Swansea University

and

École Centrale de Nantes

Master Thesis

Deflated Krylov Subspace Methods for the Poisson Pressure Equation

Author:
Héctor Hernández

Supervisor:
Dr. Michel Vissoneau

A thesis submitted in fulfilment of the requirements for the degree of Master of Science in Computational Mechanics.

June 2013

A mis padres y mis hermanas...

Contents

Chapter 1

Introduction

The solution of linear systems is currently regarded as a pillar in scientific computing. Apart that the vast majority of the ideas on which this branch of science is founded are depth and beauty form the mathematical point of view, countless applications could be mentioned. The areas requiring the solution of linear systems range from applied mathematics, physics, economics, management and economics to name only a few.

In addition, it is well recognized that a great amount of time spent in a given computation is devoted to the solution of such linear systems. It is now common the solution of systems containing millions or even billions of equations. Therefore the necessity of efficient methods have been always present.

Currently a large variety of techniques for the solution of linear systems are available. One of the most successful are iterative methods, specifically those regarded as *Krylov subspace* methods. Several techniques have been proposed over the years in order to increase the efficiency of such methods. Maybe the most important and widely extended is *preconditioning*. But this is not the unique, fast Fourier solvers, domain decomposition and multigrid, to name only a few, are also available. Even more, nowadays this is an active and productive research field.

It is one of these novel strategies for the acceleration and improvement of Krylov subspace methods in which we are interested in developing the present work. Such a technique is currently, with some degree of consensus, known as *deflation*.

1.1 Motivation

The main motivation for devote my master thesis project to deflation of Krylov subspaces goes back my past job before enrolling this master program. In that job I was in charge

of develop efficient computational programs mainly oriented in the finite element method context and fitting special client requirements. During developing such a task I had my first approach with these kind of iterative methods.

The situation could not be better after, because during the lectures on *Viscous Naval Hydrodynamics* in École Centrale de Nantes my former lecturer, and currently thesis adviser, made special emphasis in the importance of robust, efficient and reliable linear solvers for the simulation of fluid flows of practical interest.

What was even more encouraging is the fact that the conclusions of this master thesis would be applicable to a well established commercial software for the numerical simulation in fluid dynamics, a topic that deserves me a special interest. The program we refer to is ISIS. This code has been created and currently maintained and improved by a research group in the mentioned institution.

1.2 Objectives

Three general objectives in this work are pursued. The first of them is to review the state of the art of Krylov subspace methods in a very general way and with some detail deflation techniques reported as promising in the computational fluid dynamics context. The second general objective is the computational implementation of those techniques seeming more appropriates for the problem we have in mind. The third objective in this work is the application of the mentioned computational programs to some problems of our practical interest such as the linear systems arising from the ISIS software.

1.3 Structure of the Work

The present work is organized as follow:

- **Chapter 1**. Present chapter. It defines the motivation and aim of this work.
- **Chapter 2**. It provides a very general background of the numerical simulation of fluid flow. Far to pretend to be a formal presentation its only purpose is to elucidate some key ideas about the underlying problem from which arise the linear systems we are interested to solve.
- **Chapter 3**. It states the basic results of linear algebra that will be useful in the subsequent development of this work.

- **Chapter 4**. Probably this is the hearth of this thesis. Krylov subspace methods are introduced in a natural and intuitive way. We have used an informal language more focused in the algorithmic implementations. Almost nothing is said about the theoretical properties and analysis of the presented methods.

- **Chapter 5**. A large variety of the most successful and popular preconditioners is presented here. Again an informal exposition has been preferred.

- **Chapter 6**. This is maybe the most important one in the present work. It is completely devoted to the yet mentioned technique we are more interested to explore. At the contrary of the previous chapters it is slightly more formal, but special attention is payed to the algorithmic implementation.

- **Chapter 7**. Six selected examples are presented. This selection have been carried out carefully with the claim of being representative of the large amount of numerical results we have at concluding this work.

- **Chapter 8**. This concluding chapter provides a discussion of the results obtained in this work. Particular emphasis is done in the applications and limitations of the methods explored. It concludes identifying further developments that can be carried out as a direct consequence of this work.

Chapter 2

Numerical Simulation of Fluid Flow

Problems involving flows are encountered in many branches of engineering and science. Examples are flows in water supply and treatment systems, machinery, seas and rivers, and around aircraft, buildings and ships. The mathematical formulation of the laws that govern the motion of fluid is known already for well over a century. This formulation consists of some thermodynamic relations and a set of coupled partial differential equations, the so-called Navier-Stokes equations, which describe conservation of mass, momentum and energy for a fluid.

Appropriately defined boundary and initial conditions conditions completes the problem definition. In general, the resulting mathematical problem is far too difficult to be solved by analytical means. Therefore, in the past one had to rely heavily on experiments or greatly simplified mathematical models for the majority of flow problems. However, experiments are often very expensive, difficult, dangerous or even impossible to perform. On the other hand, the use of methods that solve the governing equations by numerical means is an obvious alternative. Thanks to the tremendous increase in computational power in the last decades, this approach has gained substantial significance. The new scientific discipline that has evolved is called *computational fluid dynamics* (CFD). Nowadays, CFD has, besides the more traditional experimental and analytical approaches, become an indispensable tool for the fluid dynamist.

In writing this chapter we are interested only in mention briefly a very general overview making emphasis on the topics we are concerned with in developing the present work. It does not pretend to be a detailed exposition neither of the fundamentals of mathematical fluid dynamics nor the numerical methods used to approach the problem.

2.1 The Navier-Stokes Equations

As vaguely mentioned previously, the Navier-Stokes equations are a set of partial differential equations that describes the dynamical behavior of a fluid. They can be derived from fundamental physical principles as conservation of mass, momentum and energy. We shall be particularly interested in a *Newtoinian* fluid in *incompressible* regime.

The first characteristic previously mentioned refers to a fluid property. A fluid is a substance that cannot support a shear stress at rest or in uniform flow, in fact, in a fluid, shear deformation will continue as long as any shear stress is applied. If the relationship between the applied shear stress and the dynamical response of the fluid is linear, this one is called Newtonian.

If while the fluid flows it does not experiment a noticeable change in its density, we can regard this flow as incompressible. Gases can be easily compressed while for the majority of liquids such compression is virtually impossible in room conditions.

With this considerations in mind we now write the Navier-Stokes equations

$$\nabla \cdot \mathbf{v} = 0, \tag{2.1}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v}, \tag{2.2}$$

where ρ is the fluid density and ν is the dynamic viscosity, both regarded as constant fluid properties. The fluid velocity is denoted by $\mathbf{v}$ and the pressure by p.

Equation (2.1) is directly derived from the balance of mass and it is commonly known as the continuity equation. Conservation of linear momentum leads us equation (2.2).

Solving this set of equations means precisely, determine the velocity vector $\mathbf{v}$ and the pressure p in a spatial domain $\Omega \subset \mathbb{R}^d$ of interest, with $d = 2$ or 3. For doing so, this system must be supplied with appropriate initial and boundary conditions. Then we write

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v} \quad \text{in } \Omega \times (0, T), \tag{2.3}$$

$$\nabla \cdot \mathbf{v} = 0 \quad \text{in } \Omega \times (0, T), \tag{2.4}$$

$$\mathbf{v} = \overline{\mathbf{v}} \quad \text{on } \Gamma_D \times (0, T), \tag{2.5}$$

$$\partial \mathbf{v} / \partial \hat{\mathbf{n}} = \overline{\mathbf{g}} \quad \text{on } \Gamma_D \times (0, T), \tag{2.6}$$

$$\mathbf{v}(\mathbf{x}, t = 0) = \mathbf{v}_o \quad \text{with } \nabla \cdot \mathbf{v}_o = 0. \tag{2.7}$$

This is the strong formulation of the Navier-Stokes equations which consists of the momentum equation (2.3), the continuity equation (2.4), the velocity boundary conditions $\overline{\mathbf{v}}$ on the Dirichlet boundary Γ_D, the Neumann boundary conditions $\partial\mathbf{v}/\partial\hat{\mathbf{n}} = \overline{\mathbf{g}}$ on the Neumann boundary Γ_N and the initial condition $\mathbf{v}_o$ satisfying the continuity equation. Finally $t = [0\ T]$ is the time interval of interest.

2.2 Poisson Pressure Equation

Notice that we have clearly stated that the solution of the Navier-Stokes equations implies determining the velocity and pressure fields. But notice also that the strong form of these equations does not specify nothing about the initial and boundary conditions for the pressure.

In order to clarify this situation consider a solution of the Navier-Stokes equations that, for the sake of simplicity but without loss of generality, satisfies an homogeneous Dirichlet boundary condition in the whole boundary domain. Now, we take the divergence at each side of the momentum equation (2.3)

$$\nabla \cdot \frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot (\mathbf{v} \cdot \nabla \mathbf{v}) = -\frac{1}{\rho}\nabla \cdot (\nabla p) + \nu \nabla \cdot \left(\nabla^2 \mathbf{v}\right), \tag{2.8}$$

commuting the space and time derivatives in the first term and the second and first order space derivatives in the last one and finally taking the divergence of the pressure gradient we have

$$\frac{\partial}{\partial t}(\nabla \cdot \mathbf{v}) + \nabla \cdot (\mathbf{v} \cdot \nabla \mathbf{v}) = -\frac{1}{\rho}\nabla^2 p + \nu \nabla^2 (\nabla \cdot \mathbf{v}).$$

Taking into account the continuity equation (2.4) and rearranging terms

$$\nabla^2 p = -\rho \nabla \cdot (\mathbf{v} \cdot \nabla \mathbf{v}), \tag{2.9}$$

which shows that the pressure field is a solution of the Poisson's equation with a source which is quadratic in the derivatives of the velocities.

Now, equation (2.9) must be supplied with suitable boundary conditions for p. We obtain the pressure boundary conditions taking the dot product of each term in the momentum equation (2.4) with the unit vector $\hat{\mathbf{n}}$ normal to the boundary Γ

$$\frac{\partial}{\partial t}(\mathbf{v} \cdot \hat{\mathbf{n}}) + (\mathbf{v} \cdot \nabla \mathbf{v}) \cdot \hat{\mathbf{n}} = -\frac{1}{\rho}\nabla p \cdot \hat{\mathbf{n}} + \nu \left(\nabla^2 \mathbf{v}\right) \cdot \hat{\mathbf{n}},$$

where we have yet interchanged the time derivative and the dot product in the first term. Using the homogeneous Dirichlet boundary condition for $\mathbf{v}$ we obtain

$$\nabla p \cdot \hat{\mathbf{n}} = \mu \nabla^2 \mathbf{v} \cdot \hat{\mathbf{n}},$$

or, in terms of the normal derivative

$$\frac{\partial p}{\partial \hat{\mathbf{n}}} = \mu \nabla^2 \mathbf{v} \cdot \hat{\mathbf{n}}, \tag{2.10}$$

as may be easily verified, via the divergence theorem, the right hand side of (2.8) and (2.10) satisfy the consistency condition

$$\nu \int_{\Omega} \nabla^2 p \, d\Omega = \int_{\Gamma} \frac{\partial p}{\partial \hat{\mathbf{n}}} \, d\Gamma.$$

We conclude that the pressure field p can be expressed in terms of the velocity field $\mathbf{v}$ by solving the pure Neumann problem defined by equation (2.8) supplied with the boundary condition (2.10) to within an additive constant, and we can write that

$$p = p(\mathbf{v}). \tag{2.11}$$

The initial condition for the pressure field $p(\mathbf{x}, t = 0) = p_o(\mathbf{x})$ for all $\mathbf{x} \in \Omega$, can be obtained noting that (2.11) holds at each instant of time, in particular

$$p_o = p_o\left(\mathbf{v}_o\right). \tag{2.12}$$

up to an additive constant. It is now clear by the preceding remarks, particularly (2.11) and (2.12) that the Navier-Stokes equations amount to an evolution equation for the velocity field $\mathbf{v}$ which is a functional equation and no longer a partial differential equation for p.

2.3 Numerical Methods

The strong formulation of the Navier-Stokes equations is a very complex and difficult problem to solve. Analytical solutions for such system of equations are available only for a few of extremely simplified problems with little practical interest.

Since the flows encountered in our every life experience are very complicated to be tractable by analytical means one must rely on numerical approximations. That is, the continuous mathematical model (2.3-2.7) is transferred to a *discrete* model which approximates the solution in some selected points of the domain. The process will be

referred as *discretization*, making particular emphasis in the context of partial differential equations.

Currently there exists a large variety of methods that have been devised over the years to perform such a task. Some of them are based in very different concepts, other ones share many ideas and techniques among them. As far as we known, the most popular methods have been traditionally the *Finite Difference Method*, *Finite Volume Method*, *Boundary Element Method* and *Finite Element Method*, to name only a few.

Each method has its own cons and pros, as usual, the most accurate and robust are also the most expensive. Low computational cost is usually payed with lost of numerical accuracy and robustness.

In our current knowledge, almost all the discretization methods for partial differential equations leads us a *linear system of equations* that must be solved. The complexity of such solving usually increases as the mathematical model do.

Furthermore, during this *solving process* we spend the majority of the time invested in the overall numerical approximation. For this reason, the efficient solution of such linear systems of equations is of great importance in the numerical discretization of partial differential equations.

Is precisely the fluid dynamics community one of the most demanding in this respect. It is not wear to find in current computations of flows in industrial applications systems having millions and even billions of equations. It seems that in the near future this situation will become even more and more demanding.

Several strategies are used by these numerical schemes to obtain the velocity and pressure fields. Those which obtain both at once are usually referred to *Fully Coupled Schemes*. They lead us to a linear system in which both, the discretized velocity and pressure are included as unknowns. These linear systems have a very particular form. The discretized continuity equation acts like a Lagrange multiplier, a consequence of the incompressibility condition. These linear systems are called *Saddle Point Systems* by its close relationship of those obtained in the context of restricted optimization.

Among the alternatives of this strategy are the decoupled or *Segregated* methods. They obtain the velocity and pressure fields separately. They suppose a velocity guess is known and with it they compute the corresponding pressure field. As we have see, this is possible by using the Poisson Pressure equation that, given a velocity fields determines the pressure. This pressure is used to refine the approximation for the velocity and so on.

Note that the Poisson pressure equation is a Laplacian type operator whose discretization does not represent any special difficulty. Although this direct discretization is seldom done in practice. Instead of that, an approximation of the operator is obtained in terms of those used for the pressure gradient and divergence of the velocity. The common purpose of doing that is the fulfillment of the LBB condition.

In this work we are interested in the solution of such linear systems arising in the discretization of partial differential equations in the computational fluids dynamics context. Special attention will be payed to those corresponding to the Poisson pressure equation.

Chapter 3

Background in Linear Algebra

This chapter provides a general overview of basic concepts of linear algebra that will be useful in subsequent chapters. It is also intended to introduce the notation used in the rest of this work which is quite standard. We have mainly followed the classical introductory book [121], the more focused ones in applied linear algebra [2, 3, 41, 103, 117, 132] and those oriented for the solution of linear systems [8, 64, 66, 71, 78, 87, 107, 138] where detailed proofs can be found.

3.1 Vector Spaces

We begin by defining a *vector space* $\mathcal{V}$ as a non-empty set over a numeric field $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$, whose elements are called *vectors* together with two basic operations, namely *addition* and *scalar multiplication*, that satisfies the following properties:

1. addition is commutative, that is $\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}, \quad \forall\ \boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$;
2. there are two numbers called zero 0 and unity 1 of $\mathbb{K}$ such that $\forall \boldsymbol{v} \in \mathcal{V}$ we have $0 \cdot \boldsymbol{v} = \boldsymbol{0}$ and $1 \cdot \boldsymbol{v} = \boldsymbol{v}$;
3. for each vector $\boldsymbol{v} \in V$ there exists its opposite, $-\boldsymbol{v} \in V$ such that $\boldsymbol{v} + (\boldsymbol{v}) = \boldsymbol{0}$;
4. the scalar multiplication is distributive, that is

$$\forall \alpha \in \mathbb{K},\ \forall \boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}, \quad \alpha\,(\boldsymbol{u} + \boldsymbol{v}) = \alpha \boldsymbol{u} + \alpha \boldsymbol{v}, \tag{3.1}$$

$$\forall \alpha, \beta \in \mathbb{K},\ \forall \boldsymbol{v} \in \mathcal{V}, \quad (\alpha + \beta)\,\boldsymbol{v} = \alpha \boldsymbol{v} + \beta \boldsymbol{v}, \tag{3.2}$$

5. additionally, it is also associative

$$\forall \alpha, \beta \in \mathbb{K},\ \forall \boldsymbol{v} \in \mathcal{V},\quad (\alpha\beta)\,\boldsymbol{v} = \alpha\,(\beta \boldsymbol{v})\,. \tag{3.3}$$

For the sake of simplicity, we shall mainly work with the real case, $\mathbb{K} = \mathbb{R}$, unless otherwise explicitly stated.

We are mainly interested in the set of the $n-$tuples of real numbers with $n \geq 1$, therefore $\mathcal{V} = \mathbb{R}^n$. Of great importance in this work is a nonempty part $\mathcal{W}$ of $\mathcal{V}$ which is called a *vector subspace* of $\mathcal{V}$ iff $\mathcal{W}$ is a vector space over $\mathbb{K}$.

Consider a set of p vectors of $\mathcal{V}$, then the set $\mathcal{W}$ of the linear combination of those p vectors is the *generated subspace* or *span* of the vector set denoted by

$$\begin{aligned} \mathcal{W} &= span\,\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_p\}\,, \\ &= \{\boldsymbol{v} = \alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_p \boldsymbol{v}_p \ \mid\ \alpha_i \in \mathbb{K},\ i = 1, \cdots, p\}\,. \end{aligned} \tag{3.4}$$

The set $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_p\}$ is the *set of generators* of $\mathcal{W}$.

Let $\mathcal{W}_1, \cdots, \mathcal{W}_m$ be vector subspaces if $\mathcal{V}$, then the set

$$\mathcal{S} = \{\boldsymbol{w} : \boldsymbol{w} = \boldsymbol{v}_1 + \cdots + \boldsymbol{v}_m \mid \boldsymbol{v}_i \in \mathcal{W}_i,\ i = 1, \cdots, m\} \tag{3.5}$$

is also a vector subspace of $\mathcal{V}$. This subspace $\mathcal{S}$ is the *direct sum* of the subspaces $\mathcal{W}_i$ if any element $\boldsymbol{s} \in \mathcal{S}$ can be represented as

$$\boldsymbol{s} = \boldsymbol{v}_1 + \cdots + \boldsymbol{v}_m \ \text{ with } \ \boldsymbol{v}_i \in \mathcal{W}_i,\ i = 1, \cdots, m. \tag{3.6}$$

In such a case we write

$$\mathcal{S} = \mathcal{W}_1 \oplus \cdots \oplus \mathcal{W}_m = \bigoplus_{i=1}^{m} \mathcal{W}_i. \tag{3.7}$$

A set of vectors $\boldsymbol{v}_i \in \mathcal{V}$ for $i = 1, \cdots, m$ is *linearly independent* if

$$\alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_m \boldsymbol{v}_m = \boldsymbol{0} \tag{3.8}$$

with $\alpha_1, \cdots, \alpha_m \in \mathbb{K}$ implies $\alpha_1 = \cdots = \alpha_m = 0$. Otherwise the set is called *linearly dependent.*

Any set of linearly independent generators of $\mathcal{V}$ is a *basis* of $\mathcal{V}$. Moreover, consider the set of vectors $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$ is a basis of $\mathcal{V}$, the expression

$$\boldsymbol{v} = v_1 \boldsymbol{u}_1 + \cdots + v_n \boldsymbol{u}_n \tag{3.9}$$

is the *decomposition* of $\boldsymbol{v}$ with respect to the basis and the scalars $v_1, \cdots, v_n \in \mathbb{K}$ are the components of $\boldsymbol{v}$ with respect to the given basis.

Furthermore, let $\mathcal{V}$ be a vector space which admits a basis of n vectors. Then every set of linearly independent vectors of $\mathcal{V}$ has at most n vectors. This number is the *dimension* of $\mathcal{V}$ denoted by $dim\,(\mathcal{V}) = n$. Our interest will be focused on the *finite* case.

3.2 Matrices and Vectors

Let m and n be two positive integers. A *matrix* with m rows and n columns is a set of mn scalars $a_{ij} \in \mathbb{K}$, with $i = 1, \cdots, m$ and $j = 1, \cdots, n$; represented in the rectangular array

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \tag{3.10}$$

The set of all $m \times n$ matrices is a vector space; which in the real case is denoted as $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. The main operations with matrices are

- Addition: $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, where $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are $m \times n$ matrices. The entries of $\boldsymbol{C}$ are given by
$$c_{ij} = a_{ij} + b_{ij}, \quad i = 1, \cdots, m, \quad j = 1, \cdots, n. \tag{3.11}$$

- Scalar multiplication: $\boldsymbol{C} = \alpha \boldsymbol{A}$, where $\boldsymbol{A}$ and $\boldsymbol{A}$ are $m \times n$ matrices; then
$$c_{ij} = \alpha a_{ij} + b_{ij}, \quad i = 1, \cdots, m, \quad j = 1, \cdots, n. \tag{3.12}$$

- Matrix product: $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$, where $\boldsymbol{A} \in \mathbb{R}^{m \times p}$, $\boldsymbol{B} \in \mathbb{R}^{p \times n}$ and $\boldsymbol{C} \in \mathbb{R}^{m \times n}$ with entries given by
$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}. \tag{3.13}$$

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ three special cases can be identified depending on the values of m and n. The first is when $m > 1$ and $n = 1$ leading us a *column vector*; the alternative case when $m = 1$ and $n > 1$ give us a *row vector*. Finally, when $m = n$ we have a *square matrix*; additionally we call the set $(a_{11}, \cdots, a_{nn})$ its *main diagonal*.

Of special interest is the square matrix $\boldsymbol{I}_n$ having as entries all ones in the main diagonal and zeros otherwise. It is called the *identity* of order n and represents the unity of $\mathbb{R}^{n \times n}$. When the order n is clear we shall simply write $\boldsymbol{I}$.

Given $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and the integer p, we define $\boldsymbol{A}^p$ as the product of $\boldsymbol{A}$ repeated p times. Note that $\boldsymbol{A}^0 = \boldsymbol{I}$.

Of special interest are the so called *elementary row operations* performed on a matrix. We list them here

- Pre-multiplying $\boldsymbol{A}$ by a diagonal matrix $\boldsymbol{D} = diag\,(1, \cdots, 1, \alpha, 1, \cdots, 1)$, with α occupying the $i-$th position is equivalent to multiply only the $i-$th row of $\boldsymbol{A}$ by α and left all other rows unchanged.

- Pre-multiplying $\boldsymbol{A}$ by the *elementary permutation matrix* $\boldsymbol{P}^{(i,j)}$ defined as

$$p_{rs}^{(i,j)} = \begin{cases} 1, & \text{if } r = s = 1, \cdots, i-1, i+1, \cdots, j-1, j+1, \cdots n, \\ 1, & \text{if } r = j,\ s = i \ \text{ or } \ r = i,\ s = j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.14}$$

  interchanges the $i-$th and $j-$th rows, leading all remaining entries unchanged. The successive product of elementary permutation matrices form a *permutation matrix*. All them having the special property $\boldsymbol{P}^2 = \boldsymbol{P}$, thus, they are *projectors*.

- Pre-multiplying $\boldsymbol{A}$ by the matrix $\boldsymbol{I} + \boldsymbol{N}_\alpha^{(i,j)}$ with the last one defined as

$$\left(\boldsymbol{N}_\alpha^{(i,j)}\right)_{rs} = \begin{cases} \alpha, & \text{if } r = i \ \text{ and } \ s = j \\ 0, & \text{otherwise.} \end{cases} \tag{3.15}$$

  adds α times the $j-$th row to the $i-$th.

The analogous operations can be also performed by columns by post-multiplying by similarly defined matrices.

A square matrix $\boldsymbol{A}$ is said to be *regular* if there exist a square matrix $\boldsymbol{B}$ of the same order such that $\boldsymbol{AB} = \boldsymbol{BA} = \boldsymbol{I}$. We call such $\boldsymbol{B}$ as the *inverse* of $\boldsymbol{A}$ and denote it by $\boldsymbol{A}^{-1}$. If such an inverse does not exist, then we say that $\boldsymbol{A}$ is *singular*.

The *transpose* of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is a matrix $\boldsymbol{C} \in \mathbb{R}^{n\times m}$ whose entries are given by $c_{ij} = a_{ji}$ and denoted by $\boldsymbol{A}^T$. A square matrix satisfying $\boldsymbol{A} = \boldsymbol{A}^T$ is called *symmetric*. Those for which $\boldsymbol{A} = -\boldsymbol{A}^T$ holds are called *skew-symmetric*.

Finally if $\boldsymbol{AA}^T = \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{D}$, with $\boldsymbol{D}$ a diagonal matrix having non vanishing diagonal entries, then $\boldsymbol{A}$ is *orthogonal*; moreover if $\boldsymbol{D} = \boldsymbol{I}$ it is *orthonormal* implying that $\boldsymbol{A}^{-1} = \boldsymbol{A}^T$. This kind of matrix is commonly denoted by $\boldsymbol{Q}$.

3.3 Eigenvalues and Eigenvectors

Let $\boldsymbol{A}$ be a real or complex square matrix of order n, we call $\lambda \in \mathbb{C}$ an *eigenvalue* of $\boldsymbol{A}$ if there exists a non zero vector $\boldsymbol{x} \in \mathbb{C}$ such that $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. The vector $\boldsymbol{x}$ is the *eigenvector* associated to the eigenvalue λ. The set of all eigenvalues of $\boldsymbol{A}$ is called the *spectrum* of $\boldsymbol{A}$ and it is denoted by $\sigma(\boldsymbol{A})$.

Furthermore, the eigenvalues can be determined by solving the *characteristic equation*

$$p_{\boldsymbol{A}}(\lambda) = \det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0 \tag{3.16}$$

where $p_{\boldsymbol{A}}(\lambda)$ is the *characteristic polynomial*. Since it is of degree n in λ, then there exists n eigenvalues, not necessary distinct.

An eigenvalue has *algebraic multiplicity* k_a if it is a k_a-fold root of the characteristic polynomial. Moreover, for each eigenvalue λ the set of the eigenvectors associated with it, together with the null vector is a subspace known as the *eigenspace* $\mathcal{S}$ associated with λ and dimension k_g called the *geometric multiplicity* of λ. It can not be greater than the algebraic multiplicity k_a of λ. Eigenvalues for which $k_g < k_a$ are called *defective*. A matrix having at least one defective eigenvalue is called *defective*.

We state without proof the following property

$$\det(\boldsymbol{A}) = \prod_{i=1}^{n} \lambda_i, \tag{3.17}$$

from which is easy to conclude that a singular matrix has at least one eigenvalue equals that zero.

We define the *spectral radius* as the maximum modulus of the eigenvalues and denote it by $\rho(\boldsymbol{A})$, that is

$$\rho(\boldsymbol{A}) = \max_{\lambda \in \sigma(\boldsymbol{A})} = |\lambda|. \tag{3.18}$$

Using the characteristic equation for $\boldsymbol{A}^T$ it follows that $\rho(\boldsymbol{A}) = \rho\left(\boldsymbol{A}^T\right)$. Moreover, let $\alpha \in \mathbb{C}$ we have $\rho(\alpha\boldsymbol{A}) = |\alpha|\rho(\boldsymbol{A})$ and given k a positive integer we have $\rho\left(\boldsymbol{A}^k\right) = [\rho(\boldsymbol{A})]^k$.

We say that two regular matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are *similar* if there exits a third regular matrix $\boldsymbol{C}$ such that

$$\boldsymbol{B} = \boldsymbol{C}^{-1}\boldsymbol{A}\boldsymbol{C}. \tag{3.19}$$

One of the most important and useful properties of this *similarity transformation* is the fact that $\sigma(\boldsymbol{A}) = \sigma(\boldsymbol{B})$. Moreover, it is easy to verify that, given an *eigenpair* $(\lambda, \boldsymbol{x})$ of $\boldsymbol{A}$ then $\left(\lambda, \boldsymbol{C}^{-1}\boldsymbol{x}\right)$ is an eigenpair of the similar matrix $\boldsymbol{B}$.

