.10	14	2116	1796	1.00
BFGMRESD	60	2035	1635	1.30	14	2560	2240	1.24
DMBR	46	1651	1251	1	14	2116	1796	1

Table 3.6: *Convection-diffusion equation, with $h = 1/128$ and $c = d = 256$, with 3 cycles of BGMRES(5) as variable preconditioner, restart size 15 and a number of right-hand sides given at once ranging from $p = 5$ to $p = 160$. It denotes the number of iterations, MVP the number of matrix-vector applications on a single vector, Pr the number of preconditioner applications on a single vector and $\hat{\rho}$ a scaled measure of efficiency in terms of number of matrix-vector products performed both by the method and its preconditioner.*

Convection-diffusion equation with $c = d = 256$ - $Grid : 128 \times 128$								
$m = 5$, truncation with $k_{max} = 20$								
	$p = 40$				$p = 80$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMREST	40	1015	655	1.24	60	2240	1200	1.30
DMBR	33	847	527	1	46	1800	920	1
	$p = 120$				$p = 160$			
	It	MVP	Pr	ρ	It	MVP	Pr	ρ
BFGMREST	85	3860	1700	1.37	105	5620	2100	1.34
DMBR	62	2920	1240	1	78	4280	1560	1

Table 3.7: *Convection-diffusion equation, with $h = 1/128$ and $c = d = 256$, with 5 cycles of BGMRES(5) as variable preconditioner, with a number of right-hand sides given at once ranging from $p = 40$ to $p = 160$ and using truncation ($k_{max} = 20$). It denotes the number of iterations, MVP the number of matrix-vector applications on a single vector, Pr the number of preconditioner applications on a single vector and ρ a scaled measure of efficiency in terms of number of matrix-vector products performed both by the method and its preconditioner.*

with Dirichlet boundary conditions reads as follows:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - 2i\omega c(\alpha_x \frac{\partial u}{\partial x} + \alpha_y \frac{\partial u}{\partial y}) + \omega^2(c^2 - \kappa^2)u = g_s(\mathbf{x}) \tag{3.9.3}$$

$$\mathbf{x} = (x, y) \in \Omega, \tag{3.9.4}$$

$$u = 0 \quad \text{on} \quad \partial\Omega, \tag{3.9.5}$$

where ω, c, α_x, α_y, κ are real-valued coefficients. The source term $g_s(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x_s})e^{-ic(\alpha_x x_s + \alpha_y y_s)}$ represents a harmonic point source located at (x_s, y_s) in Ω. We consider the pure advection diffusion case ($c = \kappa = 1$) and set the following values for the parameters $\omega = \pi$, $\alpha_x = 1/\sqrt{2}$, $\alpha_y = 1/\sqrt{2}$. The discrete problem is obtained after second-order finite difference discretization of (3.9.3) with a second-order upwind scheme for the treatment of the advection terms as in [66]. The right-hand side we choose for this problem correspond to Dirac sources. The inner preconditioner is based on one cycle of non preconditioned GMRES(m) corresponding to pm additional matrix-vector products when considering a linear system with p right-hand sides. Thus flexible outer Krylov subspace methods are required since the preconditioner is then variable. With this simple preconditioner we note that we can derive a purely matrix-free implementation. Both the tolerance and the deflation threshold are set as $\varepsilon = \varepsilon^d = 10^{-5}$ in the numerical experiments.

Table 3.8 collects the number of outer iterations (It) and number of preconditioner applications on a single vector (Pr) required for various restarted block flexible Krylov subspace methods performing no deflation (BFGMRES(m)), deflation at the beginning of cycle only (BFGMRESD(m)) and deflation at each iteration (BFGMRES-R(m) and DMBR(m)) for two different values of the restart parameter (respectively $m = 5$ and $m = 10$). We have also included results related to restarted flexible GMRES when solving in sequence the p linear systems independently. We note that all selected methods solve the minimization problem over a subspace of similar maximal dimension (mp). In opposition to the two previous numerical experiments, in this one we consider every floating point operation rather than simply the matrix-vector product performed by both methods. We determine the computational complexity of all algorithms including costs related to QR factorization, singular value decomposition, orthonormalization (as listed in Table 3.1), matrix-vector products and preconditioning and define a measure of efficiency τ as

$$\tau(method) = \frac{flops(FGMRES(mp))}{flops(method)}.$$

Thus a value of τ greater than one indicates that the given block subspace method leads to a computational improvement with respect to flexible GMRES applied on the given sequence of linear systems. Table 3.8 reveals that block Krylov subspace methods including deflation either at restart only or at each iteration usually are to be preferred. Indeed those methods always lead to efficiencies τ greater than one. On this application standard block Krylov subspace method (BFGMRES) is not efficient with respect to FGMRES(mp). This highlights the fact that to be effective block subspace methods must incorporate block size reduction or deflation [64]. On the two sets of numerical experiments, whatever the value of the restart parameter m, DMBR(m) always leads to the minimal number of preconditioner applications. Similarly DMBR(m) also delivers the best efficiency (see bold values in Table 3.8). Finally we note that BFGMRES-R(m) is penalized when the number of outer cycles is large since this method does not include initial deflation at the beginning of the cycle. On this academic problem we have shown the interest of using block Krylov subspace methods that include deflation at each iteration. DMBR(m) is indeed found to be competitive with respect to Krylov subspace methods including deflation at restart only.

3.9.4 Acoustic Full Waveform Inversion

We focus on a specific application in geophysics related to the simulation of wave propagation phenomena in the Earth [133]. Given a three-dimensional physical domain Ω_p, the propagation of a wave field in a

Complex-valued advection diffusion problem - *Grid* : 128×128						
$m = 5$						
	$p = 4$			$p = 8$		
Method	*It*	Pr	τ	*It*	Pr	τ
FGMRES($5p$)	75	75	1.00	155	155	1.00
BFGMRES(5)	43	172	0.41	39	312	0.37
BFGMRESD(5)	50	80	0.95	45	100	1.33
BFGMRES-R(5)	39	80	0.90	39	127	0.96
DMBR(5)	44	67	**1.06**	43	87	**1.38**
	$p = 16$			$p = 32$		
Method	*It*	Pr	τ	*It*	Pr	τ
FGMRES($5p$)	315	315	1.00	635	635	1.00
BFGMRES(5)	26	416	0.40	17	544	0.39
BFGMRESD(5)	49	150	1.27	45	235	0.97
BFGMRES-R(5)	39	214	0.85	40	386	0.62
DMBR(5)	45	121	**1.42**	43	181	**1.14**
$m = 10$						
	$p = 4$			$p = 8$		
Method	*It*	Pr	τ	*It*	Pr	τ
FGMRES($10p$)	75	75	1.00	155	155	1.00
BFGMRES(10)	22	88	0.57	20	160	0.57
BFGMRESD(10)	22	63	0.96	24	104	0.96
BFGMRES-R(10)	23	51	1.43	23	78	1.43
DMBR(10)	23	46	**1.69**	23	65	**1.69**
	$p = 16$			$p = 32$		
Method	*It*	Pr	τ	*It*	Pr	τ
FGMRES($10p$)	315	315	1.00	635	635	1.00
BFGMRES(10)	17	272	0.47	16	512	0.31
BFGMRESD(10)	26	186	0.70	30	350	0.43
BFGMRES-R(10)	23	126	1.32	22	216	1.00
DMBR(10)	23	98	**1.63**	22	156	**1.29**

Table 3.8: *Two-dimensional complex-valued advection diffusion problem. Case of* $h = 1/128$, $\omega = \pi$, $\alpha_x = 1/\sqrt{2}, \alpha_y = 1/\sqrt{2}$ *with a number of right-hand sides given at once ranging from* $p = 4$ *to* $p = 32$ *for two different values of the restart parameter* $m = 5$ *(upper part) and* $m = 10$ *(lower part). It denotes the number of iterations, Pr the number of preconditioner applications on a single vector and* τ *a scaled measure of efficiency in terms of computational operations.*

heterogeneous medium can be modelled by the Helmholtz equation written in the frequency domain:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} - \frac{(2\pi f)^2}{c^2(x,y,z)} u \quad = \quad g_s(\mathbf{x}), \quad \mathbf{x} = (x,y,z) \in \Omega_p. \tag{3.9.6}$$

u represents the pressure field in the frequency domain, c the variable acoustic-wave velocity in ms^{-1}, and f the frequency in Hertz. The source term $g_s(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x_s})$ represents a harmonic point source located at (x_s, y_s, z_s). A popular approach — the Perfectly Matched Layer formulation (PML) [13, 14] — has been used in order to obtain a satisfactory near boundary solution, without many artificial reflections. As in [23] we consider a second-order finite difference discretization of the Helmholtz equation (3.9.6) on an uniform equidistant Cartesian grid of size $n_x \times n_y \times n_z$. The same stability condition (12 points per wavelength) relating f the frequency with h the mesh grid size and $c(x,y,z)$ the heterogeneous velocity

field has been considered:

$$f = \frac{\min_{(x,y,z)\in\Omega_h} c(x,y,z)}{12\,h}.$$

In consequence A is a sparse complex matrix which is non-Hermitian and nonsymmetric due to the PML formulation that leads to complex-valued variable coefficients in the partial differential equation [96, Appendix A]. Due also to their lack of definiteness and symmetry, the resulting linear systems are known to be challenging for iterative methods [50, 52]. We refer also to Chapter 4 for more details about this problem.

We consider the same perturbed geometric two-level preconditioner presented in [23] that has been shown to be relatively efficient for the solution of three-dimensional heterogeneous Helmholtz problems in geophysics. We refer the reader to [23, Algorithm 5] for a complete description of the geometric preconditioner and to [96] for additional theoretical properties in relation with Krylov subspace methods. Also, later in Subsection 4.4.1 we detail this preconditioner. Since we are in the multiple right-hand sides scenario, we suppose that the perturbed two-grid preconditioner is applied independently once for each right-hand side. Next investigate the performance of the block flexible Krylov methods presented in Section 3.3 on this challenging real-life application. Both the tolerance and the deflation threshold are set as $\varepsilon = \varepsilon^d = 10^{-5}$ in the numerical experiments. The source terms correspond to Dirac sources. The numerical results have been obtained on Babel, a Blue Gene/P computer located at IDRIS (PowerPC 450 850 Mhz with 512 MB of memory on each core) using a Fortran 90 implementation with MPI in single precision arithmetic. This code was compiled by the IBM compiler suite with standard compiling options and linked with the vendor BLAS and LAPACK subroutines.

Acoustic full waveform inversion - *Grid* : $433 \times 433 \times 126$									
	$p = 4$			$p = 8$			$p = 16$		
Method	*It*	Pr	T	It	Pr	T	It	Pr	T
FGMRES(5*p*)	56	56	624	112	112	629	224	224	665
BFGMRES(5)	14	56	622	14	112	631	14	224	668
BFGMRESD(5)	14	43	489	15	70	401	15	120	371
BFGMRES-R(5)	16	44	503	16	74	431	16	134	417
DMBR(5)	16	*39*	**452**	16	*57*	**339**	18	*102*	328
BFGMREST(5,p/2)	24	48	542	23	80	447	20	140	410
DMBR(5,p/2)	16	40	459	15	68	392	17	124	384
Combined(5,p/2)	15	41	471	15	62	359	15	103	323
Combined(5,p/4)	18	41	474	15	59	346	15	*102*	**320**
	$p = 32$			$p = 64$			$p = 128$		
Method	*It*	Pr	T	It	Pr	T	It	Pr	T
FGMRES(5*p*)	434	434	670	1152	1152	925	2531	2531	1187
BFGMRES(5)	14	448	713	18	1152	962	19	2432	1187
BFGMRESD(5)	15	225	371	20	490	422	25	1015	509
BFGMRES-R(5)	18	283	466	25	618	537	28	1489	762
DMBR(5)	19	*181*	316	25	413	375	28	915	497
BFGMREST(5,p/2)	20	255	396	25	550	444	28	1125	524
DMBR(5,p/2)	16	189	310	24	444	396	29	976	523
Combined(5,p/2)	15	184	**305**	20	409	348	25	899	**442**
Combined(5,p/4)	20	191	320	20	*398*	**342**	25	*898*	448

Table 3.9: *Acoustic full waveform inversion (SEG/EAGE Overthrust model). Case of $f = 3.64\ Hz$ ($h = 50\ m$), with $p = 4$ to $p = 128$ right-hand sides given at once. It denotes the number of iterations, Pr the number of preconditioner applications on a single vector and T denotes the total computational time in seconds.*

As in [23] we consider the velocity field issued from the public domain SEG/EAGE Overthrust model and analyse the performance of the numerical methods at a given frequency $f = 3.64\ Hz$. Both the

problem dimension (about 23 million of unknowns) and the maximal number of right-hand sides to be considered (128) correspond to a task that geophysicists typically must face on a daily basis. Thus efficient numerical methods must be then developed for that purpose. In [23] we have considered block flexible Krylov subspace methods including deflation at restart only on this application for a reduced number of right-hand sides (from 4 to 16). We continue this detailed analysis and investigate the performance of both DMBR(m) and BFGMRES-R(m) with a larger number of right-hand sides. We also consider the variants with truncation in memory (BFGMREST(m, p_f)) and with truncation in operations DMBR(m, p_f)) with p_f set to $p/2$ in both cases. The number of cores is ranging from 32 for $p = 4$ to 1024 for $p = 128$. Since doubling the number of right-hand sides nearly doubles the memory requirement of the block methods, we also multiply the number of cores by a factor of two with respect to the number of right-hand sides. This aims at imposing the same memory constraint on each core for all numerical experiments as in [23]. The maximal memory requested is about 488 Gb for $p = 128$.

Table 3.9 collects in addition to outer iterations (It) and preconditioner applications on a single vector (Pr) the computational times in seconds (T). Among the different strategies DMBR(5) most often delivers the minimal number of preconditioner applications and computational times (see respectively italic and bold values in Table 3.9). This clearly highlights the interest of performing deflation at each iteration both in terms of preconditioner applications and computational operations on this given application. The latter result is especially important since deflating at each iteration induces an additional cost as shown in Table 3.1. DMBR(5) is thus found to be competitive with respect to methods incorporating deflation at restart only (a gain of up to 15% in terms of computational time is obtained for instance for $p = 8$) as well as DMBR(5,p/2) (gain of 27% for $p = 32$ when compared to BFGMREST(5,p/2)). This is a satisfactory improvement since methods including deflation at restart only are already quite efficient in this application as shown in [23]. We also note that the improvement over classical block flexible GMRES method is quite large as expected (a gain of up to 61% is obtained for $p = 64$).

Figure 3.1 shows the evolution of k_j - the number of Krylov directions effectively considered at iteration j - along convergence for the various block subspace methods in the case of $p = 32$. Regarding BFGMRESD(5) and BFGMREST(5,p/2) deflation is performed only at the beginning of each cycle, thus k_j is found to be constant in a given cycle. Variations at each iteration can only happen in BFGMRES-R(5) or DMBR(5). As expected DMBR(5) enjoys a nonincreasing behaviour for k_j along convergence, while peaks occur for BFGMRES-R(5) at the beginning of each cycle. On this example the use of truncation within DMBR(5) tends to delay the start of the decreasing behaviour of k_j. After a certain phase deflation is nevertheless active and proves to be useful.

We also remark that the use of truncation techniques in DMBR(m) leads to an efficient method. In certain cases DMBR(5, p/2) is as efficient as DMBR(5) in terms of computational times (see, e.g., the case $p = 32$ in Table 3.9). This feature is really important in this given application due to the large size of the linear systems. Furthermore DMBR(5, p/2) requires usually less preconditioner applications than BFGMREST(5, p/2). This satisfactory behaviour has indeed a reason: due to Corollary 3.3.2, we guarantee that the truncated variant of DMBR(m) minimizes the entire residual at each iteration (regardless of the value of p_j), whereas BFGMREST(m) chooses just a subset of the residual to be minimized at each cycle. We consider that this is indeed a critical feature of the truncated variant of DMBR(m).

Finally we investigate the convergence properties of a combination of DMBR(m) and BFGMRESD(m). At the beginning of the convergence history this method (named Combined(m, p_s) in Table 3.9) is fully equivalent to DMBR(m). Then as soon as the number of Krylov directions effectively considered at iteration j (k_j) reaches a given prescribed value (p_s) the method switches to BFGMRESD(m) at the next restart. This mainly aims at reducing the computational cost in the next cycles by performing deflation *only* at the restart instead of at each iteration. As shown in Table 3.9 this combination leads to further reductions in computational times and is especially appropriate when the number of right-hand sides becomes large on this given application.

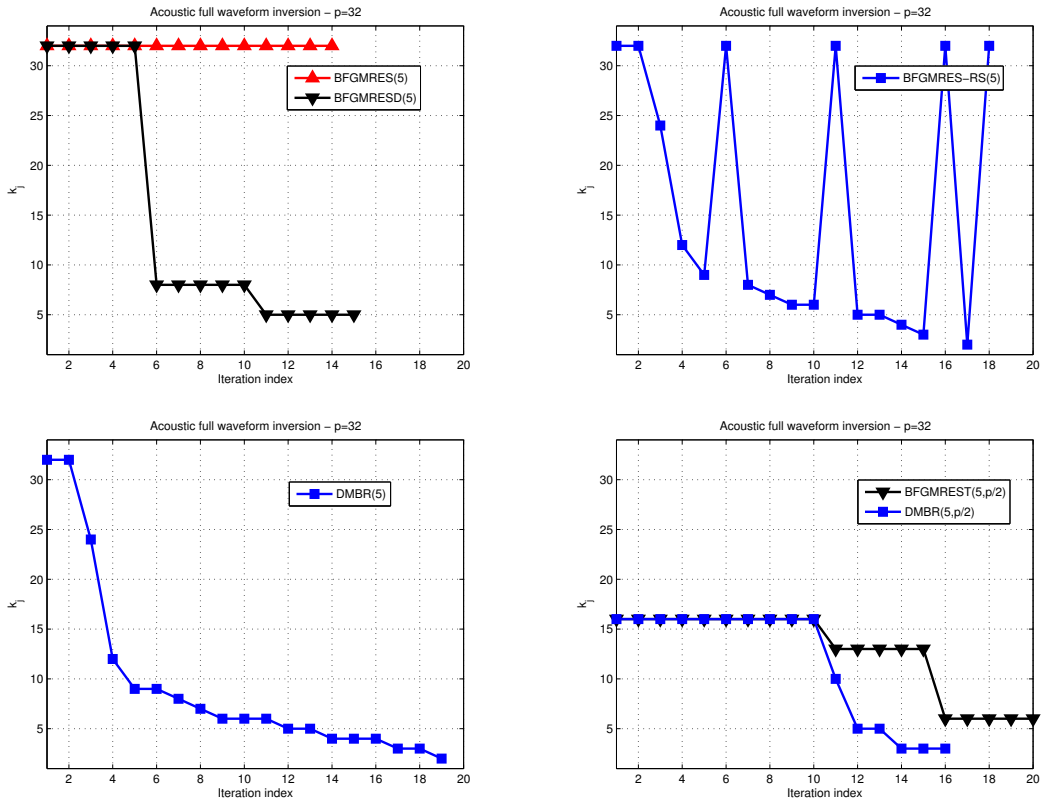


Figure 3.1: *Acoustic full waveform inversion (SEG/EAGE Overthrust model). Case of* $p = 32$. *Evolution of* k_j *versus iterations for* $p = 32$ *in BFGMRES(5), BFGMRESD(5) (top, left part), BFGMRES-R(5) (top, right part), DMBR(5) (bottom, left part) and truncated variants (BFGMREST(5,p/2), DMBR(5,p/2)) (bottom, right part).*

3.10 Conclusions

We have extended the block restarted flexible GMRES method to a variant that allows the use of deflation (i.e. block size reduction) at each iteration when solving multiple right-hand side problems given at once. The method aims at reducing the cost of each iteration of the block Krylov subspace method by judiciously choosing which information to use and which information to be postponed. We have shown that the Frobenius norm of the block residual is always nonincreasing even along cycles. Furthermore we have justified the choice of the deflation strategy that is based on a nonincreasing behaviour of the singular values of the scaled block residual (which are also nonincreasing along the iterations and cycles). We have also proposed a variant of the deflated block residual method to be used in a constrained memory environment. Numerical experiments have shown the efficiency of the new method on two different problems issued from wave propagation situations requiring the solution of multiple right-hand side problems. The block flexible method including deflation at each iteration has proven to be efficient in terms of both preconditioner applications and computational operations. It has been found superior to recent block flexible methods including deflation at restart only. This satisfactory behaviour has been observed on an industrial simulation arising in geophysics, where large indefinite linear systems with multiple right-hand sides have been successfully solved in a parallel distributed memory environment. Furthermore reductions in terms of computational times have been obtained by combining methods including deflation at each iteration and deflation at restart only in a second phase. To the best of our knowledge these results consist in one of the first illustrations of the usefulness of block Krylov subspace methods including deflation at

each iteration on a realistic three-dimensional application in a parallel distributed memory environment.

To conclude it is worthwhile to note that the theoretical properties of the deflated minimal block residual method hold for any unitary matrix $\mathscr{F}_{j+1}$. Thus we plan to investigate other possible subspace decompositions in a near future that may lead to further improvements. Finally we note that the analysis proposed in this paper can be extended as well to other block Krylov subspace methods such as block FOM [102], block GCRO [140], and block simpler GMRES [80].

Chapter 4

Acoustic Full Waveform Inversion

4.1 Introduction

In geology and geophysics, the so called *"Earth imaging"* is a technique used for scanning a delimited subsurface of Earth. The application of Earth imaging techniques is very wide; examples are civil engineering (usually requiring the knowledge of few meters of depth only), landslide analysis of a terrain, or detection and characterization of oil reservoir (often requiring several kilometers of depth) [20, p.15]. Among the strategies for performing such imaging, we highlight here in particular the use of seismic waves. It consists of positioning an acoustic wave propagator (normally called *"source"*) which is able to emit waves at a chosen frequency f through the subsurface. When encountering a layer of reflecting material in the subsurface (as for instance, a salt layer) these waves are refracted and reflected back to the surface. These reflections (sometimes called "echoes") are detected by a special kind of microphone called *"geophone"* (sometimes called simply "receiver"). See Figure 4.1 for an two-dimensional illustration.

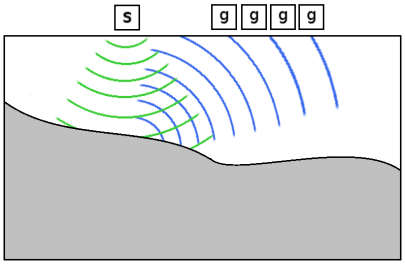


Figure 4.1: *A source for acoustic waves s propagates waves (represented in green) through the subsurface. When meeting a reflective layer (in grey) these waves are reflected back (represented in blue) to the surface, and are detected by the geophones g.*

A "survey line" is chosen on the surface of the Earth and the experiment is repeated with the source positioned in some of the points along the survey line (about every 25 meters, according to [28, p.1]). With the data gathered by the geophones on the several experiments, and knowing the position of the source and the frequency of the acoustic waves emitted, using mathematical methods, we wish then to find details about the subsurface, as density, velocity of the wave propagation, position and shape of reflective layers among others. We highlight here the search for the so called *"velocity model"*, which consists of a two-dimensional or three-dimensional model of the subsurface estimating the velocity at which the wave propagates in each point of the physical domain. See Figure 4.2 for an example of a three-dimensional velocity model.

In this chapter of this thesis, we briefly address a specific mathematical procedure for determining an approximation of the velocity model (it can also be used to approximate other information as density,

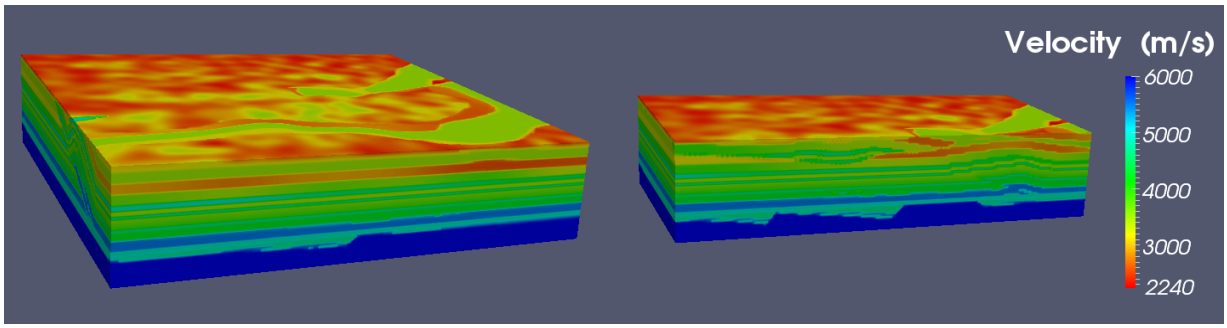


Figure 4.2: *Graphical representation of the three-dimensional academic velocity model SEG/EAGE Overthrust generated using Paraview [69]. The image on the left represents the whole velocity model, and the image on the right represents a cut of the original velocity model showing its interior.*

attenuation or anisotropy of the subsurface), the *"Full Waveform Inversion"* or FWI for short, which consists basically of an inverse problem. The solution of the linear system arising from this problem is the main motivation for the development of a block iterative solver as discussed in Chapter 2 and Chapter 3. In Section 4.2 we address an overview on the inverse problem without attaining to many details. The importance of Section 4.2 is thus to introduce the *forward* and the *backward* problems. Thereafter, we explore details concerning the forward problem, as how to discretize this problem while maintaining certain physical properties (in Section 4.3). In Section 4.4 we address methods for solving the linear problem arising from the discretization of the forward problem, and we address some known issues, as for instance, the difficulties in preconditioning the problem. In Section 4.5 we briefly discuss a software implementation we propose targeting the solution of the full waveform inversion in a massively parallel environment using a object oriented language, and in Section 4.6 we use DMBR, the method proposed in Chapter 3 and implemented into our software, to perform large (up to $\mathcal{O}(10^9)$) numerical experiments on realistic velocity models at $5Hz$ and $12Hz$, illustrating once more the interest of deflation techniques in a real life application. In Section 4.7 we present the final remarks of this chapter.

4.2 The Inverse Problem

In this section we quickly describe the full waveform inversion [20, 120, 121] which is the main motivation of our work with block iterative solvers presented in Chapter 2 and Chapter 3. Although the full waveform inversion consists of an optimization problem it gives raise to the block linear systems which we aim at solving using the methods we discussed in previous chapters.

Let the real velocity model be denoted by $\mathbf{m}^{(*)}$ and consider an initial guess $\mathbf{m}^{(1)}$ given. The inverse problem consists of an iterative procedure which aims at improving a *"synthetic velocity model"* $\mathbf{m}^{(i)}$ every iteration i, hoping to obtain a model $\mathbf{m}^{(\ell)}$ as a reliable approximation of $\mathbf{m}^{(*)}$ in a finite number of steps ℓ.

It consists of two main steps. In the first one, the so called[1] *"forward problem"* takes place. The forward problem consists of building a *"propagator"*, a procedure which simulates the propagation of the waves through $\mathbf{m}^{(i)}$ when an acoustic wave is ignited at a known position with a known frequency, obtaining the so called *wavefield*. We recall that during the geophysical experiment, the acoustic waves are ignited on several points of the physical domain along the "survey line", possibly for several different frequencies, and that the geophones gather the data for each test separately, meaning that potentially the forward problem has to solve multiple times per iteration of the inverse problem. Therefore, we represent the set of *"source positions"* $\{s_j\}_{j=1}^{p}$ and a set of frequencies $\{f_j\}_{j=1}^{n_f}$, meaning a total of $n_f \times p$ solutions have to computed for every iteration i of the inverse problem. We analyse in further sections the multiple sources issue and

[1] sometimes called *"direct problem"*.

how to exploit the relations between each case to increase the performance of the forward problem, but we do not address the multiple frequencies in practice, as we explain better in Section 4.3.

Once each wavefield $\mathbf{u}_j^{(i)}$ for $j = 1, ..., n_f \times p$ of the forward problem of the i-th iteration is known, it has to be *projected* onto the position of the geophones by the projector $\mathcal{P}_{data}$: this is done to know the data that the geophones would have gathered if $\mathbf{m}^{(i)}$ were the real velocity model. The projected synthetic solution, or *"computed data"* is then denoted by $\mathbf{d}_j^{(i)} = \mathcal{P}_{data}\left(\mathbf{u}_j^{(i)}\right)$.

The inverse problem compares the computed data with the *"observed data"*, gathered by the geophones on the real experiment, which we denote here by $\mathbf{d}_j^{obs}$, for $j = 1, ..., n_f \times p$. Figure 4.3 shows an illustration of the difference between the data gathered by the geophones in the real experiment (left) and the simulation performed by the propagator on the synthetic velocity model (right).

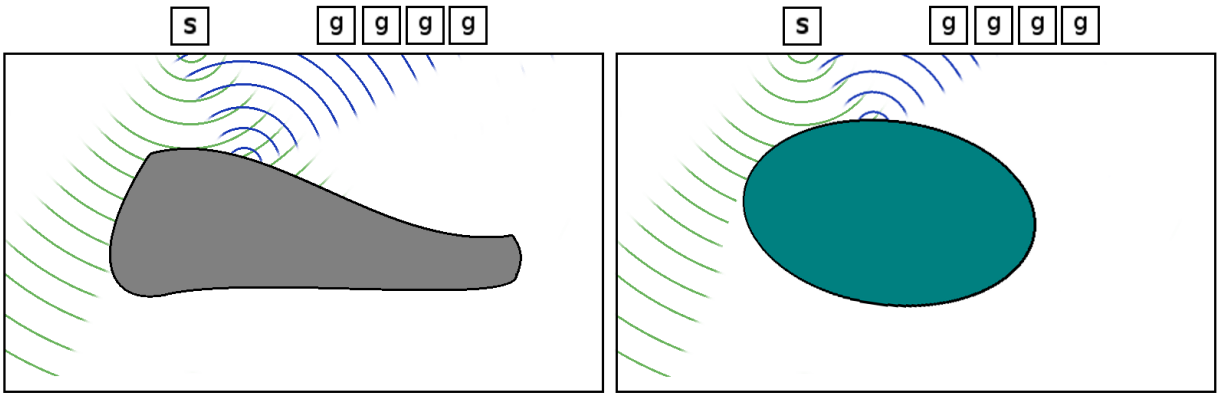


Figure 4.3: *On the left: acoustic waves propagated by a source s are reflected by a reflective layer (in grey) and are detected by the geophones g. On the right: the propagator simulates the behaviour of the waves through the synthetic velocity model. Since the information reaching the geophones is considerably different from the observed data, the inverse problem should decide to update the synthetic velocity model in order to obtain a better approximation.*

We consider that the convergence criterion of the inverse problem relies on the cost function

$$\mathcal{C}(\mathbf{m}^{(i)}) = \frac{1}{2} \sum_{j=1}^{(n_f \times p)} \left(\Delta \mathbf{d}_j^{(i)}\right)^\dagger \mathbf{S}^\dagger \mathbf{S} \Delta \mathbf{d}_j^{(i)} \tag{4.2.1}$$

(cf. [20, p.93]) where $\mathbf{S}$ is a diagonal weighting matrix[2] and $\Delta \mathbf{d}_j^{(i)} = \left|\mathbf{d}_j^{(i)} - \mathbf{d}_j^{obs}\right|$ is the *misfit vector*, or *residual vector*.

The goal of the inverse problem is thus to minimize (4.2.1), characterizing the inverse problem as a weighted least squares problem. However, not only the size of the problem is considerable (the dimension of each $\mathbf{m}^{(i)}$ is between $\mathcal{O}(10^4)$ and $\mathcal{O}(10^6)$ in a realistic application, cf. [20, p.93]) but it is also very ill-conditioned, consequently making prohibitive the use of global methods. Due to this restriction, local methods are often preferred, considering that an initial guess is known. Algorithm 4.2.1 shows a simple pseudo-code for the FWI using a generic steepest descent algorithm for solving this problem.

Remark 4.2.1. Steps 8 to 10 of Algorithm 4.2.1 refer to the choice of the model for the next iteration, and are topic of active research [133]. We briefly mention that some strategies for performing FWI consist of choosing λ models rather than only one, that is, steps 8 to 10 generate $\mathbf{m}_k^{(i+1)}$ for $1 \leq k \leq \lambda$, and during iteration $i + 1$ the algorithm is ran once for each model, being thus able to effectively search in multiple

[2] We do not address the choice of $\mathbf{S}$ in this thesis, and we simply consider it given. Moreover, in many cases a regularization term is also added to the misfit function, but we do not address this situation here.