We say that $\boldsymbol{A}$ is *diagonalizable* if it is similar to a diagonal matrix, in this particular case we write

$$\boldsymbol{A}\boldsymbol{S} = \boldsymbol{S}\boldsymbol{\Lambda}, \tag{3.20}$$

with the columns of $\boldsymbol{S}$ being the eigenvectors; which are each other orthogonal; and $\boldsymbol{\Lambda}$ a diagonal matrix containing the corresponding eigenvalues.

A special case is when $\boldsymbol{A}$ is real and symmetric since it can be proven that the spectrum is real and we order the eigenvalues in non-decreasing form

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq \lambda_n. \tag{3.21}$$

The greatest one λ_n is known as the *dominant eigenvalue* and the smallest one is the *inverse dominant eigenvalue*. The latter case comes from the fact that given a regular matrix $\boldsymbol{A}$, the eigenvalues of its inverse $\boldsymbol{A}^{-1}$ are the reciprocals of those eigenvalues of $\boldsymbol{A}$.

3.4 Norms

In subsequent chapters of the present work we will need to quantify errors or measure distances in some sense which can be carried out by computing the magnitude of a vector or a matrix. Then we introduce here the vector norm concept and after extend it to matrices.

A vector space $\mathcal{V}$ over $\mathbb{K}$ is an *inner vector space* if its is endowed with a function $\mathcal{V} \times \mathcal{V} \to \mathbb{K}$ denoted by $(\cdot, \cdot)$ called *inner product* which enjoys the following properties

- it is linear with respect to the vectors of $\mathcal{V}$, that is

$$(\alpha\boldsymbol{x} + \beta\boldsymbol{z}, \boldsymbol{y}) = \alpha(\boldsymbol{x}, \boldsymbol{y}) + \beta(\boldsymbol{z}, \boldsymbol{y}), \quad \forall\ \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathcal{V} \ \text{ and } \ \alpha, \beta \in \mathbb{K}; \tag{3.22}$$

- is is *Hermitian*, that is

$$(\boldsymbol{x}, \boldsymbol{y}) = \overline{(\boldsymbol{y}, \boldsymbol{x})}, \quad \forall\ \boldsymbol{x}, \boldsymbol{y} \in \mathcal{V} \tag{3.23}$$

  where the overbar denotes the complex conjugate. Note that when $\mathbb{K} = \mathbb{R}$ we can omit it.

- it is *positive definite*, that is

$$(\boldsymbol{x}, \boldsymbol{x}) \geq 0, \quad \forall\, \boldsymbol{x} \neq \boldsymbol{0} \tag{3.24}$$

  equality holds iff $\boldsymbol{x} = \boldsymbol{0}$.

In the particular case of the vector space $\mathcal{V} = \mathbb{K}^n$ the *Euclidean inner product* is of paramount importance. It is defined as

$$(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} x_i \overline{y}_i. \tag{3.25}$$

Moreover, consider the case $\mathcal{V} = \mathbb{R}^n$ and any square matrix $\boldsymbol{y} \in \mathbb{R}^{n \times n}$, the following relation holds

$$(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{x}, \boldsymbol{A}^T \boldsymbol{y}\right). \tag{3.26}$$

It is easy to verify that orthonormal matrices $\boldsymbol{Q}$ preserves the Euclidean inner product

$$(\boldsymbol{Q}\boldsymbol{x}, \boldsymbol{Q}\boldsymbol{y}) = \left(\boldsymbol{x}, \boldsymbol{Q}^T \boldsymbol{Q} \boldsymbol{y}\right) = (\boldsymbol{x}, \boldsymbol{y}). \tag{3.27}$$

Once a vector space has been endowed with an inner product, it is possible to endow it with a *vector norm* $\mathcal{V} \to \mathbb{K}$, denoted by $||\cdot||$, in order to obtain a *normed vector space* $(\mathcal{V}, ||\cdot||)$ having the following properties

- it is positive definite, that is

$$||\boldsymbol{x}|| \geq 0 \quad \forall\, \boldsymbol{x} \in \mathcal{V} \tag{3.28}$$

  equality holds iff $\boldsymbol{x} = \boldsymbol{0}$.

- it is *homogeneous*, that is

$$||\alpha \boldsymbol{x}|| = |\alpha|\, ||\boldsymbol{x}|| \quad \forall\, \alpha \in \mathbb{K} \;\text{ and }\; \forall\, \boldsymbol{x} \in \mathcal{V} \tag{3.29}$$

- *triangle inequality*

$$||\boldsymbol{x} + \boldsymbol{y}|| \leq ||\boldsymbol{x}|| + ||\boldsymbol{y}|| \quad \forall\, \boldsymbol{x}, \boldsymbol{y} \in \mathcal{V}. \tag{3.30}$$

Given a vector space $\mathcal{V}$, several norms can be defined in it, they will be distinguished among each other by suitable subscripts. For example, some of the most important norms for $\mathcal{V} = \mathbb{R}^n$ are the *Hölder* norms defined as

$$||\boldsymbol{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \quad \text{for} \;\; 1 \leq p \leq \infty. \tag{3.31}$$

In the limit $p \to \infty$ the norm $||\boldsymbol{x}||_p$ does exits, is finite and is called the *maximum* norm since it equals the maximum module of the components of $\boldsymbol{x}$, that is

$$||\boldsymbol{x}||_\infty = \max_{1 \leq i \leq n} |x_i|. \tag{3.32}$$

When $p = 1$ the norm $||\boldsymbol{x}||_p$ is simply the module sum of the components of $\boldsymbol{x}$, that is

$$||\boldsymbol{x}||_1 = \sum_{i=1}^{n} |x_i|, \tag{3.33}$$

Another special and very useful norm is when $p = 2$, notice that this norm is induced by the Euclidean inner product and therefore is referred as the *Euclidean norm* given by

$$||\boldsymbol{x}||_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}} = \sqrt{(\boldsymbol{x}, \boldsymbol{x})}, \tag{3.34}$$

for which the *Cauchy-Schwarz inequality* holds which reads as follows

$$|\,(\boldsymbol{x}, \boldsymbol{y})\,| \leq ||\boldsymbol{x}||_2 \; ||\boldsymbol{y}||_2, \tag{3.35}$$

where strict equality holds iff one vector is a multiple of the other one.

The definition of norm makes now clear the definition of *unitary vector* which is one with unit norm, that is $\boldsymbol{x} \in \mathcal{V}$ is unitary iff $||\boldsymbol{x}||_2 = 1$.

It makes also clear, using together with the inner product, a more precise definition of *orthogonality*. Given two non null vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{V}$ they are each other orthogonal if $(\boldsymbol{x}, \boldsymbol{y}) = 0$.

Notice also that orthogonal matrices preserves the Euclidean norm of any vector

$$||\boldsymbol{Q}\boldsymbol{x}||_2^2 = (\boldsymbol{Q}\boldsymbol{x}, \boldsymbol{Q}\boldsymbol{x}) = \left(\boldsymbol{x}, \boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{x}\right) = (\boldsymbol{x}, \boldsymbol{x}) = ||\boldsymbol{x}||_2^2. \tag{3.36}$$

During this work we will be interested in sequences of vectors $\left\{\boldsymbol{x}^{(k)}\right\}$ converging to the vector $\boldsymbol{x} \in \mathcal{V}$ such that

$$\lim_{k \to \infty} \boldsymbol{x}^{(k)} = \boldsymbol{x} \quad \text{if} \quad \lim_{k=\infty} x_i^{(k)} = x_i, \quad \text{for} \;\; i = 1, \cdots, n; \tag{3.37}$$

that is, local convergence is a necessary condition to attain global convergence. If this is the case we have

$$\lim_{k=\infty} \boldsymbol{x}^{(k)} = \boldsymbol{x} \quad \Leftrightarrow \quad \lim_{k=\infty} ||\boldsymbol{x}^{(k)} - \boldsymbol{x}|| = 0. \tag{3.38}$$

The space $\mathbb{K}^{(n\times n)}$ containing the $(n\times n)$ matrices is also a linear vector space of dimension n^2; hence, we may define norms on it. We call them as *matrix norms* and in sake of clearness the previously discussed ones are *vector norms*.

The generalization of the Euclidean vector norm leads to the *Frobenius norm* defined as

$$||\boldsymbol{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}|^2}. \tag{3.39}$$

The most important matrix norms are those *induced* by a vector norm, in such a case they are defined as

$$||\boldsymbol{A}|| = \sup_{||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||, \tag{3.40}$$

all these norms are positive definite, homogeneous and satisfy the triangle inequality.

Among the most important matrix norms are the *1-norm* and *infinite norm* defined as

$$||\boldsymbol{A}||_1 = \max_{j=1,\cdots,n}\sum_{i=1}^{n}|a_{ij}|, \quad \text{and} \quad ||\boldsymbol{A}||_\infty = \max_{i=1,\cdots,n}\sum_{j=1}^{n}|a_{ij}|. \tag{3.41}$$

Probably the most important matrix norm is the one obtained with $p = 2$ and known as the *spectral norm*, we state without prof the fact that when $\boldsymbol{A} = \boldsymbol{A}^T \in \mathbb{R}^{(n\times n)}$ we have

$$||\boldsymbol{A}||_2 = \rho\left(\boldsymbol{A}\right), \tag{3.42}$$

having the important consequence that, for orthogonal matrices $||\boldsymbol{Q}||_2 = 1$.

3.5 Special Matrices

Several kinds of special matrices have been mentioned during the present chapter depending mainly on two different criteria. The first criteria is related with the *structural* properties of the matrices at hand as square or diagonal matrices. The second criteria is related with the algebraic properties of matrices such as regularity, projection, defectiveness, symmetry or orthogonality.

In this concluding section we shall mention some special matrices endowing certain useful properties that will be exploited in subsequent chapters.

We shall consider here only real matrices $\boldsymbol{A} \in \mathbb{R}^{(m\times n)}$, the extension to the complex case is straightforward. We say that a matrix $\boldsymbol{A} \in \mathbb{R}^{(m\times n)}$ has *lower band* p if $a_{ij} = 0$ when $i > j + p$ and *upper band* q if $a_{ij} = 0$ when $j > i + q$. Note that the yet mentioned

diagonal matrices are *banded matrices* with $p = q = 0$, while *lower trapezoidal matrices* have $p = m - 1$, $q = 0$ and *upper trapezoidal matrices* have $p = 0$, $q = n - 1$.

Special cases of trapezoidal matrices arise when they are also square, in such situations we have a *lower triangular matrix* for which $a_{ij} = 0$ when $i < j$ and similarly a *upper triangular matrix* for which $a_{ij} = 0$ when $i > j$. They are usually denoted by $\boldsymbol{L}$ and $\boldsymbol{U}$, respectively, and have the generic form

$$\boldsymbol{L} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \quad \text{and} \quad \boldsymbol{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}, \tag{3.43}$$

additionally, when the entries on the main diagonal are all equal to one, then they are called *unit lower* and *unit upper* triangular matrices.

Other useful cases of banded square matrices are the *tridiagonal matrices* for which $p = q = 1$. They are frequently denoted in abbreviated form as $tridiag_n(\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{c})$, having as lower diagonal the vector $\boldsymbol{b} = [b_1, \cdots, b_{n-1}]^T$, main diagonal $\boldsymbol{d} = [d_1, \cdots, d_n]^T$ and upper diagonal $\boldsymbol{c} = [c_1, \cdots, c_{n-1}]^T$.

Of great importance in this work are the so-called *upper Hessenberg matrices*, for which $h_{ij} = 0$ for any pair i, j such that $i > j + 1$. In the square case they will be denoted as $\boldsymbol{H}_n$ and the $(n + 1) \times n$ case by $\overline{\boldsymbol{H}}_n$. They can be sketched as

$$\boldsymbol{H}_n = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & h_{n,n-1} & h_{nn} \end{bmatrix} \quad \text{and} \quad \overline{\boldsymbol{H}}_n = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & h_{n,n-1} & h_{nn} \\ 0 & 0 & 0 & h_{n+1,n} \end{bmatrix}, \tag{3.44}$$

the definition of the lower cases are straightforward.

The *Givens elementary matrices* are orthogonal rotation matrices which allow us to introduce zeros in certain position of a matrix or a vector. Given a pair of indices r and s and an angle θ, we define these matrices as

$$\boldsymbol{G}(r, s, \theta) = \boldsymbol{I} - \boldsymbol{Y} \tag{3.45}$$

where $\boldsymbol{Y} \in \mathbb{R}^{n\times n}$ is defined as

$$y_{ij} = \begin{cases} 1-\cos(\theta), & \text{if } i=j=r \text{ or } i=j=s \\ -\sin(\theta), & \text{if } i=r \text{ and } j=s \\ \sin(\theta), & \text{if } i=s \text{ and } j=r \\ 0, & \text{otherwise.} \end{cases} \tag{3.46}$$

For a given vector $\boldsymbol{x} \in \mathbb{R}^n$, the product $\boldsymbol{z} = \boldsymbol{G}^T\boldsymbol{x}$ rotates $\boldsymbol{x}$ counterclockwise by an angle θ in the plane (x_i, x_j). Then the components of $\boldsymbol{z}$ are given by

$$z_k = \begin{cases} x_k, & \text{if } k \neq i,j \\ cx_i - sx_j, & \text{if } k=i \\ sx_i + cx_j, & \text{if } k=j \end{cases} \tag{3.47}$$

where we have done $c = \cos(\theta)$ ans $s = \sin(\theta)$. If we take $\theta = \tan^{-1}(-x_j/x_i)$ we get $z_j = 0$ and $z_i = \sqrt{x_i^2 + x_j^2}$. The opposite case with $z_i = 0$ and $z_j = \sqrt{x_i^2 + x_j^2}$ is obtained taking $\theta = \tan^{-1}(x_i/x_j)$.

For concluding this section we shall review one of the fundamental concepts on which is based the overall present work, namely *sparsivity*. Going back to the basic definition of matrix we see that it contains n^2 in the square case, such a matrix is referred to as a *dense* matrix. We have seen also special cases of matrices with only some of their entries as non zeros such as banded matrices which for instance are a special case of *sparse* matrices. Although sparse matrix is defined somewhat vaguely we can regard a matrix as sparse whenever special techniques can be used to take advantage of the, possibly large, number of zero entries and their location. The most basic idea is the fact that these zeros may not be stored.

3.6 Linear Systems

A *linear system* of m equations with n unknowns is a set of algebraic linear relations of the form

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad \text{for} \quad i = 1, \cdots, m \tag{3.48}$$

where x_j are the unknowns, a_{ij} are the coefficients of the linear system and b_i are the components of the right hand side. During this work we are mainly interested in the real case when the number of unknowns is equal to the number of equations $m = n$.

Therefore the system (3.48) can be written compactly in matrix form as

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.49}$$

where $\boldsymbol{A} \in \mathbb{R}^{(n\times n)}$ is the *coefficient matrix*, $\boldsymbol{b} \in \mathbb{R}^n$ is the *right hand side* and $\boldsymbol{x} \in \mathbb{R}^n$ is the *solution vector*.

Solution a linear system means, given $\boldsymbol{A} \in \mathbb{R}^{(n\times n)}$ and $\boldsymbol{b} \in \mathbb{R}^n$ find $\boldsymbol{x}$ satisfying (3.49).The existence and uniqueness of the solution are ensured if the coefficient matrix is regular, in such a case the solution is given by

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}. \tag{3.50}$$

The solution of linear systems is among the most important, common and time consuming problems in scientific computing. These linear systems arise in a wide range of applications; the main is probably the discretization of partial differential equations but several branches of science such as chemical engineering processes, economic models and analysis of circuits, to name only a few, lead linear systems to solve.

Dense direct solvers for solving (3.49), mainly based on the factorization $\boldsymbol{A}$ into easily invertible matrices, have been the first techniques devised to handle the problem but they suffer from the significant drawback that, although the operation count is fixed and known it is usually $\mathcal{O}\left(n^3\right)$.

Important savings in storage and computational cost can be achieved if the matrix involved is sparse, leading to specialized implementations known as *sparse direct solvers* which are widely used in industrial and commercial applications. This is mainly due to the fact that they are robust, well studied and require predictable amounts of memory and operations. But even so this methods still suffers from poor scaling as the dimension of the linear systems grow even more and more.

Iterative methods represent an alternative technique for the same purpose because in principle they does not require either access or modify individual entries of the coefficient matrix. It is required only to form matrix by vector products, that is, given a vector $\boldsymbol{y}$ we need to compute $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{y}$, and in the unsymmetric case some methods also require the product $\boldsymbol{A}^T\boldsymbol{y}$. In both cases suppose the matrix has, as average, μ entries per row then we need approximately μn arithmetic operations to perform such a product.

Then an ideal iterative method is one that finds the solution $\boldsymbol{x}$ after n iterations and for which the main cost per iteration is involved in evaluating matrix by vector products, thus the arithmetic operation count is $\mathcal{O}\left(\mu n^2\right)$. If $\mu \ll n$, considerable gains in time could be achieved. Furthermore, iterative methods, in principle, does not require

more memory positions for the matrix itself and usually a few additional vectors are required. This situation is when iterative methods are competitive, faster or even the only alternative to direct methods.

Although in the preceding discussion we have segregated in a very simplistic way direct and iterative solvers, we shall have the opportunity to see that currently the border between both kind of techniques are more and more blurred, borrowing ideas and concepts mutually each other. This will be more clear in subsequent chapters, particularly in the one devoted to preconditioning.

Chapter 4

Krylov Subspace Methods

4.1 Iterative Methods

Now we are faced with the problem of solving a linear system as (3.49) by an *iterative method*. The basic idea underlying this kind of techniques is to construct a sequence of vectors $\left\{\boldsymbol{x}^{(k)}\right\}$ such that

$$\lim_{k\to\infty} \boldsymbol{x}^{(k)} = \boldsymbol{x}, \tag{4.1}$$

where $\boldsymbol{x}$ will be referred as the *exact solution* given by (3.50) and $\boldsymbol{x}^{(k)}$ is the $k-$th approximation. When $k = 0$ it is called the *initial guess*. Note that in (4.1) we have implicitly assumed that the method is *convergent* a matter we shall discuss about briefly.

In theory, the iterative process consists of an infinite number of iterations, although in practice it is stopped as soon as the inequality $||\boldsymbol{x}^{(k)} - \boldsymbol{x}|| < \epsilon$ is satisfyed, being $||\cdot||$ a suitable norm and ϵ a predefined parameter called *tolerance*. However, the exact solution $\boldsymbol{x}$ is not known making this stopping criterion useless in practice, then it is necessary to introduce suitable stopping criteria.

In doing so we define two vectors that will be useful in measuring the effectiveness of the iterative process. The first one is the *error vector* of the $k-$th approximation as

$$\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}, \tag{4.2}$$

the second one is the *residual vector* of the $k-$th approximation as

$$\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}. \tag{4.3}$$

Solving for $\boldsymbol{x}^{(k)}$ from (4.2) and substituting in (4.3) we found that both are related by

$$\boldsymbol{A}\boldsymbol{e}^{(k)} = -\boldsymbol{r}^{(k)}. \tag{4.4}$$

We consider a method of the form

$$\text{Given} \quad \boldsymbol{x}^{(0)}, \qquad \boldsymbol{x}^{(k+1)} = \boldsymbol{B}\boldsymbol{x}^{(k)} + \boldsymbol{f}, \quad k \geq 0, \tag{4.5}$$

with $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ the *iteration matrix* and $\boldsymbol{f} \in \mathbb{R}^n$ obtained from $\boldsymbol{b}$.

From (4.1) we expect that the difference between two successive approximations $\boldsymbol{x}^{(k)}$ and $\boldsymbol{x}^{(k+1)}$ becomes more and more negligible as k grows, implying that the solution $\boldsymbol{x}$ is a *fixed point* of (4.5), that is

$$\boldsymbol{x} = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{f}, \tag{4.6}$$

where, in order to compute $\boldsymbol{f}$ we use the exact solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$

$$\boldsymbol{f} = (\boldsymbol{I} - \boldsymbol{B})\,\boldsymbol{A}^{-1}\boldsymbol{b}. \tag{4.7}$$

Such a method for which (4.6) is satisfied is known to be *consistent*. Note that this analysis is only of theoretical interest since it implies the computation of the inverse of the coefficient matrix. Furthermore, consistency is a necessary but not sufficient condition for guarantee that effectively, when k grows $\boldsymbol{x}^{(k)}$ is a more accurate approximation to $\boldsymbol{x}$ in some sense; that is, if the method is *convergent*.

In order to verify if an iterative method is convergent we take $\boldsymbol{f}$ as in (4.7) and substitute it in (4.6) to get

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{B}\boldsymbol{x}^{(k)} + (\boldsymbol{I} - \boldsymbol{b})\,\boldsymbol{A}^{-1}\boldsymbol{b}, \tag{4.8}$$

substituting the exact solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ we have

$$\boldsymbol{x}^{(k+1)} - \boldsymbol{x} = \boldsymbol{B}\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}\right), \tag{4.9}$$

by the error vector definition (4.2) and induction on k we can write

$$\boldsymbol{e}^{(k+1)} = \boldsymbol{B}\boldsymbol{e}^{(k)} \quad \Rightarrow \quad \boldsymbol{e}^{(k)} = \boldsymbol{B}^k\boldsymbol{e}^{(0)}. \tag{4.10}$$

Using the diagonalization of the iteration matrix $\boldsymbol{B}$ as in (3.20) and the already mentioned result $\rho\left(\boldsymbol{A}^k\right) = \left[\rho\left(\boldsymbol{A}\right)\right]^k$ we have

$$\boldsymbol{B} = \boldsymbol{S}\Lambda\boldsymbol{S}^{-1} \quad \Rightarrow \quad \boldsymbol{B}^k = \boldsymbol{S}\Lambda^k\boldsymbol{S}^{-1}, \tag{4.11}$$

thus, for any initial guess, convergence is ensured if

$$\lim_{k\to\infty} \boldsymbol{B}^k \boldsymbol{e}^{(0)} = \boldsymbol{0}, \quad \forall\ \boldsymbol{e}^{(0)} \qquad \Leftrightarrow \qquad \rho\left(\boldsymbol{B}\right) = \max_{\lambda\in\sigma(\boldsymbol{B})} |\lambda| < 1. \tag{4.12}$$

The most intuitive way to construct an iterative method is using a *splitting* of the form $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ where $\boldsymbol{M}$ is easily invertible. Then, given $\boldsymbol{x}^{(0)}$, we compute $\boldsymbol{x}^{(k+1)}$ for $k \geq 0$ solving the systems

$$\boldsymbol{M}\boldsymbol{x}^{(k+1)} = \boldsymbol{N}\boldsymbol{x}^{(k)} + \boldsymbol{b}. \tag{4.13}$$

Multiplying by $\boldsymbol{M}^{-1}$ and substituting $\boldsymbol{N} = \boldsymbol{M} - \boldsymbol{A}$ we get

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}, \tag{4.14}$$

where $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ is the $k-$th residual vector. Furthermore it is posible to introduce a *relaxation* parameter α_k

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}, \tag{4.15}$$

which leads us to the *non stationary Richardson* method. Otherwise, if $\alpha_k = \alpha$ we obtain the *stationary Richardson* method.

We now consider the most simple case by taking $\boldsymbol{M} = \boldsymbol{I}$ and $\alpha = 1$, implying $\boldsymbol{B} = \boldsymbol{I} - \boldsymbol{A}$, we obtain the simple Richardson method which iterates as

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{r}^{(k)}, \tag{4.16}$$

multiplying by $-\boldsymbol{A}$ and adding $\boldsymbol{b}$ in both sides we have

$$\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{A}\boldsymbol{r}^{(k)}, \tag{4.17}$$

which in turns can be expressed in terms of residuals

$$\boldsymbol{r}^{(k+1)} = \left(\boldsymbol{I} - \boldsymbol{A}\right)\boldsymbol{r}^{(k)}, \tag{4.18}$$

and finally, by induction on k, we can write

$$\boldsymbol{r}^{(k+1)} = \left(\boldsymbol{I} - \boldsymbol{A}\right)^{k+1}\boldsymbol{r}^{(0)}. \tag{4.19}$$

Expression (4.19) can be modified to handle the non stationary case as

$$\boldsymbol{r}^{(k+1)} = \left(\boldsymbol{I} - \alpha_k\boldsymbol{A}\right)^{k}\boldsymbol{r}^{(0)}. \tag{4.20}$$

More important is the fact that $\boldsymbol{x}^{(k+1)}$ can be expressed as a linear combination of the initial guess and the previous residuals

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(0)} + \sum_{j=0}^{k} \alpha_j \boldsymbol{r}^{(j)}. \tag{4.21}$$

which is the departure expression for the subsequent section.

4.2 The Krylov Subspace

From (4.20) it is clear that the $k-$th residual lies in the subspace

$$\boldsymbol{r}^{(k)} \in span\left\{\boldsymbol{r}^{(0)}, \boldsymbol{A}\boldsymbol{r}^{(0)}, \boldsymbol{A}^2\boldsymbol{r}^{(0)}, ..., \boldsymbol{A}^{k-1}\boldsymbol{r}^{(0)}\right\}. \tag{4.22}$$

The $m-$dimensional subspace spanned by a given vector $\boldsymbol{v}$ and increasing powers of $\boldsymbol{A}$ applied to $\boldsymbol{v}$, until the $(m-1)-$th power is the $m-$dimensional *Krylov subspace* which we now define formally as

$$\boldsymbol{\mathcal{K}}_m\left(\boldsymbol{A}; \boldsymbol{v}\right) = span\left\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \boldsymbol{A}^2\boldsymbol{v}, ..., \boldsymbol{A}^{m-1}\boldsymbol{v}\right\}. \tag{4.23}$$

Moreover, from (4.21) it is also clear that we can find approximations to the solution using a basis constructed with this subspace as follows

$$\boldsymbol{x}^{(k+1)} \in \boldsymbol{x}^{(0)} \oplus \boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}; \boldsymbol{r}^{(0)}\right), \tag{4.24}$$

that is, the correction that makes the initial guess closer to the solution of the linear system (3.49) also lies in this Krylov subspace, therefore, in order to find better successive approximations a good alternative is explore the Krylov subspace.

Methods attempting to improve approximations using the Krylov subspace are usually referred as *Krylov subspace methods*. These methods can be roughly classified in four families depending on the manner in which they identify $\boldsymbol{x} \in \boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}; \boldsymbol{r}^{(0)}\right)$:

1. *Ritz-Galerkin.-* Constructs $\boldsymbol{x}^{(k)}$ in a way that the residual must be orthogonal to the Krylov subspace: $\boldsymbol{r}^{(k)} \perp \boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}; \boldsymbol{r}^{(0)}\right)$.

2. *Residual norm minimization.-* Selects $\boldsymbol{x}^{(k)}$ such that $||\boldsymbol{r}^{(k)}||_2$ is minimized over $\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}; \boldsymbol{r}^{(0)}\right)$.

3. *Petrov-Galerkin.-* Constructs $\boldsymbol{x}^{(k)}$ in a way that the residual is orthogonal to other subspace: $\boldsymbol{r}^{(k)} \perp \boldsymbol{\mathcal{L}}_k$.

4. *Error norm minimization.*- Selects $\boldsymbol{x}^{(k)}$ such that the error norm $||\boldsymbol{e}^{(k)}||_2$ is minimized on $\boldsymbol{A}^T\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}^T;\boldsymbol{r}^{(0)}\right)$.

4.3 Arnoldi Methods

In order to generate better approximations $\boldsymbol{x}^{(k)}$ to the solution (3.50) with any of the families already enumerated we must first construct a suitable basis for the Krylov subspace $\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A};\boldsymbol{r}^{(0)}\right)$. The most obvious one could be

$$\boldsymbol{r}^{(0)}, \boldsymbol{A}\boldsymbol{r}^{(0)}, \boldsymbol{A}^2\boldsymbol{r}^{(0)}, ..., \boldsymbol{A}^{k-1}\boldsymbol{r}^{(0)}. \tag{4.25}$$

However, this basis results being not very good from the numerical point of view because as k increases the successive generated vectors $\boldsymbol{A}^j\boldsymbol{r}^{(0)}$ align more and more in the direction of the dominant eigenvalue as done with the power method. Hence they becomes to be linear dependent doing the basis formed with them ill conditioned.

In order to generate a most suitable basis for the Krylov subspace, an process can be used and in this manner obtain an *orthogonal basis* for the Krylov subspace $\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A};\boldsymbol{r}^{(0)}\right)$.

This situation can be handle by the *Arnoldi's method* introduced in 1951 as a procedure for transforming a general matrix into Hessenberg form of dimension $m \ll n$ with the purpose to, instead of solving linear systems, approximate the extreme eigenvalues of the original matrix. The method proved its profitability even more when the matrices at hand were large and sparse. Moreover it can also be used for constructing an orthogonal basis of the Krylov subspace.

The mentioned orthogonalization process can be performed with the *classical Gram-Schmidt* procedure which for a fixed m and a unit vector $||\boldsymbol{v}||_2 = 1$, it computes for $k = 1, 2, ..., m$

$$h_{ik} = \left(\boldsymbol{v}_i, \boldsymbol{A}\boldsymbol{v}_k\right), \quad i = 1, 2, ..., k \tag{4.26}$$

$$\tilde{\boldsymbol{v}}_{k+1} = \boldsymbol{A}\boldsymbol{v}_k - \sum_{i=1}^{k} h_{ik}\boldsymbol{v}_i, \tag{4.27}$$

$$h_{k+1,k} = ||\tilde{\boldsymbol{v}}_{k+1}||_2, \tag{4.28}$$

$$\boldsymbol{v}_{k+1} = \tilde{\boldsymbol{v}}_{k+1}/||\tilde{\boldsymbol{v}}_{k+1}||_2. \tag{4.29}$$

note that, if $h_{k+1,k} = 0$ in (4.28) the algorithm stops because in such a case (4.29) is not defined; but this is not an undesired situation as we shall see shortly. The vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{k+1}$ are called *Arnoldi vectors*.

Unfortunately, it is widely known that the classical Gram-Schmidt suffers of loss of orthogonality among the computed vectors in finite precision arithmetic due to the presence of round off errors. A simple remedy of this situation leads to what it is commonly referred as the *Modified Gram-Schmidt* orthogonalization procedure which we present in the algorithm 4.3.

Given a vector $\boldsymbol{v}_1$ with $||\boldsymbol{v}_1||_2 = 1$

for $k = 1, 2, ..., m$ **do**

  $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{v}_k$

  for $i = 1, 2, ..., k$ **do**

    $h_{ik} = (\boldsymbol{w}, \boldsymbol{v}_i)$

    $\boldsymbol{w} = \boldsymbol{w} - h_{ik}\boldsymbol{v}_k$

  end for

  $h_{k+1,k} = ||\boldsymbol{w}||_2$. If $h_{k+1,k} = 0$ stop.

  $\boldsymbol{v}_{k+1} = \boldsymbol{w}/h_{k+1,k}$

end for

ALGORITHM 1: Modified Gram-Schmidt Orthogonalization.

It is important to mention that both algorithms perform the same number of arithmetic operations, thus their computational costs are the same. Additionally, it is easy to verify that in exact arithmetic without round off errors, both algorithms, the classical and modified Gram-Schmidt are equivalent. A third and even more accurate implementation uses Householder reflectors to achieve the required orthogonality but in this case the operations count is slightly increased. Other alternatives have been proposed in order to improve the orthogonality among the Arnoldi vectors, one of them consists in performing a re-orthogonalization of the last computed Arnoldi vector against the set already stored.

Now, suppose k steps of the Arnoldi algorithm has been run, then $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k$ is an orthonormal basis for $\boldsymbol{\mathcal{K}}_k(\boldsymbol{A}; \boldsymbol{v})$, if additionally we define the $n \times k$ matrix satisfying $\boldsymbol{V}_k^T \boldsymbol{V}_k = \boldsymbol{I}_k$ as

$$\boldsymbol{V}_k = [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k] \in \mathbb{R}^{n \times k}, \tag{4.30}$$

we have that

$$\boldsymbol{V}_k^T \boldsymbol{A}\, \boldsymbol{V}_k = \boldsymbol{H}_{kk} \quad \text{and} \quad \boldsymbol{V}_{k+1}^T \boldsymbol{A}\, \boldsymbol{V}_k = \boldsymbol{H}_{k+1,k}, \tag{4.31}$$

where $\boldsymbol{H}_{k+1,k} \in \mathbb{R}^{k+1 \times k}$ is the upper Hessenberg matrix with h_{ij} computed by the Arnoldi algorithm.

By setting the first Arnoldi vector as $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$, the Arnoldi algorithm will provide us with an orthonormal basis for the Krylov subspace and according

with (4.24) the $k-$th approximation will be given by

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_k \boldsymbol{y}^{(k)}, \tag{4.32}$$

where $\boldsymbol{y}^{(k)}$ is chosen in accordance with one of the four families of Krylov methods yet mentioned.

4.3.1 Full Orthogonalization Method

We first study a method based on the Ritz-Galerkin condition satisfying $\boldsymbol{r}^{(k)} \perp \boldsymbol{\mathcal{K}}_k \left(\boldsymbol{A}; \boldsymbol{r}^{(0)}\right)$, which is equivalent to

$$\boldsymbol{V}_k^T \boldsymbol{r}^{(k)} = \boldsymbol{0}, \tag{4.33}$$

since $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and taking $\boldsymbol{x}^{(k)}$ as (4.32) we obtain

$$\boldsymbol{V}_k^T \boldsymbol{A}\, \boldsymbol{V}_k \boldsymbol{y}^{(k)} = \boldsymbol{V}_k^T \boldsymbol{r}^{(0)}, \tag{4.34}$$

this system is the projection of the original system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ onto the Krylov subspace.

Given the orthogonality of $\boldsymbol{V}_k$ and the choice of $\boldsymbol{v}_1$ as the normalized initial residual, it follows that $\boldsymbol{V}_k^T \boldsymbol{r}^{(0)} = \rho \hat{\boldsymbol{e}}_1$ with $\hat{\boldsymbol{e}}_1$ being the first canonical unit vector in $\mathbb{R}^k$. Moreover, we do not need to additionally compute the $(k \times k)$ matrix $\boldsymbol{V}_k^T \boldsymbol{A}\, \boldsymbol{V}_k$ since it has already been provided by the Arnoldi method as stated in the first expression in (4.31). Then we have

$$\boldsymbol{H}_{k,k} \boldsymbol{y}^{(k)} = \rho \hat{\boldsymbol{e}}_1. \tag{4.35}$$

Now we must solve this system for $\boldsymbol{y}^{(k)}$. Once it has been determined we obtain $\boldsymbol{x}^{(k)}$ substituting it in (4.32). An obvious option is to use Gaussian elimination in order to obtain the LU factorization of the upper Hessenberg matrix $\boldsymbol{H}_{k,k}$. It is important to notice that this matrix has a well defined and regular structure which must be exploited in order to save storage and reduce the operation count.

All this together gives us the *Full orthogonalization method*, abbreviated in what follows as *FOM*. Note that, as k increases, the computational cost increases as $\mathcal{O}\left(k^2 n\right)$ and the storage requirements as $\mathcal{O}\left(kn\right)$ because of the Gram-Schmidt orthogonalization. If n is large, as in the case we are mainly interested, these upper bounds could be unacceptable and k must be limited for a practical implementation.

One way to carry out such a practical implementation relies on *restarting* the algorithm periodically. First we chose a $k \ll n$, and with the initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{A}\boldsymbol{x}^{(0)}$ run the Arnoldi algorithm to get, together the solution of the projected linear system (4.35), the $k-$th approximation $\boldsymbol{x}^{(k)}$. If the $k-$th residual corresponding to those approximation

is small enough we accept this approximation as the solution of the linear system; if not we set $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(k)}$ and repeat the process. This gives us the *Restarted full orthogonalization method*, usually abbreviated as *FOM(k)*. We present in algorithm 4.3.1 its implementation.

Given $\boldsymbol{x}^{(0)}$, compute $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$

Set $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$

for $k = 1, 2, ..., m$ **do**

  $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{v}_k$

  for $i = 1, 2, ..., k$ **do**

    $h_{ik} = (\boldsymbol{w}, \boldsymbol{v}_i)$

    $\boldsymbol{w} = \boldsymbol{w} - h_{ik}\boldsymbol{v}_k$

  end for

  $h_{k+1,k} = ||\boldsymbol{w}||_2$. If $h_{k+1,k} = 0$ stop.

  $\boldsymbol{v}_{k+1} = \boldsymbol{w}/h_{k+1,k}$

end for

Solve $\boldsymbol{H}_{k,k}\boldsymbol{y}^{(k)} = \rho\hat{\boldsymbol{e}}_1$ for $\boldsymbol{y}^{(k)}$.

Update $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_k\boldsymbol{y}^{(k)}$

Compute $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and $\rho = ||\boldsymbol{r}^{(0)}||_2$

if ρ is small enough **then**

  Take $\boldsymbol{x} \approx \boldsymbol{x}^{(k)}$ and stop.

else

  Set $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(k)}$ and restart.

end if

ALGORITHM 2: $FOM(k)$: Restarted Full Orthogonalization Method.

The situation when $h_{k+1,k} = ||\boldsymbol{w}||_2 = 0$, at least in finite precision arithmetic, which stops the Gram-Schmidt orthogonalization process is what is known as a *happy breakdown* because in such a case the Arnoldi vectors span completely the Krylov subspace, therefore the exact solution can be immediately computed.

4.3.2 Generalized Minimum Residual Method

It is now the turn to explore the second family of the Krylov subspace methods which attempts to select $\boldsymbol{x}^{(k)}$ in such a way that the norm of its corresponding residual is minimized. We take the $k-$th approximation as in (4.32), multiplying by $-\boldsymbol{A}$ and

adding $\boldsymbol{b}$ in both sides we obtain

$$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(0)} - \boldsymbol{A}\,\boldsymbol{V}_k \boldsymbol{y}^{(k)}. \tag{4.36}$$

Due to the way in which we have chosen the first Arnoldi vector, $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$, we have that $\boldsymbol{V}_{k+1}\hat{\boldsymbol{e}}_1 = \boldsymbol{v}_1$ being in this case $\hat{\boldsymbol{e}}_1 \in \mathbb{R}^{k+1}$. Additionally, exploiting now the second expression from (4.31) we obtain

$$\boldsymbol{r}^{(k)} = \boldsymbol{V}_{k+1}\left(\rho\hat{\boldsymbol{e}}_1 - \boldsymbol{H}_{k+1,k}\boldsymbol{y}^{(k)}\right), \tag{4.37}$$

Taking norms on both sides and using the orthogonality of $\boldsymbol{V}_{k+1}$ we have

$$||\boldsymbol{r}^{(k)}||_2 = ||\rho\hat{\boldsymbol{e}}_1 - \boldsymbol{H}_{k+1,k}\boldsymbol{y}^{(k)}||_2, \tag{4.38}$$

what remains is to find a vector $\boldsymbol{y}^{(k)} \in \mathbb{R}^{k+1}$ which minimizes the right hand side of (4.38); this is equivalent to solving the following *least squares* problem

$$\boldsymbol{H}_{k+1,k}\boldsymbol{y}^{(k)} = \rho\hat{\boldsymbol{e}}_1. \tag{4.39}$$

A common technique for solving the least squares problem (4.39) is performing a QR factorization, with $\boldsymbol{Q}$ being orthonormal and $\boldsymbol{R}$ upper triangular. As done with FOM, we must take into account the special structure of the upper Hessenberg matrix, being in this case rectangular. Since only the subdiagonal is non zero under the main diagonal, *Givens rotations* is clearly the most suitable method for carry out such QR factorization.

First, we observe in detail the structure of $\boldsymbol{H}_{k+1,k}$ obtained by the Arnoldi algorithm and denote it by $\overline{\boldsymbol{H}}_k^{(0)}$. Similarly it is done for the right hand side of (4.38) denoted by $\boldsymbol{z}_k^{(0)}$

$$\overline{\boldsymbol{H}}_k^{(0)} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2k} \\ 0 & h_{32} & h_{33} & \cdots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & h_{k,k-1} & h_{kk} \\ 0 & 0 & 0 & 0 & h_{k+1,k} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{z}_k^{(0)} = \begin{bmatrix} \rho \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

Our goal is to eliminate entries below the main diagonal in $\overline{\boldsymbol{H}}_k^{(0)}$. This can be achieved by plane rotations using Givens elementary matrices as defined in (3.45). Moreover, since we wish introduce zeros only in the contiguous subdiagonal to the main one the notation can be simplified. We write the $i-$th Givens rotator as $\boldsymbol{G}_i = \boldsymbol{G}(i, i+1, \theta) = \boldsymbol{I} - \boldsymbol{Y}$ with θ chosen in such a way that entries below the main diagonal are annihilated one

by one. For instance, consider the case for the first Givens rotator

$$\boldsymbol{G}_1 = \begin{bmatrix} c_1 & s_1 & 0 & \cdots & 0 \\ -s_1 & c_1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

with

$$s_1 = \frac{h_{21}}{\sqrt{h_{11}^2 + h_{21}^2}} \quad \text{and} \quad c_1 = \frac{h_{11}}{\sqrt{h_{11}^2 + h_{21}^2}}.$$

Premultiplying both, $\overline{\boldsymbol{H}}_k^{(0)}$ and $\boldsymbol{z}_k^{(0)}$ by this first Givens rotator we get

$$\overline{\boldsymbol{H}}_k^{(1)} = \begin{bmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & \cdots & h_{1k}^{(1)} \\ 0 & h_{22}^{(1)} & h_{23}^{(1)} & \cdots & h_{2k}^{(1)} \\ 0 & h_{32} & h_{33} & \cdots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & h_{k,k-1} & h_{kk} \\ 0 & 0 & 0 & 0 & h_{k+1,k} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{z}_k^{(1)} = \begin{bmatrix} c_1\rho \\ -s_1\rho \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

Applying this process recursively until $i = k$ we finally obtain $\overline{\boldsymbol{H}}_k^{(k)}$ which is upper triangular and the full vector $\boldsymbol{z}_k^{(k)}$.

Notice that all the Givens rotators are orthogonal matrices, thus any product of them is also orthogonal, then we define

$$\boldsymbol{Q}_k = \boldsymbol{G}_k \boldsymbol{G}_{k-1} \cdots \boldsymbol{G}_1.$$

Therefore, the successive rotations effectively achieve the QR factorization of the upper Hessenberg matrix since

$$\boldsymbol{R}_k = \overline{\boldsymbol{H}}_k^{(k)} = \boldsymbol{Q}_k \overline{\boldsymbol{H}}_k \quad \text{and} \quad \boldsymbol{z}_k^{(k)} = \boldsymbol{Q}_k \left(\rho \hat{\boldsymbol{e}}_1\right).$$

Using the orthonormality of $\boldsymbol{Q}_k$ we have the equivalence

$$\min ||\rho \hat{\boldsymbol{e}}_1 - \overline{\boldsymbol{H}}_k \boldsymbol{y}^{(k)}||_2 = \min ||\boldsymbol{z}_k^{(k)} - \boldsymbol{R}_k \boldsymbol{y}^{(k)}||_2,$$

therefore, the solution of this problem is obtained by simply solving the upper triangular linear system

$$\boldsymbol{R}_k \boldsymbol{y}^{(k)} = \boldsymbol{z}_k^{(k)}.$$

Once $\boldsymbol{y}^{(k)}$ has been determined, we obtain $\boldsymbol{x}^{(k)}$ by substituting in (4.32). All this together give us what is known as the *Generalized Minimal Residual* method abbreviated as *GMRes*.

When n is large, it suffers the same drawbacks in operations count and memory requirements as FOM, but the remedies are almost identical. We focus again in restarting which yield us the *Restarted Generalized Minimal Residual* method, with *GMRes(k)* used as its abbreviation. We sketch as follows its implementation in algorithm 4.3.2.

Given $\boldsymbol{x}^{(0)}$, compute $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$

Set $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$ and $\boldsymbol{z}_k^{(0)} = \rho\hat{\boldsymbol{e}}_1$

for $k = 1, 2, ..., m$ **do**

  $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{v}_k$

  for $i = 1, 2, ..., k$ **do**

    $h_{ik} = (\boldsymbol{w}, \boldsymbol{v}_i)$

    $\boldsymbol{w} = \boldsymbol{w} - h_{ik}\boldsymbol{v}_k$

  end for

  $h_{k+1,k} = ||\boldsymbol{w}||_2$. If $h_{k+1,k} = 0$ stop.

  $\boldsymbol{v}_{k+1} = \boldsymbol{w}/h_{k+1,k}$

  $\boldsymbol{H}_{k,k+1}^{(k)} = \boldsymbol{G}_k\boldsymbol{H}_{k,k+1}^{(k-1)}$

  $\boldsymbol{z}_k^{(k)} = \boldsymbol{G}_k\boldsymbol{z}_k^{(k-1)}$

end for

Solve $\boldsymbol{R}_k\boldsymbol{y}^{(k)} = \boldsymbol{z}_k^{(0)}$ for $\boldsymbol{y}^{(k)}$.

Update $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_k\boldsymbol{y}^{(k)}$

Compute $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and $\rho = ||\boldsymbol{r}^{(0)}||_2$

if ρ is small enought **then**

  Take $\boldsymbol{x} \approx \boldsymbol{x}^{(k)}$ and stop.

else

  Set $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(k)}$ and restart.

end if

ALGORITHM 3: $GMRes(k)$: Restarted Generalized Minimal Residual

As with FOM, the situation $h_{k+1,k} = 0$ is also a happy breakdown and furthermore, it is not necessary to apply the next Givens rotation because the entry which it attempts to annihilate is already zero.

4.4 Lanczos Methods

4.4.1 Symmetric Lanczos Method

The Arnoldi algorithm can be further simplified in the case when $\boldsymbol{A} = \boldsymbol{A}^T$ noting that from the first expression in (4.31) that the upper Hessenberg matrix is also symmetric $\boldsymbol{H}_{k,k} = \boldsymbol{H}_{k,k}^T$ and therefore tridiagonal, we denote it by $\boldsymbol{T}_k$

$$\boldsymbol{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{bmatrix}, \tag{4.40}$$

where $\alpha_i = h_{i,i}$ and $\beta_i = h_{i-1,i}$.

By doing the required modifications we get the *symmetric Lanczos algorithm* for which the computation of the orthogonal basis for the Krylov subspace can be performed using a *three term* recursive relation to generate the *Lanczos vectors*. That is, when computing the $(k+1)-$th Lanczos vector $\boldsymbol{v}_{k+1}$ only the two previous ones $\boldsymbol{v}_k$ and $\boldsymbol{v}_{k-1}$ are required. With these observations we sketch the symmetric Lanczos method in algorithm 4.4.1.

Given a vector $\boldsymbol{v}_1$ with $||\boldsymbol{v}_1||_2 = 1$

Set $\boldsymbol{v}_0 = \boldsymbol{0}$ and $\beta_1 = 0$

for $k = 1, 2, ..., m$ **do**

  $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{v}_k - \beta_k \boldsymbol{v}_{k-1}$

  $\alpha_k = (\boldsymbol{w}, \boldsymbol{v}_k)$

  $\boldsymbol{w} = \boldsymbol{w} - \alpha_k \boldsymbol{v}_k$

  $\beta_{k+1} = ||\boldsymbol{w}||_2$. If $\beta_{k+1} = 0$ stop.

  $\boldsymbol{v}_{k+1} = \boldsymbol{w}/\beta_{k+1}$

end for

ALGORITHM 4: Symmetric Lanczos Algorithm.

This algorithm guarantees, at least in exact arithmetic, that the Lanczos vectors are each other orthogonal. Unfortunately, exact orthogonality is only observed in early stages of the process in finite precision arithmetic, since as k increases we lose global orthogonality. Several strategies have been devised in order to overcome this problem such as complete or selective re-orthogonalization.

As with the Arnoldi algorithm, the present Lanczos algorithm can also be used for solving linear systems. By using the same arguments as those for FOM but this time with $\boldsymbol{T}_k$ instead of $\boldsymbol{H}_{kk}$ and taking advantage of the symmetry of the tridiagonal matrix $\boldsymbol{T}_k$. Then we must solve the linear system

$$\boldsymbol{T}_k \boldsymbol{y}^{(k)} = \rho \hat{\boldsymbol{e}}_1, \tag{4.41}$$

usually with Gaussian elimination. Once $\boldsymbol{y}^{(k)}$ has been computed, we obtain $\boldsymbol{x}^{(k)}$ substituting in (4.32). This yields what is known as the *Lanczos method for linear systems.*

4.4.2 Conjugate Gradient

In the previous section we have assumed symmetry of the coefficient matrix. Further simplifications and consequently improvements in efficiency can be achieved if additionally the coefficient matrix is *positive definite*, that is

$$(\boldsymbol{A}\boldsymbol{u}, \boldsymbol{u}) > 0 \quad \forall \boldsymbol{u} \in \mathbb{R}^n, \quad \boldsymbol{u} \neq \boldsymbol{0}. \tag{4.42}$$

From the first expression in (4.31) with $\boldsymbol{H}_{kk}$ substituted by $\boldsymbol{T}_k$ we have

$$\begin{aligned} (\boldsymbol{T}_k \boldsymbol{z}, \boldsymbol{z}) &= \left(\boldsymbol{V}_k^T \boldsymbol{A}\, \boldsymbol{V}_k \boldsymbol{z}, \boldsymbol{z}\right), \\ &= (\boldsymbol{A}\, \boldsymbol{V}_k \boldsymbol{z},\, \boldsymbol{V}_k \boldsymbol{z}), \\ &= (\boldsymbol{A}\boldsymbol{u}, \boldsymbol{u}), \\ &> 0, \end{aligned} \tag{4.43}$$

for all non null vector $\boldsymbol{z} \in \mathbb{R}^k$. That is, if $\boldsymbol{A}$ is symmetric positive definite, $\boldsymbol{T}_k$ is too. Hence it is now clear that Krylov subspace projections preserve the desirable property of positive definiteness.

Being that $\boldsymbol{T}_k = \boldsymbol{L}_k \boldsymbol{U}_k$ positive definite it is ensured that its LU factorizations exists and is stable even if no pivoting is strategy is employed. Hence we write $\boldsymbol{T}_k$ as the product

$$\boldsymbol{T}_k = \begin{bmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & \lambda_{k-1} & 1 & \\ & & & \lambda_k & 1 \end{bmatrix} \begin{bmatrix} \eta_1 & \beta_2 & & & \\ & \eta_1 & \beta_3 & & \\ & & \ddots & \ddots & \\ & & & \eta_{k-1} & \beta_k \\ & & & & \eta_k \end{bmatrix} \tag{4.44}$$

where $\lambda_1 = 0$, $\lambda_i = \frac{\beta_i}{\eta_{i-1}}$, for $i = 2, 3, ..., k$ and $\eta_i = \alpha_i - \lambda_i \beta_i$, for $i = 1, 2, ..., k$.

The $k-$th approximation can be written as

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_k \boldsymbol{U}_k^{-1} \boldsymbol{L}_k^{-1} \left(\rho \hat{\boldsymbol{e}}_1\right), \tag{4.45}$$

now, we let

$$\boldsymbol{P}_k = \boldsymbol{V}_k \boldsymbol{U}_k^{-1} \quad \text{and} \quad \boldsymbol{z}_k = \boldsymbol{L}_k^{-1} \left(\rho \hat{\boldsymbol{e}}_1\right), \tag{4.46}$$

postmultiplying the first expression in (4.46) by $\boldsymbol{U}_k$ we get

$$\boldsymbol{P}_k \boldsymbol{U}_k = \boldsymbol{V}_k \quad \Rightarrow \quad \boldsymbol{p}_k = \frac{1}{\eta_k} \left[\boldsymbol{v}_k - \beta_k \boldsymbol{p}_{k-1}\right], \tag{4.47}$$

where we have used the fact that $\boldsymbol{U}_k$ is upper bidiagonal. If now we premultiply the second expression in (4.46) by $\boldsymbol{L}_k$

$$\boldsymbol{L}_k \boldsymbol{z}_k = \rho \hat{\boldsymbol{e}}_1 \tag{4.48}$$

and using the fact that $\boldsymbol{L}_k$ is lower bidiagonal we get

$$z_i = -\lambda_i z_{i-1}, \quad i = 2, 3, ..., k \tag{4.49}$$

with $z_1 = \rho$. Moreover we use a partition of vector $\boldsymbol{z}_k$ as

$$\boldsymbol{z}_k = \begin{bmatrix} \boldsymbol{z}_{k-1} \\ z_k \end{bmatrix}. \tag{4.50}$$

Substituting a suitable partition of (4.47) by columns and (4.50) in (4.45) we obtain

$$\begin{aligned} \boldsymbol{x}^{(k)} &= \boldsymbol{x}^{(0)} + \boldsymbol{P}_k \boldsymbol{z}_k \\ &= \boldsymbol{x}^{(0)} + \left[\boldsymbol{P}_{k-1}, \boldsymbol{p}_k\right] \begin{bmatrix} \boldsymbol{z}_{k-1} \\ z_k \end{bmatrix} \\ &= \boldsymbol{x}^{(k-1)} + z_k \boldsymbol{p}_k. \end{aligned} \tag{4.51}$$

where going from the second to the third line recursion on k has been employed.

Now we shall make some important and useful observations. The first one is concerned with the orthogonality of the residual vectors. This property is not exclusive of the Lanczos algorithm, in fact the $k-$th residual vector generated by the Arnoldi algorithm is orthogonal to all the previous ones, at least in exact arithmetic. This can be verified

as follows

$$\begin{aligned} \boldsymbol{r}^{(k)} &= \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}, \\ &= \boldsymbol{b} - \boldsymbol{A}\left(\boldsymbol{x}^{(0)} + \boldsymbol{V}_k \boldsymbol{y}^{(k)}\right), \\ &= \boldsymbol{r}^{(0)} - \boldsymbol{A}\boldsymbol{V}_k \boldsymbol{y}^{(k)}, \\ &= \boldsymbol{r}^{(0)} - \boldsymbol{V}_{k+1}\boldsymbol{H}_{k+1,k}\boldsymbol{y}^{(k)}, \\ &= \boldsymbol{r}^{(0)} - \boldsymbol{V}_k \boldsymbol{H}_{k,k}\boldsymbol{y}^{(k)} - h_{k+1,k} y_k^{(k)} \boldsymbol{v}_{k+1}, \\ &= \rho \boldsymbol{v}_1 - \boldsymbol{V}_k \boldsymbol{H}_{k,k}\boldsymbol{y}^{(k)} - h_{k+1,k} y_k^{(k)} \boldsymbol{v}_{k+1}, \\ &= -h_{k+1,k} y_k^{(k)} \boldsymbol{v}_{k+1}. \end{aligned} \tag{4.52}$$

For the symmetric Lanczos algorithm this can be particularized by

$$\boldsymbol{r}^{(k)} = \beta_{k+1} y_k^{(k)} \boldsymbol{v}_{k+1}, \tag{4.53}$$

that is, since the Arnoldi or Lanczos $\{\boldsymbol{v}_j\}$ are mutually orthogonal, and because the residual vectors are scalar multiples of the former ones, they are also mutually orthogonal.

The second observation is particularly oriented for the Lanczos algorithm and it is about the *A-orthogonality* of the *auxiliary vectors* $\boldsymbol{p}_k$ which form, column by column, the *substitution matrix* $\boldsymbol{P}_k = \boldsymbol{V}_k \boldsymbol{U}_k^{-1}$. This can be verified as follows

$$\begin{aligned} \boldsymbol{P}_k^T \boldsymbol{A} \boldsymbol{P}_k &= \boldsymbol{U}_k^{-T} \boldsymbol{V}_k^T \boldsymbol{A} \boldsymbol{V}_k \boldsymbol{U}_k^{-1}, \\ &= \boldsymbol{U}_k^{-T} \boldsymbol{T}_k \boldsymbol{U}_k^{-1}, \\ &= \boldsymbol{U}_k^{-T} \boldsymbol{L}_k, \\ &= \boldsymbol{D}_k. \end{aligned} \tag{4.54}$$

We have used the fact that $\boldsymbol{T}_k = \boldsymbol{L}_k \boldsymbol{U}_k$ while going from the second to the third lines. The $\boldsymbol{U}_k^{-T} \boldsymbol{L}_k$ is the product of two lower triangular matrices and hence it is also lower triangular. Additionally it is equal to a symmetric matrix $\boldsymbol{P}_k^T \boldsymbol{A} \boldsymbol{P}_k$ which is also symmetric. Finally, by being a symmetric triangular matrix, it is diagonal. Therefore we have $\left(\boldsymbol{p}_i, \boldsymbol{A}\boldsymbol{p}_j\right) \neq 0$ iff $i = j$. A set of vectors satisfying this property are called *A-orthogonal* of *conjugate*.