Algorithm 4.2.1: *Generic acoustic FWI algorithm using steepest descent method*

1 Set iteration counter $i = 1$;
2 Define $n = p \times n_f$;
3 **while** *not converged* **do**
4 Solve **forward problem**: Compute incident wavefields $\mathbf{u}_j^{(i)}$ for $j = 1, ..., n$;
5 Compute misfit vectors $\Delta d_j^{(i)}$ for $j = 1, ..., n$;
6 Compute value of the cost function $\mathcal{C}(\mathbf{m}^{(i)})$ (cf. (4.2.1));
7 Build **gradient vector** $\mathcal{G}^{(i)}$;
8 Choose a perturbation vector $\delta\mathbf{m}^{(i)}$;
9 Choose a step length $\alpha^{(i)}$;
10 Update model $\mathbf{m}^{(i+1)} = \mathbf{m}^{(i)} + \alpha^{(i)}\delta\mathbf{m}^{(i)}$;
11 **end while**

directions every iteration. This however, comes at a great extra cost, since $\lambda \times p \times n_f$ forward problems have to be solved for such a case[3]. ■

It is yet to be mentioned how to perform line 7 of Algorithm 4.2.1. As we discuss in Section 4.3, the forward problem is solved by dully discretizing the wave-equation and then solving the linear system

$$\mathbf{A}^{(i)}\mathbf{u}_j^{(i)} = \mathbf{s}_j^{(i)} \tag{4.2.2}$$

where $\mathbf{s}_j^{(i)}$ represents the position of the source. Considering $\mathbf{A}^{(i)}$ given, we obtain the gradient of $\mathcal{C}(\mathbf{m}^{(i)})$ as

$$\frac{\partial\mathcal{C}(\mathbf{m}^{(i)})}{\partial\mathbf{m}^{(i)}} = \mathcal{G}(\mathbf{m}^{(i)}) = \sum_{j=1}^{n_f\times p} \mathcal{R}\left(\mathbf{u}_j^{(i)^T}\left[\frac{\partial\mathbf{A}^{(i)}}{\partial\mathbf{m}^{(i)}}\right]^T\left(\mathbf{A}^{(i)}\right)^{-T}\tilde{\mathcal{P}}_{data}\mathbf{S}^\dagger\mathbf{S}\overline{\Delta\mathbf{d}_j^{(i)}}\right),$$

where $\tilde{\mathcal{P}}_{data}$ is the operator that projects $\overline{\Delta\mathbf{d}_j^{(i)}}$ onto the forward problem space and $\mathcal{R}(*)$ is the real part of the referred vector (cf. [20, (2.11)]). We highlight here the term

$$\left(\mathbf{A}^{(i)}\right)^{-T}\tilde{\mathcal{P}}_{data}\mathbf{S}^\dagger\mathbf{S}\overline{\Delta\mathbf{d}_j^{(i)}}$$

which is referred to as the *back-propagation* of the misfit vector. Computing such a vector is equivalent to find the solution of the linear system

$$\left(\mathbf{A}^{(i)}\right)^{T}\mathbf{g}^{(i)} = \tilde{\mathcal{P}}_{data}\mathbf{S}^\dagger\mathbf{S}\overline{\Delta\mathbf{d}_j^{(i)}} \tag{4.2.3}$$

the so called *backward problem*. Notice the relation between (4.2.2) and (4.2.3), and that the solution for the later depends on the solution of the former (the solution of the forward problem is needed to compute the right-hand side of the backward problem). When the operator $\mathbf{A}^{(i)}$ is symmetric, the only difference between the backward and forward problem lies in the right-hand side.

If $(\mathbf{A}^{(i)})^{-1}$ would be known, the solution of the backward problem would take a matrix-vector multiplication operation, otherwise we would have to effectively solve the backward problem once for each frequency and each source position. When using iterative solvers, $(\mathbf{A}^{(i)})^{-1}$ is not known and therefore it is extremely important to address the solution of the backward problem. This is subject of future studies.

[3] as we mention shortly, the forward problem, along with the backward problem (whenever it is present) are the most expensive part of Algorithm 4.2.1.

4.3 Discretizing the Forward Problem

In this section we focus on the solution of the forward problem introduced in Section 4.2, and we discuss a proper formulation of this problem as well as some techniques for discretizing and some few computational issues arising when we are formulating the problem to find a numerical solution. Although this is not explicitly mentioned in depth in this section, we always have in mind the formulation for massively parallel environment, as we discuss in more detail in Section 4.4 and Section 4.5.

4.3.1 The Helmholtz Equation

We are interested in determining how the acoustic waves would propagate through the velocity model $\mathbf{m}^{(i)}$ inside the bounded parallelepiped domain $\Omega \subset \mathbb{R}^3$. Two main approaches are traditionally chosen for finding the solution for such a problem: to consider the problem in the time-domain or in the frequency-domain. Both approaches have their advantages and drawbacks which we do not discuss here in details. We refer to [31, 30, 84, 57, 126, 125] for details on the time-domain approach and to [116, 98, 99] for details on the frequency-domain approach. Since it is recognized in the literature that the frequency-domain approach is more advantageous when solving the inverse problem, in this thesis we address this approach only.

We use the discussion in [49, §1.2] to deduce the equation suitable for representing the wave propagation through the subsurface in the frequency-domain. We temporarily drop the j index from $u_j(\boldsymbol{x})$ since the discussion here is valid for every j. Consider the time-dependent wave-equation for fluids and solids in the absence of viscosity

$$\Delta p = \frac{1}{v^2}\frac{\partial^2 p}{\partial t^2} \tag{4.3.1}$$

where $p(\boldsymbol{x}, t)$ is the pressure on the point $\boldsymbol{x} \in \mathbb{R}^3$ at the instant $t \in \mathbb{R}$ and $v(\boldsymbol{x})$ is the propagation speed of compressional waves in the point $\boldsymbol{x}$. However, we consider that the waves are time-harmonic and that they can be represented as

$$p(\boldsymbol{x}, t) = u(\boldsymbol{x}) exp(-\hat{\jmath}\omega_w t), \tag{4.3.2}$$

(cf. [49, (1.22)]) where $\omega_w = 2\pi f$ is the angular frequency, $f \in \mathbb{R}$ is the frequency in Hertz, and $\hat{\jmath} = \sqrt{-1}$. Substituting (4.3.2) into (4.3.1) we then obtain

$$-\Delta u(\boldsymbol{x}) - k^2(\boldsymbol{x})u(\boldsymbol{x}) = 0 \tag{4.3.3}$$

where $k(\boldsymbol{x}) = 2\pi f / v(\boldsymbol{x})$ is called the *wavenumber* function and Δ denotes the Laplacian operator. Introducing the source term $g(\boldsymbol{x})$ and also assuming that such source term is time-harmonic, we obtain the *heterogeneous Helmholtz equation* as follows

$$-\Delta u(\boldsymbol{x}) - k^2(\boldsymbol{x})u(\boldsymbol{x}) = s(\boldsymbol{x}) \tag{4.3.4}$$

which is in turn time-independent.

Recently it was proposed in [66] a reformulation of the Helmholtz equation using the Rytov decomposition. This reformulation requires the solution of the complex advection-diffusion-reaction equation, which in turn can be solved using efficient multigrid preconditioners. The authors report their approach to be efficient in the two dimensional case. Although these results encourage further research on three dimensional cases, we do not address this situation in this thesis.

4.3.2 Perfectly Matched Layers

Because the domain Ω is supposed to be bounded whereas the geophysical domain comprises the whole Earth, in order to simulate properly the wave propagation phenomena one must add an *absorbing boundary*

condition to the equation, as for instance the *perfectly matched layers* (PML) [13, 14]. It consists of adding an artificial layer around Ω and modifying the wave-equation only inside these additional layers. For doing that, we consider solving the problem in a larger parallelepiped domain Ω_e such that $\Omega \subset \Omega_e$. Denoting the boundary of Ω_e by Γ, and $\Omega_{\text{pml}} = \Omega_e \backslash \Omega$ we then redefine the wave-equation in Ω_{pml} using *damping functions* such that the value of the pressure of the waves on Γ are always equal to zero (cf. Figure 4.4 for a two-dimensional graphical representation).

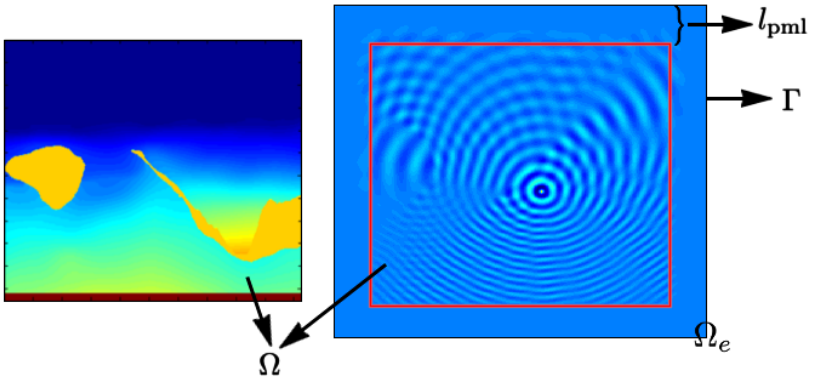


Figure 4.4: *A graphical representation of the PML in two-dimensions. Left: a slice of the SEG/EAGE Salt dome velocity model, defining Ω. Right: the wave propagation on the extended domain Ω_e containing the PML, with Ω being represented as the red box.*

Letting the vector $\boldsymbol{x} = (x^1, x^2, x^3)$, we use the same damping functions as [93]:

$$\xi_l(x^l) = \begin{cases} 1 - \hat{\jmath} \cos\left(\dfrac{\pi x^l}{2 l_{\text{pml}}}\right) & \text{if } 0 \leq x^l \leq l_{\text{pml}}, \\ 1 & \text{if } l_{\text{pml}} < x^l < l_l - l_{\text{pml}}, \\ 1 - \hat{\jmath} \cos\left(\dfrac{\pi (l_l - x^l)}{2 l_{\text{pml}}}\right) & \text{if } l_l - l_{\text{pml}} \leq x^l \leq l_l. \end{cases} \tag{4.3.5}$$

for $l \in [\![1, 3]\!]$, where $l_l \in \mathbb{R}$ is the length of Ω in the x^l direction and l_{pml} denotes the length of the PML layer (which we consider to be uniform in every direction; cf. Figure 4.4).

Using Einstein's notation, we then split (4.3.4) as

$$\begin{cases} \left[-\dfrac{\partial^2}{\partial (x^l)^2} - k(\boldsymbol{x})^2\right] u(\boldsymbol{x}) = s(\boldsymbol{x}) & \text{in } \Omega, \\ \left[-\dfrac{1}{\xi_l(x^l)} \dfrac{\partial}{\partial x^l} \dfrac{1}{\xi_l(x^l)} \dfrac{\partial}{\partial x^l} - k(\boldsymbol{x})^2\right] u(\boldsymbol{x}) = s(\boldsymbol{x}) & \text{in } \Omega_{\text{pml}} \backslash \Gamma, \\ u(\boldsymbol{x}) = 0 & \text{on } \Gamma \end{cases} \tag{4.3.6}$$

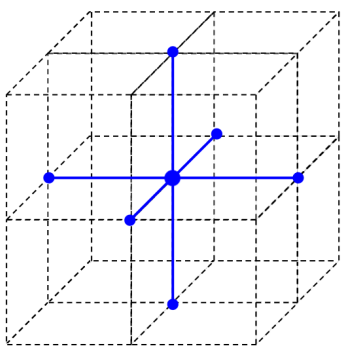
(cf. [96, p.103]). The forward problem consists in finding a solution for the problem (4.3.6) in Ω_e. Although the homogeneous form of the Helmholtz equation (4.3.3) possesses analytical solutions, this is not the case for the heterogeneous equation (4.3.4), and thus a numerical approximation must be computed.

4.3.3 Discrete Formulation

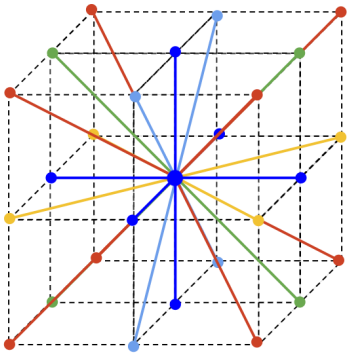
In the current literature there are several approaches for discretizing (4.3.6) and computing a numerical approximation. Techniques as finite elements and discontinuous Galerkin have been widely applied in the

literature. However, in this thesis we opt for a simplified approach, limiting our study to the uniform finite difference techniques, using either 7 or 27 points in the Cartesian grid (cf. Figure 4.5) which are cheap to generate and easily parallelizable.

Figure 4.5: *Graphical representation of three-dimensional uniform finite difference Cartesian stencils.*



7-point Cartesian stencil



27-point Cartesian stencil

Consider a uniform Cartesian grid Ω_h as the discretization of Ω_e for a given distance $h \in \mathbb{R}^+$ between each point such that $l_l/h = n_l \in \mathbb{N}$ is the number of points in the direction x^l. Then we define the discrete function

$$u_{(i,j,k)} = u_h(ih, jh, kh), \quad \text{where} \quad u_h = u\Big|_{\Omega_h}$$
$$\text{with} \quad i \in [\![1, n_1]\!], \quad j \in [\![1, n_2]\!] \quad \text{and} \quad k \in [\![1, n_3]\!]$$

and analogously for $s_{(i,j,k)}$, $v_{(i,j,k)}$ and $\xi_{(l,i)}$. Traditionally, h is chosen as to satisfy the minimal *number of points per wavelength* $n_\lambda \in \mathbb{R}^+$, a value which is scheme-dependent. Knowing that a wavelength is defined as the ratio between the velocity of the propagation of the wave and the frequency f, we find that the distance between each points must satisfy

$$h \leq \frac{v_{(i,j,k)}}{n_\lambda f}, \qquad \forall (i,j,k) \in \Omega_h \tag{4.3.7}$$

where it is normally chosen

$$h = \frac{\min\limits_{(i,j,k)\in\Omega_h} v_{(i,j,k)}}{n_\lambda f}. \tag{4.3.8}$$

Inequality (4.3.7) is often referred in the literature as *stability condition* [29] for a second order discretization scheme.

Using this notation, and the second order Taylor expansion, the discrete formulation of (4.3.6) for the

7 point Cartesian stencil can be written as

$$\begin{aligned} s_{(i,j,k)} = & \left[\frac{-\omega^2}{v^2_{(i,j,k)}} + \frac{1}{h^2} \left(\frac{1}{\xi^+_{(1,i)}} + \frac{1}{\xi^-_{(1,i)}} \right.\right. \\ & \left.\left. + \frac{1}{\xi^+_{(2,j)}} + \frac{1}{\xi^-_{(2,j)}} + \frac{1}{\xi^+_{(3,k)}} + \frac{1}{\xi^-_{(3,k)}} \right) \right] u_{(i,j,k)} \\ & - \frac{1}{h^2} \frac{1}{\xi^+_{(1,i)}} u_{(i+1,j,k)} - \frac{1}{h^2} \frac{1}{\xi^-_{(1,i)}} u_{(i-1,j,k)} \\ & - \frac{1}{h^2} \frac{1}{\xi^+_{(2,j)}} u_{(i,j+1,k)} - \frac{1}{h^2} \frac{1}{\xi^-_{(2,j)}} u_{(i,j-1,k)} \\ & - \frac{1}{h^2} \frac{1}{\xi^+_{(3,k)}} u_{(i,j,k+1)} - \frac{1}{h^2} \frac{1}{\xi^-_{(3,k)}} u_{(i,j,k-1)}. \end{aligned} \tag{4.3.9}$$

(cf. [96, p.106] for the three-dimensional formulation or [73] for a two-dimensional formulation) where

$$\begin{aligned} \xi^+_{(l,i)} &= \frac{1}{2} \xi_{(l,i)} \left(\xi_{(l,i)} + \xi_{(l,i+1)} \right) \\ \xi^-_{(l,i)} &= \frac{1}{2} \xi_{(l,i)} \left(\xi_{(l,i)} + \xi_{(l,i-1)} \right) \end{aligned}$$

for $l \in [\![1, 3]\!]$. Defining $n = n_1 \times n_2 \times n_3$ and letting $u \in \mathbb{C}^n$ (respectively $s \in \mathbb{C}^n$) denote the vectorization of $u_{(i,j,k)}$ (respectively $s_{(i,j,k)}$) in lexicographical ordering, we can write (4.3.9) as a linear system

$$Au = s$$

where $A \in \mathbb{C}^{n \times n}$ is a matrix containing the coefficients of the unknowns u whose structure is depicted in Figure 4.7. The resulting matrix A is sparse (has seven diagonals of nonzeros only), in general it is non-Hermitian, and because of the PML formulation (4.3.6) it is also non-symmetric and ill-conditioned depending on $k(\boldsymbol{x})$. Recalling from Section 4.2 that we have p source positions and for a fixed frequency f, we then obtain that the problem to be solved is

$$AU = S$$

where $U, S \in \mathbb{C}^{n \times p}$. The reason why we cannot consider multiple frequencies is that h depends on the frequency f (cf. (4.3.7)), and thus the dimension of the problem n changes for every frequency. It is beyond the scope of this manuscript to address the multiple frequencies issue, but we refer to [135, 136] for recent developments done in order to solve the Helmholtz equation for multiple frequencies, and we limit ourselves to the multi sources scenario.

We mention now some computational advantages of the discrete formulation (4.3.9). Because it is a 7-point stencil, we have to store at most seven nonzeros per row, that is, the storage cost for the discretized operator A is bounded by $7n$. Moreover, the only term dependent on $v_{(i,j,k)}$ is the central term $u_{(i,j,k)}$, meaning that in practice only these central points need to be stored in and all the other values can be computed whenever they are needed, making this scheme even cheaper. Nevertheless in our implementation we opt for storing part of the off diagonal for the sake of avoiding floating point operations. Noticing that each $\xi^+_{(l_i)}$ (or $\xi^+_{(l_i)}$) is in fact unidimensional, we store the values of each these functions along a line, totalizing $n + 2(n_1 + n_2 + n_3)$ storage for this implementation.

One of the main disadvantages of this scheme lies in the fact that it may not be stable unless we choose a particularly large value for n_λ to avoid dispersion errors in the solution, due to the fact that it is a second order scheme. In the literature we usually select $10 \leq n_\lambda \leq 12$, thus resulting in a small h and consequently a large problem size n.

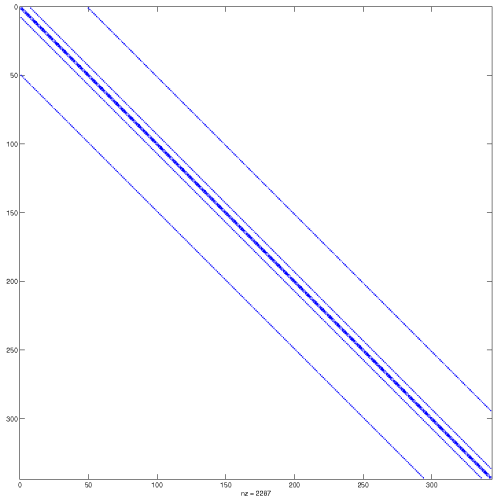


Figure 4.7: *Pattern of the coefficient matrix arising from the discrete formulation of the Helmholtz equation with PML in a Cartesian uniform grid with a 7-point stencil* (4.3.9)

4.3.4 Advanced Discretization Schemes

We briefly mention now alternative schemes for discretizing (4.3.6) which are available in the literature.

In [68] several finite difference discretization schemes are proposed for the discretization of (4.3.4) (instead of (4.3.6)), usually aiming at high accuracy, small dispersion error and anisotropy. The authors present fourth-order accurate schemes on uniform grids. A sixth-order accurate finite difference scheme is proposed in [115]. We refer to [62] for recent numerical experiments with these high-order schemes using the CARP-CG [61] iterative solver.

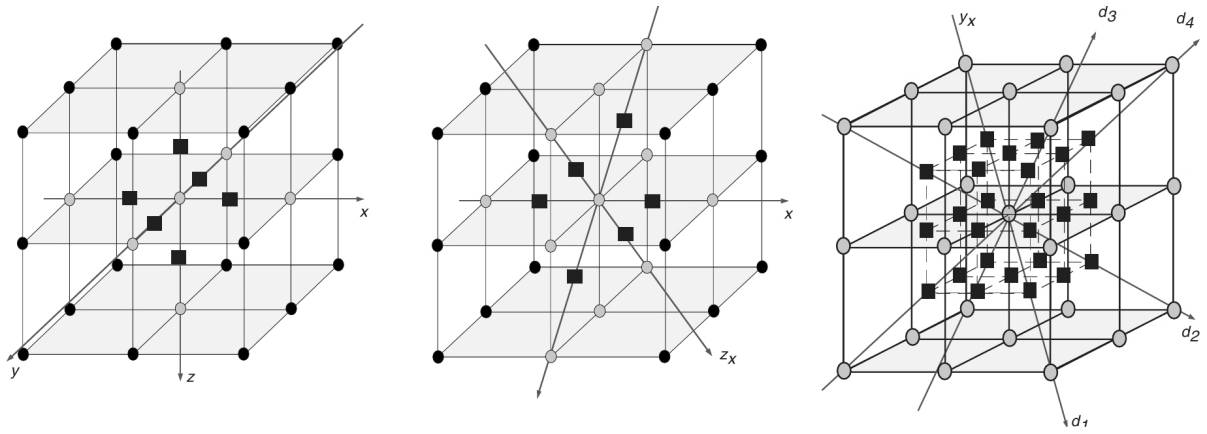


Figure 4.8: *Image taken from [93]: combination of several 7-point stencil resulting in a 27-point parsimonious staggered-grid scheme.*

In [93] a scheme consisting of a weighted combination of several rotations of 7-point stencils resulting in a 27-point scheme (cf. Figure 4.8) has been proposed. The weights are chosen such that the dispersion error is minimized for small values of n_λ. The numerical experiments in [93] show that the scheme is considered stable for $n_\lambda = 4$ in the sense that the dispersion error is still in the acceptable threshold.

This means that the discretized operator A when using this scheme is *compact*: twenty seven diagonals are stored, but the dimension n can be considerably smaller. Although in the presentation the weights are chosen to minimize the dispersion error, a priori one could choose any weighting resulting in different properties of the resulting scheme. Notice that because this scheme is formulated as a combination of second order schemes, its truncation error is also of second order. We refer to [93, 120, 121] for more numerical experiments using this scheme in two and three dimensions.

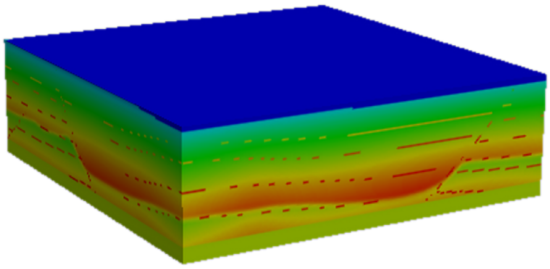
4.4 Preconditioning the Helmholtz Equation

In the current literature the difficulty in solving the Helmholtz problem is often associated with the wavenumber, which we previously defined as

$$k(\boldsymbol{x}) = \frac{2\pi f}{v(\boldsymbol{x})}$$

or with its maximal value $k_{max} = \max_{\boldsymbol{x}\in\Omega} k(\boldsymbol{x})$. In view of (4.3.8), we find out that k_{max} is inversely proportional h, that is, that larger the k_{max}, the larger must be number of points in the discretized domain n, which for high frequencies can reach the value of $\mathcal{O}(10^9)$ (see Figure 4.9 for an example with the SEG/EAGE Salt dome velocity model). Not surprisingly, the memory cost for solving the discretized problem arising from the discretization of the Helmholtz equation (4.3.4) or (4.3.6) is often the main bottleneck in this application. In [93] a sparse multifrontal direct method [4] is employed for both two-dimensional and three-dimensional formulations, the largest problem solved being a cut of the SEG/EAGE Overthrust velocity model (cf. Figure 4.2) containing $409 \times 109 \times 102$ points, for $10Hz$ using the 27-point parsimonious staggered-grid scheme mentioned in Subsection 4.3.4 (cf. Figure 4.8) with $n_\lambda = 4$ and requiring 450 GB of memory. In [18] the same full SEG/EAGE Overthrust velocity model is used, this time with multilevel LDL^T factorization preconditioner. The problem size is $409\times409\times102$ with $f = 5Hz$ only, requiring 32 GB of memory. Recent work done in [47] proposes a preconditioner based on LDL^T factorization preconditioner, which aims to eliminate the unknowns layer by layer (in 2D) or face by face (in 3D) starting from the boundaries of the domain. This preconditioner is therein called *"sweeping preconditioner"*. This work is later extended in [97] to allow parallelism, being thus able to solve for the SEG/EAGE Overthrust velocity model at $8Hz$, with $801 \times 801 \times 187$ grid, using 24 Gb of memory. It was investigated in [67] the behaviour of the algebraic additive Schwarz preconditioner when solving the Helmholtz equation for the SEG/EAGE Overthrust velocity model containing $277 \times 277 \times 73$ points, for $f = 7Hz$, requiring 150 GB of memory. In general, using a standard sparse direct solver method for solving the Helmholtz problem requires an amount of memory of the order of $\mathcal{O}(n^2 \log n)$ (cf. [121, (1)]), which can be prohibitively large for high k_{max}. Recent publications [137, 138] attempt to reduce the memory of direct solvers for Helmholtz by using low-rank approximation of sub-block of the coefficients matrix, a method therein called *parallel Hierarchically Semi-Separable (HSS) matrix compression*. The memory requirement is reported to be almost linear, between $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$, and the method is used to successfully solve realistic problems, as the SEAM velocity model with grid size $401 \times 401 \times 201$ for up to $20Hz$, using 2TB memory. More numerical experiments can be found in [3] and an extension for use with elastic waves in [139].

Since the memory seems to be the main bottleneck when solving the Helmholtz equation for standard sparse direct solvers, the use of iterative solvers becomes a feasible alternative. Krylov iterative solvers are recognized for their good scalability in massively parallel environment which is also one of the main interests when solving the Helmholtz problem due to its large dimension. Moreover, the fact that the discretized Helmholtz operator can be very sparse (e.g. when using 7-point schemes) also encourages the use of Krylov iterative solvers. However, k_{max} is also associated with difficulties in finding Krylov directions to improve the approximate solution when using GMRES method without any preconditioner, culminating in a slow convergence (cf. [52, §2.1]). When using incomplete LU factorizations [109, Chapter 10] as preconditioner, iterative solvers as GMRES [108], BiCGStab [131] and QMR [55] also show a decrease in the convergence behaviour as k_{max} grows [56]. Domain decomposition ([118] and specially



$$\text{Minimal velocity: } 1,500m/s$$
$$\text{Geophysical domain size: } 20 \times 20 \times 5km^3$$
$$n_\lambda = 12$$
$$h = \frac{1,500}{12f} = \frac{125}{f}$$
$$n = 160f \times 160f \times 40f$$
$$= 1.024 \times f^3 \times 10^6$$

Figure 4.9: *The academic SEG/EAGE Salt dome velocity model and the value of h related to f, considering that $n_\lambda = 12$. Notice that if $f = 10Hz$, $n = \mathcal{O}(10^9)$. The grid size described here does not take the PML layer into account.*

[127, §11.5.2]) techniques used as a preconditioner for GMRES are also reported to be dependent on k_{max} (cf. [52, §2.3]). The study of a preconditioner which is efficient for the Helmholtz equation (and hopefully independent of k_{max}) is subject of active research.

We highlight here the effort on using geometric multigrid techniques [83, 128] (cf. Figure 4.10) for solving the Helmholtz equation. Multigrid techniques are known to be specially efficient when the problem being solved is symmetric and positive-definite, in which case the eigenvalues are located in the positive real quadrant of the complex plane. However, the eigenvalues of the (unpreconditioned) Helmholtz operator with the PML formulation are spread along the complex plane and depending on k_{max} we may have eigenvalues clustered around the origin, characterizing a very ill-conditioned problem.

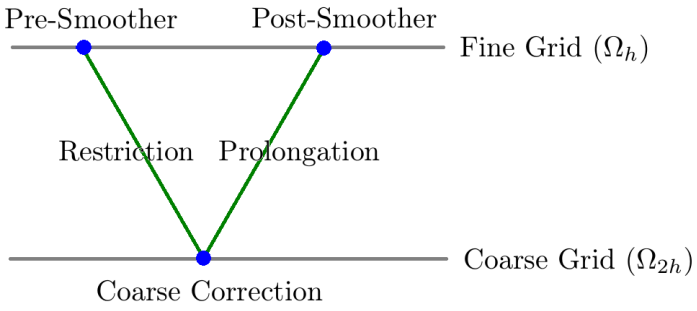


Figure 4.10: *Graphical representation of a basic V-cycle geometrical multigrid. Other advanced multigrid schemes can be deduced from the basic V-cycle.*

Another concern of the geometric multigrid approach is that on the coarsest level, the effective n_λ might be below the suggested threshold. For instance, using the 7-point stencil the advised n_λ is 12, whereas in the scheme presented in Figure 4.10, the coarsest grid will effectively be discretized with 6 points per wavelength. This means that the coarse grid correction might have no physical significance, as is explained in [52, §3.3] (cf. also Figure 9 and Figure 10 in the same publication). The dispersion and phase errors in the coarse level may be enough to invalidate the coarse grid correction, which can in some cases deteriorate the solution instead of improving.

In [46] it was proposed the use of Krylov subspace methods as FGMRES either as a solver preconditioned by geometric multigrid or as a smoother on intermediate grids. Similar numerical experiments for two-dimensional problems can be found in [39], where sparse multifrontal direct methods are used to solve the coarsest level problem. The authors of [46] report that Krylov iterative solvers are suitable as smoother on the intermediate levels, and that the convergence seems to be independent on the mesh size h, but still dependent on k_{max}. Most notably, in [46] the authors report that for some values of k_{max} the coarse grid correction may be significant even when the discretization is too coarse to have physical significance.

Other publications on the subject of geometric multigrid for Helmholtz problem are [49, 51, 77] and on algebraic multigrid [18].

In [96] it was then proposed the use of a Krylov subspace not only as a smoother but also at the coarse level of two-level multigrid algorithm. The main idea behind this proposal is that the coarse grid correction does not require high accuracy and an approximate solution suffices. This argument is then supported by rigorous Fourier analysis on two-level multigrid preconditioner therein called *perturbed geometric two-level preconditioner*. The author shows that the spectrum of the Helmholtz operator preconditioned by a two-level multigrid with an exact coarse correction is not improved when compared with the perturbed preconditioner with a very loose approximation on the coarse level (cf. [96, §3.4]). We thus conclude that the solver does not benefit from highly accurate coarse corrections. This assumption is also supported by numerical experiments where FGMRES method is used with the two-level perturbed preconditioner, where using a coarse correction relative tolerance of mere 0.6 results in faster convergence than using 10^{-12} (cf. [96, p.77], Table 3.7). We address this preconditioner in more details in Subsection 4.4.1.

Another development on preconditioning the Helmholtz operator is the use of the so called *complex shifted Laplacian* operator [49, 51]. It consists in using

$$-\Delta u(\boldsymbol{x}) - (1 - i\beta)k^2(\boldsymbol{x})u(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{where} \quad \beta \in \mathbb{R}$$

instead of (4.3.4) for the multigrid hierarchy levels. The goal of this approach is to obtain an operator which presents a better convergence behaviour when using multigrid method. The complex shifted Laplacian operator has been ever since largely used as a preconditioner for Krylov iterative solvers as GMRES or BiCGStab when solving the Helmholtz equation (cf. [52, p.32] Figure 16 for an analysis of the spectrum of the Helmholtz operator preconditioned by the complex shifted Laplacian operator). Numerical experiments in [51] show the robustness of the method when applied to realistic geophysical applications at high wavenumbers. However, in [50, 52] it is shown that the efficiency of the complex shifted Laplacian preconditioner is still linearly dependent on k_{max}. We refer to [18, 24, 49, 50, 51, 100, 101, 130] for numerical experiments with the complex shifted Laplacian using low and medium frequency ranges only and more information on how to choose the shift parameter β.

Although the use of complex shifted Laplacian preconditioners is normally advised specially when using multigrid techniques for low or mid frequency cases (about $10Hz$), the efficiency of this preconditioner may decrease for high frequencies. In the recent publication [24] it was shown that using optimal parameters deduced from Fourier analysis, the perturbed two-level preconditioner proposed in [96] shows an increased robustness on heterogeneous problems when compared to the standard approach based on complex shifted Laplacian operator. The performance is further improved when the two-level preconditioner uses a complex shifted Laplacian operator as a preconditioner on the coarse level (cf. [24, Table IV]). The numerical experiments in [24] confirm the efficiency of their approach when solving for high frequencies.