From (4.47) and (4.51) it can be seen that the $(k+1)-$th approximation, residual and auxiliary vectors can be computed using, instead of three, two term recurrence relations

that can be sketched as follows

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}, \\ \boldsymbol{r}^{(k+1)} &= \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{A}\boldsymbol{p}^{(k)}, \\ \boldsymbol{p}^{(k+1)} &= \boldsymbol{r}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)}. \end{aligned} \tag{4.55}$$

Where we have used, for the second line, the basic definition of residual. It is important to notice that the coefficients α_k and β_k and the vectors $\boldsymbol{p}^{(k)}$ are not exactly the same as those used previously.

In order to obtain α_k we take the inner product of the second line of (4.55) with $\boldsymbol{r}^{(k)}$ and use the orthogonality of the residual vectors

$$\left(\boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k+1)}\right) - \left(\boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)}\right) + \alpha_k \left(\boldsymbol{r}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right) = 0, \tag{4.56}$$

hence

$$\alpha_k = \frac{\left(\boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)}\right)}{\left(\boldsymbol{r}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}. \tag{4.57}$$

Furthermore, an alternative and more useful expression can be obtained by substituting $\boldsymbol{r}^{(k)}$ in the denominator of (4.56); solving for the residual vector from the third line in (4.55) but for the $k-$th instead of the $(k+1)-$th to get

$$\begin{aligned} \left(\boldsymbol{r}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right) &= \left(\boldsymbol{p}^{(k)} - \beta_{k-1}\boldsymbol{p}^{(k-1)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right), \\ &= \left(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right), \end{aligned} \tag{4.58}$$

where we have used the fact that the vectors $\boldsymbol{p}_j$ are mutually conjugate. Finally we obtain

$$\alpha_k = \frac{\left(\boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)}\right)}{\left(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}. \tag{4.59}$$

It remains to compute the second coefficient β_k. For that we take the inner product of the third line of (4.55) with the vector $\boldsymbol{A}\boldsymbol{p}^{(k)}$

$$\left(\boldsymbol{p}^{(k+1)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right) = \left(\boldsymbol{r}^{(k+1)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right) + \beta_k \left(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right), \tag{4.60}$$

using again the fact that the auxiliary vectors $\boldsymbol{p}_j$ are conjugate, the left hand side vanisches and we can solve for β_k getting

$$\beta_k = -\frac{\left(\boldsymbol{r}^{(k+1)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}{\left(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}. \tag{4.61}$$

Solving for $\boldsymbol{A}\boldsymbol{p}^{(k)}$ from the second line in (4.55) we have

$$\boldsymbol{A}\boldsymbol{p}^{(k)} = -\frac{1}{\alpha_k}\left(\boldsymbol{r}^{(k+1)} - \boldsymbol{r}^{(k)}\right), \tag{4.62}$$

substituting it in (4.61)

$$\begin{aligned} \beta_k &= \frac{1}{\alpha_k}\frac{\left(\boldsymbol{r}^{(k+1)}-\boldsymbol{r}^{(k)},\boldsymbol{r}^{(k+1)}\right)}{\left(\boldsymbol{p}^{(k)},\boldsymbol{A}\boldsymbol{p}^{(k)}\right)}, \\ &= \frac{\left(\boldsymbol{r}^{(k+1)},\boldsymbol{r}^{(k+1)}\right)}{\left(\boldsymbol{r}^{(k)},\boldsymbol{r}^{(k)}\right)}, \end{aligned} \tag{4.63}$$

where for passing from the first to the second line we have used the orthogonality of the residuals and the definition of the coefficient α_k given by (4.59).

Putting all this together we obtain the *Conjugate Gradient* method for which we sketch it algorithmically in 4.4.2.

Given $\boldsymbol{x}^{(0)}$ compute $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$

Set $\boldsymbol{p}^{(0)} = \boldsymbol{r}^{(0)}$

for $k = 0, 1, 2, \ldots$ **do**

$\alpha_k = \frac{\left(\boldsymbol{r}^{(k)},\boldsymbol{r}^{(k)}\right)}{\left(\boldsymbol{p}^{(k)},\boldsymbol{A}\boldsymbol{p}^{(k)}\right)}$

$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$

$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{A}\boldsymbol{p}^{(k)}$

$\beta_k = \frac{\left(\boldsymbol{r}^{(k+1)},\boldsymbol{r}^{(k+1)}\right)}{\left(\boldsymbol{r}^{(k)},\boldsymbol{r}^{(k)}\right)}$

$\boldsymbol{p}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)}$

end for

ALGORITHM 5: *CG*: Conjugate Gradient.

The *Conjugate Gradient method* is probably the most known and used Krylov subspace method. It was first published independently and almost simultaneously by Hestenes and Stiefel [74] and Lanczos [82] in 1952. The paper of Hestenes and Stieffel is completely oriented to solving linear systems of equations. They regard this method as a direct one because they proved that it converges in no more that n iterations. Nevertheless the paper of Lanczos is mainly concerned with the approximation of the extreme eigenvalues, and possibly the corresponding eigenvalues, of a matrix by means of projection and the solution of linear systems is regarded as an additional application of his method.

It is quite amazing that the power and effectiveness of this method was not noticed for more than twenty years until it has been rediscovered and demonstrated that it

can provide good approximations after performing much fewer than n iterations on well conditioned matrices.

4.4.3 Unsymmetric Lanczos Method

The previous section has been devoted to several Krylov subspace methods relying on an orthogonalization process in order to compute an approximate solution. On the other hand, this section is devoted to a class of Krylov subspace methods that are instead based on a biorthogonalization algorithm but with the same purpose of the former methods. Such kind of methods are based on the biorthogonalization Lanczos algorithm which in turns is an extension to unsymmetric matrices of the previously studied symmetric Lanczos algorithm.

First we notice that when $\boldsymbol{A} \neq \boldsymbol{A}^T$ the construction of an orthogonal basis for the Krylov subspace is not possible using three term recurrence relations. We can, however, obtain a suitable non-orthogonal basis with a three term recurrence relation, by requiring that this basis be orthogonal with respect to some other basis.

From (4.31) it is clear that we can construct a basis $\boldsymbol{V}_k$ for $\boldsymbol{\mathcal{K}}_k(\boldsymbol{A}; \boldsymbol{v})$. furthermore, suppose we can also construct another basis $\boldsymbol{W}_k$ for which $\boldsymbol{W}_k^T \boldsymbol{V}_k = \boldsymbol{D}_k$, with $\boldsymbol{D}_k$ a diagonal matrix and for which $\boldsymbol{W}_k^T \boldsymbol{v}_{(k+1)} = \boldsymbol{0}$. Then

$$\boldsymbol{W}_k^T \boldsymbol{A}\, \boldsymbol{V}_k = \boldsymbol{D}_k \boldsymbol{H}_{k,k}. \tag{4.64}$$

In order to preserve the three term recurrence relation we must be able to find a $\boldsymbol{W}_k^T$ in such a way that $\boldsymbol{H}_{k,k}$ is tridiagonal. Then

$$\boldsymbol{V}_k^T \boldsymbol{A}^T \boldsymbol{W}_k = \boldsymbol{H}_{k,k}^T \boldsymbol{D}_k, \tag{4.65}$$

must also be tridiagonal. Obviously this suggest to generate $\boldsymbol{W}_k$ with $\boldsymbol{A}^T$.

Thus, we choose two arbitrary unit vectors $\boldsymbol{v}_1$ and $\boldsymbol{w}_1$ with $(\boldsymbol{v}_1, \boldsymbol{w}_1) \neq 0$. then we generate $\boldsymbol{v}_2$ as

$$h_{2,1}\boldsymbol{v}_2 = \boldsymbol{A}\boldsymbol{v}_1 - h_{1,1}\boldsymbol{v}_1 \quad \text{with} \quad h_{1,1} = \frac{(\boldsymbol{w}_1, \boldsymbol{A}\boldsymbol{v}_1)}{(\boldsymbol{w}_1, \boldsymbol{v}_1)}$$

where $h_{1,1}$ has been chosen in such a manner to make $\boldsymbol{v}_2$ orthogonal to $\boldsymbol{w}_1$. By (3.26) we known that $(\boldsymbol{w}_1, \boldsymbol{A}\boldsymbol{v}_1) = (\boldsymbol{A}^T \boldsymbol{w}_1 \boldsymbol{v}_1)$, implying that $\boldsymbol{w}_2$ generated with

$$h_{2,1}\boldsymbol{w}_2 = \boldsymbol{A}^T \boldsymbol{w}_1 - h_{1,1}\boldsymbol{w}_1$$

is orthogonal to $\boldsymbol{v}_1$. This process can be continued for a fixed m in order to compute such biorthogonal basis leading us to the *unsymmetric Lanczos algorithm* sketched in algorithm 4.4.3.

Given two vectors $\boldsymbol{v}_1$ and $\boldsymbol{w}_1$ with $(\boldsymbol{v}_1, \boldsymbol{w}_1) = 1$

Set $\boldsymbol{v}_0 = \boldsymbol{w}_0 = \boldsymbol{0}$ and $\beta_1 = \delta_1 = 0$

for $k = 1, 2, ..., m$ **do**

$\alpha_k = (\boldsymbol{A}\boldsymbol{v}_k, \boldsymbol{w}_k)$

$\tilde{\boldsymbol{v}}_{k+1} = \boldsymbol{A}\boldsymbol{v}_k - \alpha_k \boldsymbol{v}_k - \beta_k \boldsymbol{v}_{k-1}$

$\tilde{\boldsymbol{w}}_{k+1} = \boldsymbol{A}^T \boldsymbol{w}_k - \alpha_k \boldsymbol{w}_k - \delta_k \boldsymbol{w}_{k-1}$

$\delta_{k+1} = \sqrt{|(\tilde{\boldsymbol{v}}_{k+1}, \tilde{\boldsymbol{w}}_{k+1})|}$. If $\delta_{k+1} = 0$ stop.

$\beta_{k+1} = (\tilde{\boldsymbol{v}}_{k+1}, \tilde{\boldsymbol{w}}_{k+1})/\delta_{k+1}$

$\boldsymbol{v}_{k+1} = \tilde{\boldsymbol{v}}_{k+1}/\delta_{k+1}$

$\boldsymbol{w}_{k+1} = \tilde{\boldsymbol{w}}_{k+1}/\beta_{k+1}$

end for

ALGORITHM 6: Unsymmetric Lanczos Algorithm.

Notice that the election of the scalars β_{k+1} and δ_{k+1} is not unique and in the present algorithm we choose them in such a manner that $\tilde{\boldsymbol{v}}_{k+1}$ and $\tilde{\boldsymbol{w}}_{k+1}$ are divided by two numbers with the same modulus avoiding numerical instabilities.

Unfortunately, in this case $\delta_{k+1} = 0$ is not a happy breakdown as the analogous situation with the Arnoldi or Lanczos algorithms because in the absence of a complete orthogonalization process, the projection is not optimal making the analysis and implementation of overcoming strategies for this methods harder than for the previously studied algorithms.

Now, suppose k steps of the unsymmetric Lanczos algorithm have been run, then $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k$ and $\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_k$ are two set of bi-orthogonal vectors, if additionally we define the $n \times k$ matrices satisfying $\boldsymbol{W}_k^T \boldsymbol{V}_k = \boldsymbol{D}_k$ as

$$\boldsymbol{V}_k = [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k] \quad \text{y} \quad \boldsymbol{W}_k = [\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_k], \tag{4.66}$$

then we have

$$\boldsymbol{W}_k^T \boldsymbol{A}\, \boldsymbol{V}_k = \boldsymbol{T}_k, \tag{4.67}$$

where $\boldsymbol{T}_k$ is a tridiagonal matrix whose entries have been computed by the unsymmetric Lanczos algorith and has the form

$$\boldsymbol{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \delta_k & \alpha_k \end{bmatrix}. \tag{4.68}$$

Hence it is now clear that we have generated two basis $\boldsymbol{V}_k$ and $\boldsymbol{W}_k$ for the Krylov subspaces

$$\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}; \boldsymbol{v}_1\right) = span\left\{\boldsymbol{v}_1, \boldsymbol{A}\boldsymbol{v}_1, \boldsymbol{A}^2\boldsymbol{v}_1, ..., \boldsymbol{A}^{k-1}\boldsymbol{v}_1\right\}, \tag{4.69}$$

$$\boldsymbol{\mathcal{L}}_k\left(\boldsymbol{A}^T; \boldsymbol{w}_1\right) = span\left\{\boldsymbol{w}_1, \boldsymbol{A}^T\boldsymbol{w}_1, \left(\boldsymbol{A}^T\right)^2\boldsymbol{w}_1, ..., \left(\boldsymbol{A}^T\right)^{k-1}\boldsymbol{w}_1\right\}, \tag{4.70}$$

respectively.

We can compute approximations $\boldsymbol{x}^{(k)}$ with this bi-orthogonalization procedure by choosing $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$ and $\boldsymbol{w}_1 = \boldsymbol{v}_1$ generating two basis corresponding to the Krylov subspaces $\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A}; \boldsymbol{r}^{(0)}\right)$ and $\boldsymbol{\mathcal{L}}_k\left(\boldsymbol{A}^T; \boldsymbol{r}^{(0)}\right)$.

Even more, in view that we have generated two different Krylov subspace bases it is obvious that the most natural option for such bi-orthogonalization process is a *Petrov-Galerkin* scheme. The condition $\boldsymbol{r}^{(k)} \perp \boldsymbol{\mathcal{L}}_k\left(\boldsymbol{A}^T; \boldsymbol{r}^{(0)}\right)$ implies that

$$\boldsymbol{W}_k^T\boldsymbol{r}^{(k)} = \boldsymbol{0}, \tag{4.71}$$

since $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and taking $\boldsymbol{x}^{(k)}$ as (4.32) we obtain

$$\boldsymbol{W}_k^T\boldsymbol{A}\,\boldsymbol{V}_k\boldsymbol{y}^{(k)} = \boldsymbol{W}_k^T\boldsymbol{r}^{(0)}, \tag{4.72}$$

substituting (4.66) and due to the choose of $\boldsymbol{w}_1$ we have that

$$\boldsymbol{T}_k\boldsymbol{y}^{(k)} = \rho\hat{\boldsymbol{e}}_1, \tag{4.73}$$

As with the Arnoldi method, we must solve this projected linear system for $\boldsymbol{y}^{(k)}$, once it has been determined we obtain $\boldsymbol{x}^{(k)}$ by substituting in (4.32). Also Gaussian elimination is a common choose.

Putting all this together lead us the *Bi-Lanczos algorithm for linear systems* from which a plenty variety of algorithms can be derived.

4.4.4 Bi-Conjugate Gradient

The *Bi-Conjugate Gradient*, BiCG for short, algorithm for solving unsymmetric linear systems can be derived from the unsymmetric Lanczos algorithm in exactly the same way as the Conjugate Gradient method was derived from the symmetric Lanczos algorithm. It was originally proposed by Lanczos in the celebrated paper [82] in 1952 and in a different manner by Fletcher in 1975 [53] in 1975. In what follows we present the latter implementation.

As we have seen previously, the biorthogonalization Lanczos algorithm produces two bases for the Krylov subspaces $\boldsymbol{\mathcal{K}}_k\left(\boldsymbol{A};\boldsymbol{v}_1\right)$ and $\boldsymbol{\mathcal{L}}_k\left(\boldsymbol{A}^T;\boldsymbol{w}_1\right)$. We take, as usual, $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$ and the only requirement for $\boldsymbol{w}_1$ is that $(\boldsymbol{v}_1, \boldsymbol{w}_1) \neq 0$. Suppose that we are also interested in solving the *dual system* $\boldsymbol{A}^T\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{b}'$, then an obvious option is provide $\boldsymbol{w}_1 = \boldsymbol{r}'/\rho'$ with $\rho' = ||\boldsymbol{r}'^{(0)}||_2$ being $\boldsymbol{r}'^{(0)} = \boldsymbol{b}' - \boldsymbol{A}^T\boldsymbol{x}'^{(0)}$ the *dual residual*.

As we have done in the derivation of the conjugate gradient, we write the LU factorization of the tridiagonal matrix (4.68) obtained from the biorthogonalization Lanczos algorithm as $\boldsymbol{T}_k = \boldsymbol{L}_k\boldsymbol{U}_k$ and we set

$$\boldsymbol{P}_k = \boldsymbol{V}_k\boldsymbol{U}_k^{-1}. \tag{4.74}$$

From (4.32) and (4.67) the solution can be expressed as

$$\begin{aligned} \boldsymbol{x}^{(k)} &= \boldsymbol{x}^{(0)} + \boldsymbol{V}_k\boldsymbol{T}_k^{-1}\left(\rho\hat{\boldsymbol{e}}_1\right) \\ &= \boldsymbol{x}^{(0)} + \boldsymbol{V}_k\boldsymbol{U}_k^{-1}\boldsymbol{L}_k^{-1}\left(\rho\hat{\boldsymbol{e}}_1\right) \\ &= \boldsymbol{x}^{(0)} + \boldsymbol{P}_k\boldsymbol{L}_k^{-1}\left(\rho\hat{\boldsymbol{e}}_1\right) \end{aligned} \tag{4.75}$$

We do not have to wait until the k has been completed in order to obtain intermediate approximations for $j < k$ by defining, as done with the conjugate gradient method, a vector $\boldsymbol{z}_k \in \mathbb{R}^k$ such that $\boldsymbol{L}_k\boldsymbol{z}_k = \rho$.

Remember that the vectors $\boldsymbol{v}_{k+1}$ and $\boldsymbol{w}_{k+1}$ are biorthogonal and because the vectors $\boldsymbol{r}_k$ and $\boldsymbol{r}'_k$ are in the same direction as the formers, respectively, they also form a biorthogonal sequence. Moreover, we define the matrix

$$\boldsymbol{P}'_k = \boldsymbol{W}_k\boldsymbol{L}_k^{-1}, \tag{4.76}$$

formed, column by column, by the *dual auxiliary* vectors $\boldsymbol{p}'_k$; and being conjugate with $\boldsymbol{P}_k$ because

$$\begin{aligned}(\boldsymbol{P}'_k)^T\,\boldsymbol{A}\boldsymbol{P}_k &= \boldsymbol{L}_k^{-1}\,\boldsymbol{W}_k^T\,\boldsymbol{A}\,\boldsymbol{V}_k\,\boldsymbol{U}_k^{-1}\\ &= \boldsymbol{L}_k^{-1}\,\boldsymbol{T}_k\,\boldsymbol{U}_k^{-1}\\ &= \boldsymbol{L}_k^{-1}\boldsymbol{L}_k\,\boldsymbol{U}_k\,\boldsymbol{U}_k^{-1}\\ &= \boldsymbol{I}_k\end{aligned} \tag{4.77}$$

Putting all this together we obtain the *Bi-Conjugate Gradient* method sketched in algorithm 4.4.4.

Given $\boldsymbol{x}^{(0)}$ compute $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$

and $\boldsymbol{x}'^{(0)}$ compute $\boldsymbol{r}'^{(0)} = \boldsymbol{b} - \boldsymbol{A}^T\boldsymbol{x}'^{(0)}$

Set $\boldsymbol{p}^{(0)} = \boldsymbol{r}^{(0)}$ and $\boldsymbol{p}'^{(0)} = \boldsymbol{r}'^{(0)}$

for $k = 0, 1, 2, \ldots$ **do**

$\alpha_k = \frac{\left(\boldsymbol{r}'^{(k)}, \boldsymbol{r}^{(k)}\right)}{\left(\boldsymbol{p}'^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}$

$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$

$\boldsymbol{x}'^{(k+1)} = \boldsymbol{x}'^{(k)} + \alpha_k \boldsymbol{p}'^{(k)}$

$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{A}\boldsymbol{p}^{(k)}$

$\boldsymbol{r}'^{(k+1)} = \boldsymbol{r}'^{(k)} - \alpha_k \boldsymbol{A}^T\boldsymbol{p}'^{(k)}$

$\beta_k = \frac{\left(\boldsymbol{r}'^{(k+1)}, \boldsymbol{r}^{(k+1)}\right)}{\left(\boldsymbol{r}'^{(k)}, \boldsymbol{r}^{(k)}\right)}$

$\boldsymbol{p}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)}$

$\boldsymbol{p}'^{(k+1)} = \boldsymbol{r}'^{(k+1)} + \beta_k \boldsymbol{p}'^{(k)}$

end for

ALGORITHM 7: *BiCG*: Bi-Conjugate Gradient

If the solution if the dual system is not required, it is not required to provide the corresponding initial guess $\boldsymbol{x}'^{(0)}$. Furthermore, the computation of the dual residual $\boldsymbol{r}'^{(0)}$ is not required and usually it is set as $\boldsymbol{r}'^{(0)} = \boldsymbol{r}^{(0)}$. Also the update of $\boldsymbol{x}'^{(k+1)}$ and its storage can be avoided.

4.4.5 Quasi-Minimal Residual

With a slight modification, the biorthogonalization Lanczos algorithm lead us to with the alternative relation

$$\boldsymbol{A}\,\boldsymbol{V}_k = \boldsymbol{V}_{k+1}\,\overline{\boldsymbol{T}}_k, \tag{4.78}$$

where $\overline{\boldsymbol{T}}_k \in \mathbb{R}^{(k+1)\times k}$ is defined similarly that (4.68) as

$$\overline{\boldsymbol{T}}_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \delta_k & \alpha_k \\ & & & & \delta_{k+1} \end{bmatrix}. \tag{4.79}$$

We can notice that (4.78) can be used in the same way we have done for GMRes. In order to do so we take, as usual, $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$

$$\begin{aligned} \boldsymbol{r}^{(k)} &= \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}, \\ &= \boldsymbol{b} - \boldsymbol{A}\left(\boldsymbol{x}^{(0)} + \boldsymbol{V}_k \boldsymbol{y}^k\right), \\ &= \boldsymbol{r}^{(0)} - \boldsymbol{A}\,\boldsymbol{V}_k \boldsymbol{y}^k, \\ &= \rho \boldsymbol{v}_1 - \boldsymbol{V}_{k+1}\overline{\boldsymbol{T}}_k \boldsymbol{y}^k, \\ &= \boldsymbol{V}_{k+1}\left(\rho \hat{\boldsymbol{e}}_1 - \overline{\boldsymbol{T}}_k \boldsymbol{y}^k\right). \end{aligned} \tag{4.80}$$

While going from the first to the second line we have substituted (4.32) for $\boldsymbol{x}^{(k)}$, (4.78) for going from the third to the forth lines and finally we get the last expression because the definition of the first Lanczos vector $\boldsymbol{v}_1$.

Taking norms of both sides of (4.80) we have

$$\left|\left|\boldsymbol{r}^{(k)}\right|\right|_2 = \left|\left|\boldsymbol{V}_{k+1}\left(\rho \hat{\boldsymbol{e}}_1 - \overline{\boldsymbol{T}}_k \boldsymbol{y}^k\right)\right|\right|_2 \tag{4.81}$$

But notice that unfortunately we cannot simply take off $\boldsymbol{V}_{k+1}$ as done with GMRes because it is still not orthonormal. This has the important consequence that the least squares problem can not be posed properly and then we can not aim any optimality property. However, it is still a reasonable idea just take off $\boldsymbol{V}_{k+1}$ from the Euclidean norm and then solve the least squares problem

$$\overline{\boldsymbol{T}}_k \boldsymbol{y}^{(k)} = \rho \hat{\boldsymbol{e}}_1, \tag{4.82}$$

in exactly the same manner as with GMRes using Givens rotations, but this time taking advantage of the fact that the matrix involved is now tridiagonal.

Once $\boldsymbol{y}^{(k)}$ is computed the $k-$th approximation can be computed, as usual, with (4.32). In this case we omit the algorithm implementation since it is almost immediate to obtain it from the the biorthogonalization Lanczos and GMRes algorithms.

4.4.6 Transpose Free Variants

We have seen that the unsymmetric Lanczos algorithms and hence the algorithms derived from it require at each iteration the computation of two matrix by vector products, one of them with the matrix transpose. However, if the dual solution is not required, all the computations involving the dual vectors do not contribute directly to the solution updating. We have tried to do this complete clear in the Bi-Conjugate Gradient algorithm.

Additionally, there are applications in which the matrix is not assembled and preform products with the matrix transpose could be more expensive or difficult than with the matrix itself, sometimes even impossible.

A large variety of techniques have been developed in order to avoid such iterative methods using the transpose matrix leading a quite inhomogeneous family of *transpose free variants*. The vast majority of these techniques rely in the fact that the $k-$th residual can also be expressed as the initial residual multiplied by a polynomial in $\boldsymbol{A}$ of k degree

$$\boldsymbol{r}^{(k)} = p_k\left(\boldsymbol{A}\right)\boldsymbol{r}^{(0)}, \tag{4.83}$$

with $p\left(0\right) = 1$. Similarly, polynomial expression for other recurrence vectors can be defined.

The first developed method using this idea was the *Conjugate Gradient Squared* of Sonneveld [116] by exploiteng the following observation

$$\left(p_k\left(\boldsymbol{A}\right)\boldsymbol{r}^{(0)}, p_k\left(\boldsymbol{A}^T\right)\boldsymbol{r}'^{(0)}\right) = \left(p_k^2\left(\boldsymbol{A}\right)\boldsymbol{r}^{(0)}, \boldsymbol{r}'^{(0)}\right). \tag{4.84}$$

Then the dual residual vector can be computed as

$$\boldsymbol{r}'^{(k)} = p_k^2\left(\boldsymbol{A}\right)\boldsymbol{r}^{(0)}. \tag{4.85}$$

Other dual quantities can be expressed in this manner and then the evaluation of matrix transpose by vector products could be avoided.

Notice that, if we have irregular convergence with BiCG, for instance, then this phenomena will be magnified when using CGS due to the squaring of the implicit polynomial making convergence of this method quite irregular. A possible remedy to this situation is, instead of square the polynomials in $p_k\left(\boldsymbol{A}\right)$ we take their product with other polynomial satisfying the same properties. Then the dual residual can be expressed as

$$\boldsymbol{r}^{(k)} = p_k\left(\boldsymbol{A}\right)q_k\left(\boldsymbol{A}\right)\boldsymbol{r}^{(0)}. \tag{4.86}$$

How to choose the $q_k(\boldsymbol{A})$ polynomial has lead to different Krylov subspace methods.

One of the most popular and well known of them is the *Bi-Conjugate Gradient Squared* method of van der Vorst [135] which chooses this second polynomial with the goal of stabilizing or smoothing the convergence behavior of the BiCG algorithm. The success and popularity of the method is mainly due it posses very desirable properties yielding smooth convergence curves.

The last alternative we will review is the *Transpose Free Quasi-Minimal Residual* method of Freund [57] which is derived from the Conjugate Gradient Squared method using the same principle of express recurrence vectors as polynomials or products of them but dividing the computation in two stages. The first stage is devoted to perform a Bi-conjugate Gradient search and the second a minimization of the residual. Despite its name this method is not really related to the QMR method.

Chapter 5

Preconditioning

Although the iterative methods seen in the previous chapter are well founded theoretically, they are likely to suffer deterioration of some of their desirable properties mainly in their convergence rate being irregular or slower for problems arising in practical applications such as fluid dynamics, the one which we are interested.

Moreover, lack of robustness is a widely recognized drawback of iterative solvers relative to direct solvers. This has traditionally reduced the acceptance of iterative solvers in industrial applications and commercial codes for direct methods have been more favored.

One way to improve the convergence rate of Krylov subspace methods is a family of techniques known as *preconditioning*, the basic idea is, instead of solving the original system (3.49), we solve a related one

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{5.1}$$

where $\boldsymbol{M}$ is the preconditioner which approximates $\boldsymbol{A}$ in some sense trying to improve the spectral properties of the system to be solved.

Also the direct approximation of $\boldsymbol{A}^{-1}$ is possible, but in this case we solve the preconditioned system multiplying by $\boldsymbol{M}$ directly

$$\boldsymbol{M}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}\boldsymbol{b}. \tag{5.2}$$

In both cases we have the *left* preconditioned systems, *right* and *centered* preconditioning is also possible.

Centered or *split* preconditioning will only be considered in the framework of the conjugate gradient method in the form

$$\boldsymbol{M}_1^{-1}\boldsymbol{A}\boldsymbol{M}_2^{-1}\boldsymbol{y} = \boldsymbol{M}_1^{-1}\boldsymbol{b}, \quad \text{with} \quad \boldsymbol{x} = \boldsymbol{M}_2^{-1}\boldsymbol{y}, \tag{5.3}$$

where the preconditioner is in the form $\boldsymbol{M} = \boldsymbol{M}_1\boldsymbol{M}_2$. Right preconditioning will not be considered at all in this work.

There are two major desirable properties one have to take into account when developing a preconditioning technique. In general, a good preconditioner $\boldsymbol{M}$ should meet the following requirements:

- The preconditioned system (5.1) should be easier to solve that the original one (3.49). This means that the preconditioned iteration should converge, at least, faster that the unpreconditioned one.
- The preconditioner should be easy to construct and apply. That is, the preconditoned iteration is not significantly more costly than the unpreconditioned one.

Unfortunately, these two basic requirements are in competition with each other. Determination of a good balance is highly problem dependent and far from being a trivial task. Heuristic and trial-error strategies are often used in this respect.

Taking about a good preconditioner, usually means that the overall time for constructing such a preconditioner and solve the preconditioned iteration is less than the time spent for the unpreconditioned iteration. Nevertheless there are situations in which this criteria can not be applied; the most obvious case is when the unpreconditioned iteration does not converge after it performs a reasonable number of iterations and the success of such iterative method is only possible with the preconditioned system.

Some authors assert that the preconditioning concept goes as far as the 19-th century when Jacobi used plane rotations to achieve diagonal dominance of matrices arising from least squares problems before applying the method named after him [11, 71]. After that pioneering work, a large variety if techniques have emerged, some of them prompted in quite different concepts.

We shall discuss some of the most used, popular and successful preconditioners that we have implemented and used in the tests of this work. They basically belong to three different families and only the unsymmetric case is treated in detail since the symmetric case is in many cases a special case.

5.1 Preconditioned Krylov Methods

Before going in depth to the development of specific preconditioning techniques we first review the way in which preconditioning is applied in practical applications of Krylov subspace methods and along the way explore, very generally, the effect of preconditioning when applied to such iterative methods.

With this purpose in mind we shall consider a general preconditioner in the form $\boldsymbol{M} \approx \boldsymbol{A}$ in some sense. Remember that as we have seen in the previous chapter, and in contrast with direct methods, Krylov subspace methods only requires to be able to compute matrix by vector products in the form $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{y}$, or eventually with the matrix transpose $\boldsymbol{z} = \boldsymbol{A}^T\boldsymbol{y}$, without the need of either accessing or modifying individual entries of the matrix $\boldsymbol{A}$. Moreover, even the preconditioner $\boldsymbol{M}$ is sparse, we have not any guarantee that sparsity is preserved for its inverse. Even worse the product $\boldsymbol{M}^{-1}\boldsymbol{A}$ could be completely full.

In spite of this it is easy to notice that the explicit computation of the preconditioned system $\boldsymbol{M}^{-1}\boldsymbol{A}$ is never formed in practice, doing so will destroy all efficiency that can be obtained from the sparsity of $\boldsymbol{A}$. Instead, when the computation of the preconditioned matrix by vector product is required as $\boldsymbol{z} = \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{y}$, we segregate it in two stages using an auxiliary vector $\boldsymbol{w}$. First we compute the product $\boldsymbol{A}\boldsymbol{y}$ storing it in $\boldsymbol{w}$ and then apply the operator $\boldsymbol{M}^{-1}$ to the result. This last stage will be referred generically as *apply preconditioner*. This can be clarified, in the case of left preconditioning as follows

$$\boldsymbol{w} = \boldsymbol{A}\boldsymbol{y} \quad \text{and solve} \quad \boldsymbol{M}\boldsymbol{z} = \boldsymbol{w}. \tag{5.4}$$

In the case that the preconditioner represents a explicit approximation to the inverse of $\boldsymbol{A}$ the solving in the second stage is simply substituted by other matrix by vector product

$$\boldsymbol{w} = \boldsymbol{A}\boldsymbol{y} \quad \text{and} \quad \boldsymbol{z} = \boldsymbol{M}\boldsymbol{w}. \tag{5.5}$$

Note that in both cases in putting $\boldsymbol{M} = \boldsymbol{I}$ the unpreconditioned case is recovered and then this procedure is consistent with the unpreconditioned Krylov subspace methods.

5.1.1 Preconditioned GMRes

In order to obtain the preconditioned version of the Generalized Minimal Residual method, and particularly in its restarted practical version, first take into account that instead of solving the linear system (3.49) we are interested in solving the preconditioned linear system (5.1). For such preconditioned systems it is easy to see that its

preconditioned residual is given by

$$\boldsymbol{r} = \boldsymbol{M}^{-1}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\right) \tag{5.6}$$

in the case $\boldsymbol{M} \approx \boldsymbol{A}$ which will be the one discussed hereafter. Additionally, since we are solving a linear system with coefficient matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ instead of $\boldsymbol{A}$, each time a product with the latter appears it must be substituted by a product with the former matrix. With this observations we obtain the *Preconditioned Generalized Minimal Residual* method shown in algorithm 5.1.1.

Given $\boldsymbol{x}^{(0)}$, compute $\boldsymbol{r}^{(0)} = \boldsymbol{M}^{-1}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}\right)$

Set $\boldsymbol{v}_1 = \boldsymbol{r}^{(0)}/\rho$ with $\rho = ||\boldsymbol{r}^{(0)}||_2$ and $\boldsymbol{z}_k^{(0)} = \rho\hat{\boldsymbol{e}}_1$

for $k = 1, 2, ..., m$ **do**

  $\boldsymbol{w} = \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{v}_k$

  for $i = 1, 2, ..., k$ **do**

    $h_{ik} = (\boldsymbol{w}, \boldsymbol{v}_i)$

    $\boldsymbol{w} = \boldsymbol{w} - h_{ik}\boldsymbol{v}_k$

  end for

  $h_{k+1,k} = ||\boldsymbol{w}||_2$. If $h_{k+1,k} = 0$ stop.

  $\boldsymbol{v}_{k+1} = \boldsymbol{w}/h_{k+1,k}$

  $\boldsymbol{H}_{k,k+1}^{(k)} = \boldsymbol{G}_k\boldsymbol{H}_{k,k+1}^{(k-1)}$

  $\boldsymbol{z}_k^{(k)} = \boldsymbol{G}_k\boldsymbol{z}_k^{(k-1)}$

end for

Solve $\boldsymbol{R}_k\boldsymbol{y}^{(k)} = \boldsymbol{z}_k^{(0)}$ for $\boldsymbol{y}^{(k)}$.

Update $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_k\boldsymbol{y}^{(k)}$

Compute $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and $\rho = ||\boldsymbol{r}^{(0)}||_2$

if ρ is small enough **then**

  Take $\boldsymbol{x} \approx \boldsymbol{x}^{(k)}$ and stop.

else

  Set $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(k)}$ and restart.

end if

ALGORITHM 8: $PGMRes(k)$: Preconditioned Generalized Minimal Residual

The differences from the unpreconditioned version have been remarked in red. Note that they are minimal and the general implementation does not suffer significant changes.

More important, note that the underlying Arnoldi algorithm is now constructing an orthogonal basis for the Krylov subspace defined by the preconditioned system

$$\mathcal{K}_m\left(\boldsymbol{M}^{-1}\boldsymbol{A};\boldsymbol{r}^{(0)}\right) = span\left\{\boldsymbol{r}^{(0)}, \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{r}^{(0)}, ..., \left(\boldsymbol{M}^{-1}\boldsymbol{A}\right)^{m-1}\boldsymbol{r}^{(0)}\right\}, \tag{5.7}$$

with the modified Gram-Schmidt process having the intrinsic difficulty that the true residual is not available inside the Arnoldi orthogonalization process since we are getting only the preconditioned one. Then the true residual can only be evaluated at the end of each restart before applying the preconditioner.

Derivation of the preconditioned versions of other Krylov subspace methods based on the Arnoldi orthogonalization process are very similar and, with few exemptions, they follows straightforward.

5.1.2 Preconditioned Conjugate Gradient

One of the exceptions previously mentioned applies for probably the most known, popular and used Krylov subspace method, the Conjugate Gradient. The reason is that it has been designed, as we have seen in the previous chapter, for symmetric positive definite matrices. From the previous discussion in the present chapter it is now obvious that, when preconditionig is applied we are solving for another related linear system instead of the original one. It may happen that even if $\boldsymbol{A}$ is symmetric positive definite, the preconditioned systems can suffer lack of any, or both, desirable and required properties for the success of the Conjugate Gradient method. Then, while choosing a preconditioner for such a system we must be sure that the resulting preconditioner system is symmetric positive definite if application of the conjugate gradient method is pursued.

In order to avoid such lack of desirable properties of the original system while obtaining the preconditioned one, we first suppose that a split preconditioner is available as a product of a lower triangular matrix and its transpose as

$$\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T \tag{5.8}$$

then a simple way to preserve symmetry is to use a split preconditioning of the form

$$\boldsymbol{L}_1^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}\boldsymbol{y} = \boldsymbol{L}_1^{-1}\boldsymbol{b}, \quad \text{with} \quad \boldsymbol{x} = \boldsymbol{L}^{-T}\boldsymbol{y}, \tag{5.9}$$

and if both $\boldsymbol{A}$ and $\boldsymbol{L}$ are positive definite, the preconditioned system is also positive definite.

However, it is not necessary to split the preconditioner in this manner in order to preserve symmetry. We define the $M-$inner product

$$(\boldsymbol{x}, \boldsymbol{y})_M = (\boldsymbol{M}\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}, \boldsymbol{M}\boldsymbol{y}), \tag{5.10}$$

it can be verified that $\boldsymbol{M}^{-1}\boldsymbol{A}$ is self adjoint to respect this inner product as follows

$$\left(\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x}, \boldsymbol{y}\right)_M = (\boldsymbol{A}\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}, \boldsymbol{A}\boldsymbol{y}) = \left(\boldsymbol{x}, \boldsymbol{M}\left(\boldsymbol{M}^{-1}\boldsymbol{A}\right)\boldsymbol{y}\right) = \left(\boldsymbol{x}, \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{y}\right)_M. \tag{5.11}$$

Therefore we can replace the usual Euclidean inner product by the $M-$inner product. Writing the Conjugate Gradient for this yet defined inner product and denotingthe original residual by $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and the preconditioned residual by $\boldsymbol{z}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}$ and noting that the $M-$inner products must not be computed explicitly because

$$\left(\boldsymbol{z}^{(k)}, \boldsymbol{z}^{(k)}\right)_M = \left(\boldsymbol{r}^{(k)}, \boldsymbol{z}^{(k)}\right) \quad \text{and} \quad \left(\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{p}^{(k)}, \boldsymbol{p}^{(k)}\right) = \left(\boldsymbol{A}\boldsymbol{p}^{(k)}, \boldsymbol{p}^{(k)}\right)_M. \tag{5.12}$$

This defines the *Preconditioned Conjugate Gradient* method shown in algorithm 5.1.2 in which the differences with its unpreconditioned counter part have been marked in red.

Given $\boldsymbol{x}^{(0)}$ compute $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$

Set $\boldsymbol{z}^{(0)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}$ and $\boldsymbol{p}^{(0)} = \boldsymbol{z}^{(0)}$

for $k = 0, 1, 2, \ldots$ **do**

$\alpha_k = \frac{\left(\boldsymbol{r}^{(k)}, \boldsymbol{z}^{(k)}\right)}{\left(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}$

$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$

$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{A}\boldsymbol{p}^{(k)}$

$\boldsymbol{z}^{(k+1)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(k+1)}$

$\beta_k = \frac{\left(\boldsymbol{r}^{(k+1)}, \boldsymbol{z}^{(k+1)}\right)}{\left(\boldsymbol{r}^{(k)}, \boldsymbol{z}^{(k)}\right)}$

$\boldsymbol{p}^{(k+1)} = \boldsymbol{z}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)}$

end for

ALGORITHM 9: *PCG*: Preconditioned Conjugate Gradient.

Explicit split preconditioning is also possible but it is not considered in this work.

5.2 Classical Iterative Methods

The development of classical iterative methods goes back to the pioneering work of Gauss, Seidel and Jacobi and have dominated the solution of sparse linear systems

during the 1950 to 1960's with the advent of digital computers mainly focused on the solution of systems arising from elliptic partial differential equations. Roughly speaking, they are constructed by splitting the coefficient matrix as $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ requiring $\boldsymbol{M}$ being regular and easier to invert than $\boldsymbol{A}$. Usually they are regarded as the precursors of Krylov subspace methods and, as we have seen in chapter 3 with the Richardson method, the classical methods motivate those that use the Krylov subspace.

5.2.1 Jacobi Method

The most obvious and simplest preconditioner is based on the classical iterative method of *Jacobi* which splits the coefficient matrix as

$$\boldsymbol{A} = \boldsymbol{D} + (\boldsymbol{A} - \boldsymbol{D}), \tag{5.13}$$

where $\boldsymbol{D}$ is the main diagonal of the coefficient matrix which in turns will be used as the preconditioner $\boldsymbol{M} = \boldsymbol{D}$. Thus we solve the preconditioned linear system

$$\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{b}, \tag{5.14}$$

where it is assumed $\boldsymbol{D}^{-1}$ does exists. The only requirement to ensure such existence is that entries on the main diagonal of the coefficient matrix are not zero $a_{ii} \neq 0$ for all $i = 1, \cdots, n$.

The main advantage of the Jacobi preconditioning relies on the fact that it is inexpensive to compute and apply. Additionally, being the preconditioner diagonal it is also symmetric, a fact that preserves the symmetry of the preconditioned system.

However, it is well known, even in the classical iterative methods context, that the convergence of this method is usually very slow. Furthermore, the method is ensured to converge only when the coefficient matrix is diagonal dominant.

It is important to observe that it has ones on the main diagonal, therefore it can be interpreted as a row scaling operation.

5.2.2 Symmetric Gauss-Seidel Method

Implementation of the Jacobi method compute all the n entries of the $(k+1)-$th approximation by using only the previous $k-$th approximation with no matter that, when computing the $i-$th entry $x_i^{(k+1)}$ all from the first to the $(i-1)-$th entries are already available.

Improvements in the convergence rate in this respect have been obtained by taking into account updated values of the $(k+1)-$th approximated solution. That is, the computation of the $x_i^{(k+1)}$ entry is carried out as

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right], \quad \text{for} \quad i = 1, \cdots, n, \tag{5.15}$$

which is the *forward* version of the Gauss-Seidel method. Moreover, the *backward* version can also be implemented. It reads as

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k+1)} \right], \quad \text{for} \quad i = 1, \cdots, n, \tag{5.16}$$

In the preconditioning context, the forward Gauss-Seidel method is equivalent to splitting the coefficient matrix as $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{E}$ and $\boldsymbol{N} = \boldsymbol{F}$ where $\boldsymbol{E}$ and $\boldsymbol{F}$ are the strict lower and upper parts of the coefficient matrix and, as usual, $\boldsymbol{D}$ its main diagonal. For the backward version the places of $\boldsymbol{E}$ and $\boldsymbol{F}$ are mutually interchanged.

Combining both versions together we get the *symmetric* version. The corresponding preconditioner reads as follows

$$\boldsymbol{M} = (\boldsymbol{D} - \boldsymbol{E})\, \boldsymbol{D}^{-1} (\boldsymbol{D} - \boldsymbol{F})\,. \tag{5.17}$$

Again, this preconditioner is undefined in case when at least one diagonal entry of the coefficient matrix vanishes.

Notice that the left factor in (5.15) is lower triangular while the right one is upper triangular, therefore the preconditioner $\boldsymbol{M}$ can be expressed as the product of two matrices, one lower and the other upper triangular as $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{U}$ with

$$\boldsymbol{L} = (\boldsymbol{D} - \boldsymbol{E})\, \boldsymbol{D}^{-1} \quad \text{and} \quad \boldsymbol{U} = (\boldsymbol{D} - \boldsymbol{F})\,, \tag{5.18}$$

where, for convenience, we have chosen to make $\boldsymbol{L}$ unit triangular, but the other alternative is perfectly possible.

When the coefficient matrix is also symmetric a slight modification preserves the symmetry of preconditioner which can be expressed as the product $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$, but in this case the lower matrix is no longer unit triangular and it is given by

$$\boldsymbol{L} = (\boldsymbol{D} - \boldsymbol{E})\, \boldsymbol{D}^{-\frac{1}{2}}, \tag{5.19}$$

which requires the additional condition that all the entries in the main diagonal being strictly positive.

In the classical iterative methods context *relaxed versions* of the present method are extensively used and well studied. Nevertheless plenty of numerical experiments have demonstrated that the effect of relaxation is almost unnoticeable when used as preconditioners.

5.3 Incomplete Factorizations

When a sparse matrix is expressed by a product of its lower and upper factors as $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ computed via Gaussian elimination, usually *fill-in* takes place, that is, the triangular factors $\boldsymbol{L}$ and $\boldsymbol{U}$ are denser than the corresponding parts of $\boldsymbol{A}$.

There exist several techniques devoted to reducing such fill-in when the matrix is factored in the framework of direct method. Although these techniques are sometimes not enough for the tractability of the problem at hand, specially when n is very large or the sparse matrix is unstructured. Serious problems also arises when pivoting is mandatory for the stability of such factors because it usually destroys sparsity.

However, powerful preconditioners can be obtained with an incomplete factorization of the form $\boldsymbol{M} = \overline{\boldsymbol{L}}\,\overline{\boldsymbol{U}}$, preserving sparsity by discarding some entries of the factorization process, where $\overline{\boldsymbol{L}}$ and $\overline{\boldsymbol{U}}$ are the incomplete LU factors.

Incomplete factorizations algorithms differ in the rules used in order to discard or accept entries in the incomplete factors. These rules are mainly based on several different criteria as position or values of these entries.

In order to clarify the previous discussion we let an enumeration $\mathcal{N} = \{1, \cdots, n\}$, with it we define a *sparsity pattern* as a subset $\mathcal{S}(\boldsymbol{A}) \subseteq \mathcal{N} \times \mathcal{N}$ of ordered pairs (i, j) for which $a_{ij} \neq 0$. For convenience we shall also always include in such sparsity pattern all the diagonal entries of $\boldsymbol{A}$ regardless of whether they are zero or not. Note that, when the matrix is symmetric $\boldsymbol{A} = \boldsymbol{A}^T$ it does not matter the order in the pairs (i, j) and moreover, this sparsity patter contains redundant information. Thus the pattern corresponding to the triangular lower or upper part of the original matrix is enough to describe the whole matrix. This concept can be extended to matrices that, although being *numerically unsymmetric* they are *structurally symmetric*; that is, if $a_{ij} \neq 0$ implies that $a_{ji} \neq 0$. Discretization of partial differential equations usually leads us to such kind of matrices. When the matrix used is clear from the context we shall occasionally avoid its writing, then $\mathcal{S}(\boldsymbol{A}) = \mathcal{S}_{\boldsymbol{A}}$ or simply $\mathcal{S}(\boldsymbol{A}) = \mathcal{S}$.

We can extend the definition of sparsity pattern to any factorization for which the two by two intersection of the sparsity patterns of the factors is always empty or it is regular and can be easily identified. In particular when working with LU factorization it is useful to define the *coupling matrix* as

$$\boldsymbol{C} = \overline{\boldsymbol{L}} + \overline{\boldsymbol{U}} - \boldsymbol{I}, \tag{5.20}$$

where subtraction of the identity matrix is performed with the unique purpose to avoid storing the main diagonal of the lower factor which is assumed to be unit lower triangular.

Then, given such a set of non zeros in the incomplete factors $\overline{\boldsymbol{L}}$ and $\overline{\boldsymbol{U}}$ represented by $\mathcal{S}_{\boldsymbol{C}}$, the incomplete factorization process based in Gaussian elimination can be sketched as

$$a_{ij} = \begin{cases} a_{ij} - a_{ik}a_{kj}/a_{kk}, & \text{if } (i,j) \in \mathcal{S}_{\boldsymbol{C}} \\ a_{ij}, & \text{otherwise,} \end{cases} \tag{5.21}$$

for each k and for $i, j > k$. Note that if $a_{kk} = 0$ for any k, the incomplete factorization process cannot be completed successfully yielding what is referred as a *breakdown*.

5.3.1 ILU(0) Factorization

The most intuitive way of preserving sparsity consists in computing entries of the incomplete factors only if their corresponding entries in the coefficient matrix are non zero. That is, the sparsity pattern of the coupling matrix is a simply copy of those corresponding to the original matrix. That is

$$\mathcal{S} = \{(i,j) \mid a_{ij} \neq 0\}, \tag{5.22}$$

where the absence of any subscript can not cause any confusion.

Then we have obtained what is commonly known as the the *no-fill* of *ILU(0)* preconditioner. A general implementation based on the *ikj* version of Gaussian elimination reads as shown in algorithm 5.3.1.

Note that, if we skip the lines in red we get the traditional algorithm for computing the complete factors. Practical implementations does not perform such test since the data structures used to store the matrix and the incomplete factors are adapted to handle this situation. Then it must be clear that the presented implementation has only conceptual purposes.

for $i = 1, 2, ..., n$ **do**
  for $k = 1, 2, ..., i-1$ **do**
    if $(i, k) \in \mathcal{S}$ **then**
      $a_{ik} = a_{ik}/a_{kk}$
    end if
    for $j = k+1, k+2, ..., n$ **do**
      if $(i, j) \in \mathcal{S}$ **then**
        $a_{ij} = a_{ij} - a_{ik}a_{kj}$
      end if
    end for
  end for
end for

ALGORITHM 10: General ILU Factorization.

The main advantages of this preconditioner relies on its easy implementation, almost inexpensive computation and effectiveness in some model problems such as the discretization of elliptic partial differential equations, context in which they have arisen. For several kind of matrices, diagonal dominant for example, it can be proven that the ILU(0) factorization always exists and the factors are stable.

On the other hand, as already mentioned, this factorization can breakdown if a zero pivot is encountered. Even worse is the situation when the computation can be completed but the factors are unstable due to *near breakdowns* happening when pivots of small modulus pollutes the factorization. These problems have been traditionally observed in matrices that does not come from the specific areas for which this preconditioner have been originally proposed.

5.3.2 Modified ILU Factorization

The ILU(0) preconditioner simply discards entries dropped out during the incomplete elimination process. There are techniques that attempt to reduce the effect of dropping by *compensating* the discarded entries trying to improve the performance of the preconditioner.

A popular technique which performs such a compensation takes, at the end of the k loop os the ILU algorithm, all the elements that have been dropped and subtract them from

u_{kk} giving us the *MILU(0)* factorization which now satisfies

$$\boldsymbol{A}\boldsymbol{e} = \overline{\boldsymbol{L}}\,\overline{\boldsymbol{U}}\boldsymbol{e}, \tag{5.23}$$

where $\boldsymbol{e}$ is the n vector of ones. This ensures that the row sums of $\boldsymbol{A}$ are equal to those of $\overline{\boldsymbol{L}}\,\overline{\boldsymbol{U}}$, a desirable property in the discretization of partial differential equations since the vector $\boldsymbol{e}$ represents the discretization of a constant function. This additional constrain forces the ILU factorization to be exact for some functions in some sense.

It can be proven that, when a scalar elliptic partial differential equation is descretized using finite difference or finite elements, the spectral condition number of the resulting coefficient matrix grows as $\mathcal{O}\left(h^{-2}\right)$, where h is the *mesh size*. When a factorization like ILU(0) is used, the preconditioned system reduces its condition number but it still remains in the same order making no significant improvements in reducing the number of iterations of a Krylov subspace method. Nevertheless, specifically for these kind of problems, it has been shown also that the simple compensation performed by MILU(0) with respect to ILU(0) yields a preconditioned system whose spectral condition number behaves as $\mathcal{O}\left(h^{-1}\right)$ and thus significant reductions in the number of iterations required to achieve a prescribed accuracy are expected by performing an almost inexpensive computation. Note that additionally, the implementation for MILU(0) is straightforward from those of ILU(0).

5.3.3 ILU Factorization with Fill-in

The accuracy of the ILU(0) preconditioner may be insufficient to yield an acceptable rate of convergence. More robust and accurate ILU factorizations often represent significant improvements in efficiency.

The quality of *ILU* preconditioners can be improved by allowing some fill-in in the factors. This is done by attributing to each entry that is processed by the factorization procedure a *level of fill*. For instance, consider the algorithm of the ILU factorization without taking into account the statements in red. The foundation of this idea is the level of fill-in should be an indicative of the size, the higher the level, the smaller the elements. This can be intuitively verified attributing, somewhat arbitrarily, to any element whose level if fill is k the size ϵ^k, where $\epsilon < 1$. Initially, a non zero element has a level of fill of one and a zero element has a level of fill of ∞. From the ILU factorization algorithm, notice that the element a_{ij} is modified accordingly to

$$a_{ij} = a_{ij} - a_{ik}a_{kj} \tag{5.24}$$

If lev_{ij} is the current level of the entry a_{ij}, then the its size after modification will be given by

$$\epsilon^{lev_{ij}} - \epsilon^{lev_{ik}}\epsilon^{lev_{kj}} = \epsilon^{lev_{ij}} - \epsilon^{lev_{ik}+lev_{kj}}. \tag{5.25}$$

Then, the size of a_{ij} will be proportional to the maximum of the two sizes $\epsilon^{lev_{ij}}$ and $\epsilon^{lev_{ik}+lev_{kj}}$. Therefore it is now natural to define the new level of fill as

$$lev_{ij} = \min\left\{lev_{ij} - lev_{ik} + lev_{kj}\right\}. \tag{5.26}$$

Nevertheless, we shift all the levels in the previous definitions by -1 with the only purpose to be consistent with the definition of the ILU(0) factorization. Thus we define the initial level of fill related to a matrix entry a_{ij} as

$$lev_{ij} = \begin{cases} 0, & \text{if } a_{ij} \neq 0 \text{ or } i = j \\ \infty, & \text{otherwise.} \end{cases} \tag{5.27}$$

Each time this element is modified by the ILU process, its level of fill is updated according to

$$lev_{ij} = min\left\{lev_{ij}, lev_{ik} + lev_{kj} + 1\right\}. \tag{5.28}$$

Including only entries whose level is equal or lesser than p, with p a positive integer, we obtain the *ILU(p)* preconditioner. It is important to mention now that an important drawback of this method is the impossibility to predict and then control the memory requirements for $p > 0$.

Notice that, even though the updating strategy used for the level of fill is numerically founded, the actual numerical values of the corresponding entries are not taken into account. Therefore its computation depends only in the sparsity pattern and has nothing to do with the numerical entries of the coefficient matrix. This has some pros and cons.

First we mention the advantages of this incomplete factorization strategy. Probably the most important is the possibility to break up this process in two phases that can be run sequentially, one *symbolic* and one *numeric*. The first phase is used to determine the level of fill in the factors, and the second to carry out the actual numerical factorization. Even more, similar strategies are used in the framework of direct methods in order to allocate only the necessary memory positions for the factorization of a sparse matrix [40, 45, 62].

On the other hand, the main numerical drawback of this approach is that it has been originally designed for matrices arising in the finite difference discretization of scalar elliptic partial differential equations, mainly using structured and uniform meshes in simply shaped domains which are diagonal dominant, banded and most important the

heuristic given by (5.25) is completely valid. This is not the case for matrices with irregular structure or not diagonal dominant. The situation could be worse when several physical quantities are approximated at once or anisotropy is present.

5.3.4 ILU Factorization with Dropping

Notice that the idea of compensation of dropped entries during the factorization process used for the MILU(0) factorization is quite generic and it can be adapted and implemented in several different contexts.

Additionally and maybe more importantlt, notice that in the previous variants of ILU factorization, dropping strategies are based only on the sparsity pattern of the matrix which, for certain problems, would give us a poor quality preconditioner.

An alternative strategy for controlling fill-in is to accept or discard a new fill on the basis of its size instead of such position based criterion. Then, as a first approach in this context consider that, given a *drop tolerance* τ we discard all the entries in the factors smaller than τ. It immediately arises that this strategy will de discriminatory if the matrix is badly scaled, a common situation in the already mentioned situation when anisotropy or different physical quantities are discretized.