In this thesis we limit ourselves to the use of the two-level perturbed multigrid proposed in [96], which we reproduce with more details in Subsection 4.4.1. We focus on the behaviour of the overall method when solving the Helmholtz equation (4.3.6) in the multi sources scenario, using deflation every iteration. We discuss in depth the issues arising from this particular scenario in the next section, and we refer to [24] for a recent detailed description of the perturbed preconditioner when using the complex shifted Laplacian operator as preconditioner.

4.4.1 The Perturbed Geometric Two-Level Preconditioner

In this subsection we discuss the perturbed geometric two-level preconditioner [22, 23, 96], briefly mentioned in Section 4.4. Since this is the main preconditioner we will use in our further numerical experiments (as it is the preconditioner used in the experiments in Section 3.9) we provide a more detailed explanation in this section.

Recalling the discussion in Section 4.3, A is the matrix issued from the discretization of the operator (4.3.6) on Ω_h. We then define the *coarse grid* Ω_H as the standard geometric coarsening of Ω_h in all

directions. Without loss of generality, we suppose here that h was chosen such[4] that $n/8 =: N \in \mathbb{N}$. The discretization of (4.3.6) in Ω_H is thus obtained using the same discretization scheme of choice for Ω_h, resulting in the operator $A_H \in \mathbb{C}^{N\times N}$.

We also define the restriction and the interpolation operators

$$\mathcal{R} : \mathbb{C}^n \to \mathbb{C}^N$$
$$\mathcal{I} : \mathbb{C}^N \to \mathbb{C}^n.$$

In all our numerical experiments we use $\mathcal{I}(.)$ as the trilinear interpolation, and its adjoint as the restriction operator $\mathcal{R}(.)$.

The generic perturbed two-level preconditioner [96] for single right-hand side is depicted in Algorithm 4.4.1. Unless otherwise specified, whenever we apply Algorithm 4.4.1 we use the following parameters

$$\mu = 1, \qquad m_h = 2, \qquad \mu_H = 10, \qquad \textit{and} \qquad m_H = 10. \tag{4.4.1}$$

Algorithm 4.4.1: *Perturbed geometric two-level cycle to obtain the approximation $Z_j \in \mathbb{C}^{n\times k_j}$ to the system $AZ = V_j$ for a given fixed right-hand side $V_j \in \mathbb{C}^{n\times k_j}$ with the zero initial guess*

1. **Polynomial presmoothing:** apply μ cycle(s) of FGMRES(m_h) to $AZ = V_j$ with initial approximation $0_{n\times k_j}$ and symmetric Gauss-Seidel as a right preconditioner to obtain the approximation Z_h^μ;
2. **Restrict the fine level residual:** $R_H = \mathcal{R}(V_j - AZ_h^\mu)$;
3. **Solve approximately the coarse problem** $A_H Z_H = R_H$**:** Apply μ_H cycles of FGMRES(m_H) to $A_H Z_H = R_H$ with initial approximation $0_{N\times k_j}$ and symmetric Gauss-Seidel as a right preconditioner to obtain the approximation $\tilde{Z}_H$;
4. **Correct the fine grid approximation:** $\tilde{Z}_h = Z_h^\mu + \mathcal{I}(\tilde{Z}_H)$;
5. **Polynomial postmoothing:** Apply μ cycle(s) of FGMRES(m_h) to $AZ = V_j$ with initial approximation $\tilde{Z}_h$ and symmetric Gauss-Seidel as a right preconditioner to obtain the approximation Z_h;
6. **Define** $Z_j := Z_h$

We highlight that the application of Algorithm 4.4.1 as a preconditioner characterizes a *variable preconditioner*. Writing $Z_j = \mathcal{M}(V_j)$ where the nonlinear function $\mathcal{M}(.)$ represents the action Algorithm 4.4.1 on V_j, we see that the use of the perturbed two-level preconditioner implies the use of a flexible outer method.

4.5 Software Implementation

In view of the whole discussion in this chapter, it is clear that many aspects of the full waveform inversion method require future research. Indeed so many different techniques are being applied on different levels of the problem that choosing the best strategy and staying up to date with the most recent advancements is a challenge. Also, due to the vast diversities of knowledge that this problem requires, it would be challenging to write a monolitical software to be used by geophysicists as well as computer scientists working with low level linear algebra optimization. For this reason it is needed a malleable and modular software. We thus discuss now some basic aspects of the software implementation we proposed for solving large sparse linear systems (and thus, being able to solve the forward problem) in a massively parallel environment, using FORTRAN03 object orientation techniques.

[4] or $n/4 =: N \in \mathbb{N}$ for the two-dimensional case.

In scientific computing community, FORTRAN77 and FORTRAN90 stand as two of the most efficient and popular programming languages. However, both of these languages lack of object orientation tools, as polymorphism and inheritance, which are essential building blocks for modular codes. Since our original proposal was to build a modular software able to be easily changed and adapted to exploit the new techniques being developed in the literature, we opted for using a language which would instead present at least polymorphism and inheritance features.

Among the candidate languages, we highlight here the FORTRAN03, a standard for the FORTRAN compiler published in 2004, but not yet completely implemented as a functional compiler. Instead, specific compilers implementing a limited set of FORTRAN03 standards have been developed, as ifort (Intel FORTRAN Compiler), gfortran (GNU FORTRAN compiler), pgi (The Portland Group, Inc.), among others. Nevertheless, the basic object orientation features are supported by most of these compilers.

The choice of FORTRAN03 over other object oriented languages makes sense when considering a software using only the most basic object orientation concepts, specially since we are targeting a community that is already used to FORTRAN77 and FORTRAN90 softwares, making the transition between procedural programming to object oriented programming more smooth and comfortable, in contrast with other languages which could require a deep knowledge on object orientation and software engineering.

We describe basically the advantage of using polymorphism and inheritance in Figure 4.11. Just as an example, consider we have a type *C_Solver* and that the objects *C_BFGMRES*, *C_BFGMRESD*, and *C_DMBR* which *inherit* from *C_Solver*. But more than that, the object *C_FBGMRES* requires another object of the type *C_Solver* to be used as preconditioner. This means that any of the objects *C_BFGMRESD*, and *C_DMBR* could be used as preconditioner, including *C_BFGMRES* itself. Also, if in any moment it is implemented an object of the type *C_BiCGStab* who inherits from *C_Solver*, then *C_BiCGStab* could be used as a preconditioner for *C_BFGMRES* without any modification in the code.

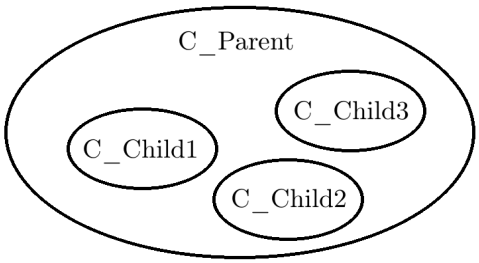


Figure 4.11: *Basic representation of polymorphism in inheritance.* **Inheritance** *guarantees that every object of the type C_Child1, C_Child2 and C_Child3 contains every component (e.g. variables, pointers, etc) that is contained in C_Parent.* **Polymorphism** *guarantees that every object of the type C_Child1, C_Child2 and C_Child3 can be treated as a C_Parent type also. For instance, a subroutine or function which requires an object of the type C_Parent as an argument, could receive also a C_Child1 or C_Child3 object instead without requiring any kind of modification in the code.*

We do not attain to details of the code in this subsection, but we refer to Appendix A for a detailed user guide of the libraries we implemented using FORTRAN03 and object orientation. We highlight that we compared the code written in FORTRAN03 with the prototypes implemented in FORTRAN90 used in the numerical experiments in Subsection 3.9.3 and Subsection 3.9.4. The FORTRAN03 version showed a marginal *speedup* of 1% to 3.5% with respect to the prototype code, thus guaranteeing that the addition of the object orientation did not bring any slow down to our code.

4.6 Numerical Experiments

This section is dedicated to numerical experiments related to acoustic full waveform inversion method problem, using the software we developed using FORTRAN03. The goal of this section is two fold: to

show that the software we developed is suitable for solving large scale problems (approximately $\mathcal{O}(10^9)$) in a massively parallel environment, and to reinforce the interest in using the deflation techniques proposed in Chapter 3 in the acoustic full waveform inversion context.

4.6.1 Forward Problem: Smoothed SEG/EAGE Salt Dome

The numerical experiments performed in Section 3.9 concern the acoustic wave propagation phenomena in frequency domain through the academic velocity field SEG/EAGE Overthrust, which is a realistic model, reflecting the real properties of the subsurface of Earth. However, for the purpose of the inverse problem, the approximated velocity model could present considerably different properties (especially during early iterations, or during the computation of an initial guess). For this reason, it turns out to be interesting to investigate the behaviour of deflation in a scenario where the velocity model is rather an approximation of a realistic model.

In this subsection we present numerical experiments related to the velocity model known as SEG/EAGE Salt Dome (already shown in Figure 4.9). This is a particularly challenging problem to be solved using iterative solvers due to a discontinuity in the velocity model (representing a dome of salt in the subsurface of Earth), which abruptly increases the velocity of propagation of the compressional waves (cf. Figure 4.12 for a graphical representation of the interior of this velocity model). We also perform perturbations on this velocity model, applying different degrees of compression techniques and obtaining smoothed approximated versions of this model (cf. Figure 4.13 to Figure 4.16 for graphical representations of the smoothed velocity field).

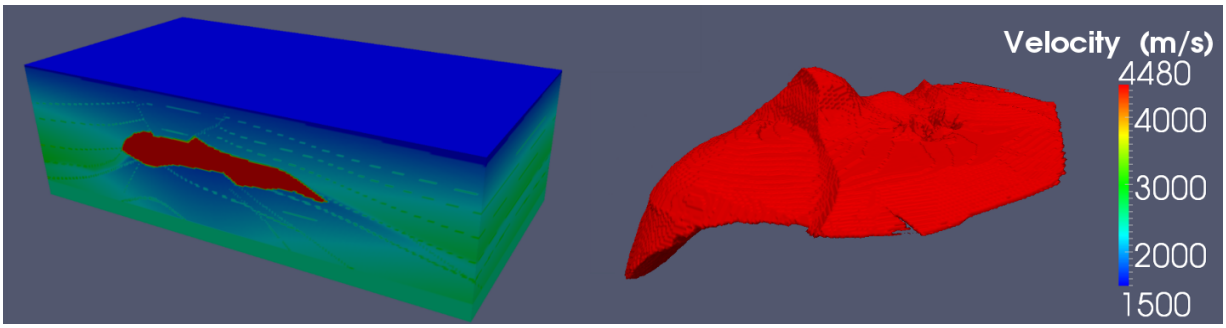


Figure 4.12: *Graphical representation of the interior of the three-dimensional academic velocity generated using Paraview [69]. The image on the left represents a plane cut at $y = 308$. The image on the right shows only the points of the domain which have velocity equal or higher than $4,000$ m/s, delineating the structure of the dome of salt.*

The experiments performed in this section aim at investigating how much the behaviour of the deflation of techniques proposed in Chapter 3 change as the velocity field turns smoother. All experiments were performed on a Bullx B510 computer located at CERFACS (two Intel Sandy Bridge with 2.6 Ghz and 32 Gb of memory per node). In all experiments we used DMBR(5) preconditioned by perturbed two-level method (cf. Subsection 4.4.1). The frequency was set to $5Hz$, and using one length of PML and 12 points per wavelength, the resulting discretized domain contained $624 \times 616 \times 240$ points[5] being thus a problem of dimension 9.2×10^7. We used 16 source positions equally spaced along a line (from point $(285, 309, 38)$ to point $(340, 309, 38)$) on the surface of the domain, and the memory cost per source point in this experiment is 11.8 Gb per right-hand side, plus the storage required to store the coefficients matrix A. The total cost including A and all right-hand sides is 189.6 Gb. The experiment was performed with 128 cores. Table 4.1 shows the results in number of iterations, number of preconditioner applications and total computational time for the test. Figure 4.13 to Figure 4.16 (left side) show a graphical representation

[5]Since we set the processor's geometry to $8 \times 4 \times 4$ we are obligated to allow $h_x \neq h_y$ due to round off, generating thus a different number of points in direction x and y.

of SEG/EAGE Salt Dome and its different smoothed versions, along with the wavefield obtained after solving the Helmholtz equation for its respective velocity model.

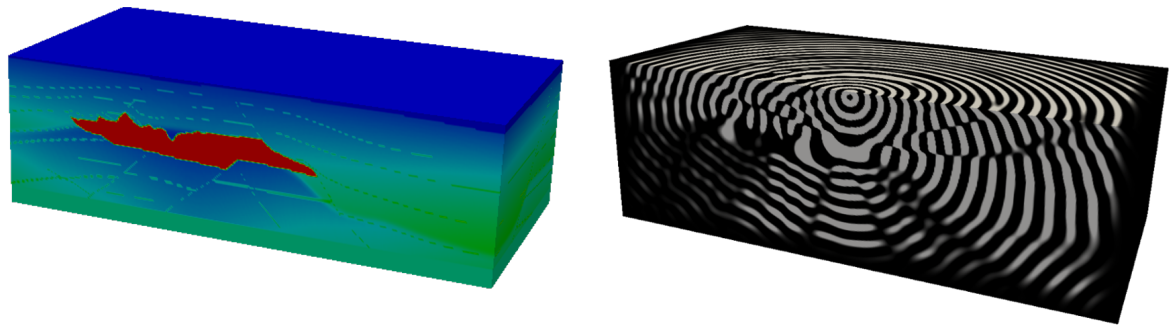


Figure 4.13: *Original SEG/EAGE Salt dome velocity field, and its respective wavefield obtained from the solution of the Helmholtz equation for* $5Hz$*. This shows the solution for the* 8*-th right-hand side.*

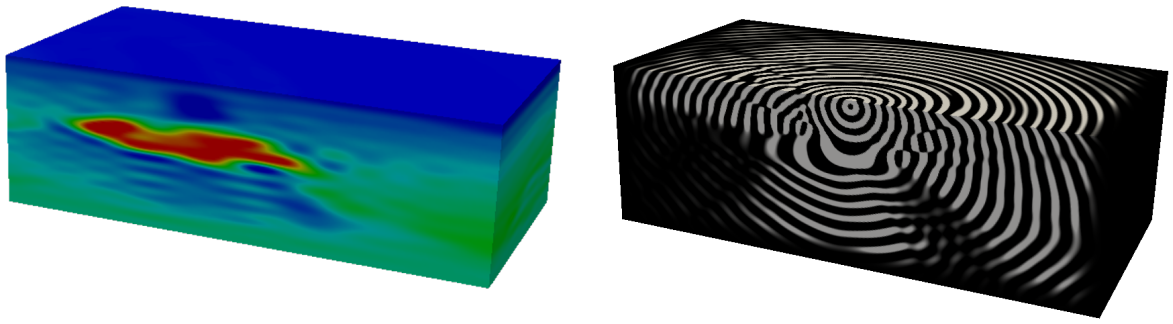


Figure 4.14: *Smoothed*$\times 1$ *version of SEG/EAGE Salt dome velocity field, and its respective wavefield obtained from the solution of the Helmholtz for* $5Hz$ *equation. This shows the solution for the* 8*-th right-hand side.*

We can verify that, as the velocity field becomes smoother, fewer iterations and fewer preconditioner applications are needed for finding an approximate solution satisfying the relative residual stopping criterion (for $\epsilon = 10^{-5}$). Figure 4.17 shows the history of k_j along the convergence of DMBR(5) for each velocity model.

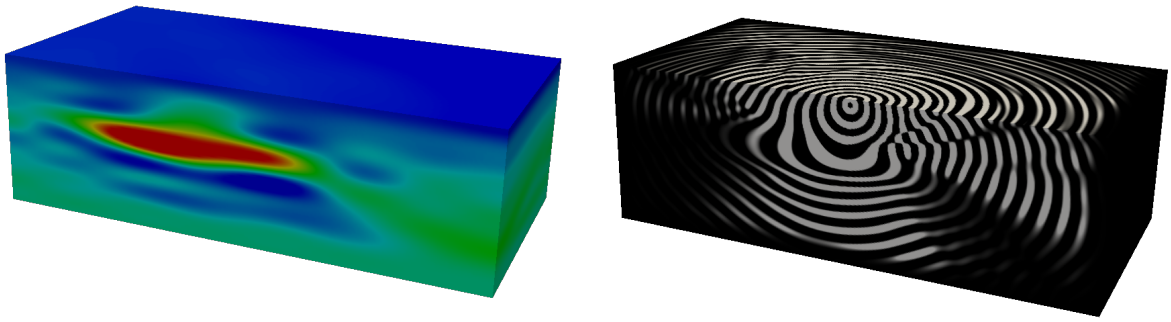


Figure 4.15: *Smoothed×2 version of SEG/EAGE Salt dome velocity field, and its respective wavefield obtained from the solution of the Helmholtz for* $5Hz$ *equation. This shows the solution for the* 8*-th right-hand side.*

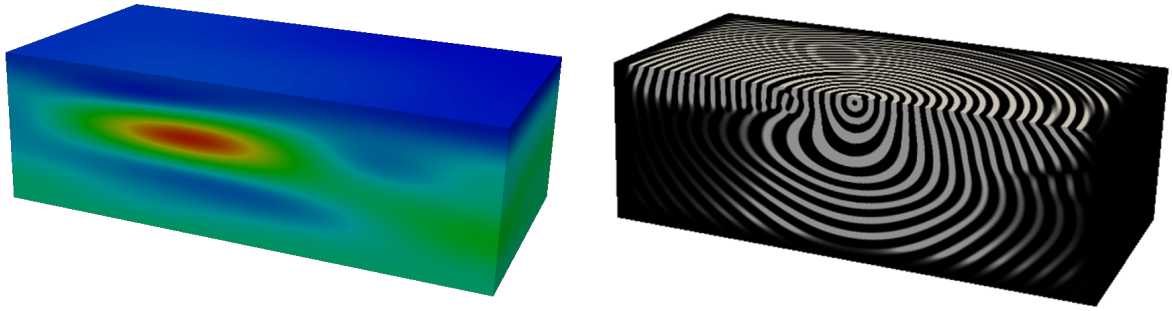


Figure 4.16: *Smoothed×3 version of SEG/EAGE Salt dome velocity field, and its respective wavefield obtained from the solution of the Helmholtz for* $5Hz$ *equation. This shows the solution for the* 8*-th right-hand side.*

As we can see in , the history of k_j for each velocity model is indeed very similar. For instance, Smoothed×3 velocity model shows a faster convergence in number of iterations for DMBR(5) when compared to the original SEG/EAGE salt dome velocity model (therein called Smoothed×0), but it starts deflating in an earlier iteration. The curves of evolution of k_j are relatively similar for all the velocity models. We consider this result very satisfactory and it encourages further research on the behaviour of deflation in the inverse problem application.

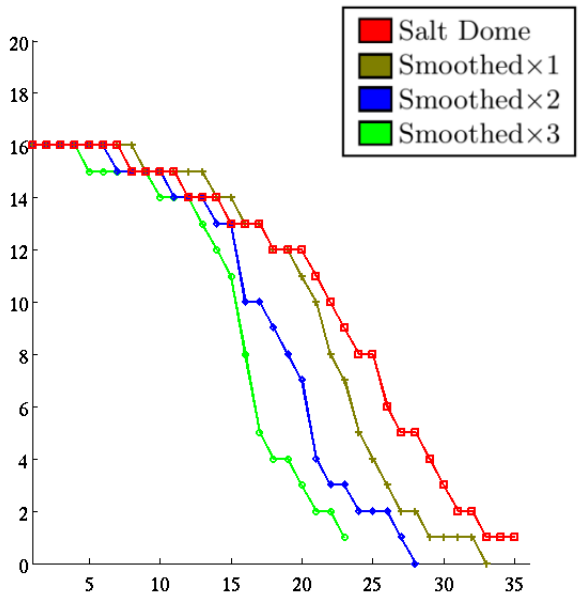


Figure 4.17: *Evolution of k_j along the iterations of DMBR(5) preconditioned by a two-level perturbed multigrid V-cycle (cf. Table 4.1) for each SEG/EAGE Salt dome velocity field and its respective smoothed versions.*

4.6.2 Forward Problem: Mid Frequency Case

In this subsection we quickly expose the solution we obtained when using our libraries to solve the Helmholtz equation for mid frequencies. The goal of this experiment is to illustrate that the software was designed for solving at higher frequencies as long as enough memory resources are available.

As in the previous section, we limit ourselves to the use of two-level techniques as described in Subsection 4.4.1, but here we use FGMRES(5) as outer solver rather than DMBR(5). The reason for this choice is that we perform this experiment using only one right-hand side. As the previous experiments, the following experiments was performed on a Bullx B510 computer located at CERFACS (two Intel Sandy Bridge with 2.6 Ghz and 32 Gb of memory per node): using the SEG/EAGE salt dome velocity field, we solve the Helmholtz equation for $12Hz$, and using one length of PML and 12 points per wavelength, the resulting discretized domain contained $1376 \times 1376 \times 480$ points being thus a problem of dimension 9.1×10^8. The only source was positioned at the very center of the domain. We used 128 cores, and the memory cost for this experiment was 141.7 Gb and with a computational time of $2,012$ seconds (approximately 33min) to find an approximate solution satisfying the relative residual smaller than 10^{-5}.

Figure 4.18 show a graphical representation of SEG/EAGE Salt Dome along with the wavefield obtained after solving the Helmholtz equation in this setting.

4.7 Conclusions

In this chapter we provided a wide overview of several aspects of the full waveform inversion problem. We focused on the challenges arising in the solution of the forward problem, and we highlight that there is no generic optimal solution, as the best method for solving the forward problem depends on characteristics of the problem, as for instance, the wavenumber (thus also the frequency), the size of the geophysical domain, among others. With this in mind, we motivate the use of a flexible object oriented code using MPI and FORTRAN03 which can be easily adapted to use new strategies or to choose which strategy is the best depending on the problem parameters. We propose a first version of this code, which implements some

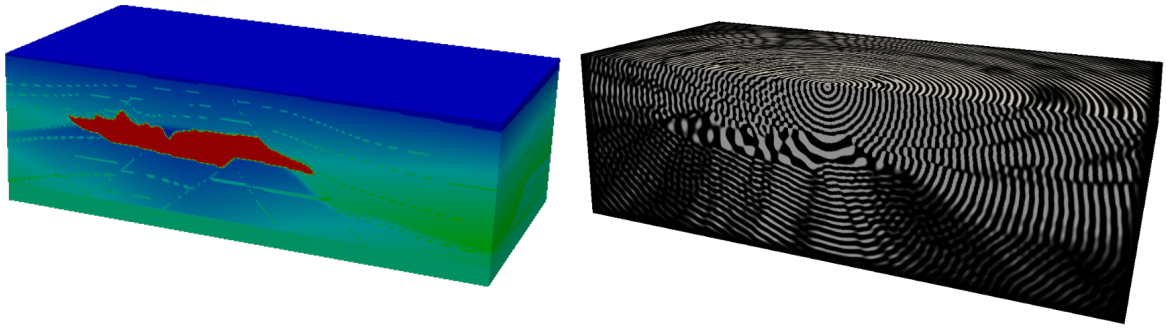


Figure 4.18: *SEG/EAGE Salt dome velocity field, and its respective wavefield obtained from the solution of the Helmholtz equation for* $12Hz$*. This experiment required* 141.7 *Gb of memory*

finite difference discretization schemes as well as some iterative solvers, as BFGMRESD [78], BGMRES-R [103] and DMBR presented in Chapter 3 as well as some preconditioners based on multigrid techniques, as the perturbed two level preconditioner [96] detailed in Subsection 4.4.1. Tests performed with this software has shown a satisfactory level of flexibility and further solvers, preconditioners and discretization schemes will be implemented in our future work. Appendix A contains a basic user guide with more detailed description of the code with examples and description of key functions, types and classes. We have shown a practical use of the code with numerical experiments performed at $5Hz$ and $12Hz$ using the academical velocity field SEG/EAGE salt dome, a challenging and realistic model. With this experiment we illustrated further the behaviour of the deflation techniques proposed in Chapter 3 when applied to the forward problem arising from the acoustic full waveform inversion problem, showing the interest in using DMBR method in real life applications.

Velocity Field	It	Pr	T
Salt Dome	35	365	1330
Smoothed×1 Salt Dome	33	337	1238
Smoothed×2 Salt Dome	28	285	1056
Smoothed×3 Salt Dome	23	246	921

Table 4.1: *Experiment at $f = 5$ Hz with $p = 16$ source points. The method used was DMBR(5) preconditioned by perturbed two-level preconditioner (cf. Subsection 4.4.1). It denotes the number of iterations, Pr the number of preconditioner applications on a single vector and T denotes the total computational time in seconds. 128 cores were used in this experiment, and the memory cost is* 11.8 *Gb per right-hand side, with a total of* 189.6 *Gb*

Chapter 5

Flexible GCRO-DR

5.1 Foreword

In this chapter we present the flexible generalized conjugate residual method with inner orthogonalization and deflated restarting (FGCRO-DR) method. It consists of an iterative solver able to *recycle subspace* information of one cycle to accelerate the convergence of the following cycle, a technique initially proposed for the solution of sequence of linear systems, that is, when we have the sequence of systems

$$Ax^{(i)} = b^{(i)}, \text{ with each } x^{(i)}, b^{(i)} \in \mathbb{C}^n$$

for $1 \leq i \leq p$, where $b^{(i)}$ may depend on some $x^{(j)}$ for $1 \leq j < i$. In such a scenario, the use of block methods as proposed in Chapter 2 and 3 is not possible.

The development of this method has been greatly inspired by the application described in Chapter 4, since subspace recycling techniques could be used to accelerate the convergence when the problem presents multiple left-hand sides, that is

$$A^{(i)}X = B, \text{ with } X, B \in \mathbb{C}^{n\times p}, A^{(i)} \in \mathbb{C}^{n\times n}$$

$1 \leq i \leq \lambda$, a situation which could arise as well from the Earth imaging scenario (cf. Remark 4.2.1). We refer to [1, 95] for more details on the use of recycling techniques for the multiple left-hand side case.

In this chapter we introduce the FGCRO-DR method for single right hand sides scenario and discuss its main properties, focusing on the comparison between FGCRO-DR and FGMRES-DR [58], a closely related method which is unable to recycle subspace information. We demonstrate that, in spite of the similarities between both methods, when a variable preconditioner is used they are only equivalent if a collinearity condition holds (cf. Theorem 5.2.11)

We let as future work the investigation of the efficiency of FGCRO-DR recycling techniques when solving the forward and backward problem or when solving for multiple models simultaneously. Another important issue we do not address in this manuscript is the generalization of FGCRO-DR for the block scenario and the potential combination of recycling technique with block techniques, which is also subject of our future research.

5.2 A Flexible Generalized Conjugate Residual Method with Inner Orthogonalization and Deflated Restarting

The title as well as the contents of this section corresponds to [27], joint work with Luiz Mariano Carvalho, Serge Gratton and Xavier Vasseur.

Abstract

This work is concerned with the development and study of a minimum residual norm subspace method based on the Generalized Conjugate Residual method with inner Orthogonalization (GCRO) method that allows flexible preconditioning and deflated restarting for the solution of non-symmetric or non-Hermitian linear systems. First we recall the main features of Flexible Generalized Minimum Residual with deflated restarting (FGMRES-DR), a recently proposed algorithm of the same family but based on the GMRES method. Next we introduce the new inner-outer subspace method named FGCRO-DR. A theoretical comparison of both algorithms is then made in the case of flexible preconditioning. It is proved that FGCRO-DR and FGMRES-DR are algebraically equivalent if a collinearity condition is satisfied. While being nearly as expensive as FGMRES-DR in terms of computational operations per cycle, FGCRO-DR offers the additional advantage to be suitable for the solution of sequences of slowly changing linear systems (where both the matrix and right-hand side can change) through subspace recycling. Numerical experiments on the solution of multidimensional elliptic partial differential equations show the efficiency of FGCRO-DR when solving sequences of linear systems.

5.2.1 Introduction

In recent years, several authors studied inner-outer Krylov subspace methods that allow variable preconditioning for the iterative solution of large sparse linear systems of equations. One of the first papers describing a subspace method with variable preconditioning is due to Axelsson and Vassilevski who proposed the Generalized Conjugate Gradient method [7]. See also [6, Section 12.3] for additional references. Since then, numerous methods have been proposed to address the symmetric, non-symmetric or non-Hermitian cases; these include Flexible Conjugate Gradient [90], Flexible GMRES (FGMRES) [105], Flexible QMR [124] and GMRESR [132] among others. This class of methods is required when preconditioning with a different (possibly nonlinear) operator at each iteration of a subspace method is considered. This notably occurs when adaptive preconditioners using information obtained from previous iterations [8, 48] are used or when inexact solutions of the preconditioning system using e.g. adaptive cycling strategy in multigrid [91] or approximate interior solvers in domain decomposition methods [127, Section 4.3] are considered. The latter situation is frequent when solving very large systems of linear equations resulting from the discretization of partial differential equations in three dimensions. Thus flexible Krylov subspace methods have gained a considerable interest in the past years and are subject to both theoretical and numerical studies [112]. We refer the reader to [114, Section 10] for additional comments on flexible methods.

When non variable preconditioning is considered, the full GMRES method [108] is often chosen for the solution of non-symmetric or non-Hermitian linear systems because of its robustness and its minimum residual norm property [107]. Nevertheless to control both the memory requirements and the computational cost of the orthogonalization scheme, restarted GMRES is preferred; it corresponds to a scheme where the maximal dimension of the approximation subspace is fixed. It means in practice that the orthonormal basis built is thrown away at the end of the cycle. Since some information is discarded at the restart, the convergence may stagnate and is expected to be slower compared to full GMRES. Nevertheless to retain the convergence rate a number of techniques have been proposed; they fall in the class of augmented and deflated methods; see e.g. [11, 35, 41, 86, 106]. Deflated methods compute spectral information at a restart and use this information to improve the convergence of the subspace method. One of the most recent procedure based on a deflation approach is GMRES with deflated restarting

(GMRES-DR) [88]. This method reduces to restarted GMRES when no deflation is applied, but may provide a much faster convergence than restarted GMRES for well chosen deflation spaces as described in [88].

Quite recently a new minimum residual norm subspace method based on GMRES allowing deflated restarting and variable preconditioning has been proposed in [58]. It mainly attempted to combine the numerical features of GMRES with deflated restarting and the flexibility property of FGMRES. Numerical experiments in [58] have shown the efficiency of Flexible GMRES with deflated restarting (FGMRES-DR) on both academic and industrial examples. In this section we study a new minimum residual norm subspace method based on the Generalized Conjugate Method with inner Orthogonalization (GCRO) [34] allowing deflated restarting and variable preconditioning. It is named Flexible Generalized Conjugate Residual Method with Inner Orthogonalization and Deflated Restarting (FGCRO-DR) and can be viewed as an extension of GCRO-DR [95] to the case of variable preconditioning. A major advantage of FGCRO-DR over FGMRES-DR is its ability to solve sequence of linear systems (where both the left and right-hand sides could change) through recycling [95]. In [95] Parks et al. mentioned that GCRO-DR and GMRES-DR were algebraically equivalent i.e. both methods produce the same iterates in exact arithmetic when solving the same given linear system starting from the same initial guess. When variable preconditioning is considered, it seems therefore natural to ask whether FGCRO-DR and FGMRES-DR could be also algebraically equivalent. We address this question in this section and the main theoretical developments that are proposed will help us to answer this question. The main contributions of the section are then twofold. First we prove that FGCRO-DR and FGMRES-DR can be considered as algebraically equivalent if a collinearity condition between two certain vectors is satisfied at each cycle. When considering non variable preconditioning, these theoretical developments will also allow us to show the algebraic equivalence between GCRO-DR and GMRES-DR that was stated without proof in [95]. Secondly we carefully analyze the computational cost of FGCRO-DR and show that the proposed method is nearly as expensive as FGMRES-DR in terms of operations per cycle. Furthermore it is explained how to include subspace recycling into FGCRO-DR and numerical experiments are reported showing the efficiency of FGCRO-DR.

This section is organized as follows. In Subsection 5.2.2 we introduce the general background of this study. We briefly recall the main properties of FGMRES-DR and then introduce the FGCRO-DR method both from a mathematical and algorithmic points of view. Subsection 5.2.7 is mainly devoted to the analysis of both flexible methods. Therein we show that both methods can be algebraically equivalent in the flexible case if a certain collinearity condition is satisfied at each cycle. In Subsection 5.2.11 we compare FGCRO-DR and FGMRES-DR in terms of computational operations per cycle and storage and discuss the solution of sequences of linear systems through subspace recycling. Finally we draw some conclusions and perspectives in Subsection 5.2.15.