In order to overcome such difficulty we slightly modify the original idea by accept or discard entries using a *relative* dropping strategy. Therefore we store and use in the subsequent factorization process the entry a_{ij} in the i-th row if it satisfies the criterion

$$|a_{ij}| > \tau \left(\frac{1}{n} \sum_{k=1}^{n} |a_{ik}| \right). \tag{5.29}$$

The reason for choosing row sums and not column sums in evaluating the average is due to the fact that our generic implementation of the incomplete factorization is based in the *ikj* version of Gaussian elimination which computes the factors row by row; but other options are possible.

Furthermore, a *relaxed* diagonal compensation can be implemented by subtracting a portion of the sum of all dropped entries s to the k-th diagonal entry of $\boldsymbol{U}$

$$a_{ii} = a_{ii} - \alpha s, \quad \text{with} \quad 0 \leq \alpha \leq 1. \tag{5.30}$$

Note that ILU(0) and MILU(0) are two extreme cases of a similarly defined method with $\alpha = 0$ and $\alpha = 1$ respectively.

Putting all this together gives us the ILU factorization with standard dropping $ILUD(\alpha, \tau)$. which has been demonstrated to outperform ILU(p) preconditioners with about the same memory usage.

The major drawback of this strategy is that it is virtually impossible to predict and directly control the fill-in in the factors of the preconditioner. Additionally, choosing good values for τ is strongly problem dependent making the situation quite complicated because a smaller τ will, at least intuitively, lead us a more stable and effective preconditioner but increasing the cost in both computation and application of the present incomplete factorization.

5.3.5 ILU Factorization with Threshold

As already stated in the previous subsection, when using ILUD preconditioners, memory requirements might be unpredictable because the dropping is based only on a numerical criterion. A remedy for this situation is to limit the number of entries to be allowed in each row of the incomplete $\boldsymbol{LU}$ factors.

First a dropping strategy is used to accept or discard entries in the factorization based on their magnitude,as done similarly with ILUD but this time using the Euclidean norm of the current row as

$$|a_{ij}| > \tau \left(\sum_{k=1}^{n} |a_{ik}|^2 \right)^{\frac{1}{2}}. \tag{5.31}$$

Once they have been *filtered*, we store and use only the remaining p greatest entries in magnitude, apart from the diagonal entry, in the current row of the coupled matrix $\boldsymbol{C}$.

This give us the ILU with dual threshold strategy, $ILUT(\tau, p)$ which have been demonstrated to be quite powerful even for problems not arising from the discretization of partial differential equations making it robust and reliable.

Although the selection of the parameters p and τ is also problem dependent this dependency is not so strong as the observed with the ILUD preconditioner. Even more, if the computation of the ILUT preconditioner breaks down or the resulting factors are unstable for a given a pair of such parameters, it will often succeed by taking a smaller value of τ or increasing p, usually a combination of these two modifications will markedly increase the effectiveness of the preconditioner.

5.3.6 ILUD and ILUT factorization with Pivoting

The presence of zero or small pivots during the Gaussian elimination process is a serious and well studied problem even in the case of dense complete factorization methods. An exactly zero pivot will immediately cause a breakdown in the factorization process and the LU factors in this case do not exist even with $\boldsymbol{A}$ being regular. The second situation, when a small pivot is encountered in the factorization process, could be even worse because, although the factorization process can be completed the resulting factor happens to be unstable and therefore useless.

This situation has been extensively studied in the frame of complete factorization and the direct solution of general, and mainly dense, linear systems. The simplest and most popular remedy has traditionally been the use of *pivoting*, which consist in interchanging rows and columns in order to move entries better suited to be taken as pivots, which generally are those with the greatest magnitude. Note that row interchange is equivalent to pre-multiplication by a elementary permutation matrix $\boldsymbol{P}$ as those defined by (3.14) while column interchanges are achieved with post-multiplication of similarly defined matrices.

The simplest strategy consists in looking for, at the $k-$th stage of the factorization process, the greatest module entry in the $k-$th column below the main diagonal; this is what is usually called *partial pivoting* and for the vast majority of situations it is enough to improve the stability of the resulting factors. In case the mentioned strategy is not enough *total pivoting* can be used requiring row as column interchanges in order to move the greatest module entry contained in the square sub matrix known as the $k-$th *Schur complement* of $\boldsymbol{A}$ respect Gaussian elimination. As a cheaper but effective strategy *rook pivoting* can be alternatively used.

Therefore in order to increase the quality of incomplete LU factorizations it is quite intuitive that pivoting strategies will be useful in this regard. Unfortunately performing total and rook pivoting is only possible when the kij of kji versions of Gaussian elimination are used. Performing partial pivoting by rows, as described previously, is possible when any of the kij, kji, jik or jki versions of Gaussian elimination is used. In view that the incomplete factorization algorithms described previously are based on the ikj version of Gaussian elimination pivoting can only be performed by columns.

Furthermore, a relaxed pivoting strategy has been implemented in order to determine whether or not to permute two columns based on a *pivoting tolerance* δ. At step i columns i and j are permuted when

$$\delta|a_{ij}| > |a_{ii}|, \tag{5.32}$$

note that $\delta = 0$ corresponds to the unpivoted versions.

Additionally, the search for pivots could be performed only in a block of size r previously specified, a feature useful with systems arising from discretization of partial differential equations with several physical quantities per node or cell.

The implementation of the present pivoting strategy with the ILUD and ILUT methods gives us the $ILUDP(\alpha, \tau; r, \delta)$ and $ILUTP(\tau, p; r, \delta)$ incomplete factorizations.

Since the pivoting strategy does not imply arithmetical operations, except n divisions of the form a_{ii}/δ, it is expected that this pivoting strategy does not significantly increase the cost of computing the corresponding preconditioners in comparison to the unpivoted versions. On the other hand, if there is reliable evidence that the entries with the greatest modules are close to the main diagonal we can restrict the magnitude of the parameter r reducing the number of comparisons to be done.

In general, since the pivoted version requires two more parameters than their unpivoted counterparts they are even more problem dependent and a good combination of all these parameters could be a trial and error task. Usually this tuning process is carried out with sample matrices when a reliable and robust implementation is sought.

5.4 Sparse Approximate Inverses

The ILU techniques discussed in the previous section were originally developed for matrices arising from the discretization of elliptic partial differential equations in one variable. As we have seen such factorization processes can completely break down if a zero pivot is encountered during the elimination process. Even in the presence of non zero but small pivots the provided factors might be unstable and they can not improve, or even worse, the convergence rate of the Krylov subspace method used to solve the related preconditioned system.

This situation has been observed experimentally when lack of diagonal dominance takes place. A common situation in the discretization of convection diffusion problems dominated by the former or in the discretization of partial differential equations where different physical quantities are approximated at once, a classical example being the Navier-Stokes equations in which we are interested. Other branches having these types of difficulties are circuit simulation, chemistry and economy to name only a few.

One possible remedy is to try to find a preconditioner that does not require the solution of a linear system as done for the incomplete LU factors. One option is the direct approximation of the inverse of the coefficient matrix $\boldsymbol{A}^{-1}$ by a matrix $\boldsymbol{M}$ and to use it

a a preconditioner by performing its product with $\boldsymbol{A}^{-1}$ removing the necessity of solving a linear system when the preconditioner is applied. This yields a variety of techniques referred to as *sparse approximate inverse* preconditioning.

This kind of technique relies on the assumption that for a given sparse matrix $\boldsymbol{A}$ it is possible to find a sparse matrix $\boldsymbol{M}$ which approximates its inverse $\boldsymbol{A}^{-1}$ in some sense. Although this is, at least at the first glace, not obvious at all because even in the case when $\boldsymbol{A}$ is sparse its inverse is usually completely full. Nevertheless, it is often the case that many of the entries in such an inverse are small in modulus, thus we can take and approximation by *sparsifying* using a dropping strategy.

The most important result in this concern is for banded symmetric positive definite matrices is the classical result of Demko, Moss and Smith [42]. It states that for such kind of matrix the magnitudes in its inverse are bounded in an exponentially decaying manner as

$$\left|\left[\boldsymbol{A}^{-1}\right]_{ij}\right| \leq C\rho^{|i-j|} \quad \forall \quad i,j = 1,2,\cdots,n; \tag{5.33}$$

where $0 \leq \rho(\kappa,\beta) \leq 1$ and also $C = C(\kappa,\beta)$ with κ the spectral condition number and β the bandwidth of the matrix $\boldsymbol{A}$. Thus, it follows that the most influential entries, those having greatest modulus, are confined in a narrow region around the main diagonal. Unfortunately, as the condition number grows the factor ρ approaches to one quickly doing the decay almost imperceptible but the basic idea can be, even in this unfavorable case, exploited.

It is important to note that the most attractive feature of this sparse approximate inverse approach is its easy and intuitive parallel implementation, since its application consists of matrix-vector products. In what follows we shall see also that for some schemes, also the computation of the preconditioner requires a high degree of ingenuity.

5.4.1 Sparse Inverse Preconditioner

This is one of the oldest suggested and investigated preconditioning schemes in the context of sparse inverse approximation and belongs to a family of methods referred as *Frobenius norm minimization.*

The basic idea in this framework is to compute a direct approximation of the inverse solving the constrained minimization problem

$$\min_{\boldsymbol{M}\in\mathcal{S}} ||\boldsymbol{I} - \boldsymbol{A}\boldsymbol{M}|\,|_F = \min_{\boldsymbol{M}\in\mathcal{S}} ||\boldsymbol{A}\boldsymbol{M} - \boldsymbol{I}|\,|_F \tag{5.34}$$

with $\mathcal{S}$ being a set of matrices with a given sparsity pattern.

This minimization problem can be decoupled in n independent least squares problems because

$$||\boldsymbol{I} - \boldsymbol{A}\boldsymbol{M}||_F^2 = \sum_{j=1}^{n} ||\boldsymbol{e}_j - \boldsymbol{A}\boldsymbol{m}_j||_2^2, \tag{5.35}$$

where $\boldsymbol{e}_j$ and $\boldsymbol{m}_j$ denote the j-th columns of the identity and the approximate inverse respectively. From (5.35) it is now clear the reason by choosing the Frobenius norm. Then we must solve, possibly is parallel, n minimization problems of the form

$$\min_{\mathcal{S}(\boldsymbol{m}_j)} ||\boldsymbol{e}_j - \boldsymbol{A}\boldsymbol{m}_j||_2^2, \quad \text{for} \quad j = 1, \cdots, n. \tag{5.36}$$

where $\mathcal{S}(\boldsymbol{m}_j)$ is the sparsity pattern of each of the columns $\boldsymbol{m}_j$ of the approximate inverse, which is completely defined from the sparsity pattern of the overall approximate inverse $\mathcal{S}(\boldsymbol{M})$.

Notice that (5.36) is in fact a least squares problem of small size, although n can be large, due to the sparsity of $\boldsymbol{A}$.

The main issue with this approach is to choose the sparsity pattern $\mathcal{S}(\boldsymbol{M})$. Early implementations of this approach choose the sparsity pattern $\mathcal{S}(\boldsymbol{M})$ in advance, for example setting it the same as the original matrix or a power of it $\boldsymbol{A}^k$ with k small . Although this strategy has proved to be quite inefficient an *adaptive* strategies has been devised and implemented. First, in the sake of simplicity, we shall assume that such sparsity pattern has been suitable choose in advance, after we shall explain how it is updated in order to improve the approximate inverse.

Let the column vector $\boldsymbol{m}_j$ have $k_j \ll n$ non zeros and let $\mathcal{J} = \{i \mid m_{ij} \neq 0\}$ be the set of non zeros in such a column $\boldsymbol{m}_j$, then (5.36) is effectively reduced to a problem involving a submatrix $\boldsymbol{A}_j$ of $\boldsymbol{A}$ consisting of the k_j columns of $\boldsymbol{A}$ corresponding to the non zero positions of $\boldsymbol{m}_j$, that is $\boldsymbol{A}_j = \boldsymbol{A}(\mathcal{N}, \mathcal{J})$. As $\boldsymbol{A}$ is sparse $\boldsymbol{A}_j$ has only r_j rows that are non zeros expected to be much smaller than n to sparsity of $\boldsymbol{A}$. With this definitions we now let $\mathcal{I}$ the set of such r_j indices and denote by $\hat{\boldsymbol{A}} = \boldsymbol{A}(\mathcal{I}, \mathcal{J})$ the submatrix extracted from $\boldsymbol{A}$, the right hand side $\hat{\boldsymbol{e}}_j = \boldsymbol{e}_j(\mathcal{I})$ and the unknown vector $\hat{\boldsymbol{m}}_j = \boldsymbol{m}_j(\mathcal{J})$ and solve the least squares problem

$$\left\| \hat{\boldsymbol{e}}_j - \hat{\boldsymbol{A}}\hat{\boldsymbol{m}}_j \right\|_2 = min, \tag{5.37}$$

for $\hat{\boldsymbol{m}}_j$ by QR factorization in order to determine the non zero entries in $\boldsymbol{m}_j$. Note that, if we force that the sparsity pattern $\mathcal{S}(\boldsymbol{M})$ contains at least the main diagonal we get that $r_j \geq k_j$ and the *reduced least squares* problem defined by (5.37) is well suited for such QR factorization.

Now, we set the entries in the $j-th$ column of $\boldsymbol{M}$ according to $\boldsymbol{m}_j(\mathcal{J}) = \hat{\boldsymbol{m}}_j$ and define the residual of the least squares solution provided by (5.37) as

$$\begin{aligned} \boldsymbol{r} &= \boldsymbol{e}_j - \boldsymbol{A}\boldsymbol{m}_j, \\ &= \boldsymbol{e}_j - \boldsymbol{A}_j\hat{\boldsymbol{m}}_j, \\ &= \boldsymbol{e}_j - \boldsymbol{A}(\mathcal{N}, \mathcal{J})\,\hat{\boldsymbol{m}}_j. \end{aligned} \tag{5.38}$$

Taking norms on both sides, and given a tolerance ϵ we expect that

$$\|\boldsymbol{r}\|_2 = \left\|\hat{\boldsymbol{e}}_j - \hat{\boldsymbol{A}}\hat{\boldsymbol{m}}_j\right\|_2 < \epsilon, \tag{5.39}$$

is satisfied, otherwise we can try again by modifying the originally proposed sparsity pattern $\mathcal{S}(\boldsymbol{m}_j)$ suggesting such a yet mentioned adaptive strategy.

The underlying idea of this adaptive strategy is to enlarge $\mathcal{S}(\boldsymbol{m}_j)$ systematically to get a better approximation for $\boldsymbol{m}_j$ for which (5.39) is eventually satisfied. The most natural and used strategy for enlarging $\mathcal{S}(\boldsymbol{m}_j)$ is done by defining the set of non zeros in the residual vector given by (5.38) as

$$\mathcal{L} = \{\ell \mid r_\ell \neq 0\}, \tag{5.40}$$

which will be referred as the *set of candidates* form which will be obtained the indices of the most profitable columns to be appended while enlarging $\mathcal{S}(\boldsymbol{m}_j)$. This profitability is evaluated by computing the norm of the residual vector when the k index, contained in $\mathcal{L}$, is added to $\mathcal{J}$ for each $k \in \mathcal{L}$. Note that doing so is extremely costly and since what we need is an approximation we can use an heuristic to approximates this computation with the expression

$$\tilde{\rho}_k = \rho - \frac{(\boldsymbol{r}, \boldsymbol{A}\boldsymbol{e}_k)^2}{\|\boldsymbol{A}\boldsymbol{e}_k\|_2^2} \tag{5.41}$$

where $\rho = \|\boldsymbol{r}\|_2$. Then we append the index k to $\mathcal{J}$ for which $\min_{k\in\mathcal{L}} \tilde{\rho}_k$, or possibly several of them. with this enlarged set of columns indices $\mathcal{J} = \mathcal{J} \cup k$ we get its corresponding set of row indices $\mathcal{I}$ and compute again the problem given by (5.37). We continue this process iteratively until (5.39) is satisfied or we reach the limitation in the number of rows allowed in $\hat{\boldsymbol{A}}$ which is equal the cardinality of the set of column indices $|\mathcal{J}_k| < p$.

This lead us to the $SPAI(\epsilon, p)$ preconditioner, which have proved to be very effective even for weakly diagonal dominant matrices and which computation and applying in parallel is almost trivial.

5.4.2 Approximate Inverse by Minimal Residual

Setup time for SPAI preconditioners is usually very high even when performed in parallel. This has motivated the use of cheaper strategies in order to reduce the time spent while computing such approximate inverse.

One of those strategies is, instead of the exact minimization of $||\boldsymbol{I} - \boldsymbol{AM}||_F$ performed column by column, we replace it by an approximate one.

This can be done with a few iterations of a one dimensional projection method such as *Minimum Residual* applied to each of the systems

$$\boldsymbol{A}\boldsymbol{m}_j = \boldsymbol{e}_j \quad \text{for} \quad j = 1, \cdots, n, \tag{5.42}$$

using an initial guess $\boldsymbol{M}_0$ for the approximate inverse we get algorithm 5.4.2.

Given $\boldsymbol{M} = \boldsymbol{M}_0$
for $j = 0, 1, 2, ...$ **do**
  Define $\boldsymbol{m}_j = \boldsymbol{M}\,\boldsymbol{e}_j$
  for $i = 0, 1, 2, ..., n_j$ **do**
    Compute $\boldsymbol{r}_j = \boldsymbol{e}_j - \boldsymbol{A}\boldsymbol{m}_j$
    $\alpha_j = \frac{(\boldsymbol{r}_j, \boldsymbol{A}\boldsymbol{r}_j)}{(\boldsymbol{A}\boldsymbol{r}_j, \boldsymbol{A}\boldsymbol{r}_j)}$
    $\boldsymbol{m}_j = \boldsymbol{m}_j + \alpha_j \boldsymbol{r}_j$
    Drop small entries in $\boldsymbol{m}_j$
    If $||\boldsymbol{r}||_2 < \epsilon$ stop
  end for
end for

ALGORITHM 11: $AIMR(p, \tau)$: Approximate Inverse by Minimal Residual.

Where n_j is the number of iterations allowed in the residual minimization of the corresponding $\boldsymbol{m}_j$ column and ϵ is a previously specified *residual tolerance*.

In order to preserve sparsity the small entries are dropped and memory limiting can be included similarly as done with ILUT techniques by allowing only the p largest entries in each column $\boldsymbol{m}_j$ of the approximate inverse $\boldsymbol{M}$.

It has been experimentally demonstrated that we do not need to specify any sparsity pattern in advance since large entries in $\boldsymbol{m}_j$ emerge automatically after few iterations, then we take $\boldsymbol{M}_0 = \boldsymbol{I}$.

It is important to mention that all the operations in the previous algorithm must be performed in *sparse-sparse* mode in order to improve the speed and keep as low as possible the memory requirements. Then the residual vector $\boldsymbol{r}$ and the columns of the approximate inverse $\boldsymbol{m}_j$ are stored and operated with in sparse mode.

This gives us the $AIMR(p, \tau)$ preconditioner which reduces substantially the cost of computing the approximate inverse compared to the SPAI algorithm. It has been proved its defectiveness specially with diagonal dominant and even weakly diagonal dominant matrices. Although, notice that the accuracy of this preconditioner is subjected to convergence of the minimal residual method representing a drawback of this method. As with ILUT techniques, the selection of the parameters p and τ is very problem dependent and even more, this algorithm requires two additional input parameters n_j and ϵ. In our current implementation we have set $t = n_j$.

5.4.3 Incomplete Bi-Conjugation

It is now the time to review and alternative of the Frobenius norm minimization schemes presented in the two previous subsections. This techniques relies in a generalization of the Gram-Schmidt algorithm is a very similar way in which the orthogonalization Lanczos algorithm for symmetric matrices has been generalized to the biorthogonalization Lanczos algorithm for unsymmetric matrices.

furthermore, we are now interested in the possibility to express the preconditioner $\boldsymbol{M} \approx \boldsymbol{A}^{-1}$ as the product of k matrices as

$$\boldsymbol{M} = \prod_{i=1}^{k} \boldsymbol{M}_i; \tag{5.43}$$

giving us a family of preconditioners usually referred as *factored sparse approximate inverses*.

As previously mentioned, a generalization of the Gram-Schmidt process know as *bi-conjugation* provides a natural way to compute a triangular factorization of $\boldsymbol{A}^{-1}$ working only and directly with $\boldsymbol{A}$.

Suppose that $\boldsymbol{A}$ admits the factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\,\boldsymbol{V}$, where $\boldsymbol{L}$ is unit lower triangular, $\boldsymbol{D}$ is diagonal and $\boldsymbol{V}$ is unit upper triangular, then $\boldsymbol{A}^{-1}$ can be factored in the form

$$\boldsymbol{A}^{-1} = \boldsymbol{V}^{-1}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1} = \boldsymbol{Z}\,\boldsymbol{D}^{-1}\,\boldsymbol{W}^{T}, \tag{5.44}$$

where, for instance, $\boldsymbol{Z}$ and $\boldsymbol{W}$ are unit upper triangular matrices. Note that, by construction, the sets of vectors $\{\boldsymbol{z}_1, \cdots, \boldsymbol{z}_n\}$ and $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_n\}$ forming the columns of $\boldsymbol{Z}$

and $\boldsymbol{W}$ are *A-biconjugate*, that is, the following relationship among them holds

$$(\boldsymbol{w}_i, \boldsymbol{A}\boldsymbol{z}_j) = \begin{cases} \delta_i, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{5.45}$$

Obtaining a factored sparse inverse, means that the factors, and not its product, are sparse matrices. The diagonal matrix $\boldsymbol{D}^{-1}$ meets this requirement by definition. Although there is no reason to expect that any or both of these upper triangular matrices $\boldsymbol{Z}$ and $\boldsymbol{W}$ are sparse even when $\boldsymbol{A}$ is.

Here we claim again to the result of Demko, Moss and Smith (5.33) and suppose the modulus of the entries in the factors $\boldsymbol{Z}$ and $\boldsymbol{W}$ decay from the main diagonal in a nice behaved manner. Therefore we set a drop tolerance τ and a maximum number of allowed entries in each column of $\boldsymbol{Z}$ and $\boldsymbol{W}$ p which are applied in the same manner as we have done for ILUT or AIMR.

If the triangular factors $\boldsymbol{Z}$ and $\boldsymbol{W}$ are sparsified by means of these dropping strategies, we get the factored approximate inverse of the form

$$\boldsymbol{M} = \overline{\boldsymbol{Z}}\boldsymbol{D}^{-1}\overline{\boldsymbol{W}}^T \approx \boldsymbol{A}^{-1}. \tag{5.46}$$

Putting all this together leads us algorithm 5.4.3 for computing the $AINV(p, \tau)$ preconditioner.

It has been proved that, when the matrix $\boldsymbol{A}$ is regular, the biconjugation algorithm can always be completed, at least in exact arithmetical. Unfortunately, because incompleteness divisions by zero may occur even when regularity of $\boldsymbol{A}$ is ensured; even worse is the situation when the the algorithm can be completed but the resulting factors are unstable yielding no acceleration when using it in conjunction with a Krylov subspace method. There is plenty of experimental evidence that this preconditioner is quite effective in the same cases when ILU(p) with a small p is. It is also reported that it is possible to implement diagonal modifications to force the success of the computation, we have not explored this techniques in the present work. It is very important to notice that this algorithm is naturally sequential because the columns of $\boldsymbol{Z}$ and $\boldsymbol{W}$ are biorthogonalized each other by using all the previously computed columns making the parallel implementation of this algorithm far from being trivial.

5.4.4 Approximate Inverse via Bordering

As we have seen, an important drawback of the AINV preconditioner is its sequential manner in which it operates difficult its parallelization. For this reason alternative

Given $\boldsymbol{Z}_0$ and $\boldsymbol{W}_0$

for $i = 1, 2, ...n$ **do**

  for $j = i, i+1, ..., n$ **do**

    $\delta_j^{(i-1)} = \left(\boldsymbol{A}^T \boldsymbol{e}_i, \boldsymbol{z}_j^{(i-1)}\right)$

    $\gamma_j^{(i-1)} = \left(\boldsymbol{A}\, \boldsymbol{e}_i, \boldsymbol{w}_j^{(i-1)}\right)$

  end for

  for $j = i+1, i+2, ..., n$ **do**

    $\boldsymbol{z}_j^{(i)} = \boldsymbol{z}_j^{(i-1)} - \frac{\delta_j^{(i-1)}}{\delta_i^{(i-1)}} \boldsymbol{z}_i^{(i-1)}$

    $\boldsymbol{w}_j^{(i)} = \boldsymbol{w}_j^{(i-1)} - \frac{\gamma_j^{(i-1)}}{\gamma_i^{(i-1)}} \boldsymbol{w}_i^{(i-1)}$

  end for

  Drop small entries in $\boldsymbol{z}_j^{(i)}$ and $\boldsymbol{w}_j^{(i)}$

  Retain only the p largest entries in $\boldsymbol{z}_j^{(i)}$ and $\boldsymbol{w}_j^{(i)}$

end for

Take $\boldsymbol{z}_i = \boldsymbol{z}_i^{(i-1)}$ and $\boldsymbol{w}_i = \boldsymbol{w}_i^{(i-1)}$

ALGORITHM 12: *AINV*(p, τ): Approximate Inverse.

techniques methods more suitable for parallelization have been developed. An alternative strategy for the factored sparse inverses approach is derived by seeking two unit triangular matrices such that

$$\boldsymbol{L}\boldsymbol{A}\,\boldsymbol{V} = \boldsymbol{D} \quad \Rightarrow \quad \boldsymbol{A}^{-1} = \boldsymbol{V}\,\boldsymbol{D}^{-1}\boldsymbol{L}. \tag{5.47}$$

But notice that such expression is exactly the same as (5.44) apart of the proper changes in notation. Now $\boldsymbol{L}$ is unit lower triangular, $\boldsymbol{V}$ is unit upper triangular and $\boldsymbol{D}$ remains as a diagonal matrix. In this case we call to $\boldsymbol{L}$ and $\boldsymbol{V}$ the *inverse LV factors* of $\boldsymbol{A}$.

Furthermore, these matrices can be built one column or row at a time. In order to clarify the subsequent discussion we introduce the notation referring to sequence of matrices in the form

$$\boldsymbol{A}_{k+1} = \begin{bmatrix} \boldsymbol{A}_k & \boldsymbol{v}_k \\ \boldsymbol{w}_k^T & \alpha_{k+1} \end{bmatrix}, \tag{5.48}$$

in which, by definition $\boldsymbol{A}_n = \boldsymbol{A}$.

Suppose we have computed the inverse factors until stage k and then $\boldsymbol{L}_k$ and $\boldsymbol{V}_k$ are available. Thus, the inverse factors in the next stage $k+1$ can be obtained by writing

$$\begin{bmatrix} \boldsymbol{L}_k & \boldsymbol{0} \\ -\boldsymbol{y}_k^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_k & \boldsymbol{v}_k \\ \boldsymbol{w}_k^T & \alpha_{k+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_k & -\boldsymbol{z}_k \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{D}_k & \boldsymbol{0} \\ \boldsymbol{0}^T & \delta_{k+1} \end{bmatrix}. \tag{5.49}$$

From which follows immediately that at each k step we must solve two linear systems of size k in order to determine the next row $-\boldsymbol{y}_k^T$ of $\boldsymbol{L}_{k+1}$ and the next column $-\boldsymbol{z}_k$ of $\boldsymbol{V}_{k+1}$. These linear systems are the following

$$\boldsymbol{A}_k \boldsymbol{z}_k = \boldsymbol{v}_k \quad \text{and} \quad \boldsymbol{A}_k^T \boldsymbol{y}_k = \boldsymbol{w}_k. \tag{5.50}$$

In turns, the $k+1$ entry of the diagonal matrix $\boldsymbol{D}$ is given by

$$\delta_{k+1} = \alpha_{k+1} - \boldsymbol{w}_k^T \boldsymbol{z}_k - \boldsymbol{y}_k^T \boldsymbol{v}_k + \boldsymbol{y}_k^T \boldsymbol{A}_k \boldsymbol{z}_k. \tag{5.51}$$

Notice that (5.51) can be evaluated once the solution of both systems in (5.50) are available. These solutions can be sought using a approximate method requiring a relative low accuracy. More important such linear systems are closely related, the second is the *dual* of the first one; this is a becoming situation for methods based on biorthogonalization Lanczos algorithms, in fact, in chapter 3 we have stated a version being able to produce both solutions in the algorithm corresponding to the Bi-Conjugate Gradient method.

Therefore performing few BiCG iterations together with numerical dropping and memory limiting we get the $AIB(p,\tau)$ preconditioner which is sketched in algorithm 5.4.4.

---

for $k = 1, 2, \ldots n$ **do**

Solve $\boldsymbol{A}_k \boldsymbol{z}_k = \boldsymbol{v}_k$ and $\boldsymbol{A}_k^T \boldsymbol{y}_k = \boldsymbol{w}_k$

Compute $\delta_{k+1} = \alpha_{k+1} - \boldsymbol{w}_k^T \boldsymbol{z}_k - \boldsymbol{y}_k^T \boldsymbol{v}_k + \boldsymbol{y}_k^T \boldsymbol{A}_k \boldsymbol{z}_k$

Drop small entries in $\boldsymbol{z}_k$ and $\boldsymbol{y}_k$

Retain only the p largest entries in $\boldsymbol{z}_k$ and $\boldsymbol{y}_k$

end for

ALGORITHM 13: $AIB(p,\tau)$: Approximate Inverse by Bordering.

In order to save memory usage and retain the computational cost low, the implementation of the BiCG method used with this algorithm has been tuned in order to take advantage of this particular situation, for example, the vectors used are set as sparse vectors, then all the operations like inner products, matrix by vector product and vector updates are performed in sparse-sparse mode.

Notice that a parallel implementation is possible in this situation since all the k steps are independent each other, although the computational cost for all them is not exactly the same as we can see with the two extreme cases $k = 1$ and $k = n$, the first representing a single equation while the second involves the whole matrix $\boldsymbol{A}$.

A major drawback of this method is the fact that it is subjected to convergence of the BiCG method for which it is known to be quite irregular even in favorable situations. This could pollute the stability of the inverse factors obtained once the algorithm is completed. Even more, notice that strictly speaking this methods requires all the sequence matrices $\boldsymbol{A}_k$ being regular; if any of these matrices is singular the preconditioner is not properly defined, although alternative strategies could be implemented in order to overcome such situation.

As usual, a good choice of the dropping parameters τ and p is strongly problem dependent. Moreover, the BiCG algorithm requires by itself the maximum number of iterations allowed and the residual tolerance, even the dual residual tolerance could be required. If we take both tolerances equal it still requires two parameters. As we have seen in the previous chapter, the number of iterations required for a Krylov subspace methods to converge is strongly related with the size of the linear system involved, and since in this case the size of such linear systems is changing for every k it would be more convenient to specify a factor ϕ controlling the number of iterations in each k step as $t_k = \phi k$. since what we are looking for is an approximation values for ϕ much smaller than one would be enough.

Chapter 6

Deflation

This chapter is devoted to the main topic of the present work, a family of techniques we will call *deflation*. The motivation of these techniques is quite similar that those for preconditioning but by different means. Since the convergence rate of Krylov subspace methods is directly and intricately affected by the spectral properties of the coefficient matrix. Therefore preconditioning attempts to improve such spectral properties in an indirect way based on the fact that, if the preconditioner $\boldsymbol{M}$ matches exactly with the coefficient matrix $\boldsymbol{A}$, then any Krylov subspace method will perform a single iteration to deliver the exact solution of the system. The important observation here is that in such ideal, and non practical, situations the preconditioned system is precisely the identity matrix having a condition number equal to one all its spectrum clustered at one. What preconditioning techniques do is in fact an attempt to close the preconditioned system as much as possible to the identity matrix which has an ideally suited spectrum for Krylov subspace methods.

On the other hand, deflation works by trying to directly modify favorably the spectrum of the *deflated linear system*, a concept to be clarified in this chapter, with respect to the spectrum of the original coefficient matrix. While preconditioning works globally and indirectly with the spectrum of the matrix, deflation techniques work directly with some specific members of the spectrum or a portion of it.

Despite early developments of such techniques which took place in the late eighties and early nineties they do not enjoy the unification and common background like preconditioning techniques which in turn have achieved a high maturity level. Clear evidence of this fact is the vast terminology used for these methods. The theoretical, and sometimes heuristic, different points of view from which these kind of techniques are approached is also an indicator of this absence of a general consensus.

Even more, many authors have regarded, and some others still do, deflation techniques as a special technique of preconditioning. It seems that this situation is changing in order to regard deflation techniques as a different strategy not subordinated to another family of techniques although relationships and similarities have been identified and recently studied from the practical and theoretical point of view. We support the later one, for this reason we have devoted an independent chapter for a such technique.

6.1 Evolution of Deflation

The word *deflation* has been used traditionally in the spectral environment, devoted to study and compute the eigenvalues and eigenvectors of a matrix, for long time. Several techniques have been devised in order to improve the performance and accuracy of iterative methods for the solution of the eigenvalue problem, deflation is one of them. Although deflation techniques are always used, in our knowledge, in conjunction with other techniques.

The concepts used in deflation for spectral computations posses a high degree of elegance, being simple but very powerful. Then, it seems like a natural idea to take advantage of them for the problem with which we are concerned. Now we are interested in *deflation for the solution of linear systems.*

Roughly speaking, and in accordance with our current knowledge on this subject, the term *deflation* focused for solving linear systems of equations first appeared in the paper of Nicolaides [96]. In this paper a methodology is developed attempting to accelerate the conjugate gradient method. An example is presented where it is used for the solution of the linear system arising from the discretization of the Poisson equation and from this ground a method very similar to what we call *domain* deflation is explained, which will be discussed in what follows.

After, Guillard and Désidéri presented a paper [67] in which a *spectral preconditioner* is developed and applied to a linear system arising from Chebyshev approximation of a generalized Helmholtz problem in conjunction with the Richardson method. This methodology is completely based on the spectral properties of the continuous operator. It claims to use as much as possible the spectral information from the continuum model in the solution of the discretized one.

In 1996 two papers appeared which explicitly stated they were attempting to remove the negative impact in the convergence rate of Krylov subspace methods of small eigenvaues of the coefficient matrix. In the first of these papers Erhel, Burrage and Pohl [52] the term *preconditioning by deflation* is used to refer to such a technique used in conjunction

with GMRes. The second of these papers was written by Bielawski, Mulyarchik and Popov [18] who regarded their method as an *algebraically reduced system* whose operators may be viewed as the prolongation and restriction operators used in multigrid.

Champan and Saad published in 1997 a paper in which the GMRes method in endowed with an *augmenting* strategy, also with the specific purpose of removing the eigenvalues that hamper the convergence rate of Krylov subspace methods. Clear examples of *deflation vectors* are given and ways of reducing the computational cost invested in computing them are suggested. Additionally, some theoretical considerations are presented supporting more firmly these kinds of techniques. Maybe the most important contribution of this paper, in our particular point of view, is the fact that the method is applied to a large variety of matrices from the Matrix Market collection coming from different applications. This last point is of paramount importance because it demonstrates that the method is suitable for general matrices even if not very much information of the original problem from where they arose is available.

In the same year, Erhel and Guyomarc'h presented in their research report [51] an *augmented subspace* conjugate gradient method with a very clear and easy to follow algorithmic implementation and more importantly an elegant theoretical discussion in terms of matrix polynomials.

A year after, Burrage, Erhel, Pohl and Williams presented an acceleration technique explicitly called *deflation* for the solution of linear systems but more oriented to work together with classical iterative methods.

Finally, in 2000 a paper appeared authored by Saad, Yeung, Erhel and Guyomarc'h. It seems, to our particular criteria, that this paper attempted to unify the practical implementation of the conjugate gradient method given previously in the report of the two latter authors with the elegant theoretical background presented in the paper of Saad. More important to mention is the fact that in both previous papers the term *augmentation* was preferred over *deflation* but from this paper a slight consensus can be noticed about the latter term.

Around the same time a paper by Vuik, Segal and Meijerink [143] appeared in which the title does not refer at all to deflation and this technique is not classified as a special case of another procedure such as preconditioning or augmentation. Moreover they implement this methodology in conjunction or not with incomplete Cholesky preconditioning, segregating both methodologies and providing this family of techniques a proper personality. The method is applied with much success to very specialized problems coming from the oil industry, one that has traditionally supported the development of iterative methods, and suggests the construction of the deflation vectors on physical grounds.

Soon after, in 2001, the same authors together with Wijma presented in [144] a detailed comparative study about the way the deflation vectors are chosen. This paper follows the same aim as the previous, it does not subordinate deflation to other techniques. Also the problems presented all came from the oil industry and are strongly related with those of the previous paper.

Although the same year Vuik in conjunction with Frank presented in [56] a *deflation based preconditioner*. They presented what is now almost uniformly know as *sub-domain* deflation and applied it to linear systems arising from the discretization of the two dimensional Laplacian in square domains. The case of large differences in the coefficients of this Laplacian is analyzed and, very shallowly, the unsymetric case is mentioned.

More advanced applications of deflation techniques began to appear from the mid-two thousands, among those focused on problems in computational fluid dynamics we can mention [6, 25, 86, 123–127, 151]. The first three completely focused on the pressure Poisson equation in which we are particularly interested.

Finally, we mention the paper of Gutknecht [70] which provides an elegant and exhaustive theoretical foundation of *spectral deflation* definitively regarding it as an independent technique not subordinated to any other. Although from a different point of view, Tang, Vuik and coworkers have extensively studied the relationships of deflation with domain decomposition and multigrid methods regarding it as a second level preconditioner [128–130]. A detailed study where applications are also presented can be found in the PhD thesis of Tang [122].

6.2 Hotelling Power Method

The determination of eigenvalues, and possibly their corresponding eigenvectors, is of great importance in scientific computing not only for theoretical reasons. Spectral information is extensively used in several branches of engineering and science; vibration in mechanical devises, natural frequencies in structures subjected to earthquakes or fluid flow stability to name only a few.

As we have seen in chapter 3, determining the eigenvectors of a given matrix is equivalent to solving the characteristic polynomial (3.16). The Abel theorem states that such a solving is impossible for polynomials of order greater than four using formulas employing only arithmetical operations. Then methods for determining the eigenvalues, and eventually also the eigenvectors, of a matrix are in essence iterative. This has motivated one of the most active and solid branches of numerical analysis.

One of the first methods devised for this task is the *power method* which delivers the *dominant* eigenpair λ_n and $\boldsymbol{v}_n$ by successive approximations. In doing so, this method neither access nor modifies individual entries of the matrix $\boldsymbol{A}$, what it needs with the coefficient matrix is only to perform its product with a vector in each iteration. As we have mentioned previously in chapter 4 this method can be regarded as one exploiting information contained in the Krylov subspace.

The main success of the basic implementation of the power method relies on its low cost in terms of operations and storage requirements which apart of the matrix and the required eigenvector are virtually null. Additionally, in plenty of applications we are interested only in the dominant eigenpair delivered by this method. But this situation can pass to be a drawback if more eigenpairs are required. For such a task there have been developed several variants of the basic power method. The *inverse power method* computes the dominant eigenpair of $\boldsymbol{A}^{-1}$ which in turns consists in the reciprocal of the eigenvalue of smallest module in $\sigma(\boldsymbol{A})$ and its corresponding eigenvector. The *shifted power method* takes a value μ and apply the power method to the *shifted system* $\boldsymbol{A}-\mu\boldsymbol{I}$ with the purpose to get the eigenvalue of $\boldsymbol{A}$ closest to μ.

The method in which are now interested have emerged from the necessity to get the $n-1$ eigenpair once the corresponding to the dominant eigenvalue has been computed. Such situation is common in the dynamic analysis of structures where it has been extensively used even in commercial software [10]. Suppose such a dominant eigenpair is already available, we take $n-1$ unit vectors $\boldsymbol{p}_i$, being among each other orthogonal, and define an orthonormal matrix whose first column is the dominant eigenvector as

$$\boldsymbol{P}_1 = [\boldsymbol{v}_n, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_n], \quad \text{with} \quad (\boldsymbol{v}_n, \boldsymbol{p}_i) = 0 \ \ \forall \ \ i = 2, 3, \cdots, n. \tag{6.1}$$

Consider now the deflated matrix

$$\boldsymbol{P}_1 \boldsymbol{A} \boldsymbol{P}_1^T = \begin{bmatrix} \lambda_n & \boldsymbol{0} \\ \boldsymbol{0}^T & \boldsymbol{A}_1 \end{bmatrix}. \tag{6.2}$$

Since $\boldsymbol{P}_1$ is an orthonormal matrix $\boldsymbol{P}_1^T = \boldsymbol{P}_1^{-1}$, the deflated matrix (6.2) is similar to $\boldsymbol{A}$, and thus their spectrum are the same

$$\sigma(\boldsymbol{A}) = \sigma\left(\boldsymbol{P}_1 \boldsymbol{A} \boldsymbol{P}_1^T\right) = \{\lambda_1, \lambda_2, \cdots, \lambda_n\}, \tag{6.3}$$

even what is more important for us is the fact that the spectrum of the $n-1$ principal minor of the deflated matrix is exactly the same as the matrix $\boldsymbol{A}$ with the dominant

eigenvalue removed, that is

$$\sigma\left(\boldsymbol{A}_1\right) = \{\lambda_1, \lambda_2, \cdots, \lambda_{n-1}\}. \tag{6.4}$$

Then, by applying the power method to this $n-1$ principal minor $\boldsymbol{A}_1$ we will get the eigenvalue with second largest module and its corresponding eigenvector. Once the $n-1$ eigenpair has been found we can repeat the process to get the desired number of eigenpairs. The similarity transformation (6.2) is usually referred to as *Hotelling deflation*.

Note that the matrix $\boldsymbol{P}_1$ is not unique. Several matrices can perform such similarity transformation. This gives us the advantage that we can chose it in such a manner that some desirable properties, like sparsity, are retained. But it has some implicitly drawbacks, for instance, the orthonormalization of a set of vectors could be expensive to perform and could be even more difficult if conditions for preserve sparsity are imposed.

6.3 Wielandt Method

Now we turn our attention to a generalization of the Hotelling deflation discussed in the previous section. Only for notational purposes we are going to reverse the ordering of the matrix spectrum by changing the $\leq$ signs by $\geq$ in (3.21), then λ_1 is now the dominant eigenvector.

Suppose again once we have available the dominant eigenpair of the matrix $\boldsymbol{A}$ we wish to obtain the second dominant eigenpair with eigenvalue λ_2. It is possible to displace the eigenvalue λ_1 by applying a rank one modification to the original matrix while keeping the rest of the spectrum unchanged. This rank one modification is chosen so that the eigenvalue λ_2 becomes the one with largest module of the modified matrix.

After applying this rank one modification we obtain the *deflated matrix*

$$\boldsymbol{A}_1 = \boldsymbol{A} - \alpha \boldsymbol{v}_1 \boldsymbol{u}^T, \tag{6.5}$$

where $\boldsymbol{v}_1$ is the dominant eigenvector of $\boldsymbol{A}$, $\boldsymbol{u}$ is an arbitrary vector satisfying $(\boldsymbol{v}_1, \boldsymbol{u}) = 1$ and α is an appropriate shift. We take the product of this deflated matrix by the

dominant eigenvector

$$\begin{aligned} \boldsymbol{A}_1\boldsymbol{v}_1 &= \left(\boldsymbol{A} - \alpha\boldsymbol{v}_1\boldsymbol{u}^T\right)\boldsymbol{v}_1, \\ &= \boldsymbol{A}\boldsymbol{v}_1 - \alpha\boldsymbol{v}_1\boldsymbol{u}^T\boldsymbol{v}_1, \\ &= \boldsymbol{A}\boldsymbol{v}_1 - \alpha\left(\boldsymbol{u}, \boldsymbol{v}_1\right)\boldsymbol{v}_1, \\ &= \lambda_1\boldsymbol{v}_1 - \alpha\boldsymbol{v}_1, \\ &= \left(\lambda_1 - \alpha\right)\boldsymbol{v}_1. \end{aligned} \tag{6.6}$$

This proves that $(\lambda_1 - \alpha)$ is an eigenvalue of the deflated matrix. Now we take the product of its transpose with $\boldsymbol{v}_i$ for any $i = 2, \cdots, n$

$$\begin{aligned} \boldsymbol{A}_1^T\boldsymbol{v}_i &= \left(\boldsymbol{A}^T - \alpha\boldsymbol{u}\boldsymbol{v}_1^T\right)\boldsymbol{v}_i, \\ &= \boldsymbol{A}^T\boldsymbol{v}_i - \alpha\boldsymbol{u}\boldsymbol{v}_1^T\boldsymbol{v}_i, \\ &= \boldsymbol{A}^T\boldsymbol{v}_i - \alpha\left(\boldsymbol{v}_1, \boldsymbol{v}_i\right)\boldsymbol{u}, \\ &= \lambda_i\boldsymbol{v}_i. \end{aligned} \tag{6.7}$$

Where we have used the fact that the eigenvectors are each other orthogonal. This proves that the $n-1$ eigenpairs for $i = 2, \cdots, n$ of the original and the deflated matrices are exactly the same. Therefore, the spectrum of this deflated matrix is given by

$$\sigma\left(\boldsymbol{A}_1\right) = \left\{\lambda_1 - \alpha, \lambda_2, \cdots, \lambda_n\right\}. \tag{6.8}$$

Furthermore, taking $\alpha = \lambda_1$ and $\boldsymbol{u} = \boldsymbol{v}_1$ we get

$$\boldsymbol{A}_1 = \left(\boldsymbol{I} - \boldsymbol{v}_1\boldsymbol{v}_1^T\right)\boldsymbol{A}, \tag{6.9}$$

for which λ_1 has been annihilated. Note that in all this discussion the choice of the dominant eigenvector has been completely arbitrary but it does not imply any sacrifice of generality. Instead of choosing the dominant eigenpair, any other eigenpair could be taken in order to perform such deflation process. At the end, its corresponding eigenvalue will be annihilated in the spectrum of the deflated matrix.

Methods computing several eigenpairs at once have also been developed. Many of them use Krylov subspaces and only matrix-vector products are required. Without going into the details of such methods we only wish to clarify that they usually provides the extreme values of the spectrum which correspond to the smallest and largest eigenvalues. Suppose one of them has been used to get m eigenpairs and we are interested in compute with the method previously used other eigenpairs corresponding to interior eigenvalues.

We can apply (6.9) recursively as

$$\boldsymbol{A}_m = \left(\boldsymbol{I} - \sum_{i=1}^{m} \boldsymbol{v}_i \boldsymbol{v}_i^T \right) \boldsymbol{A} = \boldsymbol{P}\boldsymbol{A}, \tag{6.10}$$

for which the m eigenvalues corresponding to the eigenvectors $\boldsymbol{v}_i$ used have been annihilated. In what follows we shall refer to the matrix $\boldsymbol{P}$ as the *deflator*

$$\boldsymbol{P} = \boldsymbol{I} - \sum_{i=1}^{m} \boldsymbol{v}_i \boldsymbol{v}_i^T. \tag{6.11}$$

Although this method is not widely known in dense eigenproblem computations it has been extensively used when the involved matrix is large and sparse. From the point of view of the practical implementation an important consideration is that we never need to form the deflated matrix explicitly. This is important because in general, even if $\boldsymbol{A}$ is sparse, $\boldsymbol{A}_m$ is dense. Furthermore, in many algorithms for eigenproblem computation, the only operation required is matrix by vector products $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{y}$. Suppose the most basic case with the deflated matrix as in (6.5). First we compute $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{y}$, then the scalar $\varphi = \alpha\,(\boldsymbol{u}, \boldsymbol{y})$ and finally $\boldsymbol{z} = \boldsymbol{z} - \varphi \boldsymbol{v}_1$.

6.4 Spectral Deflation

From the Wielandt method we have devised a technique that given a matrix $\boldsymbol{A}$ and m of its eigenpairs it annihilates the m corresponding eigenvalues after premultiplying by the deflator $\boldsymbol{P}$ without explicitly forming the deflated matrix $\boldsymbol{A}_m$. It is well known that the convergence rate of Krylov subspace methods depends directly on the condition number of the coefficient matrix. Then it seems a natural and obvious idea to annihilate the extreme eigenvalues by deflation. It has been observed that are the smallest eigenvalues those having a worse effect in the convergence behavior of iterative methods.

In order to obtain an alternative expression for the deflator $\boldsymbol{P}$ more suitable for its application in the context of Krylov subspace methods let us define the matrix $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ whose columns are the m eigenvectors. Also define the *coarse matrix* $\hat{\boldsymbol{A}} = \boldsymbol{W}^T \boldsymbol{A}\, \boldsymbol{W}$. Then the *deflation matrix* can also be written as

$$\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{A}\, \boldsymbol{W}\, \hat{\boldsymbol{A}}^{-1}\, \boldsymbol{W}^T. \tag{6.12}$$

In order to verify the equivalence between (6.11) and (6.12) we write $\boldsymbol{W}$ by columns

$$\begin{aligned}
\boldsymbol{P} &= \boldsymbol{I} - \boldsymbol{A}\left[\boldsymbol{v}_1 \cdots \boldsymbol{v}_m\right]\left[\begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_m^T \end{bmatrix} \boldsymbol{A}\left[\boldsymbol{v}_1 \cdots \boldsymbol{v}_m\right]\right]^{-1}\begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_m^T \end{bmatrix}, \\
&= \boldsymbol{I} - \left[\boldsymbol{A}\boldsymbol{v}_1 \cdots \boldsymbol{A}\boldsymbol{v}_m\right]\begin{bmatrix} \boldsymbol{v}_1^T\boldsymbol{A}\boldsymbol{v}_1 & \cdots & \boldsymbol{v}_1^T\boldsymbol{A}\boldsymbol{v}_m \\ \vdots & \ddots & \vdots \\ \boldsymbol{v}_m^T\boldsymbol{A}\boldsymbol{v}_1 & \cdots & \boldsymbol{v}_m^T\boldsymbol{A}\boldsymbol{v}_m \end{bmatrix}^{-1}\begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_m^T \end{bmatrix}, \\
&= \boldsymbol{I} - \left[\lambda_1\boldsymbol{v}_1 \cdots \lambda_m\boldsymbol{v}_m\right]\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_m \end{bmatrix}^{-1}\begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_m^T \end{bmatrix}, \\
&= \boldsymbol{I} - \left[\boldsymbol{v}_1 \cdots \boldsymbol{v}_m\right]\begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_m^T \end{bmatrix}, \\
&= \boldsymbol{I} - \textstyle\sum_{i=1}^{m} \boldsymbol{v}_i\boldsymbol{v}_i^T.
\end{aligned} \tag{6.13}$$

Now, it is easy to see that we can apply any Krylov subspace method to the *deflated linear system*

$$\boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{P}\boldsymbol{b}, \tag{6.14}$$

from this and the previous comments it is clear that we do not need to form explicitly the deflated matrix. The effect of deflation is obtained by forming the product of $\mathbf{P}$ with a vector exactly in the same way as preconditioning.

6.5 Domain Deflation

Although several efficient methods for the eigenproblem have been developed over the years, the computation of eigenpairs is usually a very expensive task, specially those in the leftmost extreme of the spectrum. In addition, if get the desired eigenvalues with the only purpose of construct the deflator, its speed up time could be prohibitive.

Several techniques take advantage of the information provided by the orthogonalization process when a long term recurrence method like $FOM(k)$ of $GMRes(k)$ is used. At the end, the underlying orthoormalization algorithm on which all this methods rely was originally proposed for the approximation of eigenvalues. Suppose an iteration of the $FOM(k)$ method have been run. We compute the eigenvectors of the upper Hessenberg matrix $\boldsymbol{H}_k$ denoted by $\boldsymbol{z}_i$. The approximation to m eigenvectors $\boldsymbol{v}_i$ of the original

coefficient matrix is given by

$$\boldsymbol{v}_i = \boldsymbol{V}\boldsymbol{z}_i. \tag{6.15}$$

The vectors $\boldsymbol{z}_i$ are the *Ritz vectors* while its corresponding eigenvalues θ_i are the *Ritz values*. This is in fact, the basis a one of the most successful family of methods for computing eigenvalues and eigenvectors of large and sparse matrices. They are known generically as *Implicitly Restarted Arnoldi Methods*.

This alternative has the disadvantage that it can only be used when a short recurrence method is employed to solve the linear system. In addition, the deflation is not applied in the first iteration and formally speaking we are changing to a different system inside the iteration. The price to pay is that a correction at the end is required as done with *flexible* preconditioning.

A cheap alternative to overcome these problems consist in going back to the origins of this technique when it was proposed for the solution of linear systems. Nicolades have proposed in his paper to divide the domain in which the partial differential equation, which after discretization, provides the linear system we are interested to solve, in m several disjoint subdomains $\Omega_i \cap \Omega_j =$ for $i \neq j$. All they together forming the original domain $\bigcup_{i=1}^{m} \Omega_i = \Omega$ [96]. After discretization, all the points where the unknown variable of the original partial differential equation belong to one and only one subdomain. Being x_i the unknown associated tho point i. We take m vectors, each one corresponding to one subdomain. Their entries are given by

$$w_i^{(j)} = \begin{cases} 1, & \text{if point } i \text{ belongs to } \Omega_j \\ 0, & \text{Otherwise.} \end{cases} \tag{6.16}$$

Although Nicolaides did not present numerical results with such a strategy it has been reported in the literature it has some considerable degree of success [6, 56]. Several other variants oriented in this manner have been proposed. Many of them inspired by the physics of the underlying problem. In our knowledge one of the most studied problems is the case in which the coefficients of the partial differential equation vary drastically in space [140, 143, 144]. An inherent characteristic of layered groundwater reservoirs in which the flow of water is modeled by the Darcy equation. Other example in which this approach have been applied successfully is the numerical simulation of bubbly flows [127].

When no information about the coefficients of the partial differential equation yielding the linear system with must solve a similar strategy can be obtained by setting the first q

entries of the first deflation vector w_1 equal to one and zero otherwise, being $q = \lfloor n/m \rfloor$:

$$w_j^{(1)} = \begin{cases} 1, & \text{for } \; j = 1, 2, ..., q \\ 0, & \text{for } \; j = q+1, q+2, ..., n. \end{cases} \tag{6.17}$$

The second deflation vector is defined as follows

$$w_j^{(2)} = \begin{cases} 0, & \text{for } \; j = 1, 2, ..., q \\ 1, & \text{for } \; j = q+1, q+2, ..., 2q \\ 0, & \text{for } \; j = 2q+1, 2q+2, ..., n. \end{cases} \tag{6.18}$$

The definition of the $m - 2$ remaining deflation vectors is straightforward.

In the present work we have used the last approach. It is important to mention that these options for choosing the deflation vectors in $\boldsymbol{W}$ are in some way related to domain decomposition and multigrid methods. Detailed discussion, analysis and comparison from the theoretical and experimental points of view can be found in [128, 129].

6.6 Implementation

As we have seen and stressed in the previous sections, explicit computation of the deflated system is never formed explicitly. Consider the case when eigenvectors, or some approximation, are used as deflation vectors. Being these deflator vectors dense we have not many reasons, if any, to expect the deflator $\boldsymbol{P}$ being sparse. With this in mind would be completely naive to expect sparsity in the deflated matrix $\boldsymbol{P}\boldsymbol{A}$.

Furthermore, note from 6.12 that the deflator application requires the inverse of the coarse matrix $\hat{\boldsymbol{A}}$ given by

$$\hat{\boldsymbol{A}} = \boldsymbol{W}^T \boldsymbol{A}\, \boldsymbol{W}. \tag{6.19}$$

Taking the column partition of the matrix containing the deflation vectors will be very elucidating to see the compuatational cost of such *coarsening* operation

$$\begin{aligned}\hat{\boldsymbol{A}} &= \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_m^T \end{bmatrix} \boldsymbol{A} \left[\boldsymbol{w}_1 \cdots \boldsymbol{w}_m\right], \\ &= \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_m^T \end{bmatrix} \left[\ \boldsymbol{A}\boldsymbol{w}_1 \cdots \boldsymbol{A}\boldsymbol{w}_m\right], \\ &= \begin{bmatrix} \boldsymbol{w}_1^T\boldsymbol{A}\boldsymbol{w}_1 & \cdots & \boldsymbol{w}_1^T\boldsymbol{A}\boldsymbol{w}_m \\ \vdots & \ddots & \vdots \\ \boldsymbol{w}_m^T\boldsymbol{A}\boldsymbol{w}_1 & \cdots & \boldsymbol{w}_m^T\boldsymbol{A}\boldsymbol{w}_m \end{bmatrix}.\end{aligned} \tag{6.20}$$

As we can see, obtaining such coarse matrix requires the evaluation of m matrix by vector products and m^2 inner products. When the coefficient matrix is symmetric the amount of inner products is reduced to $m(m+1)/2$. Note that any of the parameters used in such process change as the iterations run. Then we need to perform such coarsening only one time.