5.2.2 Flexible Krylov methods with restarting

5.2.3 General setting

Notation Throughout this section we denote by $\|.\|$ the Euclidean norm, $I_k \in \mathbb{C}^{k\times k}$ the identity matrix of dimension k and $0_{i\times j} \in \mathbb{C}^{i\times j}$ the zero rectangular matrix with i rows and j columns. Given $N \in \mathbb{C}^{n\times m}$ $\Pi_{N^\perp} = I_n - N\,N^\dagger$ will represent the orthogonal projector onto range $(N)^\perp$, where the superscript $\dagger$ refers to the Moore-Penrose inverse. Finally given $Z_m = [z_1, \cdots, z_m] \in \mathbb{C}^{n\times m}$, we will usually decompose Z_m into two submatrices defined as $Z_k = [z_1, \cdots, z_k] \in \mathbb{C}^{n\times k}$ and $Z_{m-k} = [z_{k+1}, \cdots, z_m] \in \mathbb{C}^{n\times(m-k)}$.

Setting We focus on minimum residual norm based subspace methods that allow flexible preconditioning for the iterative solution of

$$Ax = b, \quad A \in \mathbb{C}^{n\times n}, \quad x, b \in \mathbb{C}^n \tag{5.2.1}$$

given an initial vector $x_0 \in \mathbb{C}^n$. In this section A is supposed to be nonsingular. Flexible methods refer to a class of methods where the preconditioner is allowed to vary at each iteration. We refer the reader

to e.g. [114] for a general introduction on Krylov subspace methods and to [114, Section 10] and [107, Section 9.4] for a review on flexible methods. The minimum residual norm GMRES method [108] has been extended by Saad [105] to allow variable preconditioning. The resulting algorithm known as FGMRES(m) relies on the Arnoldi relation

$$AZ_m = V_{m+1}\bar{H}_m, \tag{5.2.2}$$

where $Z_m \in \mathbb{C}^{n\times m}$, $V_{m+1} \in \mathbb{C}^{n\times(m+1)}$ has orthonormal columns and $\bar{H}_m \in \mathbb{C}^{(m+1)\times m}$ is upper Hessenberg. We denote by $\mathcal{M}_j$ the preconditioning operator at iteration j and remark that $\mathcal{M}_j$ may be a nonlinear preconditioning function. We will then denote by $\mathcal{M}_j(v)$ the action of $\mathcal{M}_j$ on a vector v. In (5.2.2), the columns of V_{m+1} form an orthonormal basis of the subspace spanned by the following vectors

$$\{r_0, Az_1, \cdots, Az_m\} \quad \text{with} \quad r_0 = b - Ax_0$$

whereas $Z_m = [z_1, \cdots, z_m]$ and $V_m = [v_1, \cdots, v_m]$ are related by

$$Z_m = [\mathcal{M}_1(v_1), \cdots, \mathcal{M}_m(v_m)] \quad \text{with} \quad v_1 = \frac{r_0}{\|r_0\|}.$$

At the end of the cycle an approximate solution $x_m \in \mathbb{C}^n$ is then found by minimizing the residual norm $\|r_0 - AZ_m y\|$ over the space $x_0 + \text{range}\,(Z_m)$. Thus we obtain that

$$x_m = x_0 + Z_m y^*,$$

where y^* is the solution of the following least-squares problem of size $(m+1)\times m$

$$y^* = \text{argmin}_{y\in\mathbb{C}^m}\|r_0 - AZ_m y\| = \text{argmin}_{y\in\mathbb{C}^m}\left\|\|r_0\|\ e_1 - \bar{H}_m y\right\|,$$

where e_1 is the first canonical vector of $\mathbb{C}^{m+1}$. Flexible subspace methods with restarting are based on a procedure where the construction of the subspace is stopped after a certain number of steps (denoted by m in this section with $m < n$). The method is then restarted mainly to control both the memory requirements and the cost of the orthogonalization scheme. In FGMRES(m) the restarting consists in taking as an initial guess the last iterate of the cycle (x_m).

The main focus of this section is to present minimum residual norm subspace methods with *deflated* restarting that allow *flexible* preconditioning. Deflated restarting aims at determining an approximation subspace of dimension m as a direct sum of two subspaces of smaller dimension, where one of these subspaces will contain relevant spectral information that will be kept for the next cycle. We refer the reader to e.g. [106] and [114, Section 9] for a review on augmented and deflated methods. Flexible methods with deflated restarting will notably satisfy the following flexible Arnoldi relation

$$AZ_m = V_{m+1}\bar{H}_m \quad \text{with} \quad V_{m+1}^H\, V_{m+1} = I_{m+1}, \tag{5.2.3}$$

where $\bar{H}_m \in \mathbb{C}^{(m+1)\times m}$ is not necessarily of upper Hessenberg form. In this section we call this relation a flexible Arnoldi-like relation due to its similarity to relation (5.2.2).

Stagnation and breakdown We refer the reader to [112, Section 6] for general comments and a detailed discussion on the possibility of both breakdown and stagnation in flexible inner-outer Krylov subspace methods. Although important, these issues are not addressed in this section and we assume that no breakdown occurs in the inner-outer subspace methods that will be proposed.

5.2.4 Flexible GMRES with deflated restarting

A number of techniques have been proposed to compute spectral information at a restart and use this information to improve the convergence rate of the Krylov subspace methods; see, e.g., [86, 87, 88, 106].

These techniques have been exclusively developed in the case of a fixed preconditioner. Among others GMRES-DR is one of those methods. It focuses on removing (or deflating) the eigenvalues of smallest magnitude. A full subspace of dimension k, $k < m$ (and not only the approximate solution with minimum residual norm) is now retained at the restart and the success of this approach has been demonstrated on many academic examples [86]. Approximations of eigenvalues of smallest magnitude are obtained by computing harmonic Ritz pairs of A with respect to a certain subspace [88]. We present here a definition of a harmonic Ritz pair equivalent to the one introduced in [94, 117]; it will be of key importance when defining appropriate deflation strategies.

Definition 5.2.1. Harmonic Ritz pair. Consider a subspace $\mathcal{U}$ of $\mathbb{C}^n$. Given $B \in \mathbb{C}^{n\times n}$, $\theta \in \mathbb{C}$ and $y \in \mathcal{U}$, (θ, y) is a harmonic Ritz pair of B with respect to $\mathcal{U}$ if and only if

$$By - \theta\, y \perp B\mathcal{U}$$

or equivalently, for the canonical scalar product,

$$\forall w \in \text{range}\,(B\mathcal{U}) \quad w^H\,(By - \theta\, y) = 0.$$

We call y a harmonic Ritz vector associated with the harmonic Ritz value θ.

As in the case of fixed preconditioning, deflated restarting may also improve the convergence rate of flexible subspace methods. In [58] a deflated restarting procedure has been proposed for the FGMRES algorithm. The i-th cycle of the resulting algorithm called FGMRES-DR is now briefly described and we denote by $r_0^{(i-1)} = b - Ax_0^{(i-1)}$, V_{m+1}, $\bar{H}_m$ and Z_m the residual and matrices obtained at the end of the $(i-1)$-th cycle.

Based on the Arnoldi-like relation (5.2.3), the deflation procedure proposed in [58, Proposition 1] relies on the use of k harmonic Ritz vectors $Y_k = V_m P_k$ of $AZ_m V_m^H$ with respect to range (V_m), where $Y_k \in \mathbb{C}^{n\times k}$ and $P_k \in \mathbb{C}^{m\times k}$. In Lemma 5.2.2 shown in [58, Lemma 3.1], we recall a useful relation satisfied by the harmonic Ritz vectors.

Lemma 5.2.2. *In flexible GMRES with deflated restarting, the harmonic Ritz vectors are given by $Y_k = V_m P_k$ with corresponding harmonic Ritz values λ_k. $P_k \in \mathbb{C}^{m\times k}$ satisfies the following relation:*

$$AZ_m P_k = V_{m+1}\left[\begin{bmatrix} P_k \\ 0_{1\times k}\end{bmatrix}, c - \bar{H}_m y^*\right]\begin{bmatrix} \text{diag}(\lambda_1,\ldots,\lambda_k) \\ \alpha_{1\times k}\end{bmatrix}, \tag{5.2.4}$$

$$AZ_m P_k = [V_m P_k, r_0^{(i-1)}]\begin{bmatrix} \text{diag}(\lambda_1,\ldots,\lambda_k) \\ \alpha_{1\times k}\end{bmatrix}, \tag{5.2.5}$$

where $r_0^{(i-1)} = V_{m+1}(c - \bar{H}_m y^*)$ and $\alpha_{1\times k} = [\alpha_1,\ldots,\alpha_k] \in \mathbb{C}^{1\times k}$.

Next, the QR factorization of the $(m+1)\times(k+1)$ matrix appearing on the right-hand side of relation (5.2.4) is performed as

$$\left[\begin{bmatrix} P_k \\ 0_{1\times k}\end{bmatrix}, c - \bar{H}_m y^*\right] = QR \tag{5.2.6}$$

where $Q \in \mathbb{C}^{(m+1)\times(k+1)}$ has orthonormal columns and $R \in \mathbb{C}^{(k+1)\times(k+1)}$ is upper triangular. We write the matrix Q obtained in relation (5.2.6) as

$$Q = \left[\begin{bmatrix} Q_{m\times k} \\ 0_{1\times k}\end{bmatrix}, \frac{\bar{\rho}}{\|\bar{\rho}\|}\right], \tag{5.2.7}$$

where $Q_{m\times k} \in \mathbb{C}^{m\times k}$ and $\bar{\rho} \in \mathbb{C}^{m+1}$ is defined as

$$\bar{\rho} = (I_{m+1} - \begin{bmatrix} Q_{m\times k} \\ 0_{1\times k} \end{bmatrix} \begin{bmatrix} Q_{m\times k} \\ 0_{1\times k} \end{bmatrix}^H)(c - \bar{H}_m y^*). \tag{5.2.8}$$

Proposition 5.2.3. *In flexible GMRES with deflated restarting, the flexible Arnoldi relation*

$$\begin{aligned} A\ Z_k &= V_{k+1}\bar{H}_k, && (5.2.9)\\ V_{k+1}^H V_{k+1} &= I_{k+1}, && (5.2.10)\\ \text{range}\left([Y_k, r_0^{(i-1)}]\right) &= \text{range}\,(V_{k+1}) && (5.2.11) \end{aligned}$$

holds at the i-th cycle with matrices $Z_k, V_k \in \mathbb{C}^{n\times k}$ and $\bar{H}_k \in \mathbb{C}^{(k+1)\times k}$ defined as

$$\begin{aligned} Z_k &= Z_m Q_{m\times k}, && (5.2.12)\\ V_{k+1} &= V_{m+1} Q, && (5.2.13)\\ \bar{H}_k &= Q^H \bar{H}_m Q_{m\times k}, && (5.2.14) \end{aligned}$$

where V_{m+1}, Z_m and $\bar{H}_m$ refer to matrices obtained at the end of the $(i-1)$-th cycle.

Proof. Relations (5.2.9), (5.2.10), (5.2.12), (5.2.13) and (5.2.14) have been shown in [58, Proposition 2]. Respectively, from relations (5.2.13) and (5.2.6), we deduce

$$\begin{aligned} V_{k+1}R &= V_{m+1}\left[\begin{bmatrix} P_k \\ 0_{1\times k} \end{bmatrix}, c - \bar{H}_m y^*\right] \\ V_{k+1}R &= [V_m P_k, r_0^{(i-1)}] && (5.2.15) \end{aligned}$$

which finally shows that range $\left([Y_k, r_0^{(i-1)}]\right) = \text{range}\,(V_{k+1})$ since R is supposed to be nonsingular. □

FGMRES-DR then carries out $m-k$ Arnoldi steps with flexible preconditioning and starting vector v_{k+1} while maintaining orthogonality to V_k leading to

$$A\,[z_{k+1}, \cdots, z_m] = [v_{k+1}, \cdots, v_{m+1}]\,\bar{H}_{m-k} \quad \text{and} \quad V_{m+1}^H\,V_{m+1} = I_{m+1}.$$

We note that $\bar{H}_{m-k} \in \mathbb{C}^{(m-k+1)\times(m-k)}$ is upper Hessenberg. At the end of the i-th cycle this gives the flexible Arnoldi-like relation

$$A\,[Z_k, Z_{m-k}] = [V_{m+1}]\left[\begin{bmatrix} \bar{H}_k \\ 0_{m-k\times k} \end{bmatrix}\begin{bmatrix} B_{k\times m-k} \\ \bar{H}_{m-k} \end{bmatrix}\right]$$

where $V_{m+1} \in \mathbb{C}^{n\times(m+1)}$, $\bar{H}_m \in \mathbb{C}^{(m+1)\times m}$ and $B_{k\times m-k} \in \mathbb{C}^{k\times(m-k)}$ results from the orthogonalization of $[v_{k+2}, \cdots, v_{m+1}]$ against V_{k+1}. We note that $\bar{H}_m$ is no more upper Hessenberg due to the leading dense $(k+1)\times k$ submatrix $\bar{H}_k$. At the end of the i-th cycle, an approximate solution $x_0^{(i)} \in \mathbb{C}^n$ is then found by minimizing the residual norm $\left\|b - A(x_0^{(i-1)} + Z_m y)\right\|$ over the space $x_0^{(i-1)} + \text{range}(Z_m)$, the corresponding residual being $r_0^{(i)} = b - Ax_0^{(i)}$, with $r_0^{(i)} \in \text{range}(V_{m+1})$. We refer the reader to [58] for the complete derivation of the method and numerical experiments showing the efficiency of FGMRES-DR on both academic and industrial examples.

5.2.5 Flexible GCRO with deflated restarting

GCRO-DR [95] - a combination of GMRES-DR and GCRO - is a Krylov subspace method that allows deflated restarting and subspace recycling simultaneously. This latter feature is particularly interesting when solving sequences of linear systems with possibly different left or right-hand sides. As pointed out in [95], GCRO-DR is attractive because any subspace may be recycled. In this section we restrict the presentation to the case of a single linear system as proposed in (5.2.1).

GCRO and GCRO-DR belong to the family of inner-outer methods [6, Ch. 12] where the outer iteration is based on GCR, a minimum residual norm method proposed by Eisenstat, Elman and Schultz [43]. To this end GCR maintains a correction subspace spanned by range (Z_m) and an approximation subspace spanned by range (V_m), where $Z_m, V_m \in \mathbb{C}^{n\times m}$ satisfy

$$\begin{aligned} A\ Z_m &= V_m, \\ V_m^H\ V_m &= I_m. \end{aligned}$$

The optimal solution of the minimization problem $\min\|b - Ax\|$ over the subspace $x_0 + \text{range}\,(Z_m)$ is then found as $x_m = x_0 + Z_m\ V_m^H\ r_0$. Consequently $r_m = b - A\ x_m$ satisfies

$$r_m = r_0 - V_m\ V_m^H r_0 = \Pi_{V_m^\perp} r_0, \quad r_m \perp \text{range}\,(V_m)\,.$$

In [34] de Sturler proposed an improvement to GMRESR [132], an inner-outer method based on GCR in the outer part and GMRES in the inner part. He suggested that the inner iteration takes place in a subspace orthogonal to the outer Krylov subspace. In this inner iteration the projected residual equation

$$(I_n - V_m\ V_m^H)Az \quad = \quad r_m$$

is solved only approximately. If a minimum residual norm subspace method is used in the inner iteration to solve this projected residual linear system, the residual over both the inner and outer subspaces are minimized. This leads to the GCRO (Generalized Conjugate Residual method with inner Orthogonalization) Krylov subspace method [34]. Numerical experiments [34] indicate that the resulting method may perform better than other inner-outer methods (without orthogonalization) in some cases.

The GCRO method with deflated restarting (named GCRO-DR) based on harmonic Ritz value information has been proposed in [95]. An approximate invariant subspace is used for deflation following closely the GMRES-DR method. We refer the reader to [95] for a description of this method, algorithms and implementation details. We present now a new variant of GCRO-DR that allows flexible preconditioning by explaining the different steps occurring during the i-th cycle. Again we denote by $r_0^{(i-1)} = b - Ax_0^{(i-1)}$, V_{m+1}, $\bar{H}_m$ and Z_m the residual and matrices obtained at the end of the $(i-1)$-th cycle.

We suppose that a flexible Arnoldi-like relation of type (5.2.3) holds. As in Subsection 5.2.4 an important point is to specify which harmonic Ritz information is selected. Given a certain matrix $W_m \in \mathbb{C}^{n\times m}$ to be specified later on, such as range (W_m) = range (V_m), the deflation procedure relies on the use of k harmonic Ritz vectors $Y_k = W_m P_k$ of $AZ_m W_m^\dagger$ with respect to range (W_m), where $Y_k \in \mathbb{C}^{n\times k}$ and $P_k \in \mathbb{C}^{m\times k}$. W_m will notably satisfy a property detailed in Lemma 5.2.8 and we point out that the calculation of $W_m^\dagger$ is not needed in the practical implementation of the algorithm (see Section 5.2.12). In Lemma 5.2.4 we detail a useful relation satisfied by the harmonic Ritz vectors.

Lemma 5.2.4. *In flexible GCRO with deflated restarting, the harmonic Ritz vectors are given by $Y_k = W_m P_k$ with corresponding harmonic Ritz values θ_k. The matrix $P_k = [p_1, \cdots, p_k] \in \mathbb{C}^{m\times k}$ satisfies the following relation:*

$$AZ_m P_k = [W_m P_k, r_0^{(i-1)}] \begin{bmatrix} \text{diag}(\theta_1, \ldots, \theta_k) \\ \beta_{1\times k} \end{bmatrix}, \tag{5.2.16}$$

where $r_0^{(i-1)} = V_{m+1}(c - \bar{H}_m y^*)$ and $\beta_{1\times k} = [\beta_1, \ldots, \beta_k] \in \mathbb{C}^{1\times k}$.

Proof. According to Definition 5.2.1, the harmonic residual vectors $AZ_m W_m^\dagger W_m p_j - \theta_j W_m p_j$ and the residual vector $r_0^{(i-1)} = V_{m+1}(c - \bar{H}_m y^*)$ all belong to a subspace of dimension $m+1$ (spanned by the columns of V_{m+1}) and are orthogonal to the same subspace of dimension m (spanned by the columns of AZ_m subspace of range (V_{m+1})), so they must be collinear. Consequently there exist k coefficients noted $\beta_j \in \mathbb{C}$ with $1 \leq j \leq k$ such that

$$\forall j \in \{1, \ldots, k\} \quad AZ_m p_j \quad - \quad \theta_j W_m p_j = \beta_j r_0^{(i-1)}. \tag{5.2.17}$$

Setting $\beta_{1\times k} = [\beta_1, \ldots, \beta_k] \in \mathbb{C}^{1\times k}$, the collinearity expression (5.2.17) can be written in matrix form as

$$AZ_m P_k = [W_m P_k, r_0^{(i-1)}] \begin{bmatrix} \mathrm{diag}(\theta_1, \ldots, \theta_k) \\ \beta_{1\times k} \end{bmatrix}.$$

Due to the flexible Arnoldi-like relation (5.2.3), relation (5.2.16) can be also expressed as

$$V_{m+1}\bar{H}_m P_k = [W_m P_k, r_0^{(i-1)}] \begin{bmatrix} \mathrm{diag}(\theta_1, \ldots, \theta_k) \\ \beta_{1\times k} \end{bmatrix}. \tag{5.2.18}$$

If required, $\beta_{1\times k}$ can be deduced from (5.2.18) by

$$(c - \bar{H}_m y^*)^H (\bar{H}_m P_k - V_{m+1}^H W_m P_k \mathrm{diag}(\theta_1, \ldots, \theta_k)) = (c - \bar{H}_m y^*)^H (c - \bar{H}_m y^*)\beta_{1\times k}. \tag{5.2.19}$$

□

Next, the QR factorization of the $(m+1)\times k$ matrix $\bar{H}_m P_k$ appearing in relation (5.2.18) is performed as $\bar{H}_m P_k = QR$ with $Q \in \mathbb{C}^{(m+1)\times k}$ and $R \in \mathbb{C}^{k\times k}$.

Proposition 5.2.5. *In flexible GCRO with deflated restarting, the relation $AZ_k = V_k$ with $V_k^H V_k = I_k$ holds at the i-th cycle with matrices $Z_k, V_k \in \mathbb{C}^{n\times k}$ defined as*

$$\begin{aligned} Z_k &= Z_m P_k R^{-1}, \\ V_k &= V_{m+1} Q, \end{aligned}$$

where V_{m+1} and Z_m refer to matrices obtained at the end of the $(i-1)$-th cycle. In addition $V_k^H r_0^{(i-1)} = 0$ holds during the i-th cycle.

Proof. By using information related to the QR factorization of $\bar{H}_m P_k$ and the flexible Arnoldi relation (5.2.3) exclusively, we obtain

$$\begin{aligned} A\ Z_k &= AZ_m P_k R^{-1}, \\ &= V_{m+1}\bar{H}_m P_k R^{-1}, \\ &= V_{m+1} Q, \\ &= V_k. \end{aligned}$$

Since both V_{m+1} and Q have orthonormal columns, V_k satisfies $V_k^H V_k = I_k$. Finally since $r_0^{(i-1)}$ is the optimum residual at the $i-1$-th cycle, i.e. $(AZ_m)^H r_0^{(i-1)} = 0$ we obtain

$$\begin{aligned} P_k^H (AZ_m)^H r_0^{(i-1)} &= 0, \\ (V_{m+1}\bar{H}_m P_k)^H r_0^{(i-1)} &= 0, \\ R^H V_k^H r_0^{(i-1)} &= 0. \end{aligned}$$

This finally shows that $V_k^H r_0^{(i-1)} = 0$ since R is supposed to be nonsingular. □

To complement the subspaces, the inner iteration is based on the approximate solution of

$$(I_n - V_k\ V_k^H)Az = (I_n - V_k\ V_k^H)r_0^{(i-1)} = r_0^{(i-1)},$$

where the last equality is due to Proposition 5.2.5. For that purpose FGCRO-DR then carries out $m-k$ steps of the Arnoldi method with flexible preconditioning leading to

$$\begin{aligned}(I_n - V_k V_k^H)\ A\ [z_{k+1}, \cdots, z_m] &= [v_{k+1}, \cdots, v_{m+1}]\ \bar{H}_{m-k} \\ (I_n - V_k V_k^H)\ A\ Z_{m-k} &= V_{m-k+1}\ \bar{H}_{m-k}\end{aligned}$$

with $v_{k+1} = r_0^{(i-1)}/||r_0^{(i-1)}||$. At the end of the cycle this gives the flexible Arnoldi-like relation

$$\begin{aligned}A\ [Z_k, Z_{m-k}] &= [V_k, V_{m-k+1}] \begin{bmatrix} I_k & V_k^H\ A\ Z_{m-k} \\ 0_{m-k+1\times k} & \bar{H}_{m-k} \end{bmatrix} \\ A\ Z_m &= V_{m+1}\ \bar{H}_m,\end{aligned}$$

where $Z_m \in \mathbb{C}^{n\times m}$, $V_{m+1} \in \mathbb{C}^{n\times(m+1)}$ and $\bar{H}_m \in \mathbb{C}^{(m+1)\times m}$. At the end of the i-th cycle, an approximate solution $x_0^{(i)} \in \mathbb{C}^n$ is then found by minimizing the residual norm $\left\|b - A(x_0^{(i-1)} + Z_m y)\right\|$ over the space $x_0^{(i-1)} + \text{range}(Z_m)$, the corresponding residual being $r_0^{(i)} = b - Ax_0^{(i)}$, with $r_0^{(i)} \in \text{range}(V_{m+1})$.

5.2.6 Algorithms

Details of the FGCRO-DR method are given in Algorithm 5.2.1, where Matlab-like notations are adopted (for instance in step 7b, $Q(1:m, 1:k)$ denotes the submatrix made of the first m rows and first k columns of matrix Q noted $Q_{m\times k}$ in equation (5.2.7)). For the sake of completeness the FGMRES-DR algorithm has been also described with notations chosen as close as possible to FGCRO-DR to make a code comparison as easy as possible. Concerning Algorithm 5.2.1 we make the following comments:

- As discussed later the computation of $W_m^\dagger$ in step 5a is not required thanks to the definition of the harmonic Ritz pair (see Definition 5.2.1).
- As pointed out by Morgan [88] and Parks et al. [95] we might have to adjust k during the algorithm to include both the real and imaginary parts of complex eigenvectors.
- In steps 10a and 10b $\mathcal{M}_j^{(i)}$ denotes the possibly nonlinear preconditioning operator at iteration j during the i-th cycle.

5.2.7 Analysis of FGMRES-DR and FGCRO-DR

We compare now the flexible variants of GMRES-DR and GCRO-DR introduced respectively in Subsection 5.2.4 and 5.2.5 . In the following we use the superscript $\widetilde{}$ to denote quantities related to the FGMRES-DR algorithm e.g. $\widetilde{Y}_k$ denotes the set of harmonic Ritz vectors computed in the FGMRES-DR algorithm. When analyzing both algorithms we will suppose that identical preconditioning operators are used in steps 10a and 10b i.e.

$$\forall i, \forall j \in \{k+1, \cdots, m\}, \quad \mathcal{M}_j^{(i)}(.) = \widetilde{\mathcal{M}}_j^{(i)}(.) \quad . \tag{5.2.20}$$

Algorithm 5.2.1: *Flexible GCRO-DR(m, k) and Flexible GMRES-DR(m, k)*

1: choose m, k, tol and x_0

2: $r_0 = b - Ax_0$, $\beta = \|r_0\|$, $v_1 = r_0/\beta$, $c = \beta e_1$, $i \leftarrow 0$

3: Apply FGMRES(m) to obtain $\bar{H}_m$, Z_m, V_{m+1} such that $AZ_m = V_{m+1}\bar{H}_m$, $y^* = \arg\min_{y\in\mathbb{C}^m} \|c - \bar{H}_m y\|$, $x_0^{(0)} = x_0 + Z_m y^*$, $r_0^{(0)} = b - Ax_0^{(0)} = V_{m+1}(c - \bar{H}_m y^*)$, $W_m = V_m$

4: **while** $\left\|r_0^{(i)}\right\| > \|b\| \times tol$ **do** $i \leftarrow i + 1$

FGCRO-DR

5a: Compute k harmonic Ritz vectors of $AZ_m W_m^\dagger$ with respect to range (W_m) and store them in Y_k. Define P_k such that $Y_k = W_m P_k$.

6a: $Q\ R = \bar{H}_m P_k$

7a: $W_k = W_m P_k R^{-1}$

8a: $V_k = V_{m+1} Q$

9a: $Z_k = Z_m P_k R^{-1}$

10a: Apply $m-k$ flexible preconditioned Arnoldi steps with $(I_n - V_k V_k^H)A$ and $v_{k+1} = r_0^{(i-1)} / \left\|r_0^{(i-1)}\right\|$ such that $(I_n - V_k V_k^H)A\left[z_{k+1}, \ldots, z_m\right] = \left[v_{k+1}, \ldots, v_{m+1}\right]\bar{H}_{m-k}$ with $z_j = \mathcal{M}_j^{(i)}(v_j)$

11a: Set $\bar{H}_m = \begin{bmatrix} I_k & V_k^H A Z_{m-k} \\ 0_{m-k+1\times k} & \bar{H}_{m-k} \end{bmatrix}$ yielding $A\left[z_1, \ldots, z_m\right] = \left[v_1, \ldots, v_{m+1}\right]\bar{H}_m$ and define $W_m = \left[W_k \quad V_m(1:n, k+1:m)\right]$

FGMRES-DR

5b: Compute k harmonic Ritz vectors of $AZ_m V_m^H$ with respect to range (V_m) and store them in Y_k. Define P_k such that $Y_k = V_m P_k$.

6b: $QR = \begin{bmatrix} \begin{bmatrix} P_k \\ 0_{1\times k} \end{bmatrix} & c - \bar{H}_m y^* \end{bmatrix}$

7b: $\bar{H}_k = Q^H \bar{H}_m Q(1:m, 1:k)$

8b: $V_{k+1} = V_{m+1} Q$

9b: $Z_k = Z_m Q(1:m, 1:k)$

10b: Apply $m-k$ flexible preconditioned Arnoldi steps with A and v_{k+1} while maintaining orthogonality to V_k such that $A\left[z_{k+1}, \ldots, z_m\right] = \left[v_{k+1}, \ldots, v_{m+1}\right]\bar{H}_{m-k}$ with $z_j = \mathcal{M}_j^{(i)}(v_j)$ and $V_{m+1}^H V_{m+1} = I_{m+1}$

11b: Set $\bar{H}_m = \begin{bmatrix} \begin{bmatrix} \bar{H}_k \\ 0_{m-k\times k} \end{bmatrix} & \begin{bmatrix} B_{k\times m-k} \\ \bar{H}_{m-k} \end{bmatrix} \end{bmatrix}$ yielding $A\left[z_1, \ldots, z_m\right] = \left[v_1, \ldots, v_{m+1}\right]\bar{H}_m$

12: $y^* = \arg\min_{y\in\mathbb{C}^m} \|c - \bar{H}_m y\|$ with $c = V_{m+1}^H r_0^{(i-1)}$

13: $x_0^{(i)} = x_0^{(i-1)} + Z_m y^*$

14: $r_0^{(i)} = b - Ax_0^{(i)}$

15: **end while**

5.2.8 Equivalent preconditioning matrix

Definition 5.2.6. Equivalent preconditioning matrix. Suppose that $V_p = [v_1, \cdots, v_p] \in \mathbb{C}^{n\times p}$ and $Z_p = [\mathcal{M}_1(v_1), \cdots, \mathcal{M}_p(v_p)] \in \mathbb{C}^{n\times p}$ obtained during a cycle of a flexible method with (standard or deflated) restarting (with $1 \le p \le m < n$) are both of full rank i.e. $\operatorname{rank}\left(V_p\right) = \operatorname{rank}\left(Z_p\right) = p$. We will then denote by $M_{V_p} \in \mathbb{C}^{n\times n}$ a nonsingular equivalent preconditioning matrix defined as

$$Z_p \stackrel{\text{def}}{=} M_{V_p}\ V_p. \tag{5.2.21}$$

Such a matrix represents the action of the nonlinear operators $\mathcal{M}_j$ on the set of vectors v_j (with $j = 1, \cdots, p$). It can be chosen e.g. as $M_{V_p} = [Z_p\ \underline{Z_p}][V_p\ \underline{V_p}]^{-1}$ where $\underline{Z_p}$ (respectively $\underline{V_p}$) denotes an orthogonal complement of Z_p (respectively V_p) in $\mathbb{C}^n$.

5.2.9 Relations between Z_m and W_m and $\widetilde{Z}_m$ and $\widetilde{V}_m$

We denote by $M_{W_m}^{(0)}$ and $\widetilde{M}_{\widetilde{V}_m}^{(0)}$ the equivalent preconditioning matrices used in the initialization phase of both algorithms (step 3 in Algorithm 5.2.1). With this notation we remark that the following relations hold

$$Z_m = M_{W_m}^{(0)} W_m = \widetilde{Z}_m = \widetilde{M}_{\widetilde{V}_m}^{(0)} \widetilde{V}_m. \tag{5.2.22}$$

We first analyze the relation between $\widetilde{Z}_m$ and $\widetilde{V}_m$.

Lemma 5.2.7. *At the end of the i-th cycle of the FGMRES-DR method $\widetilde{Z}_m$ and $\widetilde{V}_m$ satisfy*

$$\widetilde{Z}_m = \widetilde{M}_{\widetilde{V}_m}^{(i)} \widetilde{V}_m = [\widetilde{M}_{\widetilde{V}_m}^{(i-1)} \widetilde{V}_k,\ \widetilde{M}_{\widetilde{V}_{m-k}}^{(i)} \widetilde{V}_{m-k}]. \tag{5.2.23}$$

Proof. The initialization phase leads to the relation $\widetilde{Z}_m = \widetilde{M}_{\widetilde{V}_m}^{(0)} \widetilde{V}_m$. We suppose that at the end of the $i-1$th cycle the following relation holds: $\widetilde{Z}_m = \widetilde{M}_{\widetilde{V}_m}^{(i-1)} \widetilde{V}_m$. At step 9b of the i-th cycle $\widetilde{Z}_k$ is defined as

$$\widetilde{Z}_k = \widetilde{Z}_m \widetilde{Q}_{m\times k} = \widetilde{M}_{\widetilde{V}_m}^{(i-1)} \widetilde{V}_m \widetilde{Q}_{m\times k} = \widetilde{M}_{\widetilde{V}_m}^{(i-1)} \widetilde{V}_k.$$

The proof is then completed since $\widetilde{Z}_{m-k} = [\widetilde{\mathcal{M}}_{k+1}^{(i)}(\widetilde{v}_{k+1}), \cdots, \widetilde{\mathcal{M}}_m^{(i)}(\widetilde{v}_m)] = \widetilde{M}_{\widetilde{V}_{m-k}}^{(i)} \widetilde{V}_{m-k}$ at the end of step 10b. □

The next lemma details a relation between Z_m and W_m that is satisfied in the FGCRO-DR method.

Lemma 5.2.8. *At the end of the i-th cycle of the FGCRO-DR method Z_m and W_m satisfy*

$$Z_m = M_{W_m}^{(i)} W_m = [M_{W_m}^{(i-1)} W_k,\ M_{W_{m-k}}^{(i)} W_{m-k}]. \tag{5.2.24}$$

Proof. The initialization phase leads to the relation $Z_m = M_{W_m}^{(0)} W_m$. We suppose that at the end of the $i-1$th cycle the following relation holds: $Z_m = M_{W_m}^{(i-1)} W_m$. At step 9a of the i-th cycle Z_k is defined as

$$\begin{aligned} Z_k &= Z_m P_k R^{-1} \\ &= M_{W_m}^{(i-1)} W_m P_k R^{-1} \\ &= M_{W_m}^{(i-1)} W_k. \end{aligned}$$

The proof is then completed since $Z_{m-k} = [\mathcal{M}_{k+1}^{(i)}(w_{k+1}), \cdots, \mathcal{M}_m^{(i)}(w_m)] = M_{W_{m-k}}^{(i)} W_{m-k}$ at the end of step 11a. □

Lemma 5.2.7 and 5.2.8 show that $\widetilde{Z}_m$, $\widetilde{V}_m$, Z_m and W_m satisfy relations that will play a central role in Subsection 5.2.10. We investigate next the relation between Z_m and V_m.

Lemma 5.2.9. *At the end of the i-th cycle of the FGCRO-DR method Z_m and V_m satisfy*

$$[AZ_k, Z_{m-k}] = [V_k,\ M_{V_{m-k}}^{(i)} V_{m-k}]. \tag{5.2.25}$$

Proof. We use the relation $AZ_k = V_k$ satisfied in the FGCRO-DR method shown in Proposition 5.2.5. The proof is then completed since $Z_{m-k} = [\mathcal{M}_{k+1}^{(i)}(v_{k+1}), \cdots, \mathcal{M}_m^{(i)}(v_m)] = M_{V_{m-k}}^{(i)} V_{m-k}$ at the end of step 11a. □

We conclude this section by presenting a technical lemma related to the FGMRES-DR method.

Lemma 5.2.10. *During the i-th cycle of the FGMRES-DR method, $\widetilde{v}_{k+1}$ satisfies the following relation*

$$\widetilde{v}_{k+1} = \widetilde{\widetilde{v}}_{k+1}/||\widetilde{\widetilde{v}}_{k+1}|| \quad with \quad \widetilde{\widetilde{v}}_{k+1} = \Pi_{[\widetilde{Y}_k]^\perp}\widetilde{r}_0^{(i-1)} \tag{5.2.26}$$

where $\widetilde{r}_0^{(i-1)} = b - A\widetilde{x}_0^{(i-1)}$ denotes the residual obtained at the end of the $(i-1)$-th cycle.

Proof. Using Proposition 5.2.3 and relation (5.2.8) we obtain

$$\begin{aligned}
\widetilde{\widetilde{v}}_{k+1} &= \widetilde{V}_{m+1}\bar{\rho} = \widetilde{r}_0^{(i-1)} - \widetilde{V}_{m+1}\begin{bmatrix}\widetilde{Q}_{m\times k}\\ 0_{1\times k}\end{bmatrix}\begin{bmatrix}\widetilde{Q}_{m\times k}\\ 0_{1\times k}\end{bmatrix}^H \widetilde{V}_{m+1}^H\widetilde{r}_0^{(i-1)},\\
\widetilde{\widetilde{v}}_{k+1} &= \widetilde{V}_{m+1}\bar{\rho} = \widetilde{r}_0^{(i-1)} - \widetilde{V}_m\widetilde{Q}_{m\times k}(\widetilde{V}_m\widetilde{Q}_{m\times k})^H\widetilde{r}_0^{(i-1)}.
\end{aligned}$$

Since $\widetilde{V}_m\widetilde{Q}_{m\times k}$ has orthonormal columns this last expression now becomes

$$\bar{v}_{k+1} = \Pi_{[\widetilde{V}_m\widetilde{Q}_{m\times k}]^\perp}\widetilde{r}_0^{(i-1)}.$$

Since $\widetilde{Q}_{m\times k}$ is the orthogonal factor of the QR decomposition of $\widetilde{P}_k$, we obtain

$$\text{range}\left(\widetilde{V}_m\widetilde{P}_k\right) = \text{range}\left(\widetilde{V}_m\widetilde{Q}_{m\times k}\right).$$

Since from Lemma 5.2.4 $\widetilde{Y}_k = \widetilde{V}_m\widetilde{P}_k$, the proof is then completed. □

5.2.10 Analysis of the FGMRES-DR and FGCRO-DR methods

Lemma 5.2.8 has already described an important property satisfied by W_m in the FGCRO-DR method proposed in Algorithm 5.2.1. We will analyze further the relation between the FGMRES-DR and FGCRO-DR methods. The next theorem states that the two flexible methods generate the same iterates in exact arithmetic under some conditions involving notably two vectors.

Theorem 5.2.11. *We denote by $r_0^{(i)} = b - Ax_0^{(i)}$ the residual obtained at the end of the i-th cycle of the FGCRO-DR method (see step 14 of Algorithm 5.2.1). We suppose that Definition 5.2.6 holds and that the same equivalent preconditioning matrix is obtained at the end of the i-th cycle of both FGCRO-DR and FGMRES-DR algorithms i.e. $M_{W_m}^{(i)} = \widetilde{M}_{\widetilde{V}_m}^{(i)}$. Under this assumption the harmonic Ritz vectors $\widetilde{Y}_k$ and Y_k can be chosen equal during the $i+1$-th cycle. If in addition there exists a real-valued positive coefficient η_{i+1} such that*

$$\Pi_{[Y_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]^\perp} A\mathcal{M}_{k+1}^{(i+1)}(\Pi_{Y_k^\perp}r_0^{(i)}/\left\|\Pi_{Y_k^\perp}r_0^{(i)}\right\|) = \eta_{i+1}\ \Pi_{[Y_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]^\perp} A\mathcal{M}_{k+1}^{(i+1)}(r_0^{(i)}/\left\|r_0^{(i)}\right\|) \tag{5.2.27}$$

in the FGCRO-DR algorithm, then both algorithms generate the same iterates in exact arithmetic and

$$\text{range}\left(V_{m+1}\right) = \text{range}\left(\widetilde{V}_{m+1}\right), \tag{5.2.28}$$

$$\text{range}\left(Z_m\right) = \text{range}\left(\widetilde{Z}_m\right), \tag{5.2.29}$$

with

$$V_{m+1} = [\widetilde{V}_{k+1}\widehat{Q}, v_{k+2}, \cdots, v_{m+1}], \quad \widetilde{V}_{m+1} = [\widetilde{V}_{k+1}, v_{k+2}, \cdots, v_{m+1}], \tag{5.2.30}$$

$$Z_m = [\widetilde{Z}_{k+1}\widehat{X}, z_{k+2}, \cdots, z_m], \quad \widetilde{Z}_m = [\widetilde{Z}_{k+1}, z_{k+2}, \cdots, z_m], \tag{5.2.31}$$

where $\widehat{Q} \in \mathbb{C}^{(k+1)\times(k+1)}$ is a unitary matrix and $\widehat{X} \in \mathbb{C}^{(k+1)\times(k+1)}$ is a nonsingular triangular matrix.

Proof. The whole proof is performed in three parts assuming that we analyze the $i+1$-th cycle of each algorithm. Suppose that at the beginning of the $i+1$-th cycle (step 4) there exist a unitary matrix $\widehat{Q} \in \mathbb{C}^{(k+1)\times(k+1)}$ and a nonsingular matrix $\widehat{X} \in \mathbb{C}^{(k+1)\times(k+1)}$ such that the following relations hold

$$V_{k+1} = \widetilde{V}_{k+1}\widehat{Q}, \tag{5.2.32}$$
$$Z_{k+1} = \widetilde{Z}_{k+1}\widehat{X}, \tag{5.2.33}$$
$$[v_{k+2}, \cdots, v_{m+1}] = [\widetilde{v}_{k+2}, \cdots, \widetilde{v}_{m+1}], \tag{5.2.34}$$
$$[z_{k+2}, \cdots, z_m] = [\widetilde{z}_{k+2}, \cdots, \widetilde{z}_m]. \tag{5.2.35}$$

We will then prove the existence of a unitary matrix $\widehat{Q}' \in \mathbb{C}^{(k+1)\times(k+1)}$ and of a nonsingular matrix $\widehat{X}' \in \mathbb{C}^{(k+1)\times(k+1)}$ such that at the end of the $i+1$-th cycle

$$V_{k+1} = \widetilde{V}_{k+1}\widehat{Q}', \tag{5.2.36}$$
$$Z_{k+1} = \widetilde{Z}_{k+1}\widehat{X}', \tag{5.2.37}$$
$$[v_{k+2}, \cdots, v_{m+1}] = [\widetilde{v}_{k+2}, \cdots, \widetilde{v}_{m+1}], \tag{5.2.38}$$
$$[z_{k+2}, \cdots, z_m] = [\widetilde{z}_{k+2}, \cdots, \widetilde{z}_m]. \tag{5.2.39}$$

Regarding FGCRO-DR we assume that at the beginning of the $i+1$-th cycle (step 4)

$$\text{range}\,(W_m) = \text{range}\,(V_m). \tag{5.2.40}$$

We will also prove that relation (5.2.40) holds at the end of the $i+1$-th cycle. Note that relations (5.2.28), (5.2.29) and (5.2.40) are obviously satisfied before the first cycle, because steps 1 to 3 are identical in both algorithms yielding $V_{m+1} = \widetilde{V}_{m+1}$, $Z_m = \widetilde{Z}_m$ and $W_m = V_m$. Finally a consequence of (5.2.32), (5.2.34), (5.2.33) and (5.2.35) is that the residual of the linear system $Ax = b$ in both algorithms are equal at the beginning of the $i+1$-th cycle i.e. $r_0^{(i)} = \widetilde{r}_0^{(i)}$. We will denote r_0 this residual for ease of notation.

Part I - Steps 5a and 5b In this part, we prove that we can choose $\widetilde{Y}_k = Y_k$ with $Y_k = W_m P_k = \widetilde{V}_m \widetilde{P}_k$.

FGCRO-DR Let $y_j = W_m p_j$ be the j-th column of Y_k. Since y_j is a harmonic Ritz vector of $AZ_m W_m^\dagger$ with respect to range (W_m), the following relation holds (see Definition (5.2.1))

$$Z_m^H A^H \left(AZ_m p_j - \theta_j W_m p_j\right) = 0. \tag{5.2.41}$$

Due to (5.2.33) and (5.2.35) there exists a nonsingular matrix $X \in \mathbb{C}^{m\times m}$ that relates Z_m and $\widetilde{Z}_m$

$$Z_m = \widetilde{Z}_m X. \tag{5.2.42}$$

Using the last equality (5.2.42), the harmonic Ritz relation (5.2.41) now becomes

$$X^H \widetilde{Z}_m^H A^H \left(A\widetilde{Z}_m X p_j - \theta_j W_m p_j\right) = 0.$$

From Lemma 5.2.8 and relation (5.2.42) we deduce

$$X^H \widetilde{Z}_m^H A^H \left(A\widetilde{Z}_m X p_j - \theta_j M_{W_m}^{(i)^{-1}} Z_m p_j\right) = 0,$$
$$X^H \widetilde{Z}_m^H A^H \left(A\widetilde{Z}_m X p_j - \theta_j \widetilde{M}_{\widetilde{V}_m}^{(i)^{-1}} \widetilde{Z}_m X p_j\right) = 0,$$

where we have used explicitly the assumption on the equivalent preconditioning matrix obtained at the end of the i-th cycle i.e. $M_{W_m}^{(i)} = \widetilde{M}_{\widetilde{V}_m}^{(i)}$. Next, the application of Lemma 5.2.7 leads to

$$X^H \widetilde{Z}_m^H A^H \left(A\widetilde{Z}_m \widetilde{V}_m^H \widetilde{V}_m X p_j - \theta_j \widetilde{V}_m X p_j\right) = 0. \tag{5.2.43}$$

Since X is nonsingular the last equality proves that $\widetilde{V}_m X p_j$ is a harmonic Ritz vector of $A\widetilde{Z}_m\widetilde{V}_m^H$ with respect to range $\left(\widetilde{V}_m\right)$ associated to the Ritz value θ_j. From relations (5.2.41) and (5.2.43) we deduce that the harmonic Ritz vectors can be chosen to be equal and correspond to the same harmonic Ritz values. In this case they notably satisfy the following equality

$$\forall j \in \{1, \cdots, k\}, \quad \widetilde{V}_m X p_j \quad = \quad W_m p_j \quad \text{i.e.} \quad \widetilde{p}_j = X p_j. \tag{5.2.44}$$

We will then denote by $Y = \widetilde{Y}_k = Y_k$ the k harmonic Ritz vectors computed in either FGCRO-DR or FGMRES-DR. We assume that the harmonic Ritz values θ_j $(1 \leq j \leq k)$ are non zero.

Part IIa - Steps 6a to 10a, 6b to 10b We show that at the end of steps 10a and 10b the following relations hold: range (V_{k+1}) = range $\left(\widetilde{V}_{k+1}\right)$ = range $\left([Y, r_0^{(i)} / \left\|r_0^{(i)}\right\|]\right)$. This result will help us to prove the existence of the matrix $\widehat{Q}'$ introduced in relation (5.2.36).

FGCRO-DR Since $AZ_m P_k = V_k R$ (Proposition 5.2.5), we deduce from Lemma 5.2.4

$$\left[V_k, r_0^{(i)} / \left\|r_0^{(i)}\right\|\right] \quad = \quad \left[Y, r_0^{(i)} / \left\|r_0^{(i)}\right\|\right] \begin{bmatrix} \text{diag}(\theta_1, \ldots \theta_k) R^{-1} & 0_{k\times 1} \\ \left\|r_0^{(i)}\right\| \beta_{1\times k} R^{-1} & 1 \end{bmatrix}. \tag{5.2.45}$$

This relation leads to the following result

$$\text{range}\,(V_{k+1}) = \text{range}\left([Y, r_0^{(i)} / \left\|r_0^{(i)}\right\|]\right). \tag{5.2.46}$$

Similarly $W_{k+1} = [W_k, \frac{r_0^{(i)}}{\left\|r_0^{(i)}\right\|}]$ can be written as, using $Y = W_m P_k$

$$\begin{aligned} [W_k, r_0^{(i)} / \left\|r_0^{(i)}\right\|] \quad &= \quad [W_m P_k R^{-1}, \frac{r_0^{(i)}}{\left\|r_0^{(i)}\right\|}] \\ &= \quad [Y R^{-1}, r_0^{(i)} / \left\|r_0^{(i)}\right\|] \\ &= \quad [Y, r_0^{(i)} / \left\|r_0^{(i)}\right\|] \begin{bmatrix} R^{-1} & 0_{k\times 1} \\ 0_{1\times k} & 1 \end{bmatrix}. \end{aligned} \tag{5.2.47}$$

From relations (5.2.47) and (5.2.46) we deduce that

$$\text{range}\,(W_{k+1}) = \text{range}\,(V_{k+1}). \tag{5.2.48}$$

This last result also proves that range (W_m) = range (V_m) at the end of the cycle.

FGMRES-DR In Proposition 5.2.3 we have shown that

$$\text{range}\left(\widetilde{V}_{k+1}\right) = \text{range}\left([Y, r_0^{(i)} / \left\|r_0^{(i)}\right\|]\right). \tag{5.2.49}$$

Since both V_{k+1} and $\widetilde{V}_{k+1}$ have orthonormal columns, we deduce from (5.2.46) and (5.2.49) that there exists a unitary matrix $\widehat{Q}'$ such that

$$V_{k+1} = \widetilde{V}_{k+1} \widehat{Q}' \tag{5.2.50}$$

which proves the relation proposed in equation (5.2.36).

Part IIb - Steps 6a to 10a, 6b to 10b We show that at the end of steps 10a and 10b the following relation holds: $\text{range}\,(Z_{k+1}) = \text{range}\left(\widetilde{Z}_{k+1}\right)$. This result will help us to prove the existence of the matrix $\widehat{X}'$ introduced in relation (5.2.37).

FGCRO-DR Concerning $Z_{k+1} = [Z_k, z_{k+1}]$, there exists a nonsingular matrix $M^{(i+1)}_{[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]} \in \mathbb{C}^{n\times n}$ (see Definition 5.2.6) such that

$$Z_{k+1} \quad = \quad M^{(i+1)}_{[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]}[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|].$$

If $T \in \mathbb{C}^{(k+1)\times(k+1)}$ denotes the following triangular matrix

$$T = \begin{bmatrix} R & 0_{k\times 1} \\ 0_{1\times k} & 1 \end{bmatrix}$$

due to relation (5.2.47), $Z_{k+1}T$ can be written as

$$Z_{k+1}T \quad = \quad M^{(i+1)}_{[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]. \tag{5.2.51}$$

FGMRES-DR Similarly from Lemma 5.2.7, $\widetilde{Z}_{k+1}$ can be expressed as

$$\widetilde{Z}_{k+1} \quad = \quad \widetilde{M}^{(i+1)}_{\widetilde{V}_{k+1}}\widetilde{V}_{k+1}$$

where $\widetilde{M}^{(i+1)}_{\widetilde{V}_{k+1}} \in \mathbb{C}^{n\times n}$ is nonsingular (see Definition 5.2.6). If $\widetilde{T} \in \mathbb{C}^{(k+1)\times(k+1)}$ denotes the following triangular matrix

$$\widetilde{T} = \widetilde{R}\begin{bmatrix} I_k & 0_{k\times 1} \\ 0_{1\times k} & 1/\left\|r_0^{(i)}\right\| \end{bmatrix}$$

$\widetilde{Z}_{k+1}\widetilde{T}$ can be expressed as

$$\widetilde{Z}_{k+1}\widetilde{T} \quad = \quad \widetilde{M}^{(i+1)}_{\widetilde{V}_{k+1}}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|] \tag{5.2.52}$$

thanks to the relation (5.2.15). Relations (5.2.51) and (5.2.52) characterize $Z_{k+1}T$ and $\widetilde{Z}_{k+1}\widetilde{T}$ with respect to $[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]$. We can further improve this result by showing the following equality

$$M^{(i+1)}_{[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|] \quad = \quad \widetilde{M}^{(i+1)}_{\widetilde{V}_{k+1}}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]. \tag{5.2.53}$$

Lemma 5.2.8 and Lemma 5.2.7 respectively give us two useful relations for $M^{(i+1)}_{[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]$ and $\widetilde{M}^{(i+1)}_{\widetilde{V}_{k+1}}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]$ i.e.

$$M^{(i+1)}_{[W_k, r_0^{(i)}/\left\|r_0^{(i)}\right\|]}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|] \quad = \quad [M^{(i)}_{W_m}Y,\ \mathcal{M}^{(i+1)}_{k+1}(r_0^{(i)}/\left\|r_0^{(i)}\right\|)], \tag{5.2.54}$$

$$\widetilde{M}^{(i+1)}_{\widetilde{V}_{k+1}}[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|] \quad = \quad [\widetilde{M}^{(i)}_{\widetilde{V}_m}Y,\ \widetilde{\mathcal{M}}^{(i+1)}_{k+1}(r_0^{(i)}/\left\|r_0^{(i)}\right\|)]. \tag{5.2.55}$$

Using the assumption on the equivalent preconditioning matrix obtained at the end of the i-th cycle i.e. $M^{(i)}_{W_m} = \widetilde{M}^{(i)}_{\widetilde{V}_m}$ we have

$$M^{(i)}_{W_m}Y \quad = \quad \widetilde{M}^{(i)}_{\widetilde{V}_m}Y. \tag{5.2.56}$$

The fact that identical (possibly nonlinear) preconditioning operators are used in steps 10a and 10b of Algorithm 5.2.1 (see relation (5.2.20)) allows us to write

$$\mathcal{M}_{k+1}^{(i+1)}(r_0^{(i)}/\left\|r_0^{(i)}\right\|) = \widetilde{\mathcal{M}}_{k+1}^{(i+1)}(r_0^{(i)}/\left\|r_0^{(i)}\right\|). \tag{5.2.57}$$

Relations (5.2.56) and (5.2.57) finally show the relation (5.2.53). Consequently from relations (5.2.51), (5.2.52) and (5.2.53) we deduce that there exists a nonsingular matrix $\widehat{X}' \in \mathbb{C}^{(k+1)\times(k+1)}$ such that

$$Z_{k+1} = \widetilde{Z}_{k+1}\widehat{X}'. \tag{5.2.58}$$

This proves the relation proposed in equation (5.2.37). Since T and $\widetilde{T}$ are both triangular, we note that $\widehat{X}' = \widetilde{T}T^{-1}$ is also triangular.

Part IIIa - Steps 10a and 10b We first show that $\widetilde{v}_{k+2} = v_{k+2}$ by expressing these two quantities in function of $r_0^{(i)}$ and Y.

FGCRO-DR The Arnoldi relation (step 10a) yields $v_{k+2} = \bar{v}_{k+2}/||\bar{v}_{k+2}||$, where $\bar{v}_{k+2} = (I_n - v_{k+1}v_{k+1}^H)(I_n - V_kV_k^H)A\mathcal{M}_{k+1}^{(i+1)}(r_0^{(i)}/\left\|r_0^{(i)}\right\|)$. Since from Proposition 5.2.5 $V_k^H r_0^{(i)} = 0$ in the $i+1$-th cycle, $(I_n - v_{k+1}v_{k+1}^H)$ and $(I_n - V_kV_k^H)$ commute and from Part IIa of the proof, the following expression can be derived

$$\bar{v}_{k+2} = \Pi_{V_{k+1}^\perp} A\mathcal{M}_{k+1}^{(i+1)}(r_0^{(i)}/\left\|r_0^{(i)}\right\|) = \Pi_{[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]^\perp} A\mathcal{M}_{k+1}^{(i+1)}(r_0^{(i)}/\left\|r_0^{(i)}\right\|). \tag{5.2.59}$$

FGMRES-DR The following expression for $\widetilde{v}_{k+2} = \widetilde{\bar{v}}_{k+2}/||\widetilde{\bar{v}}_{k+2}||$ is obtained using Lemma 5.2.10

$$\widetilde{\bar{v}}_{k+2} = (I_n - \widetilde{V}_{k+1}\widetilde{V}_{k+1}^H)A\mathcal{M}_{k+1}^{(i+1)}(\widetilde{v}_{k+1}) = \Pi_{[Y, r_0^{(i)}/\left\|r_0^{(i)}\right\|]^\perp} A\mathcal{M}_{k+1}^{(i+1)}(\Pi_{Y^\perp} r_0^{(i)}/\left\|\Pi_{Y^\perp} r_0^{(i)}\right\|). \tag{5.2.60}$$

Due to the assumption (5.2.27) of Theorem 5.2.11 we deduce from (5.2.59) and (5.2.60) that $\bar{v}_{k+2} = \eta_{i+1}\ \widetilde{\bar{v}}_{k+2}$ with η_{i+1} positive and therefore $v_{k+2} = \widetilde{v}_{k+2}$.

Part IIIb - Steps 10a and 10b In this part we continue the analysis of the Arnoldi procedure with flexible preconditioning and show that $v_{k+2+j} = \widetilde{v}_{k+2+j}$ for $j = 1, \ldots, m-k-1$.

For the case $j = 1$, we introduce $\bar{v}_{k+3}$ and $\widetilde{\bar{v}}_{k+3}$ such that $v_{k+3} = \bar{v}_{k+3}/||\bar{v}_{k+3}||$ and $\widetilde{v}_{k+3} = \widetilde{\bar{v}}_{k+3}/||\widetilde{\bar{v}}_{k+3}||$. The application of the Arnoldi procedure in both algorithms leads to

$$\begin{aligned}
\bar{v}_{k+3} &= (I_n - v_{k+2}v_{k+2}^H)(I_n - V_{k+1}V_{k+1}^H)\ A\mathcal{M}_{k+2}^{(i+1)}(\bar{v}_{k+2}) \\
\widetilde{\bar{v}}_{k+3} &= (I_n - \widetilde{v}_{k+2}\widetilde{v}_{k+2}^H)(I_n - \widetilde{V}_{k+1}\widetilde{V}_{k+1}^H)\ A\mathcal{M}_{k+2}^{(i+1)}(\widetilde{\bar{v}}_{k+2}).
\end{aligned}$$

Thus from Parts II and IIIa of the proof we obtain that v_{k+3} and $\widetilde{v}_{k+3}$ are equal. The proof can then be completed by induction.

Results from Parts II and III justify the relation (5.2.38) i.e. $[v_{k+2}, \cdots, v_{m+1}] = [\widetilde{v}_{k+2}, \cdots, \widetilde{v}_{m+1}]$. Consequently from Lemma 5.2.7, Lemma 5.2.9 and relation (5.2.20) we deduce the relation (5.2.39). This finally shows the main relations (5.2.28) and (5.2.29) of Theorem 5.2.11 that are satisfied at the end of the $i+1$-th cycle. □

First consequence of Theorem 5.2.11

Corollary 5.2.12. *If the same flexible preconditioning operators are used in both Arnoldi procedures (steps 10a and 10b of Algorithm 5.2.1) and if at each cycle i there exists a real-valued positive coefficient η_i such that*

$$\Pi_{[Y,r_0^{(i-1)}/\left\|r_0^{(i-1)}\right\|]^\perp} A\mathcal{M}_{k+1}^{(i)}(\Pi_{Y^\perp} r_0^{(i-1)}/\left\|\Pi_{Y^\perp} r_0^{(i-1)}\right\|) = \eta_i\ \Pi_{[Y,r_0^{(i-1)}/\left\|r_0^{(i-1)}\right\|]^\perp} A\mathcal{M}_{k+1}^{(i)}(r_0^{(i-1)}/\left\|r_0^{(i-1)}\right\|),$$

or equivalently (from relations (5.2.59) and (5.2.60)) such that $\widetilde{\bar{v}}_{k+2} = \eta_i\ \bar{v}_{k+2}$, FGCRO-DR and FGMRES-DR are algebraically equivalent.

Proof. We have already emphasized that $M_{W_m}^{(0)} = \widetilde{M}_{\widetilde{V}_m}^{(0)}$ in relation (5.2.22). In Theorem 5.2.11 we have analyzed the i+1-th cycle of both algorithms assuming that $M_{W_m}^{(i)} = \widetilde{M}_{\widetilde{V}_m}^{(i)}$. First we have proved in Part IIb the relation (5.2.53) and secondly, respectively in Parts IIIa and IIIb, that $[v_{k+2}, \cdots, v_m] = [\widetilde{v}_{k+2}, \cdots, \widetilde{v}_m]$ and $[z_{k+2}, \cdots, z_m] = [\widetilde{z}_{k+2}, \cdots, \widetilde{z}_m]$. Consequently the same equivalent preconditioner matrix is obtained at the end of the $i+1$-th cycle i.e. $M_{W_m}^{(i+1)}$ and $\widetilde{M}_{\widetilde{V}_m}^{(i+1)}$ can be chosen equal. We deduce that FGCRO-DR and FGMRES-DR are algebraically equivalent. □

About GCRO-DR and GMRES-DR

We propose a second consequence of Theorem 5.2.11 analyzed now with a fixed preconditioning matrix M.

Corollary 5.2.13. *When a fixed right preconditioner is used, the GCRO-DR and GMRES-DR methods sketched in Algorithm 5.2.1 are algebraically equivalent.*

Proof. We denote by M the fixed right preconditioning operator. A straightforward reformulation of Lemma 5.2.8 in this context leads to the relation $Z_m = MW_m$ in GCRO-DR. Exploiting now Lemma 5.2.4 allows us to derive the following relation that holds during the $i+1$-th cycle:

$$AMW_m P_k = AMY = [Y, r_0^{(i)}] \begin{bmatrix} \mathrm{diag}(\theta_1, \ldots, \theta_k) \\ \beta_{1\times k} \end{bmatrix}.$$

Thus

$$\Pi_{[Y,r_0^{(i)}]^\perp} AMY = 0. \tag{5.2.61}$$

Due to (5.2.61) and Part IIIa of Theorem 5.2.11 we deduce the following development

$$\begin{aligned} \bar{v}_{k+2} &= \Pi_{[Y,r_0^{(i)}]^\perp} AM(r_0^{(i)} - YY^\dagger r_0^{(i)}), \\ \bar{v}_{k+2} &= \Pi_{[Y,r_0^{(i)}]^\perp} AM\Pi_{Y^\perp} r_0^{(i)}, \\ \bar{v}_{k+2} &= \widetilde{\bar{v}}_{k+2}. \end{aligned}$$

By induction it is possible to deduce the rest of the proof regarding $\bar{v}_{k+j}, j > 2$. Using $\mathrm{range}\left(\widetilde{V}_{k+1}\right) = \mathrm{range}\,(V_{k+1})$ obtained in Part IIa we deduce that

$$\mathrm{range}\left(\widetilde{V}_m\right) = \mathrm{range}\,(V_m) = \mathrm{range}\,(W_m)\,. \tag{5.2.62}$$

A straightforward reformulation of Lemma 5.2.7 leads to the relation $\widetilde{Z}_m = M\widetilde{V}_m$ in GMRES-DR. From relation (5.2.62) we finally deduce that

$$\mathrm{range}\left(\widetilde{Z}_m\right) = \mathrm{range}\,(Z_m)\,.$$

Consequently the minimization problem $\min\left\|r_0^{(i)} - AZ_m y\right\|$ leads to the same solution for both algorithms at each cycle: GCRO-DR and GMRES-DR sketched in Algorithm 5.2.1 are thus algebraically equivalent. □

A numerical illustration

In this section we intend to illustrate the results shown in Section 5.2.10 and 5.2.10 on a simple numerical example. We consider a real symmetric positive definite matrix $A = Q\ D\ Q^T$ of size 200 with Q orthonormal and D diagonal with entries ranging from 10^{-4} to 1. The spectrum of A contains eigenvalues of small magnitude[1] and consequently the use of deflation techniques should improve the convergence rate of Krylov subspace methods if the harmonic Ritz values of smallest modulus are taken into account. In this experiment we consider a polynomial preconditioner represented by two iterations of unpreconditioned GMRES for the solution of $Ax = b$ with b given by $b = \frac{Ae}{\|Ae\|_2}$ ($e \in \mathbb{R}^{200}$ denoting the vector with all components equal to one) starting from a zero initial guess. Figure 5.1 shows the histories of convergence of various flexible methods minimizing over a subspace of same dimension i.e. respectively FGMRES(10), FGMRES-DR(10,6), FGCRO-DR(10,6) and full flexible GMRES with such a variable preconditioner. Flexible methods with deflated restarting are found to be efficient since they are close to the full flexible GMRES method in terms of performances. We also remark that the convergence histories of FGCRO-DR(10,6) and FGMRES-DR(10,6) are different. According to Corollary 5.2.12 we compute the scalar product of v_{k+2} and $\widetilde{v}_{k+2}$ (which are both vectors of unit norm) to determine the cosinus of the angle between these two vectors. The values are reported in Table 5.1 for the first five cycles. With such a variable preconditioner it is found that the methods are not equivalent in the first cycle already since the collinearity condition between v_{k+2} and $\widetilde{v}_{k+2}$ is not fulfilled. The situation is similar during the next cycles and it explains why different convergence histories for FGMRES-DR(10,6) and FGCRO-DR(10,6) observed in Figure 5.1 are obtained in such a case. As expected from Section 5.2.10, if a fixed right preconditioner is used, the convergence histories of GMRES-DR(10,6) and GCRO-DR(10,6) are found to be exactly similar (results not shown here). In such a case v_{k+2} and $\widetilde{v}_{k+2}$ fulfill the collinearity condition; this is confirmed in Table 5.1 when a diagonal preconditioning is used.

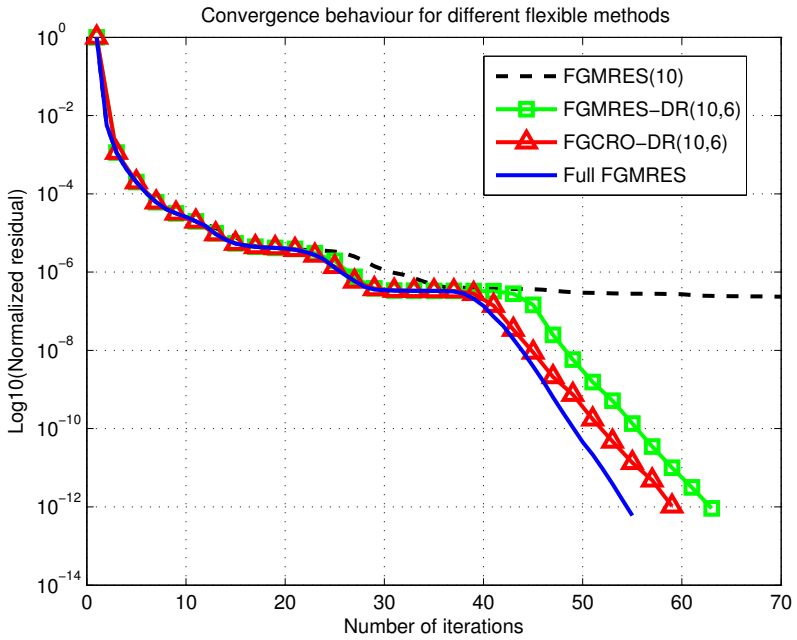


Figure 5.1: *Convergence histories of different flexible methods applied to $Ax = b$ where $A \in \mathbb{R}^{200\times200}$ is symmetric positive definite with some eigenvalues of small magnitude.*

[1]The eigenvalues of A are logarithmically spaced $(10^{-4}, 10^{-3}, 10^{-2})$ and linearly distributed between 0.02 and 1 with step 1/200.

Table 5.1: *Scalar product $v_{k+2}^T \widetilde{v}_{k+2}$ during the first five cycles of FGCRO-DR(10,6) when solving the linear system considered in Section 5.2.10. Case of a variable preconditioner (two iterations of GMRES) and of a fixed right preconditioner (diagonal preconditioning).*

Cycle index	1	2	3	4	5
Variable preconditioner	0.92	0.89	0.45	0.90	0.90
Fixed right preconditioner	1.00	1.00	1.00	1.00	1.00

5.2.11 Further features of FGCRO-DR(m, k)

In this section we first compare FGCRO-DR(m, k) with FGMRES-DR(m, k) presented in Algorithm 5.2.1 from both a computational and storage point of view. Then we detail how subspace recycling can be used in FGCRO-DR(m, k) when solving a sequence of linear systems.

5.2.12 Computational cost

We first analyze the computational cost related to the generalized eigenvalue problem to deduce harmonic Ritz information and then detail the total cost of the proposed method.

Harmonic Ritz information

The generalized eigenvalue problem (5.2.41) can be also written as

$$\bar{H}_m^H \ \bar{H}_m y = \theta \bar{H}_m^H V_{m+1}^H W_m y. \tag{5.2.63}$$

Since $W_m = [W_{k+1}, v_{k+2}, \cdots, v_m]$ $V_{m+1}^H W_m$ can be decomposed at the end of the cycle as

$$V_{m+1}^H W_m = \begin{bmatrix} V_{k+1}^H \ W_{k+1} & 0_{(k+1)\times(m-k-1)} \\ 0_{(m-k-1)\times(k+1)} & I_{m-k-1} \\ 0_{1\times(k+1)} & 0_{1\times(m-k-1)} \end{bmatrix}, \tag{5.2.64}$$

where the structure of the $(k+1)\times(k+1)$ block $V_{k+1}^H \ W_{k+1}$ is as follows

$$V_{k+1}^H \ W_{k+1} = \begin{bmatrix} V_k^H \ W_k & V_k^H \ w_{k+1} \\ v_{k+1}^H \ W_k & v_{k+1}^H \ w_{k+1} \end{bmatrix} = \begin{bmatrix} V_k^H \ W_k & 0_{k\times 1} \\ v_{k+1}^H \ W_k & 1 \end{bmatrix}.$$

$V_k^H \ W_k$ is a $k \times k$ matrix that satisfies the following relation at the end of the i-th cycle

$$(V_k^H W_k)^{(i)} = Q^H \ (V_{m+1}^H W_m)^{(i-1)} \ P_k R^{-1}$$

where the superscript is related to the cycle index. Thus storing the $(m+1)\times m$ matrix $(V_{m+1}^H W_m)^{(i-1)}$ can be used to obtain $(V_k^H W_k)^{(i)}$ at a cost that is independent of n. Next we analyze how to compute efficiently $v_{k+1}^H \ W_k$ during the i-th cycle. From relation (5.2.18) shown respectively in Lemma 5.2.4 and Proposition 5.2.5, we deduce the relation

$$v_{k+1}^H V_k R \ = \ v_{k+1}^H W_k R \ \mathrm{diag}(\theta_1, \ldots, \theta_k) + v_{k+1}^H r_0^{(i-1)} \beta_{1\times k}. \tag{5.2.65}$$

Due to Proposition 5.2.5 and the definition of v_{k+1}, we have $v_{k+1}^H V_k = 0$. Thus we finally obtain that

$$v_{k+1}^H W_k \ = \ -\|(c - \bar{H}_m y^*)^{(i-1)}\|_2 \ \beta_{1\times k} \ (R \ \mathrm{diag}(\theta_1, \ldots, \theta_k))^{-1}, \tag{5.2.66}$$

where $\beta_{1\times k}$ is obtained from relation (5.2.19) which does only involve projected quantities. This allows us to deduce $v_{k+1}^H W_k$ at a cost independent of n. From this development we draw two important consequences from a computational point of view. First, $(V_{m+1}^H W_m)^{(i)}$ can be obtained recursively at a cost that is independent of the problem size n. Secondly storing W_m (that would represent m additional vectors of size n) is not mandatory, only $V_{m+1}^H W_m$ - matrix of size $(m+1)\times m$ - is required.

Cost of a cycle

We summarize in Table 5.2 the main computational costs associated with each generic cycle of FGMRES-DR(m,k) and FGCRO-DR(m,k). In FGCRO-DR(m,k), an Arnoldi method based on the modified Gram-Schmidt procedure has been assumed[2]. We have only included the costs proportional to the size of the original problem n which is supposed to be much greater than m and k. We denote respectively by op_A and op_M the floating point operation counts for the matrix-vector product and the preconditioner application.

Table 5.2: *Computational cost of a generic cycle of FGMRES-DR(m,k) and FGCRO-DR(m,k). C represents the total cost of FGCRO-DR(m,k) and corresponds to $C = (m-k)(op_M + op_A) + n(2(m+1)k + 1 + 2mk + (m-k)(2m+2k+6))$.*

Computation of	FGMRES-DR(m,k)	FGCRO-DR(m,k)
$V_m(:, 1:k+1)$	$2n(m+1)(k+1)$	$2n(m+1)k + n$
$Z_m(:, 1:k)$	$2nmk$	$2nmk$
$V_m(:, k+2:m+1)$	$(m-k)op_A +$ $n(m-k)(2m+2k+5)$	$(m-k)op_A +$ $n(m-k)(2m+2k+6)$
$Z_m(:, k+1:m)$	$(m-k)op_M$	$(m-k)op_M$
Total cost	$C + n\ (m+k+1)$	C

The generalized eigenvalue problem in FGCRO-DR(m,k) has been ignored in Table 5.2 since it can be performed at a cost independent of n as outlined in Section 5.2.12. Furthermore the computation of c (required at step 12 of Algorithm 5.2.1) has not been included in Table 5.2 since in both methods it can be obtained at a cost independent of n (see Proposition 3 in [58] for FGMRES-DR). From Table 5.2 we deduce that FGCRO-DR(m,k) requires slightly less operations per cycle than FGMRES-DR(m,k).

5.2.13 Storage requirements

We only consider the storage proportional to the size of the original problem n. Similarly as in FGMRES-DR(m,k) (see [58, Section 3.2.2]), if the matrix multiplications $V_{m+1}Q$ and $Z_m P_k R^{-1}$ at steps 8a and 9a of Algorithm 5.2.1 are performed *in place* (i.e. respectivel overwriting V_k and Z_k y), FGCRO-DR(m,k) only requires the storage of Z_m and V_{m+1} which corresponds to $(2m+1)$ vectors of length n. The same storage cost is needed in FGMRES-DR(m,k) as detailed in [58].

5.2.14 Solution of sequence of linear systems

As advocated in [95], GCRO-DR(m,k) is suited for the solution of a sequence of slowly changing linear systems defined as $A^l x^l = b^l$ where both the matrix $A^l \in \mathbb{C}^{n\times n}$ and the right-hand side $b^l \in \mathbb{C}^n$

[2]In FGCRO-DR(m,k) (step 10a of Algorithm 5.2.1) the action of $(I_n - V_k V_k^H)$ requires $\sum_{j=k+1}^{m}(4nk+n)$ operations, the Arnoldi method based on modified Gram-Schmidt requires $\sum_{j=k+1}^{m}\sum_{i=k+1}^{j}(4n)$ operations whereas norm computation and normalization cost $\sum_{j=k+1}^{m}(3n)$ operations. In FGMRES-DR(m,k) (step 10b of Algorithm 5.2.1) the Arnoldi method based on modified Gram-Schmidt requires $\sum_{j=k+1}^{m}\sum_{i=1}^{j}(4n)$ operations due to maintaining orthogonality to V_k whereas norm computation and normalization cost $\sum_{j=k+1}^{m}(3n)$ operations.

change from one system to the next, and the linear systems may typically not be available simultaneously. Next, we analyze how subspace recycling can be used in FGCRO-DR(m,k). We suppose that FGCRO-DR(m,k) has been applied for the solution of a given linear system (indexed by $s-1$) in this sequence and that appropriate subspaces to be recycled Z_k^{s-1} and W_k^{s-1} have been selected during a given cycle. As explained in Proposition 5.2.5, the relations $A^{s-1}Z_k^{s-1} = V_k^{s-1}$ with ${V_k^{s-1}}^H V_k^{s-1} = I_k$ and $range(W_k^{s-1}) = range(V_k^{s-1})$ are then supposed to hold. Proposition 5.2.14 details how to consider subspace recycling in the initial phase of FGCRO-DR(m,k), when solving the new linear system $A^s x^s = b^s$ with x_0 as an initial guess.

Proposition 5.2.14. *Suppose that Z_k^{s-1} and W_k^{s-1} are defined from solving a previous linear system $A^{s-1}x^{s-1} = b^{s-1}$ with FGCRO-DR(m,k) and that $A^s x^s = b^s$ is the new linear system to be solved. In the initial phase of flexible GCRO with deflated restarting and subspace recycling, the relation $A^s Z_k^s = V_k^s$ with ${V_k^s}^H V_k^s = I_k$ holds with matrices $V_k^s, Z_k^s \in \mathbb{C}^{n\times k}$ defined as*

$$\begin{aligned} V_k^s &= Q, \\ Z_k^s &= Z_k^{s-1}R^{-1} \end{aligned}$$

with $QR = A^s Z_k^{s-1}$ where $Q \in \mathbb{C}^{n\times k}$ has orthonormal columns and $R \in \mathbb{C}^{k\times k}$ is upper triangular. In addition we define $W_k^s \in \mathbb{C}^{n\times k}$ as $W_k^s = W_k^{s-1}R^{-1}$.

Proof. By using information related to the reduced QR factorization of $A^s Z_k^{s-1}$ and respectively the relation $A^{s-1}Z_k^{s-1} = V_k^{s-1}$, we easily obtain

$$\begin{aligned} A^s Z_k^s &= A^s Z_k^{s-1}R^{-1} = Q, \\ &= V_k^s. \end{aligned}$$

Since Q has orthonormal columns, V_k^s satisfies ${V_k^s}^H V_k^s = I_k$. Finally $W_k^s = W_k^{s-1}R^{-1}$ is imposed to make sure that the relation shown in Lemma 5.2.8 will hold at the end of the initial phase of FGCRO-DR(m,k) with subspace recycling. □

Algorithm 5.2.2: *Initial generation of V_{m+1}^s, Z_m^s and W_m^s when subspace recycling is used to solve $A^s x^s = b^s$*

1 Suppose that V_k^{s-1}, Z_k^{s-1} and W_k^{s-1} are defined from solving a previous linear system $A^{s-1}x^{s-1} = b^{s-1}$ and satisfy $A^{s-1}Z_k^{s-1} = V_k^{s-1}$ with ${V_k^{s-1}}^H V_k^{s-1} = I_k$ and $range(W_k^{s-1}) = range(V_k^{s-1})$;
2 $r_0 = b^s - A^s x_0$;
3 $Q\ R = A^s Z_k^{s-1}$;
4 $V_k^s = Q$;
5 $Z_k^s = Z_k^{s-1}R^{-1}$;
6 $W_k^s = W_k^{s-1}R^{-1}$;
7 $x_0^{(0)} = x_0 + Z_k^s {V_k^s}^H r_0$;
8 $r_0^{(0)} = r_0 - V_k^s {V_k^s}^H r_0$;
9 Apply $m-k$ flexible preconditioned Arnoldi steps with $(I_n - V_k^s {V_k^s}^H)A^s$ and $v_{k+1}^s = r_0^{(0)} / \left\| r_0^{(0)} \right\|$ such that $(I_n - V_k^s {V_k^s}^H)A^s \left[z_{k+1}^s, \ldots, z_m^s \right] = \left[v_{k+1}^s, \ldots, v_{m+1}^s \right] \bar{H}_{m-k}$ with $z_j^s = \mathcal{M}_j^{(i)}(v_j^s)$;
10 $d^* = \arg\min_{d\in Z_m^s} \left\| r_0^{(0)} - A^s d \right\|$, $x_0^{(1)} = x_0^{(0)} + d^*$, $r_0^{(1)} = b^s - A^s x_0^{(1)}$;
11 $W_m^s = \left[W_k^s \quad V_m^s(1:n, k+1:m) \right]$;

In the case of a sequence where only the right-hand sides are changing, we note that the reduced QR factorization (step 2 in Algorithm 5.2.2) is not required. The complete construction of the initial

generation of subspaces V^s_{m+1}, Z^s_m, W^s_m is sketched in Algorithm 5.2.2. Once V^s_{m+1}, Z^s_m and W^s_m have been obtained, the main cycle of FGCRO-DR(m,k) (lines 4 to 15 of Algorithm 5.2.1) can be applied straightforwardly. Subspace recycling can thus be easily used in FGCRO-DR(m,k) to solve sequences of linear systems.

A numerical illustration

As a numerical illustration we consider sequences of linear systems arising from the finite difference discretization of multidimensional elliptic partial differential equations (isotropic Laplace operator) posed on the $[0,1]^d$ hypercube with homogeneous Dirichlet boundary conditions. These sequences correspond to situations where only the right-hand sides are changing for a given dimension d. An efficient solution method is of primary interest in certain applications related to e.g. financial engineering, molecular biology or quantum dynamics [15, 16]. In the numerical experiments reported here (performed in Matlab) we have used second order finite difference discretization schemes leading to sparse matrices with at most $2d+1$ nonzero elements per row. We analyze the performances of various flexible methods used with four iterations of unpreconditioned GMRES as a preconditioner. This polynomial preconditioner is a variable nonlinear function which thus requires a flexible Krylov subspace method as an outer method [113]. Table 5.3 collects the number of matrix-vector products of some flexible methods minimizing over a subspace of same dimension respectively i.e. FGMRES(20), FGMRES-DR(20,10), FGCRO-DR(20,10) and FGCRO-DR(20,10) with subspace recycling. Using deflation helps to improve the convergence rate of flexible GMRES in this application since a reduction of approximately 20% to 25% in terms of matrix-vector products is obtained for FGMRES-DR(20,10) independently of the dimension d. FGCRO-DR(20,10) leads to numbers of matrix-vector products which are similar to FGMRES-DR(20,10) although the convergence histories are found to be different. Finally using both deflation and recycling in FGCRO-DR leads to a significant decrease in terms of matrix-vector products. A reduction in the range of 40% to 45% is indeed obtained versus another flexible Krylov subspace method with deflated restarting (FGMRES-DR(m,k)). This can be considered as a primary advantage over FGMRES-DR(m,k) since FGMRES-DR(m,k) does not allow subspace recycling. It nicely extends to the flexible setting the advantage of GCRO-DR versus GMRES-DR previously illustrated in [95]. We note that the resulting method is factorization free and mostly relies on matrix-vector products, a nice feature if distributed memory platforms are targeted to address numerical problems of larger size in higher dimension.

Table 5.3: *Solution of a d-dimensional elliptic partial differential equation problem on a 16^d grid with homogeneous Dirichlet boundary conditions ($d = 2, \cdots, 5$). Total number of matrix-vector products ($\#Mvp$) required to solve a sequence of twelve linear systems with different flexible methods. The variable preconditioner is based on four iterations of unpreconditioned GMRES. The stopping criterion corresponds to a reduction of six orders of magnitude of the normalized residual in the Euclidean norm. Harmonic Ritz values of smallest modulus have been considered when deflating.*

Grid Problem size (n)	16^2 (225)	16^3 (3375)	16^4 (50625)	16^5 (759375)
Method	$\#Mvp$	$\#Mvp$	$\#Mvp$	$\#Mvp$
FGMRES(20)	972	1176	1272	1128
FGMRES-DR(20,10)	732	948	1020	876
FGCRO-DR(20,10) (no recycling)	732	948	1020	876
FGCRO-DR(20,10) (with recycling)	457	541	547	529

5.2.15 Conclusion and perspectives

In this section we have studied a new minimum residual norm subspace method with deflated restarting that allows flexible preconditioning based on the GCRO subspace method. The resulting method named

FGCRO-DR has been presented together with FGMRES-DR, a recently proposed algorithm of the same family but based on the GMRES subspace method. A theoretical comparison analysis of both algorithms has been performed in Subsection 5.2.7, where Theorem 5.2.11 - the main result of this section - proves the algebraic equivalence of FGMRES-DR and FGCRO-DR if a certain collinearity condition holds at each cycle. Corollary 5.2.13 has also proved that GMRES-DR and GCRO-DR are algebraically equivalent when a fixed right preconditioner is used. Furthermore we have carefully analyzed the computational cost of a given cycle of FGCRO-DR and have shown that FGCRO-DR is nearly as expensive as FGMRES-DR in terms of operations. FGCRO-DR offers the additional advantage to be suitable for the solution of sequences of slowly changing linear systems (where both the matrix and right-hand side can change) through subspace recycling.

In [26] variants of FGCRO-DR have been proposed which only differ in the formulation of the projected generalized eigenvalue problem. As future work we plan to investigate the numerical properties of these variants on realistic problems of large size for both single and multiple left or right-hand side situations. Of interest are applications related to e.g. steady or unsteady simulations of nonlinear equations [25] or stochastic finite element methods [42, 129] in three dimensions where variable preconditioning using approximate solvers has to be usually considered. We also note that when all right-hand sides are available simultaneously and when the matrix is fixed, block subspace methods may be also suitable. Thus a perspective could be to propose a block variant of FGCRO-DR.

Acknowledgments

We would like to thank the referees and the associate editor for their careful readings and valuable suggestions that helped us to improve our manuscript significantly. We also thank Iain S. Duff and Xavier Pinel for fruitful discussions and comments. The participation of the first author in the initial preparation of this article was guaranteed thanks to grant CNPq-473420/2007-4, coordinated by Professor Nelson Maculan. The first author would like to acknowledge the warm welcome he received at CERFACS, in the Parallel Algorithms Team, during his sabbatical leave where the initial work was completed.

Chapter 6

Conclusions

The earth imaging problem related to geophysics consists of one of the most difficult problems in today's high performance computing community. The challenges are spread on several domains as geophysics, optimization, numerical analysis, among others. In this thesis we focused on aspects concerning the forward problem arising from the full waveform inversion technique, most notably the multi source scenario. We focused on the solution of very challenging large sparse non-Hermitian linear systems arising from the discretization of the three-dimensional Helmholtz equation. Due to the large size of the problem we chose then the use of Krylov iterative solvers which are known for optimal memory control and good scalability in massively parallel environment, but require flexible preconditioners based on multigrid techniques to be efficient in terms of computational cost. The goals of this thesis were originally the following: develop a flexible block Krylov iterative solver able to efficiently handle the multiple right-hand sides situation, investigate subspace recycling possibilities for accelerating the convergence when multiple models are used, and the implementation of a modular software using object orientation paradigms (specially polymorphism) for solving the proposed problem using the method we developed in a massively parallel environment .

Regarding the development of a flexible block Krylov iterative solver, we focused on a GMRES based method due its minimal residual property and monotonically decreasing behaviour of the norm of the residual. In a first moment we extended the needed theory on flexible Krylov solvers for the multiple right-hand sides in order to obtain a detailed and unified vision on this scenario. The conclusions we drew from this extension greatly influenced and facilitated the understanding of the multiple right-hand sides case, to later on exploit deflation techniques. Deflation techniques are recognized to be necessary when using Krylov solvers for multiple right-hand sides. We were then able to extend the deflation techniques already available in the literature and finally propose a variant of a BGMRES based block Krylov subspace method able to perform deflation at the beginning of every iteration based on information associated with the singular values of the scaled residual, a method we named in this thesis deflated minimal block residual, or DMBR. To justify the new variant, we performed numerical experiments on both small academical cases and on large real life application. Most notably, we compared DMBR with other methods known to be efficient for solving the forward problem arising from the full waveform inversion application. In this comparison using the SEG/EAGE Overthrust velocity field, we observed a speedup in terms of computational time of up to 30%, for low frequency cases using hundreds of right-hand sides.

We implemented an object oriented code using FORTRAN03 targeting performance in a massively parallel environment using standard MPI partitioning, as well as being as close as possible to FORTRAN90 programming. This was achieved by using only the most basic FORTRAN03 object orientation features, which are ultimately *encapsulated* and thus not imposing any software engineering knowledge upon the final user. The code is flexible enough to accommodate changes without requiring rewriting of current routines. The numerical experiments in this chapter show not only the interest of using such a software in large scale real life application, but we also reinforced the conclusion that deflation techniques are essential when using block Krylov solvers for the multisources scenario.

Later on we investigated a family of flexible methods performing *deflated restart* for the single right-

hand side case, a technique to accelerate the convergence of inner-outer iterative methods using harmonic Ritz values. We have discussed the similarities between the recently proposed FGMRES-DR with the method we then proposed, FGCRO-DR. We show that although in the fixed preconditioner setting both methods are algebraically equivalent, in the flexible setting this is not true unless a collinearity condition holds for every iteration, a phenomena unlike to happen in practice. We verified with numerical experiments the different convergence history of the two methods. Later on we showed that when using FGCRO-DR, it is possible to recycle information when solving sequence of linear systems, a feature not present in FGMRES-DR. We performed numerical experiments on simple academic problems showing that in this case the gain obtained through subspace recycling can be large.

Therefore, we consider the enhancements we proposed for the iterative solver are significant for the solution of the forward problem arising from the full waveform inversion optimization problem. However, some further investigation could bring an even more significant improvement. Whereas the strategy for handling multiple right-hand sides has shown great performance for hundreds of right-hand sides, in the earth imaging scenario often it is required the solution of tens of thousands of right-hand sides. For solving problems with such a large number of right-hand sides, we consider in our future research to investigate the combination of recycling strategies with deflation strategies, that is, partitioning the tens of thousands of right-hand sides into *"clusters"* of hundreds of right-hand sides, solving each cluster using block strategies and then using recycling strategies to recycle information from the solution of one cluster to improve the convergence of the solution of the next cluster of right-hand sides.

Another possibility for our future research is the use of block and subspace recycling strategies inside the preconditioner. For the moment, only our solver uses block and deflation strategies, whereas the preconditioner performs independent iterations of GMRES for each right-hand side. It was not yet investigated in the literature the use of block and deflation techniques in the coarse level or smoother of the geometric multigrid. Likewise, we are unaware of any result related to the use of block and deflation techniques in the geometric multigrid cycle preconditioner when using complex shifted Laplacian operator, a result that could be indeed very interesting for the inverse problem. Similarly, subspace recycling could greatly speed up the solution on coarse levels, which remains the most expensive part of the preconditioner.

Additionally, in our future research we would like to consider advanced high-order accurate schemes and discontinuous Galerkin discretization techniques as well as other PDE formulations of the forward problem (e.g. visco-elastic equation). Notably the recent reformulation of the Helmholtz equation using the Rytov decomposition seems attractive to us, as our academical numerical experiments involving the solution of the advection-diffusion equation showed the interest of using deflation techniques for this scenario. Since the convergence of iterative solvers is highly dependent on the numerical properties of the coefficient matrix being used, it is of our interest to investigate the behaviour of both recycling strategies and block deflation strategies in this scenario. In the same sense, although in this thesis we investigate the behaviour of block Krylov iterative solvers when using a two-level multigrid preconditioner, we would like to investigate in the near future the deflation and recycling behaviour for Helmholtz equation when using non-multigrid based preconditioners, such as the sweeping preconditioner, CARP-CG, or low-rank approximation techniques as the characteristics of the properties of the preconditioned Krylov subspace depends intrinsically on the matrix and (variable) preconditioner particularities.

Appendix A

User Guide

Introduction

We describe now with some more details the library we implemented using FORTRAN03 and MPI for solving the forward problem arising from the acoustic full waveform inversion method. These libraries were written targeting compatibility with *Depth Imaging and Velocity Analysis* (or DIVA), a proprietary software maintained by TOTAL. However, a priori they could be used for any high performance computing purpose, not necessarily related to acoustic full waveform inversion. As it was discussed in Section 4.6, one of the main concerns was to produce routines to be used by programmers who are used to FORTRAN90 instructions, but would produce extra modularity and code reuse due to object orientation, reason why we choose the FORTRAN03 language. The current source code is compatible with Intel Fortran Compiler version 12.1.5 20120612.

In this chapter we thus discuss basic aspects of the technical knowledge necessary to make the transition between procedural to object oriented programming, to then show with few details how to use the libraries we implemented. We suppose that the reader is familiar with both FORTRAN90 and MPI use. In the current version most of the routines associated with MPI are hidden from the final user, but we mention that only the standard data partition is used.

This appendix is organized as follows. First we introduce in the Section FORTRAN03 Basic Guidelines the features from FORTRAN03 that are not contained in FORTRAN90 standard. In Section Polymorphism and Inheritance we explain these two key concepts. Once they are established, we describe the libraries we proposed to be used by DIVA software: the *libEina*, *libOperator* and *libSolver*. We proposed a fourth library called *libDiscretizer* but unlike the other libraries we propose, *libDiscretizer* is not independent of DIVA software, and for this reason we do not address it here. Finally, in Section Conclusions we propose the final remarks of this appendix.

We highlight that this is not a complete reference guide, it instead contains an introductory material for using the libraries we proposed.

FORTRAN03 Basic Guidelines

In this section we briefly discuss aspects of FORTRAN03 programming which are not contained in FORTRAN90 standard. Since the purpose is to have as few changes as possible (the FORTRAN90 syntax is compatible with FORTRAN03), we just use a limited set of extra features from FORTRAN03, namely the keywords EXTENDS, ABSTRACT, PASS and NOPASS, DEFERRED and CLASS, which we describe next.

EXTENDS Keyword

Defines a new type as an extension of another type. This is the basic building block for *inheritance* and *polymorphism*. In Figure A.1, both **C_ChildType1** and **C_ChildType2** have all the variables declared inside the **C_ParentType**, but they also have their own exclusive variables. Notice that both **C_ChildType1** and **C_ChildType2** could be extended to a third generation type. In addition, the variables child1 (respectively, child2) has *multiple types*, that is, it is a **C_ParentType** and a **C_ChildType1** (respectively, **C_ChildType2**) at the same time.

```
TYPE ::  C_ParentType
  CHARACTER(LEN=128)   ::   Name1
  INTEGER              ::   IProperty1
  INTEGER              ::   IProperty2
END TYPE C_ParentType

TYPE, EXTENDS(C_ParentType) ::  C_ChildType1
  CHARACTER(LEN=128)   ::   Name2
  REAL                 ::   RProperty1
  INTEGER              ::   IProperty3
END TYPE C_ChildType1

TYPE, EXTENDS(C_ParentType) ::  C_ChildType2
  CHARACTER(LEN=128)   ::   Name3
  COMPLEX              ::   ZProperty1
END TYPE C_ChildType2
```

```
TYPE(C_ChildType1) ::  child1
TYPE(C_ChildType2) ::  child2

child1%Name1        =   'This is valid'
child2%Name1        =   'This is also valid'

child1%IProperty1   =   4     !  Again, valid
child2%IProperty1   =   4     !  and this also

child1%RProperty1   =   5.0   !  and this also...
child2%RProperty1   =   5.0   !  But this is invalid!
```

Figure A.1: *Example of* **EXTENDS** *usage*

ABSTRACT Keyword

Used when defining a type. If a type is defined as being **ABSTRACT**, then it cannot be used for any purpose except being extended to another type using the **EXTENDS** keyword. In Figure A.2 we have an illustration of the use of **ABSTRACT** keyword.

```
TYPE, ABSTRACT ::  C_AbsParentType
  CHARACTER(LEN=128)   ::   Name4
  COMPLEX              ::   ZProperty2
END TYPE C_AbsParentType

TYPE, EXTENDS(C_AbsParentType) ::  C_ChildType3
  CHARACTER(LEN=128)   ::   Name5
  COMPLEX              ::   ZProperty3
END TYPE C_ChildType3
```

```
!  ------------ Example 1 ------------
TYPE(C_ParentType) ::  parenttype
parenttype%Name1        =   'This is valid'
parenttype%IProperty1   =   4
!  ------------ Example 2 ------------
TYPE(C_ChildType3) ::  child3
child3%Name4        =   'This is valid'
child3%Name5        =   'This is also valid'
child3%ZProperty2   =   CMPLX(1.0,1.0)
child3%ZProperty3   =   CMPLX(1.0,1.0)
!  ------------ Example 3 ------------
!  This example does not even compile
TYPE(C_AbsParentType) ::  absparenttype
absparenttype%Name3        =   'Invalid'
absparenttype%ZProperty2   =   CMPLX(1.0,1.0)
```

Figure A.2: *Example of* **ABSTRACT** *usage, using the types defined in Figure A.1*

A priori the usefulness of **ABSTRACT** may not be clear, but it is essential to the use of **DEFERRED PROCEDURES**. Just as an example, consider that an abstract type is just an interface for a type and not a type itself.

PASS and NOPASS Keyword

As in FORTRAN90, FORTRAN03 allows the use of pointer to functions to be set inside types using the keyword **PROCEDURE**. A **PROCEDURE** acts exactly like a variable belonging to the type it is defined, but it also requires the **CALL** keyword (in case it is a subroutine) and the referred arguments. However, in FORTRAN03 it was introduced the possibility of using the **PASS** and **NOPASS** keywords. If **PASS** is

active, it will send the type itself as one of the arguments of the function, reducing thus the number of arguments that need to be passed when calling the procedure. Figure A.3 illustrates both cases.

```
TYPE:: C_Class1
 CHARACTER(LEN=128)   ::   Name1
 REAL                 ::   RProperty1
 CONTAINS
   PROCEDURE, PASS(arg1) ::  procedure1
END TYPE C_Class1

SUBROUTINE procedure1(arg1)
 CLASS(C_Class1), INTENT(INOUT) ::  arg1
 (...)
END SUBROUTINE procedure1

(...)

TYPE(C_Class1) ::  class1
CALL class1%procedure1()
```

```
TYPE:: C_Class2
 CHARACTER(LEN=128)   ::   Name2
 COMPLEX              ::   ZProperty2
 CONTAINS
   PROCEDURE, NOPASS ::  procedure2
END TYPE C_Class2

SUBROUTINE procedure2(arg2)
 CLASS(C_Class2), INTENT(INOUT) ::  arg2
 (...)
END SUBROUTINE procedure2

(...)

TYPE(C_Class2) ::  class2
CALL class2%procedure2(class2)
```

Figure A.3: *Example of* **PROCEDURE** *usage with PASS and NOPASS. On the left side with* **PASS** *keyword and on the right side with the* **NOPASS** *keyword*

DEFERRED Keyword

Pretty much like the **ABSTRACT** keyword, the **DEFERRED** keyword defines a procedure as an interface. The deferred procedure contains nothing but a name and the type and structure of the types it refers to. It must be declared inside an **ABSTRACT INTERFACE** statement. It is important to notice that a **DEFERRED** procedure can be defined only inside an abstract type, otherwise a compilation error is thrown. In Figure A.4 we show an example of declaration of a deferred procedure.

```
TYPE, ABSTRACT ::  C_Class3
 CHARACTER(LEN=128)   ::   Name3
 REAL                 ::   RProperty3
 CONTAINS
   PROCEDURE, PASS(arg) ::  procedure3
   PROCEDURE(interface_procedure4), DEFERRED, PASS(arg) ::  procedure4
END TYPE C_Class3

ABSTRACT INTERFACE  SUBROUTINE interface_procedure4(arg)
    CLASS(C_Class3), INTENT(INOUT) ::  arg
 END SUBROUTINE interface_procedure4
END INTERFACE

SUBROUTINE procedure3(arg)
 CLASS(C_Class3), INTENT(INOUT) ::  arg

 PRINT *,'=>', arg%Name3
 PRINT *,'=>', arg%RProperty4
END SUBROUTINE procedure3
```

Figure A.4: *Example of deferred procedure usage*

To be able to use a deferred procedure, it has to be implemented before. In Figure A.5 we show two types that extend **C_Class3**, and each implement **procedure4** in a different manner. Both methods are valid and both methods could coexist.

CLASS Keyword

Finally we refer to the **CLASS** declaration. We define the arguments that a specific function or subroutine receives, an argument declared as a **CLASS(C_Class)** can be of type **C_Class** or any other type that extends **C_Class** using the **EXTENDS** keyword. Although not mentioned, we used the **CLASS** declaration

```
TYPE, EXTENDS(C_Class3) ::  C_SubClass1
  COMPLEX ::  ZProperty4
  CONTAINS
    PROCEDURE(procedure4_1), PASS(arg) ::  procedure4
END TYPE C_SubClass1

SUBROUTINE procedure4_1(arg)
  CLASS(C_SubClass1), INTENT(INOUT) ::  arg

  PRINT *,'=>', arg%Name3
  PRINT *,'=>', arg%ZProperty4
END SUBROUTINE procedure4_1

(...)
!  ---------- Example of Use ----------
TYPE(C_SubClass1) ::  subclass1
CALL subclass1%procedure3()
CALL subclass1%procedure4()
```

```
TYPE, EXTENDS(C_Class3) ::  C_SubClass2
  INTEGER ::  IProperty5
  CONTAINS
    PROCEDURE(procedure4_2), PASS(arg) ::  procedure4
END TYPE C_SubClass2

SUBROUTINE procedure4_2(arg)
  CLASS(C_SubClass2), INTENT(INOUT) ::  arg

  PRINT *,'=>', arg%Name3
  PRINT *,'=>', arg%IProperty5
END SUBROUTINE procedure4_2

(...)
!  ---------- Example of Use ----------
TYPE(C_SubClass2) ::  subclass2
CALL subclass2%procedure3()
CALL subclass2%procedure4()
```

Figure A.5: *Implementing a deferred procedure. In this example two types are extended from* **C_Class3** *(from Figure A.4) and each implements* **procedure4** *in a different manner*

in Figure A.3, Figure A.4 and in Figure A.5. Notice that in Figure A.4, **procedure3** receives a **CLASS** (and not a **TYPE**) as argument. This means that any of the following types can be passed as argument: **C_Class3**, **C_SubClass1** and **C_SubClass2** (from Figure A.5).

Polymorphism and Inheritance

We quickly address these two terms which we are going to use from now on. In the previous section we already showed examples of these two features of FORTRAN03, although we did not address them specifically.

Inheritance stands for the ability to draw a hierarchy of types, in which each type possesses all the procedures and variables of the types above them in the hierarchy. For instance, in Figure A.1, we say that the type **C_ChildType1** *inherits* the variables **Name1**, **IProperty1** and **IProperty2** from **C_ParentType**. In Figure A.5, **C_ChildType3** inherits no only **Name3**, **RProperty3** from **C_Class3**, but also the procedures **procedure3** and **procedure4** (although the later is not implemented).

Polymorphism stands for the ability of admitting multiple different types to be used for a specific purpose. In Figure A.4, **procedure3** prints **Name3** and **RProperty4** on the screen. Since due to inheritance **C_Class3** and *every type extending* **C_Class3** possesses these two properties, this function can receive a type **C_Class3** as argument or any type extending **C_Class3**.

Empiric discussion Consider that we have a function called **GMRES** implemented. **GMRES** needs, among other things, a matrix in order to apply the Arnoldi procedure. In practice there are many ways to store a matrix, specially when a massively parallel software is being used. Some of these formats exploit special structures of the matrix, some others aim at avoiding communication between nodes and etc.

However, regardless on how the matrix is stored, some attributes of a matrix always exist. For instance, every matrix has a number of rows, number of nonzeros, etc. Every matrix also should have a *procedure* for performing matrix-vector product. This procedure might be very different from the traditional matrix-vector routine, depending on how the matrix is stored. Nevertheless, it is not important for **GMRES** function to know how the matrix-vector product works, but instead, to receive the result of a matrix-vector product routine.

In this scenario, we could create an abstract type **C_Matrix**. This abstract type would contain some basic information, as the number of rows, the number of the MPI rank, communication handler, etc. This abstract type would contain a **DEFERRED PROCEDURE** called **matrix_vector** which would receive nothing but the vector that needs to be multiplied and the vector in which we want to store the product (as in Figure A.4).

We then extend **C_Matrix** to a type **C_CSR**, a very common matrix format. In this extension we finally add the necessary information for storing the matrix: an array of nonzero elements, array of index of first nonzero element of each row, etc. We then implement the **matrix_vector** procedure for performing a matrix-vector product using these structures.

Aside, we could extend **C_Matrix** to another type, for instance, **C_HarwellBoeing**. Similarly we add the structures for storing the matrix in this new type, and we implement the matrix-vector routine for this data type format.

Now that we have both these types implemented, it would make a lot of sense to pass to **GMRES** a type **C_Matrix** instead of **C_CSR** or **C_HarwellBoeing**. The point is that **GMRES** does not need to know how the matrix-vector product is performed, it just needs to know it can be performed and which arguments it needs to pass to the **matrix_vector** routine.

This was the main idea behind the libraries and modules we proposed. The iterative solver receives a class which acts like a matrix, and then uses it to perform matrix vector products. Additionally, the solver itself is a class, and it uses another solver class as preconditioner. For instance, BGMRES can be preconditioned by BGMRES itself.

The following sections are dedicated to the specification and description of the libraries and their basic usage.

libEina Basic Documentation

We now describe the main classes and types defined in the libEina library.

Modules M_Eina, M_OptimizationFlag and M_Topology

We quickly describe the following modules without attaining to details, as their knowledge is not crucial for using the main routines of this and the next libraries we describe.

M_Eina: This module contains a series of simple routines. Normally they are really simple and are meant to improve the readability of the code. No crucial functions or types to be mentioned, as this module is mainly used internally.

M_OptimizationFlag: Contains the specification of a special kind of flag, the **OFLAG**. It is used in some functions to tweak some very specific optimization options. We do not cover this in this user guide, since it is an advanced topic.

M_Topology: Contains the specification of some constants used in stencil definition.

Module M_Error

Contains the error handler type (**T_Error**) definition and some few functions to control it. This type is meant to be used mainly with the subroutines **Print_Error**, **Print_Warning** and **Print_Bug** which we discuss later in the description of the module **M_Class**.

MODULE M_Error most important functions:

New_Error(id, msg)	a function that returns a type **T_Error** with the passed ID and message.
No_Error()	a function that returns a type **T_Error** containing no error at all
Unexpected_Error()	a function that returns a type **T_Error** containing an error without any specific description

We describe now the **T_Error** type contained in **M_Error** module.

TYPE T_Error

Description:
The error handling type. Used to print errors, warning and to throw back the error to the caller.
Important Components:

INTEGER	**ID**	A UNIQUE error ID
CHARACTER*512	**orig**	A string (normally optional) containing the name of the function that generated this error.
CHARACTER*1024	**msg**	A string (normally mandatory) containing the error message concerning this error
T_Error POINTER	**stack**	This error might have been caused by another error - literally. So it would be a good idea do attach it such that the called will have the full information regarding the causer of the problem.
LOGICAL	**IsFatal**	Logical flag telling if we should immediately abort the run due to this error

Notice that since every **T_Error** has a pointer to another **T_Error**, it is possible to create a complex stack of errors showing where the error originally occurred. As we are going to mention later, the function **Print_Error** will print every error in the stack recursively. The error id, message, and everything else is purely optional. However, the following advices should be kept in mind, as following always the same standards is normally a good idea:

6 digits rule : Create error IDs with exactly 6 digits. Probably no one has a code with more than 999.999 errors possibilities; Better to keep this limit!

Unique ID : Do not use the same ID for different circumstances, even if the error message is the same. If your error message will be print on the screen and the error ID is unique, we can search the ID in the source code; if it is not unique, we will have to figure out which part of the code threw that error.

Precompiler : In the libraries we write, the error handling is done in a separate file. For instance m_diag7pts.err contain the error handling of m_diag7pts.f90. This is because when reading m_diag7pts.f90 we are usually not interested in reading every single possibility of error in a given function. To clean the code, we decided to use:

```
IF (this.NE.that)                                    ERROR_012345
```

where ERROR_012345 is a macro defined in m_diag7pts.err. Note that since the IF is still declared inside m_diag7pts.f90, whoever reads this file will know that if "this" is not equal to "that", an error will be thrown; and if the reader is interested in knowing more about this error, he can easily access m_diag7pts.err and read the full handling of this situation.

Severe Error: By default, the errors are printed only if they happen in the master process - a good idea if you don't want to read "Failed to initialize" once for every processor. However, some errors are quite severe and/or can happen in individual processes. For these errors, just put a negative ID number. These are treated as "severe errors" and can be printed by any process, not necessarily the master.

Fatal: Not every error aborts the run. Some of them might be even expected. For the error abort the run, the variable **IsFatal** Errors that should normally be fatal: - Dimension Mismatch: something should be bigger or smaller than it is; if this error was detected, then it is avoiding a segmentation fault. The normal strategy is to print the dimensions found, the ones expected and abort the run immediately. - Not enough memory: another one that prevents a disaster. This error should always be fatal as after trying to allocate something the code usually expects to use it.

Module M_Class

The most primitive classes that is implemented in this and the subsequent libraries. Every single class we implemented extends this one (or extends a class which extends this one) and must follow the shape established here.

TYPE, ABSTRACT C_Class

Description:
The abstract class, to be inherited by every other class.

Important Components:

INTEGER	**MyID**	Rank of the MPI process
INTEGER	**CommHandler**	Integer containing the Communicator Handler for MPI routines
INTEGER	**MemUsed**	Contains the size in bytes that this object is allocating from the memory at the moment.
INTEGER	**MemNeeded**	Contains the size in bytes that this object will take from the memory if initialized (allocated).
LOGICAL	**IsVerbose**	Tells if this object should print informations.

Important Procedures:

Print_Info()	It is a "hello world" subroutine. It prints some basic info regarding this **C_Class** on the screen. A generic one is provided here, but it should normally be overridden inside the extended class for additional information.
Print_Error()	Useful for printing errors on the screen. If the error is too severe (number below zero) the message is printed for every process regardless if MyID=0 or not.
Destroy	**DEFERRED** Deallocates current object. Do NOT destroy pointers (since they may be used somewhere else)
DestroyAll()	Like Destroy, but also destroys every pointer recursively. This might break your code if not used wisely.
Initialize(MemAvailable)	**DEFERRED** This subroutine considers that all parameters necessary for performing the allocation of the object and all its structures are already set, and that the allocation can proceed. It receives MemAvailable as parameter, which is the maximum memory that the user has available for allocating. It is updated after the allocation of everything.

libOperator Basic Documentation

This library contains the modules for manipulating different operators (or matrices) and also the interpolation and restriction routines used by the geometric multigrid algorithm.

Module M_Operator

This module contains the definition of the abstract **C_Operator** class. This is the class which is going to be used to create the *matrices* later. Most modules and routines which require a coefficients matrix will include this module (and not the ones extending the **C_Operator**).

TYPE, ABSTRACT EXTENDS(C_Class) C_Operator

Description:
C_Operator is an abstract class able to accommodate any kind of function the function $y = f(x)$, given x and y of proper dimensions. We dispose of a certain number of procedures inside this class, but the core procedure is apply(), which does exactly what was specified before: apply the function f on x. In fact, in the apply routine, the variable y is supposed to be INTENT(INOUT), so this function can be performing in fact a $y = f(x, y)$. Nevertheless, we keep this class simplified.
Always have in mind that the purpose is to have this as a parallel operator, that is, to be used in MPI application, therefore several parameters regarding MPI are present here.

Important Components:

INTEGER	**LocalLen**	Basic length of vectors which do not have a communication layer.
INTEGER	**CommLen**	Basic length of vectors which do have a communication layer.
INTEGER(:)	**Neighbor**	Array containing the MPI rank of each process that the operator have to communicate with.

Important Procedures:

Apply(y,yLen,x,xLen,p)	**DEFERRED** The core of the **C_Operator** class. This procedure has to be compulsorily implemented. It simply performs $y = f(x)$, and it depends completely on the purpose of the operator.
GetResidual(y,yLen,b, bLen,x, xLen, p)	**DEFERRED** Dedicated function for performing $y = b - f(x)$. It is important to optimize it because it is quite a common operation and usually an expensive one.
UpdateBound(x, xLen, p)	**DEFERRED** Used to update the communication layer of vectors after applying this operator.

Module M_StencilCollection

Extends the **C_Operator** to the **C_StencilCollection** class, which is also abstract. In this class, however, we already have some assumptions, as for instance, that the referred operator is coming from a finite difference discretization scheme (for the moment, it always assumes it is a three-dimensional discretization).

TYPE, ABSTRACT, EXTENDS(C_Operator) C_StencilCollection

Description:
This is an extension of **C_Operator** which adds some basic structures associated with finite differences discretization three-dimensional discretization schemes.

Important Components:

INTEGER	nLocX, nLocY, nLocZ	Number of points in the local domain respectively in the direction x, y and z. This excludes the communication layer.
INTEGER	nComX, nComY, nComZ	Number of points in the local domain respectively in the direction x, y and z, with communication layer included.

Important Procedures:

UpdateBound(x, xLen, p)	**IMPLEMENTED** This procedure is defined in M_Operator, and implemented here, as for the moment, all the finite difference discretization schemes use exactly the same routine for performing the communication. This might be not true if more sophisticated schemes are implemented in the future.

Module M_Diag7pts

This module contains the definition of an almost matrix-free implementation of a 7 point stencil operator. It is specifically tuned for the Helmholtz operator, discretized with PML boundary condition. We use a special characteristic of such discretized operators and store only the main diagonal, using an economic technique for storing off-diagonal elements in small vectors.

TYPE, **EXTENDS(C_StencilCollection) C_Diag7pts**

Description:
This is the class for storing the the almost matrix-free 7 point stencil operator, tuned for using with the Helmholtz operator with PML boundary conditions.

Important Components:

COMPLEX(:)	A	Contains the points related to the central position in the stencil. If no PML is used, this is the only term to be stored.
COMPLEX(:,2)	A_X	Contains the coefficients for the neighbours in the X axis
COMPLEX(:,2)	A_Y	Contains the coefficients for the neighbours in the Y axis
COMPLEX(:,2)	A_Z	Contains the coefficients for the neighbours in the Z axis

Important Procedures:

Apply(y,yLen,x,xLen,p)	**IMPLEMENTED** Implements the deferred procedure inherited from C_Operator
GetResidual(y,yLen,b, bLen,x, xLen, p)	**IMPLEMENTED** Implements the deferred procedure inherited from C_Operator
Initialize(MemAvailable)	**IMPLEMENTED** Implements the deferred procedure inherited from C_Class
Destroy()	**IMPLEMENTED** Implements the deferred procedure inherited from C_Class

In addition to **C_Diag7pts**, in this module we find the following very important function. In Figure A.6 we show how to use this class and this function.

MODULE M_Diag7pts most important functions:

New_Diag7pts(MyID, Name, o_nlocx, o_nlocy, o_nlocz, CommHandler)	Returns an object of the type **C_Diag7pts**.

```
!  ------------------------------------------------------------------------
!  Suppose that the following is known:
!  MyID: the rank of current MPI process
!  CommHandler:  the communication handler
!  MemAvailable:  current memory available
!  fill_matrix():  a subroutine which receives any C_Operator
!  and fills it coefficients with some values
!  ------------------------------------------------------------------------
(...)
CLASS(C_Operator), POINTER   ::   matrix
TYPE(T_Error)                ::   err
COMPLEX, ALLOCATABLE         ::   vecx(:,:),vecy(:,:)
INTEGER                      ::   xlen,ylen,nlocx,nlocy,nlocz,p
nlocx   =   10
nlocy   =   11
nlocz   =   12
p       =   1
xlen    =   nlocx*nlocy*nlocz
ylen    =   (nlocx+2)*(nlocy+2)*(nlocz+2)

ALLOCATE(vecx(xlen,p))
ALLOCATE(vecy(ylen,p))
vecx = CMPLX(1.0,1.0)
vecy = CMPLX(0.0,0.0)

!  This creates the C_Operator as a C_Diag7pts.  It uses polymorphism.
matrix => New_Diag7pts(   MyID,               &
                          'New Test Matrix',  &
                          nlocx,              &
                          nlocy,              &
                          nlocz,              &
                          CommHandler)

!  Now we initialize the C_Operator we just created.
!  Remember that this C_Operator is ALSO a C_Diag7pts!!
CALL matrix%Initialize(MemAvailable)

!  Now it is initialized and allocated.  We can fill it with values
CALL fill_matrix(matrix)

!  Now we can use it; with apply() function for instance
CALL matrix%Apply(vecy,ylen,vecx,xlen,p,0,err)

!  We are done using this C_Operator.  We just destroy it and finish
CALL matrix%Destroy()
DEALLOCATE(vecx,vecy)
```

Figure A.6: *Example of usage of* **C_Diag7pts**, *using polymorphism*

Module M_Standard27pts

This module contains the definition of a standard implementation of a 27 point stencil operator. So far it has been tested with Operto et. al discretization scheme [93].

TYPE, **EXTENDS(C_StencilCollection) C_Standard27pts**

Description:
This is the class for storing the the almost matrix-free 7 point stencil operator, tuned for using with the Helmholtz operator with PML boundary conditions.

Important Components:

COMPLEX(27,:,:,:)	**A**	For each point (i,j,k), $A(:,i,j,k)$ contains the coefficient of the central point plus the coefficients of its 26 neighbours.

Important Procedures:

Apply(y,yLen,x,xLen,p)	**IMPLEMENTED** Implements the deferred procedure inherited from **C_Operator**

GetResidual(y,yLen,b, bLen,x, xLen, p)	**IMPLEMENTED** Implements the deferred procedure inherited from **C_Operator**
Initialize(MemAvailable)	**IMPLEMENTED** Implements the deferred procedure inherited from **C_Class**
Destroy()	**IMPLEMENTED** Implements the deferred procedure inherited from **C_Class**

As the modules **M_Diag7pts**, this module also contains a **New_Standard27pts** function. In Figure A.7 we show how to use this class and this function. Notice, however, that the only difference lies in the name of the function being called to create the matrix.

MODULE M_Standar27pts most important functions:	
New_Standard27pts(MyID, Name, o_nlocx, o_nlocy, o_nlocz, CommHandler)	Returns an object of the type **C_Standard27pts**.

Module M_Transformation, M_FullInterpolation and M_FullRestriction

We quickly describe the following modules without attaining to details, as their knowledge is not crucial for using the main routines of this and the next libraries we describe.

M_Transformation: Contains the abstract type used for interpolations and restrictions. This is basically meant to accommodate any rectangular matrix. It is used by the C_GeoMultigrid class which we describe later.

M_FullInterpolation: Contains the full weighted interpolation object. Used by C_GeoMultigrid.

M_FullRestriction: Contains the full weighted restriction object. Used by C_GeoMultigrid.

```
!  ------------------------------------------------------------------------
!  Suppose that the following is known:
!  MyID: the rank of current MPI process
!  CommHandler:  the communication handler
!  MemAvailable:  current memory available
!  fill_matrix():  a subroutine which receives any C_Operator
!  and fills it coefficients with some values
!  ------------------------------------------------------------------------
(...)
CLASS(C_Operator), POINTER   ::   matrix
TYPE(T_Error)                ::   err
COMPLEX, ALLOCATABLE         ::   vecx(:,:),vecy(:,:)
INTEGER                      ::   xlen,ylen,nlocx,nlocy,nlocz,p
nlocx   =   10
nlocy   =   11
nlocz   =   12
p       =   1
xlen    =   nlocx*nlocy*nlocz
ylen    =   (nlocx+2)*(nlocy+2)*(nlocz+2)

ALLOCATE(vecx(xlen,p))
ALLOCATE(vecy(ylen,p))
vecx = CMPLX(1.0,1.0)
vecy = CMPLX(0.0,0.0)

!  This creates the C_Operator as a C_Standard27pts.  It uses polymorphism.
matrix => New_Standard27pts(   MyID,                &
                               'New Test Matrix',   &
                               nlocx,               &
                               nlocy,               &
                               nlocz,               &
                               CommHandler)

!  Now we initialize the C_Operator we just created.
!  Remember that this C_Operator is ALSO a C_Standard27pts!!
CALL matrix%Initialize(MemAvailable)

!  Now it is initialized and allocated.  We can fill it with values
CALL fill_matrix(matrix)

!  Now we can use it; with apply() function for instance
CALL matrix%Apply(vecy,ylen,vecx,xlen,p,0,err)

!  We are done using this C_Operator.  We just destroy it and finish
CALL matrix%Destroy()
DEALLOCATE(vecx,vecy)
```

Figure A.7: *Example of usage of* **C_Standard27pts**, *using polymorphism. This example is identical to Figure A.6, except that here we call* **New_Standard27pts** *instead of* **New_Diag7pts**

libSolver Basic Documentation

In this library we implement several iterative solvers as well as preconditioners. Thanks to polymorphism, every solver implemented here can also be used as a preconditioner.

Module M_Solver

This module contains the definition of the abstract **C_Solver** class. This is the class which is going to be used to create iterative solver classes and routines later.

TYPE, ABSTRACT, EXTENDS(C_Class) C_Solver

Description:
The solver class has to be as flexible as possible to accommodate every possible future implementation, of an iterative solver, therefore only the strictly generic entities are declared here. For the sake of generality also, the preconditioner is as well considered as a solver in the sense that they also come from a **C_Solver** class.
In a bigger picture, solvers can be seen as a function; they receive a right-hand side b, an initial guess x, and a coefficient matrix A to then compute $x = f(x, b)$, which is supposedly an approximation of $A^{-1}b$. So far, nothing prevents the use of this class for implementing direct solvers, but this has never been the original intention.

Important Components:

C_Operator POINTER	**MyOperator**	This object defines several internal values for the solver. For instance, the allocation of the vectors $\mathcal{V}_j$ in BFGMRES are dependant of **MyOperator%LocalLen** and **MyOperator%CommLen**
INTEGER	**Maxp**	Maximum number of right-hand sides that this solver is going to deal with at once. Used to compute the maximum size of some structures before allocating it.
LOGICAL	**IsVariable**	Every object which inherits from **C_Solver** should be considered as a flexible solver by default. Set this flag to **.FALSE.** in a specific solver if this is not true, since sometimes it is possible to optimize memory cost and operations when using fixed preconditioner.

Important Procedures:

Apply(x,xLen,b,bLen,p)	**DEFERRED** As in the **C_Operator** type, this is the core of the **C_Solver** class. It simply starts the solver execution.

Recalling the inheritance that we discussed earlier, since **C_Solver** extends **C_Class**, it also possesses the component **IsVerbose**. This flag is particularly useful for the solver, although it is also present in any **C_Operator**, for instance. If it is active, the solver will print the history of the relative residual on the screen, for instance. Also, because **C_Solver** extends **C_Class**, it also possesses the **Initialize()** function (which was deferred in **C_Class** and is also deferred in **C_Solver**).

Module M_BFGMRES

This module contains the first implementation of a **C_Solver**, the **C_BFGMRES**.

Notice that **C_BFGMRES**, as the **C_Diag7pts** and **C_Standard27pts** implement both **Initialize()** and **Destroy()** functions from **C_Class**. Likewise, in this module some extra functions for handling the new

TYPE, ABSTRACT, EXTENDS(C_Solver) C_BFGMRES

Description:
The first solver we implement. It consists of BFGMRES method proposed by Vital [134]. It requires some extra parameters to be allocated, like the tolerance and the restart size. The stopping criterion of this solver is fixed: it always stops when the $\|.\|_\psi$ of the relative residual is smaller than the tolerance, or when the maximum number of cycles has been performed.

Important Components:

C_Solver POINTER	**MyPC**	This object defines the preconditioner to be used by this **C_BFGMRES**. Notice that it could be **C_BFGMRES** itself, since **C_BFGMRES** is also a **C_Solver**. If no preconditioner is going to be used, one can use the **C_NoSolver** which we describe later in the module **M_NoSolver**.
INTEGER	**MaxCycles**	Maximum number of cycles to be performed. The solver stops and returns its final approximation of the solution even if it did not converge
INTEGER	**MaxzDim**	BFGMRES will restart whenever the number of iterations reach **MaxRestartSize** or whenever the dimension of the subspace $\mathcal{Z}_j$ reaches **MaxzDim**, whichever comes first.
INTEGER	**MaxRestartSize**	BFGMRES will restart whenever the number of iterations reach **MaxRestartSize** or whenever the dimension of the subspace $\mathcal{Z}_j$ reaches **MaxzDim**, whichever comes first.
REAL	**Tol**	Convergence threshold for BFGMRES. It will return whenever the $\|.\|_\psi$ of the relative residual is smaller than **Tol** or **MaxCycles** have been performed.

Important Procedures:

Apply(x,xLen,b,bLen,p)	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Solver**
Initialize(MemAvailable)	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Class**
Destroy()	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Class**

class are provided.

MODULE M_BFGMRES most important functions:

New_BMGRES()	An interface accepting multiple types of parameter set. See Figure A.8 for usage. Most notably, we highlight here the existence of the parameter **O_PCToCreate**. It can receive, for the moment, one of the following values: **NoSolver**, **GaussSeidel** or **BFGMRES**. The first one is used when no preconditioner is desired, the second one uses one iteration of local symmetric Guass-Seidel iteration (to be covered later with more details in the **M_GaussSeidel**) and the later uses 10 cycles of unpreconditioned BGMRES(5) as preconditioner. Alternatively, instead of providing the O_PCToCreate, one can provide a pointer to a **C_Solver**, which is then going to be used as preconditioner. In Figure A.9 we illustrate how to use this situation.

```
!  ------------------------------------------------------------------------
!  Suppose that the following is known:
!  MemAvailable:  current memory available
!  Matrix:  a C_Operator with the coefficient matrix already ready for use
!  vecx:  a (xlen,p) vector containing the initial guess
!  vecb:  a (blen,p) vector containing the right-hand side
!  ------------------------------------------------------------------------
(...)
CLASS(C_Solver), POINTER   ::   solver
TYPE(T_Error)              ::   err

!  Since we supposed everything else is known, it is
!  left to be create the C_BGMRES. Here we use
!  50 cycles of BGMRES(10) with p right-hand sides
!  and no preconditioner (due to O_PCToCreate='NoSolver')

solver => New_BFGMRES(   Name            =   'My testing BFGMRES',   &
                         MyOperator      =   matrix,                 &
                         MaxCycles       =   50,                     &
                         MaxzDim         =   10*p,                   &
                         MaxRestartSize  =   10,                     &
                         Maxp            =   p,                      &
                         Tol             =   1e-6,                   &
                         O_PCToCreate    =   'NoSolver' )

!  Now we initialize it.  The initialization allocates everything
CALL solver%Initialize(MemAvailable)

!  Now we can use it; with apply() function.  We set it to verbose
!  just to make sure to see some convergence information
solver%IsVerbose = .TRUE.
CALL solver%Apply(vecb,blen,vecx,xlen,p,0,err)

!  We are done using this C_Solver.  We just destroy it.
CALL solver%Destroy()
```

Figure A.8: *Example of usage of* **C_BFGMRES**, *using no preconditioner. There is a strong similarity between this and Figure A.6 and Figure A.7 examples*

```
!  ------------------------------------------------------------------------
!  Suppose that the following is known:
!  MemAvailable:  current memory available
!  Matrix:  a C_Operator with the coefficient matrix already ready for use
!  vecx:  a (xlen,p) vector containing the initial guess
!  vecb:  a (blen,p) vector containing the right-hand side
!  ------------------------------------------------------------------------
(...)
CLASS(C_Solver), POINTER   ::   solver, preconditioner
TYPE(T_Error)              ::   err

!  We create a C_BFGMRES to be used as a preconditioner
!  Only 5 cycles of BGMRES(3) are going to be used
preconditioner => New_BFGMRES(   Name            =   'This is a Preconditioner',   &
                                 MyOperator      =   matrix,                       &
                                 MaxCycles       =   5,                            &
                                 MaxzDim         =   3*p,                          &
                                 MaxRestartSize  =   3,                            &
                                 Maxp            =   p,                            &
                                 Tol             =   0.0,                          &
                                 O_PCToCreate    =   'NoSolver' )

!  And now we create the solver...
solver => New_BFGMRES(   Name            =   'My testing BFGMRES',   &
                         MyOperator      =   matrix,                 &
                         MaxCycles       =   50,                     &
                         MaxzDim         =   10*p,                   &
                         MaxRestartSize  =   10,                     &
                         Maxp            =   p,                      &
                         Tol             =   1e-6,                   &
                         MyPC            =   preconditioner )
!  We can initialize the preconditioner and then the solver
!  or initialize only the solver, as it will recursively initialize
!  the preconditioner if needed
CALL preconditioner%Initialize(MemAvailable)
CALL solver%Initialize(MemAvailable)

!  Now we can use it; with apply() function.  We set it to verbose
!    just to make sure to see some convergence information
solver%IsVerbose = .TRUE.
!  We can also make the preconditioner verbose; in that case we
!    see both convergence information being printed
preconditioner%IsVerbose = .TRUE.
CALL solver%Apply(vecb,blen,vecx,xlen,p,0,err)

!  We are done.  We just destroy everything.  Alternatively we could
!  use the DestroyAll procedure which destroys recursively every pointer.
CALL preconditioner%Destroy()
CALL solver%Destroy()
```

Figure A.9: *Example of usage of* **C_BFGMRES**, *using a customized* **C_BFGMRES** *as preconditioner.*

Modules M_NoSolver, M_GaussSeidel, M_FGMRES and M_LinAlg

We do not advise the user to deal directly with any of these modules, and for this reason we quickly address them here before proceeding to the discussion of other modules.

M_NoSolver Internally, every **C_Solver** like **C_BFGMRES** need a pointer to another **C_Solver** to use as preconditioner, even in the unpreconditioned case, because it calls the procedure **preconditioner%Apply()**. This module *fills this gap*, providing a "solver" that simply copies the input vector b to vector x whenever the **Apply()** procedure is called. Although one can actually use **C_NoSolver**, the current implementation of libSolver automatically creates one whenever it is needed such that the user do not have to mind this issue.

M_GaussSeidel Provides a **C_Solver**, meant to be used as preconditioner for another solver. It is intrinsically dependent on the structure of the **C_Operator** being used. Because of that, the actual routines for performing it are currently implemented inside the respective operator. Both **C_Diag7pts** and **C_Standard27pts** have an implementation of the local symmetric Guass-Seidel algorithm. The **C_GaussSeidel** class calls this routine directly from the **C_Operator**, acting as some sort of interface. This means that this module in fact need to be updated whenever a new **C_Operator** is added to libOperator library, which is undesirable. As **C_NoSolver**, there is no need for the user to create its own **C_GaussSeidel** as the routines provided in libSolver create it automatically if needed. An important information about this **C_Solver** is that it contains a component **MaxIt**, meaning that several iterations of the local symmetric Gauss-Seidel algorithm could be used as preconditioner. However, libSolver uses by default only one iteration.

M_FGMRES This module is analogous to **M_BFGMRES**. Noticeable differences are the absence of the **MaxzDim** component and the replacement of O_PCToCreate='BFGMRES' by O_PCToCreate='FGMRES'. This module executes the single right-hand side FGMRES algorithm for each one of the p right-hand side and thus it is non-optimal. It is often advised to use **M_BFGMRES** instead.

M_LinAlg Provides important linear algebra routines implemented in parallel tailored to be used with **M_Solver**. We mention here the **Get_Norm** routine for computing the Euclidean norm of each column of a block vector, the **CGS2** which implements the classical Gram-Schmidt algorithm with reorthogonalization, and **QR_CGS2** which computes the QR decomposition of a matrix using the **CGS2** function. Since these are low level routines, we do not address them deeply here.

Module M_DMBR

This module contains the main iterative solver we propose, the **C_DMBR**.

In an overall, the module **M_DMBR** is analogous to the module **M_BFGMRES**.

TYPE, ABSTRACT, EXTENDS(C_Solver) C_DMBR

Description:
The DMBR solver, performing deflation at the beginning of every iteration. It contains all the components of **C_BFGMRES** plus some extra parameters for handling deflation.

Important Components:

C_Solver POINTER	**MyPC**	This object defines the preconditioner to be used by this **C_DMBR**. Notice that it could be **C_DMBR** itself or **C_BFMGRES**, for instance.
INTEGER	**MaxCycles**	Maximum number of cycles to be performed. The solver stops and returns its final approximation of the solution even if it did not converge
INTEGER	**MaxzDim**	DMBR will restart whenever the number of iterations reach **MaxRestartSize** or whenever the dimension of the subspace $\mathcal{Z}_j$ reaches **MaxzDim**, whichever comes first.
INTEGER	**MaxRestartSize**	DMBR will restart whenever the number of iterations reach **MaxRestartSize** or whenever the dimension of the subspace $\mathcal{Z}_j$ reaches **MaxzDim**, whichever comes first.
REAL	**Tol**	Convergence threshold for DMBR. It will return whenever the $\|.\|_\psi$ of the relative residual is smaller than **Tol** or **MaxCycles** have been performed.
REAL	**DeflationTol**	Deflation threshold for DMBR. Used to determine the value of k_j in the beginning of every iteration.
INTEGER	**Maxkj**	Used to enforce a maximum value for k_j even if no small singular values were detected.
INTEGER	**Deflation**	Flag used to set the kind of deflation. F_DMBR_DEFLATION_NONE does not deflate (equivalent to **C_BFGMRES**), F_DMBR_DEFLATION_IT deflates only at the end of each iteration (equivalent to [103] R-criterion algorithm), F_DMBR_DEFLATION_CY deflates at the beginning of each cycle.

Important Procedures:

Apply(x,xLen,b,bLen,p)	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Solver**
Initialize(MemAvailable)	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Class**
Destroy()	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Class**

MODULE M_DMBR most important functions:

New_DMBR()	An interface accepting multiple types of parameter set. The parameter **O_PCToCreate** is the same used in **M_BFGMRES**. See Figure A.10 for usage.

```
!  ------------------------------------------------------------------------
!  Because this example is perfectly analogous to C_BFGMRES examples
!  We just show here an example of use of the interface New_DMBR
! -------------------------------------------------------------------------
!  This is is valid.  Since Maxkj was not passed it is set to Maxp,
!  DeflationTol is set to Tol and Deflation=F_DMBR_DEFLATION_STD
solver => New_DMBR(   Name             =   'My testing DMBR',   &
                      MyOperator       =   matrix,              &
                      MaxCycles        =   50,                  &
                      MaxzDim          =   10*p,                &
                      MaxRestartSize   =   10,                  &
                      Maxp             =   p,                   &
                      Tol              =   1e-6,                &
                      O_PCToCreate     =   'NoSolver' )
!  We could alternatively pass all the parameters
solver => New_DMBR(   Name             =   'My testing DMBR',      &
                      MyOperator       =   matrix,                 &
                      MaxCycles        =   50,                     &
                      MaxzDim          =   10*p,                   &
                      MaxRestartSize   =   10,                     &
                      Maxp             =   p,                      &
                      Tol              =   1e-6,                   &
                      DeflationTol     =   1e-8,                   &
                      Maxkj            =   p/2,                    &
                      Deflation        =   F_DMBR_DEFLATION_STD,   &
                      MyPC             =   preconditioner )
```

Figure A.10: *Example of usage of* **C_DMBR**. *It is analogous to examples in Figure A.8 and Figure A.9*

Module M_GeoMultigrid

The last module belonging to libSolver, which implements a V cycle of geometric multigrid algorithm. Due to the polymorphism, however, this could be used to implement several levels of multigrid

TYPE, ABSTRACT, EXTENDS(C_Solver) C_GeoMultigrid

Description:
The **C_GeoMultigrid**, meant to be used as a preconditioner. A priori it is nothing but a two-level V cycle of geometric multigrid, but it could be any arbitrary geometric multigrid due to polymorphism, as we explain later.

Important Components:

C_Solver POINTER	**PreSmoother**	A **C_Solver** to use as pre-smoother. It could be, for instance, **C_BGMRES** or **C_GeoMultigrid** itself (yielding a W cycle in this case).
C_Solver POINTER	**PostSmoother**	A **C_Solver** to use as post-smoother. It could be, for instance, **C_BGMRES** or **C_GeoMultigrid** itself (yielding a W cycle in this case).
C_Solver POINTER	**Coarse**	A **C_Solver** to use as coarse grid corrector. It could be, for instance, **C_DMBR** or **C_GeoMultigrid** itself (yielding a three-level cycle in this case).
INTEGER	**MaxIt**	The **C_GeoMultigrid** performs MaxIt and then exists. It has no stopping criterion implemented at the moment.

Important Procedures:

Apply(x,xLen,b,bLen,p)	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Solver**
Initialize(MemAvailable)	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Class**
Destroy()	**IMPLEMENTED**	Implements the deferred procedure inherited from **C_Class**

As **M_DMBR** and **M_BFGMRES**, this module also provides some interfaces for dealing with internal values. However, they require the knowledge of the interpolation and restriction routines. Since we are trying to attain a basic aspects, we stick only with the simplest form, which also assumes that **PreSmoother**=**PostSmoother**.

MODULE M_GeoMultigrid most important functions:

New_GeoMultigrid(Name,MaxIt, Maxp,Smoother,Coarse)	Returns a **C_GeoMultigrid** ready to be initialized and used as a preconditioner. The **Smoother** passed is used as both **PreSmoother** and **PostSmoother**.

```
FUNCTION BlockPerturbed_TwoGrid(FineOP, CoarseOP, p) RESULT(self)

 CLASS(C_Operator), POINTER, INTENT(INOUT)   ::   FineOP, CoarseOP
 INTEGER , INTENT(IN)                        ::   p

 TYPE(C_Class), POINTER                      ::   self, Smoother, Coarse
 !  Create the Smoother and the Coarse solver.  We choose BFGMRES for both.
 Smoother => New_BFGMRES(   Name             =   'SmootherBGMRES',   &
                            MyOperator       =   FineOP,             &
                            MaxCycles        =   2,                  &
                            MaxzDim          =   2*p,                &
                            MaxRestartSize   =   2,                  &
                            Maxp             =   p,                  &
                            Tol              =   0.0,                &
                            O_PCToCreate     =   'GaussSeidel' )
 Coarse => New_BFGMRES(   Name             =   'CoarseBGMRES',   &
                          MyOperator       =   CoarseOP,         &
                          MaxCycles        =   10,               &
                          MaxzDim          =   5*p,              &
                          MaxRestartSize   =   5,                &
                          Maxp             =   p,                &
                          Tol              =   0.0,              &
                          O_PCToCreate     =   'GaussSeidel' )
 !  Finally, creates the C_GeoMultigrid itself
 self => New_GeoMultigrid(   Name       =   'BlockPerturbed_TwoGrid',   &
                             MaxIt      =   1,                          &
                             Smoother   =   Smoother,                   &
                             Coarse     =   Coarse )
END FUNCTION BlockPerturbed_TwoGrid
```

Figure A.11: *Example of usage of* **C_GeoMultigrid** *for creating a variant of the two-level perturbed geometric multigrid preconditioner proposed in [96] (cf. Algorithm 4.4.1). In this variant we use BFGMRES instead of FGMRES for the smoothing and coarse correction steps.*

```
FUNCTION Example_ThreeLevel_Multigrid(FineOP, MiddleOP, CoarseOP, p) RESULT(self)

 CLASS(C_Operator), POINTER, INTENT(INOUT)   ::   FineOP, MiddleOP, CoarseOP
 INTEGER , INTENT(IN)                        ::   p

 TYPE(C_Solver), POINTER                     ::   self, First, SecondAndThird
 !  Use the routine from previous example to create a C_GeoMultigrid
 SecondAndThird => BlockPerturbed_TwoGrid(   FineOP     =   MiddleOP,   &
                                             CoarseOP   =   CoarseOP,   &
                                             p          =   p )
 !  Creates the smoother to be used on the finest level
 First => New_BFGMRES(   Name             =   'FirstBGMRES',   &
                         MyOperator       =   FineOP,          &
                         MaxCycles        =   15,              &
                         MaxzDim          =   7*p,             &
                         MaxRestartSize   =   7,               &
                         Maxp             =   p,               &
                         Tol              =   0.0,             &
                         O_PCToCreate     =   'NoSolver' )
 !  Finally, creates the C_GeoMultigrid itself
 self => New_GeoMultigrid(   Name       =   'ThreeLevel_Multigrid',   &
                             MaxIt      =   1,                        &
                             Smoother   =   First,                    &
                             Coarse     =   SecondAndThird )
END FUNCTION Example_ThreeLevel_Multigrid
```

Figure A.12: *Example of usage of* **C_GeoMultigrid** *for creating a three-level multigrid. Here we we the example in Figure A.11 to build the two bottom levels, and use* 15 *cycles of unpreconditioned BGMRES(*7*) as smoother on the finest level.*

Conclusions

In this User Guide we provided the basic tools for making the transition between FORTRAN90 procedural programming to FORTRAN03 object oriented programming. We explained the most essential concepts and then we introduced the libraries we implemented using only the most basic concepts. These libraries could be used, a priori, for kind of application using MPI routines. Although the results we obtained with this library were satisfactory, we aim at improving the number of options available, as adding new solvers, preconditioners and matrix storage formats.

Bibliography

[1] K. Ahuja, E. de Sturler, S. Gugercin, and E. R. Chang. Recycling BiCG with an application to model reduction. *SIAM J. Scientific Computing*, 34(4), 2012.

[2] J. I. Aliaga, D. L. Boley, R. W. Freund, and V. Hernández. A Lanczos-type method for multiple starting vectors. *Mathematics of Computation*, 69:1577–1601, 2000.

[3] P. R. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. Improving multifrontal methods by means of block low-rank representations. Rapport de recherche RR-8199, INRIA, Jan. 2013. Submitted for publication to SIAM.

[4] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[5] W. E. Arnoldi. "the principle of minimized iterations in the solution of the matrix eigenvalue problem". *"Quarterly of Applied Mathematics"*, 9(3):17–29, 1951.

[6] O. Axelsson. *Iterative solution methods*. Cambridge University Press, 1994.

[7] O. Axelsson and P. S. Vassilevski. A black box generalized conjugate gradient solver with inner iterations and variable-step preconditioning. *SIAM J. Matrix Analysis and Applications*, 12(4):625–644, 1991.

[8] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel. Adaptively preconditioned GMRES algorithms. *SIAM J. Scientific Computing*, 20:243–269, 1998.

[9] Z. Bai, D. Day, and Q. Ye. ABLE: an adaptive block Lanczos for non hermitian eigenvalue problems. *SIAM J. Matrix Analysis and Applications*, 20(4):1060–1082, 1999.

[10] A. H. Baker, J. M. Dennis, and E. R. Jessup. An efficient block variant of GMRES. *SIAM J. Scientific Computing*, 27(5):1608–1626, 2006.

[11] A. H. Baker, E. R. Jessup, and T. Manteuffel. A technique for accelerating the convergence of restarted GMRES. *SIAM J. Matrix Anal. Appl.*, 26(4):962–984, 2005.

[12] G. Barbella, F. Perotti, and V. Simoncini. Block Krylov subspace methods for the computation of structural response to turbulent wind. *Comput. Meth. Applied Mech. Eng.*, 200(23-24):2067–2082, 2011.

[13] J.-P. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physiscs*, 114(2):185–200, 1994.

[14] J.-P. Berenger. Three-dimensional perfectly matched layer for absorption of electromagnetic waves. *Journal of Computational Physiscs*, 127:363–379, 1996.

[15] G. Beylkin and M. J. Mohlenkamp. Algorithms for numerical analysis in high dimensions. *SIAM J. Scientific Computing*, 26(6):2133–2159, 2005.

[16] H. bin Zubair, C. W. Oosterlee, and R. Wienands. Multigrid for high dimensional elliptic partial differential equations on nonequidistant grids. *SIAM J. Scientific Computing*, 29(4):1613–1636, 2007.

[17] Å. Björck. *Numerical methods for least squares problems*. SIAM, 1996.

[18] M. Bollhöfer, M. J. Grote, and O. Schenk. Algebraic multilevel preconditioner for the solution of the Helmholtz equation in heterogeneous media. *SIAM J. Scientific Computing*, 31:3781–3805, 2009.

[19] W. Boyse and A. Seidl. A block QMR method for computing multiple simultaneous solutions to complex symmetric systems. *SIAM J. Scientific Computing*, 17(1):263–274, 1996.

[20] R. Brossier. *Imagerie Sismique à Deux Dimensions des Milieux Visco-Élastiques par Inversion des Formes d'ondes: développements méthodologiques et application*. PhD thesis, Université de Nice-Sophia Antipolis, November 2009.

[21] P. A. Businger and G. Golub. Linear least squares solutions by Householder transformations. *Numerische Mathematik*, 7:269–276, 1965.

[22] H. Calandra, S. Gratton, R. Lago, X. Pinel, and X. Vasseur. Two-level preconditioned Krylov subspace methods for the solution of three-dimensional heterogeneous Helmholtz problems in seismics. *Numerical Analysis and Applications*, 5:175–181, 2012.

[23] H. Calandra, S. Gratton, J. Langou, X. Pinel, and X. Vasseur. Flexible variants of block restarted GMRES methods with application to geophysics. *SIAM J. Scientific Computing*, 34(2):A714–A736, 2012.

[24] H. Calandra, S. Gratton, X. Pinel, and X. Vasseur. An improved two-grid preconditioner for the solution of three-dimensional Helmholtz problems in heterogeneous media. *Numerical Linear Algebra with Applications*, 2013. to appear.

[25] M. H. Carpenter, C. Vuik, P. Lucas, M. B. van Gijzen, and H. Bijl. A general algorithm for reusing Krylov subspace information. I. Unsteady Navier-Stokes. NASA/TM 2010216190, NASA, Langley Research Center, 2010.

[26] L. M. Carvalho, S. Gratton, R. Lago, and X. Vasseur. A flexible generalized conjugate residual method with inner orthogonalization and deflated restarting. Technical Report TR/PA/10/10, CERFACS, Toulouse, France, 2010.

[27] L. M. Carvalho, S. Gratton, R. Lago, and X. Vasseur. A flexible generalized conjugate residual method with inner orthogonalization and deflated restarting. *SIAM J. Matrix Analysis and Applications*, 32(4):1212–1235, 2011.

[28] J. F. Clærbout. *Imaging the earth's interior*. Blackwell Scientific Publications, Inc., Cambridge, MA, USA, 1985.

[29] G. Cohen. *Higher-order Numerical Methods for Transient Wave Equations*. Springer, 2002.

[30] E. Crase, A. Pica, M. Noble, J. McDonald, and A. Tarantola. Robust elastic non-linear waveform inversion: application to real data. *Geophysics*, 55:527–538, 1990.

[31] E. Crase, C. Wideman, M. Noble, and A. Tarantola. Nonlinear elastic inversion of land seismic reflection data. *Journal of Geophysical Research*, 97:4685–4705, 1992.

[32] J. Cullum and T. Zhang. Two-sided Arnoldi and non-symmetric Lanczos algorithms. *SIAM J. Matrix Analysis and Applications*, 24:303–319, 2002.

[33] R. D. Cunha and D. Becker. Dynamic block GMRES: an iterative method for block linear systems. *Advances in Computational Mathematics*, 27(4):423–448, 2006.

[34] E. de Sturler. Nested Krylov methods based on GCR. *J. Comput. Appl. Math.*, 67(1):15–41, 1996.

[35] E. de Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM J. Numerical Analysis*, 36(3):864–889, 1999.

[36] E. de Sturler and M. Kilmer. Recycling subspace information for diffuse optical tomography. *SIAM J. Scientific Computing*, 27:2140–2166, 2004.

[37] E. de Sturler, C. Le, S. Wang, and G. Paulino. Large scale topology optimization using preconditioned Krylov subspace recycling and continuous approximation of material distribution. *Int J. Numerical Methods in Engineering*, 69:2441–2468, 2007.

[38] L. Du, T. Sogabe, B. Yu, Y. Yamamoto, and S.-L. Zhang. A block IDR(s) method for nonsymmetric linear systems with multiple right-hand sides. *J. Comput. Appl. Math.*, 235:4095–4106, 2011.

[39] I. S. Duff, S. Gratton, X. Pinel, and X. Vasseur. Multigrid based preconditioners for the numerical solution of two-dimensional heterogeneous problems in geophysics. *International Journal of Computer Mathematics*, 84-88:1167–1181, 2007.

[40] M. Eiermann and O. G. Ernst. Geometric aspects in the theory of Krylov subspace methods. *Acta Numerica*, 10(10):251–312, 2001.

[41] M. Eiermann, O. G. Ernst, and O. Schneider. Analysis of acceleration strategies for restarted minimal residual methods. *Journal of Computational and Applied Mathematics*, 123(1-2):261–292, 2000.

[42] M. Eiermann, O. G. Ernst, and E. Ullmann. Computational aspects of the stochastic finite element method. *Comput. Visual. Sci.*, 10(1):3–15, 2007.

[43] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric system of linear equations. *SIAM J. Numerical Analysis*, 20(2), April 1983.

[44] L. Elbouyahyaoui, A. Messaoudi, and H. Sadok. Algebraic properties of the block GMRES and block Arnoldi methods. *Electron. Trans. Numer. Anal.*, 33:207–220, 2008/09.

[45] H. Elman, O. Ernst, D. O'Leary, and M. Stewart. Efficient iterative algorithms for the stochastic finite element method with application to acoustic scattering. *Comput. Methods Appl. Mech. Engrg.*, 194(1):1037–1055, 2005.

[46] H. C. Elman, O. G. Ernst, and D. P. O'Leary. A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations. *SIAM J. Scientific Computing*, 23:1291–1315, 2001.

[47] B. Engquist and L. Ying. Sweeping preconditioner for the helmholtz equation: Moving perfectly matched layers. *Multiscale Modeling & Simulation*, 9(2):686–710, 2011.

[48] J. Erhel, K. Burrage, and B. Pohl. Restarted GMRES preconditioned by deflation. *Journal of Computational and Applied Mathematics*, 69:303–318, 1995.

[49] Y. A. Erlangga. *A robust and efficient iterative method for the numerical solution of the Helmholtz equation.* PhD thesis, Technische Universiteit Delft, December 2005.

[50] Y. A. Erlangga. Advances in iterative methods and preconditioners for the Helmholtz equation. *Archives of Computational Methods in Engineering*, 15:37–66, 2008.

[51] Y. A. Erlangga, C. Oosterlee, and C. Vuik. A novel multigrid based preconditioner for heterogeneous Helmholtz problems. *SIAM J. Scientific Computing*, 27:1471–1492, 2006.

[52] O. Ernst and M. J. Gander. Why it is difficult to solve Helmholtz problems with classical iterative methods. In O. L. I. Graham, T. Hou and R. Scheichl, editors, *Numerical Analysis of Multiscale Problems*. Springer, 2011.

[53] R. W. Freund. Krylov-subspace methods for reduced-order modeling in circuit simulation. *Journal of Computational and Applied Mathematics*, 123(1-2):395–421, 2000.

[54] R. W. Freund and M. Malhotra. A block QMR algorithm for non-Hermitian linear systems with multiple right-hand sides. *Linear Algebra and its Applications*, 254:119–157, 1997.

[55] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60(1):315–339, December 1991.

[56] M. J. Gander and F. Nataf. An incomplete LU preconditioner for problems in acoustics. *Journal of Computational Acoustics*, 13(3):1–22, 2005.

[57] O. Gauthier, J. Virieux, and A. Tarantola. Two-dimensional nonlinear inversion of seismic waveforms: numerical results. *Geophysics*, 51:1387–1403, 1986.

[58] L. Giraud, S. Gratton, X. Pinel, and X. Vasseur. Flexible GMRES with deflated restarting. *SIAM J. Scientific Computing*, 32(4):1858–1878, 2010.

[59] L. Giraud and J. Langou. A robust criterion for the modified Gram-Schmidt algorithm with selective reorthogonalization. *SIAM J. Scientific Computing*, 25(2):417–441, Feb. 2003.

[60] G. H. Golub and C. F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, October 1996.

[61] D. Gordon and R. Gordon. CARP-CG: A robust and efficient parallel solver for linear systems, applied to strongly convection dominated pdes. *Parallel Computing*, 36(9):495–515, 2010.

[62] D. Gordon and R. Gordon. Parallel solution of high frequency Helmholtz equations using high order finite difference schemes. *Applied Mathematics and Computation*, 218(21):10737–10754, 2012.

[63] A. E. Guennouni, K. Jbilou, and H. Sadok. A block version of BICGSTAB for linear systems with multiple right-hand sides. *Electronic Transactions on Numerical Analysis*, 16:129–142, 2003.

[64] M. H. Gutknecht. Block Krylov space methods for linear systems with multiple right-hand sides: An introduction. *Modern Mathematical Models, Methods and Algorithms for Real World Systems*, page 420–447, 2006.

[65] M. H. Gutknecht and T. Schmelzer. The block grade of a block Krylov space. *Linear Algebra and its Applications*, 430(1):174 – 185, 2009.

[66] E. Haber and S. MacLachlan. A fast method for the Helmholtz equation. *Journal of Computational Physiscs*, 230:4403–4418, 2011.

[67] A. Haidar. *On the parallel scalability of hybrid linear solvers for large 3D problems*. PhD thesis, CERFACS, 2009.

[68] I. Harari and E. Turkel. Accurate finite difference methods for time-harmonic wave propagation. *Journal of Computational Physiscs*, 119:252–270, 1995.

[69] A. Henderson. Paraview guide, a parallel visualization application, 2007.

[70] N. J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[71] R. Horn and C. Johnson. *Topics in Matrix Analysis*. Topics in Matrix Analysis. Cambridge University Press, 91.

[72] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, February 1990.

[73] B. Hustedt, S. Operto, and J. Virieux. Mixed-grid and staggered-grid finite difference methods for frequency-domain acoustic wave modelling. *Geophys. J. Int.*, 157:1269–1296, 2004.

[74] I. C. F. Ipsen and C. D. Meyer. The idea behind Krylov methods. *American Mathematical Monthly*, 105:889–899, 1998.

[75] W. Joubert. A robust GMRES-based adaptive polynomial preconditioning algorithm for nonsymmetric linear systems. *SIAM J. Scientific Computing*, 15:427–439, 1994.

[76] A. Khabou. Solveur itératif haute performance pour les systèmes linéaires avec seconds membres multiples. Master's thesis, University of Bordeaux I, 2009.

[77] S. Kim and S. Kim. Multigrid simulations for high-frequency solutions of the Helmholtz problem in heterogeneous media. *SIAM J. Scientific Computing*, 24:359–392, 2002.

[78] J. Langou. *Iterative methods for solving linear systems with multiple right hand sides*. Ph.D. dissertation, INSA Toulouse, June 2003. TH/PA/03/24.

[79] R. B. Lehoucq and K. J. Maschhoff. Implementation of an implicitly restarted block Arnoldi method. Technical report, 1997.

[80] H. Liu and B. Zhong. A simpler block GMRES for nonsymmetric systems with multiple right-hand sides. *Electronic Transactions on Numerical Analysis*, 30:1–9, 2008.

[81] D. Loher. *Reliable Nonsymmetric Block Lanczos Algorithms*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETHZ), Switzerland, 2006. Number 16337.

[82] MATLAB. version 7.10.0 (r2010a), 2010.

[83] S. F. McCormick. *Multigrid methods*. SIAM, 1987.

[84] P. Mora. Inversion = migration + tomography. *Geophysics*, 54:1575–1586, 1989.

[85] R. B. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra and its Applications*, 154–156:289–309, 1991.

[86] R. B. Morgan. A restarted GMRES method augmented with eigenvectors. *SIAM J. Matrix Anal. Appl.*, 16(4):1154–1171, 1995.

[87] R. B. Morgan. Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. *SIAM J. Matrix Anal. Appl.*, 21(4):1112–1135, 2000.

[88] R. B. Morgan. GMRES with deflated restarting. *SIAM J. Scientific Computing*, 24(1):20–37, 2002.

[89] A. A. Nikishin and A. Y. Yeremin. Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, i: General iterative scheme. *SIAM J. Matrix Analysis and Applications*, 16(4):1135–1153, 1995.

[90] Y. Notay. Flexible conjugate gradients. *SIAM J. Scientific Computing*, 22(4):1444–1460, 2000.

[91] Y. Notay and P. S. Vassilevski. Recursive Krylov-based multigrid cycles. *Numerical Linear Algebra with Applications*, 15:473–487, 2008.

[92] D. P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications*, 29:293–322, 1980.

[93] S. Operto, J. Virieux, P. Amestoy, J.-Y. L'Excellent, L. Giraud, and H. B. H. Ali. 3d finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study. *Geophysics*, 72(5):SM195–SM211, 2007.

[94] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numerical Linear Algebra with Applications*, 2:115–134, 1995.

[95] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM J. Scientific Computing*, 28(5):1651–1674, Sept. 2006.

[96] X. Pinel. *A Perturbed Two-level Preconditioner for the Solution of Three-Dimensional Heterogeneous Helmholtz problems with Applications to Geophysics.* Ph.D. dissertation, Institut National Polytechnique de Toulouse, CERFACS, May 2010.

[97] J. Poulson, B. Engquist, S. Fomel, S. Li, and L. Ying. A parallel sweeping preconditioner for high frequency heterogeneous 3d helmholtz equations. *CoRR*, abs/1204.0111, 2012.

[98] R. G. Pratt. Inverse theory applied to multi-source cross-hole tomography. Part II: elastic wave-equation method. *Geophysical Prospecting*, 38:311–330, 1990.

[99] R. G. Pratt and M. H. Worthington. Inverse theory applied to multi-source cross-hole tomography. Part I: acoustic wave-equation method. *Geophysical Prospecting*, 38:287–310, 1990.

[100] C. D. Riyanti, Y. A. Erlangga, R.-E. Plessix, W. A. Mulder, C. Vuik, and C. Oosterlee. A new iterative solver for the time-harmonic wave equation. *Geophysics*, 71:57–63, 2006.

[101] C. D. Riyanti, A. Kononov, Y. A. Erlangga, R.-E. Plessix, W. A. Mulder, C. Vuik, and C. Oosterlee. A parallel multigrid-based preconditioner for the 3D heterogeneous high-frequency Helmholtz equation. *Journal of Computational Physiscs*, 224:431–448, 2007.

[102] M. Robbé and M. Sadkane. Exact and inexact breakdowns in block versions of FOM and GMRES methods. Technical Report, Université de Bretagne Occidentale. Département de Mathématiques, 2004. Available at http://www.math.univ-brest.fr/archives/recherche/prepub/Archives/2005/breakdowns.pdf.

[103] M. Robbé and M. Sadkane. Exact and inexact breakdowns in the block GMRES method. *Linear Algebra and its Applications*, 419(1):265–285, 2006.

[104] A. Ruhe. Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices. *Mathematics of Computation*, 33:680–687, 1979.

[105] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm, 1993.

[106] Y. Saad. Analysis of augmented Krylov subspace methods. *SIAM J. Matrix Analysis and Applications*, 18:435–449, 1997.

[107] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition.* Society for Industrial and Applied Mathematics, 2nd edition, April 2003.

[108] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing*, 7(3):856–869, 1986.

[109] Y. Saad and H. A. van der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123:1–33, 2000.

[110] T. Sakurai, H. Tadano, and Y. Kuramashi. Application of block Krylov subspace algorithms to the Wilson-Dirac equation with multiple right-hand sides in lattice QCD. *Computer Physics Communications*, 181(1):113–117, 2010.

[111] V. Simoncini and E. Gallopoulos. Convergence properties of block GMRES and matrix polynomials. *Linear Algebra and its Applications*, 247:97–119, 1996.

[112] V. Simoncini and D. B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM J. Numerical Analysis*, 40(6):2219–2239, 2003.

[113] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM J. Sci. Comput.*, 25:454–477, February 2003.

[114] V. Simoncini and D. B. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14:1–59, 2007.

[115] I. Singer and E. Turkel. Sixth order accurate finite difference schemes for the Helmholtz equation. *Journal of Computational Acoustics*, 14:339–351, 2006.

[116] L. Sirgue and R. G. Pratt. Efficient waveform inversion and imaging: a strategy for selecting temporal frequencies. *Geophysics*, 69:231–248, 2004.

[117] G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Analysis and Applications*, 17(2):401–425, 1996.

[118] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[119] P. Soudais. Iterative solution methods of a 3-D scattering problem from arbitrary shaped multidielectric and multiconducting bodies. *IEEE Trans. on Antennas and Propagation*, 42 (7):954–959, 1994.

[120] F. Sourbier, S. Operto, J. Virieux, P. Amestoy, and J. Y. L. Excellent. FWT2D : a massively parallel program for frequency-domain full-waveform tomography of wide-aperture seismic data - part 1: algorithm. *Computer & Geosciences*, 35:487–495, 2009.

[121] F. Sourbier, S. Operto, J. Virieux, P. Amestoy, and J. Y. L. Excellent. FWT2D : a massively parallel program for frequency-domain full-waveform tomography of wide-aperture seismic data - part 2: numerical examples and scalability analysis. *Computer & Geosciences*, 35:496–514, 2009.

[122] G. W. Stewart. *Matrix Algorithms: Volume 2, Eigensystems.* Society for Industrial Mathematics, 2001.

[123] X. Sun and C. Bischof. A basis-kernel representation of orthogonal matrices. *SIAM J. Matrix Analysis and Applications*, 16(4):1184–1196, 1995.

[124] D. B. Szyld and J. A. Vogel. FQMR: A flexible quasi-minimal residual method with inexact preconditioning. *SIAM J. Scientific Computing*, 23(2):363–380, 2001.

[125] A. Tarantola. Inversion of seismic reflection datain the acoustic approximation. *Geophysics*, 49:1259–1266, 1984.

[126] A. Tarantola. *Inversion Problem Theory: Methods for data fitting and model parameter estimation.* Springer, New York, 1987.

[127] A. Toselli and O. Widlund. *Domain Decomposition methods - Algorithms and Theory*, volume 34. Springer Series on Computational Mathematics, Springer, New-York, 2004.

[128] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press Inc., 2001.

[129] E. Ullmann. *Solution Strategies for Stochastic Finite Element Discretizations*. PhD thesis, Technische Universität Bergakademie Freiberg, Germany, 2008.

[130] N. Umetani, S. P. MacLachlan, and C. W. Oosterlee. A multigrid-based shifted Laplacian preconditioner for fourth-order Helmholtz discretization. *Numerical Linear Algebra with Applications*, 16:603–626, 2009.

[131] H. A. van der Vorst. BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsmmetric linear systems. *SIAM J. Scientific and Statistical Computing*, 13(2):631–644, March 1992.

[132] H. A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numerical Linear Algebra with Applications*, 1:369–386, 1994.

[133] J. Virieux and S. Operto. An overview of full waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC127–WCC152, 2009.

[134] B. Vital. *Étude de Quelques Méthodes de Résolution de Problèmes Linéaires de Grande Taille Sur Multiprocesseur*. PhD thesis, Université de Rennes, November 1990.

[135] M. M. Wagner, P. Pinsky, and M. Malhotra. Application of Padé via Lanczos approximations for efficient multifrequency solution of Helmholtz problems. *Journal of the Acoustical Society of America*, 113(1):313–9, 2003.

[136] M. M. Wagner, P. M. Pinsky, A. A. Oberai, and M. Malhotra. A Krylov subspace projection method for simultaneous solution of Helmholtz problems at multiple frequencies. *Comput. Methods Appl. Mech. Eng.*, 192(41-42):4609–4640, 2003.

[137] S. Wang, M. V. de Hoop, and J. Xia. Acoustic inverse scattering via Helmholtz operator factorization and optimization. *Journal of Computational Physics*, 229(22):8445 – 8462, 2010.

[138] S. Wang, M. V. de Hoop, and J. Xia. On 3d modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver. *Geophysical Prospecting*, 59(5):857–873, 2011.

[139] S. Wang, M. V. de Hoop, J. Xia, and X. S. Li. Massively parallel structured multifrontal solver for time-harmonic elastic waves in 3-d anisotropic media. *Geophysical Journal International*, 191(1):346–366, 2012.

[140] R. Yu, E. de Sturler, and D. D. Johnson. A block iterative solver for complex non-hermitian systems applied to large-scale electronic-structure calculations. Technical Report UIUCDCS-R-2002-2299, University of Illinois at Urbana-Champaign, Department of Computer Science, 2002.