For this reason we have segregated the deflation process in two stages, one devoted to *compute* the deflator and the second to *apply* it. Note that the first one is independent of any particular Krylov subspace implementation, or even other kind of methods, while the second must be included inside those iterative methods.

In what follows we shall assume the deflation vectors are available when we start the *computing* phase, although several comments will be done when they are convenient in order to clarify some fine detals about the implementation. After we will back again in describing how the deflation vectors in the framework of spectral deflation have been obtained in the present work.

From (6.20) we have obtained a rough estimative of the operation count required for the coarsening process. But more important, it exemplifies the general procedure in which it is actually carried out. First the m matrix by vector products are evaluated, then we compute the m^2 inner products. Once the coarse matrix $\hat{\boldsymbol{A}}$ is evaluated we proceed to factorize it instead of explicitly compute its inverse. For such a task several options and observations can be done, we list them as follows

1. If the matrix $\boldsymbol{A}$ is symmetric positive definite we compute its Cholesky factorization, alternatively the root free and diagonal pivoted versions have been implemented.

2. If the matrix $\boldsymbol{A}$ is symmetric but positive semidefinite we compute its LDL^T factorization, the diagonal pivoted version has been implemented.

3. If the matrix $\boldsymbol{A}$ is symmetric but indefinete or if we known, or at least suspect, that orthogonality of the columns of $\boldsymbol{A}$ is not well conditioned, we compute the Bunch-Kaufmann factorization where two by two block pivots are allowed.

4. If the matrix $\boldsymbol{A}$ is unsymmetric and strongly diagonal dominant, LU factorization without pivoting is applied.

5. If the matrix $\boldsymbol{A}$ is unsymmetric but not diagonal dominant, or possibly in a weakly manner, an adaptive strategy has been implemented. LU factorization with partial pivoting is run with monitoring of the *grown factor* ρ. If such ρ increases above a predefined tolerance we switch to a rook pivoting strategy. If the grown factor increases even more a last switch to total pivoting is done. At each switch it is possible to continue the factorization from the current stage or recompute it from scratch.

Once the deflation vectors and an adequate factorization of the coarse matrix are ready we are in position to apply deflation together with any Krylov subspace method. As done with preconditioning, what we need is to compute the product of the projector $\boldsymbol{P}$ with a given vector. This relies on the fact that Krylov subspace methods only need to perform matrix by vector products and because we are now interested in solving a system with the deflated matrix $\boldsymbol{PA}$ as the coefficient matrix. Therefore products of the form $\boldsymbol{z} = \boldsymbol{PAy}$ are now required. As done with preconditioning, we segregate this double product in two steps as follows

$$\boldsymbol{w} = \boldsymbol{Ay} \quad \text{and} \quad \boldsymbol{z} = \boldsymbol{Pw}. \tag{6.21}$$

But remember again that the deflator $\boldsymbol{P}$ is not explicitly formed. Its product with a vector can be preformed as

$$\begin{aligned} \boldsymbol{Pw} &= \left[\boldsymbol{I} - \boldsymbol{A}\boldsymbol{W}\hat{\boldsymbol{A}}^{-1}\boldsymbol{W}^T\right]\boldsymbol{w}, \\ &= \boldsymbol{w} - \boldsymbol{A}\boldsymbol{W}\hat{\boldsymbol{A}}^{-1}\boldsymbol{W}^T\boldsymbol{w}, \\ &= \boldsymbol{w} - \boldsymbol{A}\boldsymbol{W}\hat{\boldsymbol{A}}^{-1}\boldsymbol{d}_0, \\ &= \boldsymbol{w} - \boldsymbol{A}\boldsymbol{W}\boldsymbol{d}, \\ &= \boldsymbol{w} - \boldsymbol{Ap}, \\ &= \boldsymbol{w} - \boldsymbol{t}. \end{aligned} \tag{6.22}$$

Overwriting could be used in the solution of the, presumably small, coarse linear system. Then the same storage used for $\boldsymbol{d}$ can be used for $\boldsymbol{d}_0$.

Note that passing from the second to the third lines, m inner products are computed. Then a forward and backward substitution is done for passing to the fourth line. Going to the fifth requires m scalar by vector products. Finally, a matrix by vector product and a vector addition are needed to complete the process.

Of course the operation $\boldsymbol{P}^T\boldsymbol{w}$ for methods requiring the matrix transpose by vector product is perfectly defined and can be easily derived in a similar way that (6.22).

It seems convenient now at closing the present chapter describe in some detail the methods used for the computation of the deflation vectors. Of course we refer to the spectral deflation case since the domain deflation case have been clearly stated in the previous subsection and because there is not precisely a computation involved. Although we must take into account that domain deflation vectors are by definition sparse, a fact exploited in our current implementations. Furthermore, the coarse matrix is banded, its bandwidth depends on the connectivity of the subdomains. Although this feature has been sacrified for the sake numerical stability in the matrix factorization, due to inherent difficulties in the implementation of pivoting techniques with a matrix storage format different to the explicit storage of a dense matrix. This prioritization has been done having in mind that the number of subdomains m would be much smaller that the dimension of the linear system.

For the computation of the eigenvalues used for the spectral deflation we have an implementation of the *Implicitly Restarted Arnoldi Method*, which is nowadays deserved as one of the most reliable procedures for the eigenproblem calculations. Generally speaking it takes a vector, makes it unit and run ℓ steps of the Arnoldi algorithm. Then it proceed by computing the eigenvalues of the upper Hessenberg matrix, that is, the Ritz values. This is done with the QR method without shifts but an *aggressive early deflation* technique using a *neighbor wise* criterion as described in [81]. We found such a technique much faster that the shifted QR algorithm at the cost of some accuracy. Although the application we are concerned with does not require a high degree of such accuracy in the computation of the eigenvectors.

It is important to mention that plenty of difficulties have been encountered in such an implementation when applied to unsymmetric matrices apart the need of work in complex arithmetic. We observed that in the symmetric case, which reduces to the symmetric Lanczos algorithm, not very much difficulties were found. Maybe the most important one has been the lack of othogonality among the Lanczos vectors. This situation can be remedied using complete or selective re-orthogonalization. We have

choose the latter one due to its lower computational cost, as usual sacrificing some degree of accuracy that in the present application does not have a great negative impact.

Chapter 7

Applications

In this chapter we describe the most relevant and interesting cases of the large amount of numerical experiments performed during this work. The selection of these examples has been done by trying to emphasize as much as possible the pros an cons of the methods previously described and implemented. In doing this selection, information of other problems have been taken also into account. Therefore we claim this selection is a good representative. Putting in a document all the available information would be virtually impossible due to space reasons.

We have tried also to present the information we are interested as much as possible in graphical form. Doing so we are not following the traditional way in which it is done in the iterative methods literature. The purpose of that is to avoid tables plagued by numbers which are difficult to follow, specially when a large amount of information is available as in the present case. Thus we hope this presentation being more friendly with the reader.

We have implemented all the methods described previously in this work in a function oriented fashion. The purpose of use only implemented code by ourselves has been mainly to get an objective comparison. This has the added value that we are now able to modify and possibly improve such algorithms.

The elected programming language has been Matlab, due to its easy implementation and flexibility in the development stage. Although all the algorithms have ben coded in a way easy to translate to Fortran, minimizing or avoiding completely the use of intrinsic and closed Matlab functions.

All the test were run in a personal computer DELL Optiplex 980 with 8Gb of RAM and eight Intel 2.8GHz Intel Core processor. Although any parallelism has been implemented in our algorithms that are essentially sequential.

For all the matrices we have taken as the right hand side and artificially created vector. It has been computed as follows. First a vector with random entries is created, having zero mean and deviation of one. This vector is multiplied by the coefficient matrix and the result is normalized. Finally we take this vector as the right hand side.

Then the *true* solution is obtained by using the backslash command of Matlab. Is this vector the one used to compute the error of the iteration when Krylov subspace methods run. For all the test we have set a stooping criterion based on norm of the residual. The tolerance has been set as 10^{-6}. The maximum number of iterations allowed has been set, for symmetric matrices as one an a half times the size of the linear system and for unsymmetric matrices as twice the same number.

As we have seen in chapter 5 many of the preconditioners requires as input one or more parameters. It is well known that the selection of such values is strongly problem dependent. Nevertheless we have used generic values trying to match them with the most widely used in the literature. But we claim that tuning the preconditioners for a particular application will improve their performance.

Something similar can be said about deflation. For symmetric matrices we have chose the number of deflation vectors someway arbitrary while we were performing our numerical experiments. For unsymmetric matrices we have always chose the number of delation vectors as a function of the size of the linear system. Until a maximum of a tenth of this size. For both cases spectral and domain deflation have been tested.

7.1 Symmetric Matrices

The matrix we are mainly interested is those arising from the discretization of the Poisson pressure equation which in turns is symmetric. For this kind of matrices we have tested only two preconditioning techniques, namely Jacobi and incomplete Cholesky factorization with no fill in.

Additionally, two renumbering strategies has been tested for this kind of matrices. The first one is the well known Reverse Cuthill-McKee method which is a bandwidth reducing method. The second is the the also well known Minimum Degree algorithm which in turns is a fill in reducing method.

7.1.1 Two Dimensional Laplacian

The first of the examples we will show is basically a toy problem which does not represent almost any difficulty for its solution by Krylov Subspace methods. It has been tested

i	λ_i
1	0.04467669509947992
2	0.11119273597746462
3	0.11119273597747378
4	0.17770877685544381
5	0.22040061174490241
6	0.22040061174490935

TABLE 7.1: The six smallest eigenvalues of 20×20 Laplacian matrix.

and included here with the only purpose of validation since we have reproduced exactly the parameters used by Champan and Saad in [26].

It consist in the discretization of the Laplace equation in a square domain using twenty cells in each direction by a five point stencil finite difference method. This lead us a pentadiagonal symmetric matrix with size $n = 400$.

We have solved such a matrix with the conjugate Gradient method in accordance with the mentioned paper. No prenconditioning is used and only spectral deflation with $1, 2, 3, 4, 5$ and 6 deflation vectors have been tested.

As have done Champan and Saad, we show the six smallest eigenvalues of the matrix at hand in table 7.1.1.

In figure (7.1) we show the convergence curves of the Conjugate Gradient using spectral deflation. The case when zero deflation vectors are used correspond to the classical Conjugate Gradient method without deflation and shown with a black line.

Note that the iterations corresponding to the cases when one and two deflation vectors are used are practically identical. This is due to the fact that, at leas in finite precision arithmetic, the coefficient matrix is defective. From the table 7.1.1 we can see that the second and third eigenvalues are equal up to thirteen figures. In terms of the iterations this difference is completely unnoticeable.

A similar effect happens when four and five deflation vectors are used, but this time it takes place almost at the end of the iteration. This is due the fifth and sixth eigenvectors are almost equal, in this case within one more significant figure.

The results obtained with our implementation are in completely agree with those presented in the mentioned paper. Furthermore the convergence of the iterative method have been significantly accelerated fulfilling our purposes. This gives us the insight that our implementation works as expected.

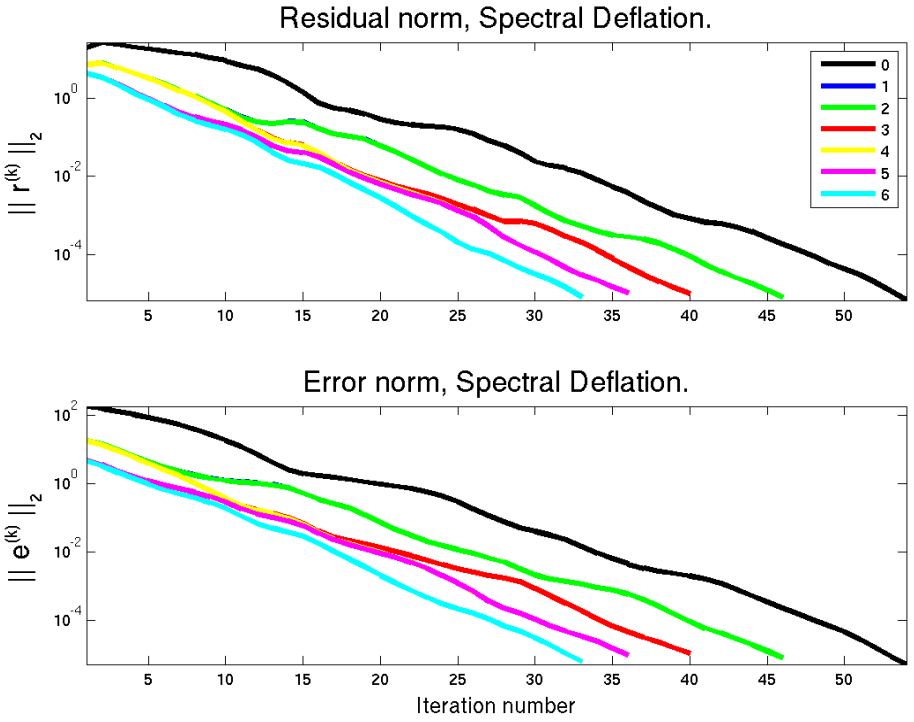

FIGURE 7.1: Conjugate Gradient convergence curves for two dimensional 20×20 Laplacian using the eigenvectors orresponding to the $0, 1, 2, 3, 4, 5$ and 6 smallest eigenvalues as deflation vectors.

7.1.2 Matrix Market Possion Pressure

The second example we present corresponds to the *PressPoisson* matrix taken from the University of Florida sparse matrix collection collection available on the internet. It comes from the discretization of the Poisson pressure of a back step flow using finite elements. It size is $n = 14,822$ and its lower part contains $\eta = 715,804$ non zeros. Its condition number is $\kappa = 2.03665 \times 10^6$, then it is regarded as not very ill-conditioned.

In figure 7.2 we show it original sparsity pattern together those corresponding to the minimum degree and reverse Cuthill-McKee reordering. Figure 7.3 displays the decimal logarithm of its ordered eigenvalues. Notice the sharpness in the left extreme of the plot indicating that smallest eigenvalues are separated far away from the bulk of the spectrum. From this we can conclude that taking the eigenvectors corresponding to the smallest eigenvectors will be more efficient that taking those corresponding to the largest ones, since after deflation the reduction of the condition number will be better.

Figures 7.4 shows the number of iterations performed in order to obtain the required reduction in the norm of the residual using the original, minimum degree and reverse Cuthill-McKee orderings for the unprecnditioned and preconditioned conjugate gradient

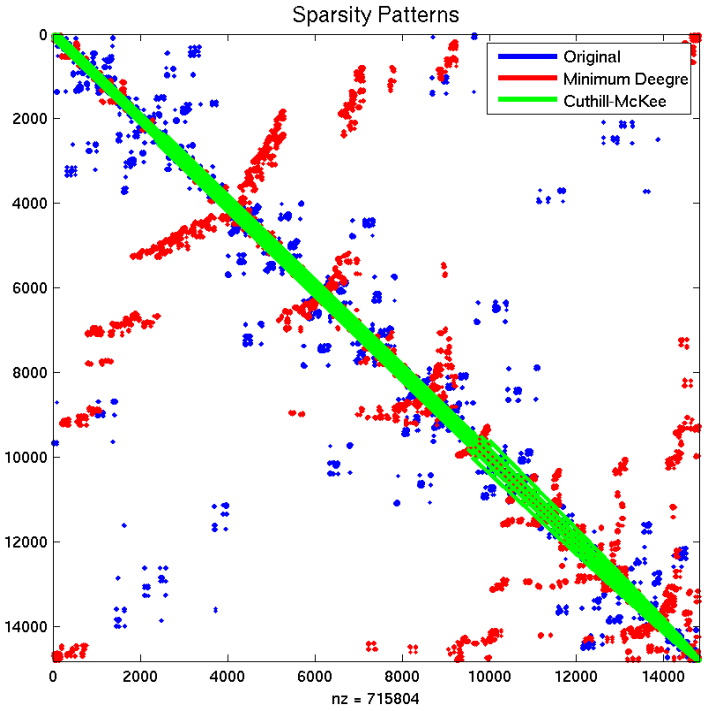

FIGURE 7.2: Sparsity pattern of Pres_Poisson with 14, 822 unknowns.

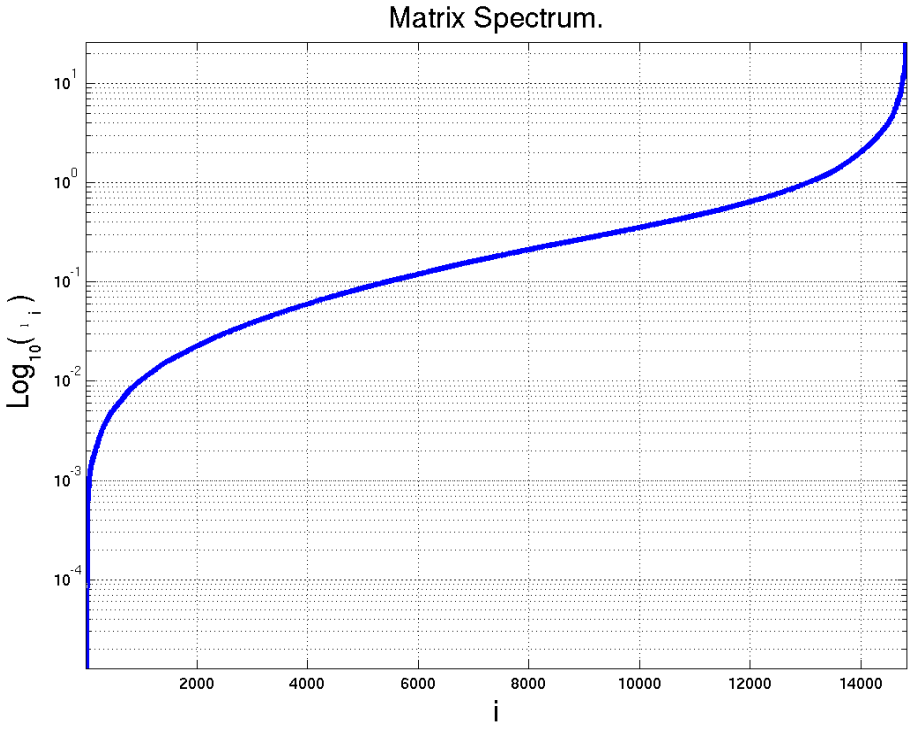

FIGURE 7.3: Logarithm of the spectrum for Pres_Poisson with 14, 822 unknowns.

iterations. Figure 7.5 shows their corresponding convergence curves. Note that, as expected the unpreconditioned and Jacobi preconditioned iterations are ordering invariant. For this particular problem reverse Cuthill-McKee outperforms the original reordering when the incomplete Choleski preconditioner were used.

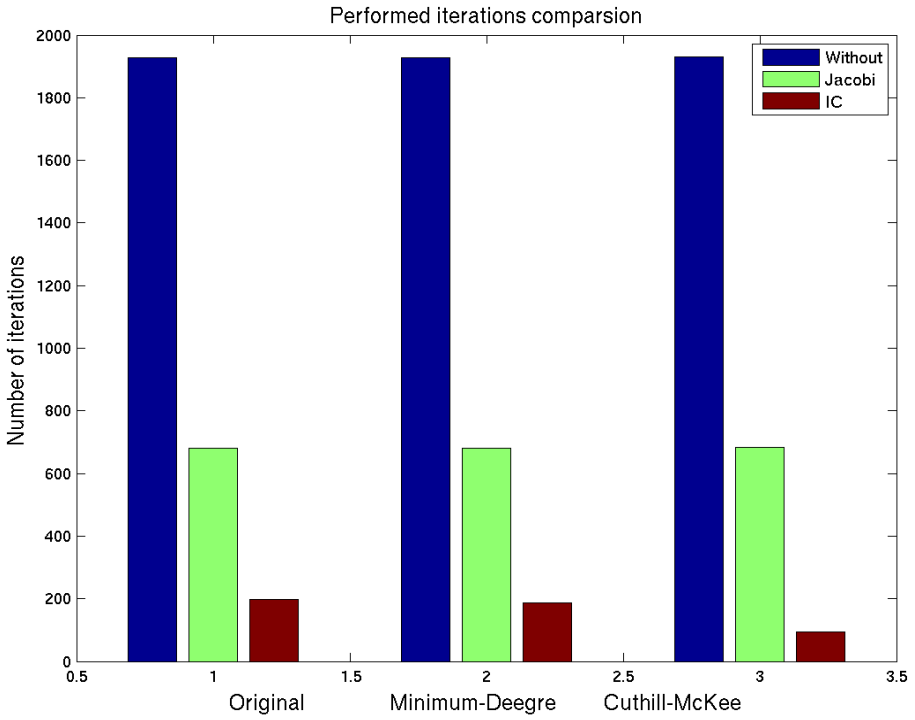

FIGURE 7.4: Number of performed Conjugate Gradient iterations for solving the Pres_Poisson with 14,822 unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky.

Figures 7.6 and 7.7 present the analogous information when, additionally to preconditioning and reordering, spectral deflation hes been used employing $m = 500$ spectral deflation vectors. Note that for the three preconditioning options employed the original ordering always outperforms the other two reordering. Again the incomplete Cholesky preconditioner fairly outperforms the Jacobi preconditioner and, as clearly expected, the unpreconditioned iteration.

Finally, in figure 7.8 we present the convergence curves corresponding to the incomplete Cholesky preconditioned iterations using $0, 20, 40, 60, 80, 100$ and 120 deflation vectors. Note that with the only exception when taking 100 deflation vectors, the number of iterations is always reduced.

Figure 7.9 shows the overall elapsed time in the whole solution process with an apportionment for each stage. The color key means Computing Preconditioner (CP), Computing

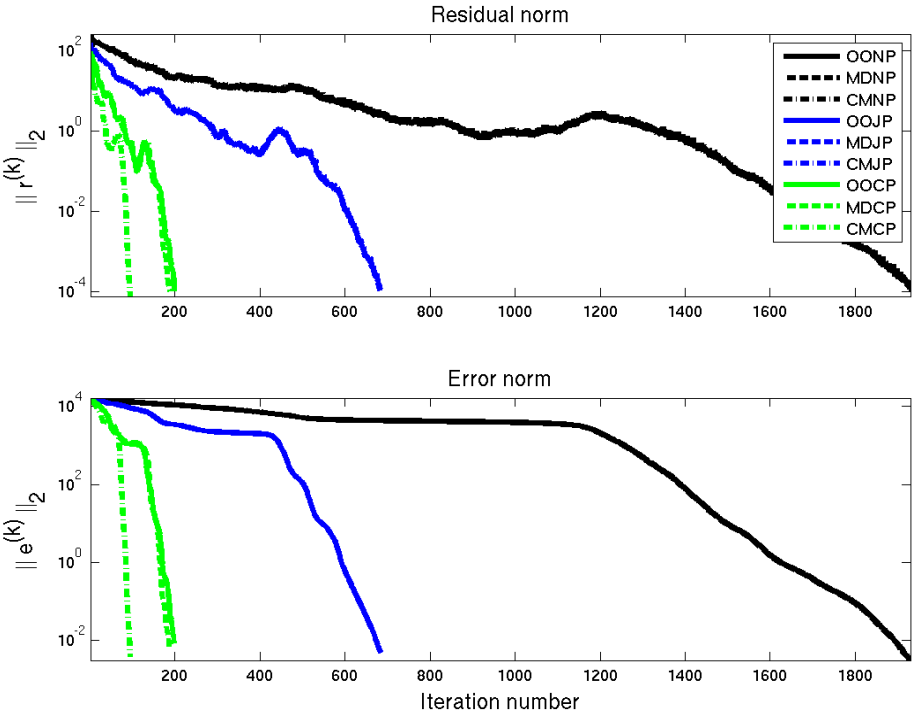

FIGURE 7.5: Conjugate Gradient convergence curves for solving the Pres_Poisson with 14, 822 unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky.

Deflator (CD) Iterative Solver (IS), Apply Preconditioner (AP) and Apply Deflation (AD).

It is very important to notice that despite the elapsed time by the iterative solver reduces as more deflation vectors are used the elapsed time for the overall computation increases due to the cost of computing and applying the deflator. Notice also that even when comparing only the times when deflation is used a minimum is attained in the middle of the plot.

7.1.3 ISIS Poisson Pressure

The third example we present in concluding for symmetric matrices is a matrix that comes from the discretization of the Poisson pressure equation in a driven cavity flow using finite volumes with the *ISIS* software. In turns, this is a commercial software created and maintained at École Central de Nantes. Then we pretend that the study developed during the present work will be incorporated in such a package.

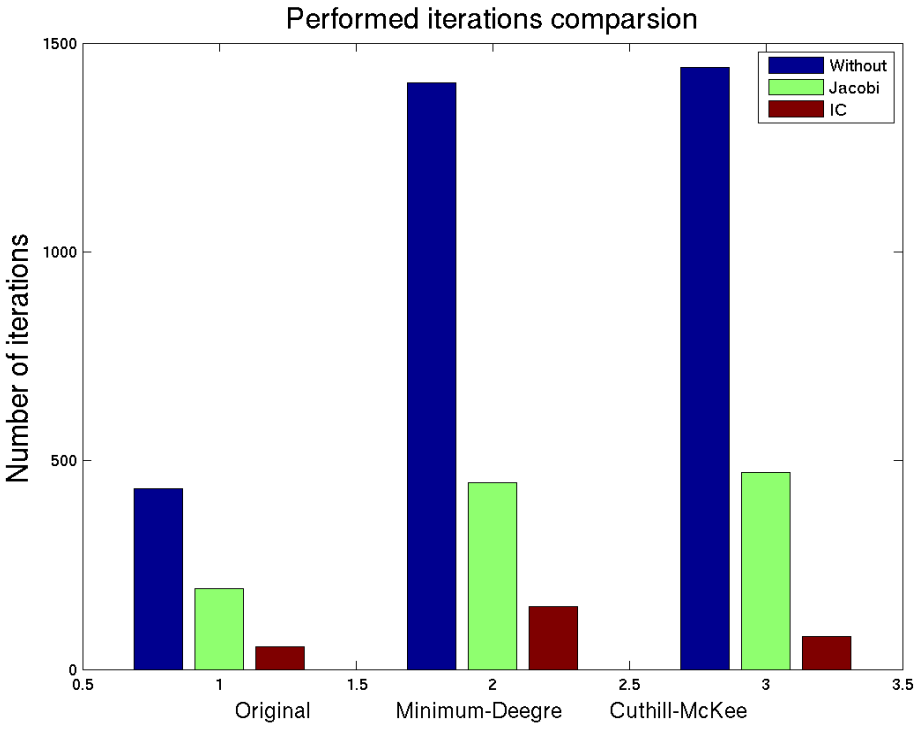

FIGURE 7.6: Number of performed Conjugate Gradient iterations for solving the Pres_Poisson with $14,822$ unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky; together with spectral deflation, $p = 500$.

The matrix has dimension $n = 6,400$. Its condition number is in the order $\kappa = \mathcal{O}(14)$ which represents a challenging problem being considered ill-conditioned. It original sparsity pattern is shown in figure 7.10 together the other two used reorderings.

Figure 7.11 shows the decimal logarithm of its ordered spectrum. Note that in this case the observation made for the previous matrix in this regard is even more pronounced. The leftmost part of the spectrum is even more segregated from the bulk of the spectrum being almost flat in the opposite extreme.

Figure 7.12 shows the number of iterations performed when only preconditioning is used for the three orderings. Note that for this matrix Jacobi preconditioning does not represent a significant improvement respect the unpreconditioned iteration. This is because the matrix at hand is far from being diagonal dominant.

The incomplete Cholesky preconditioner performs satisfactorily but its application to the original ordering clearly outperforms the corresponding to minimum degree. There is not noticeable difference between the reverse Cuthill-Mckee and the original reordering when this preconditioner is used.

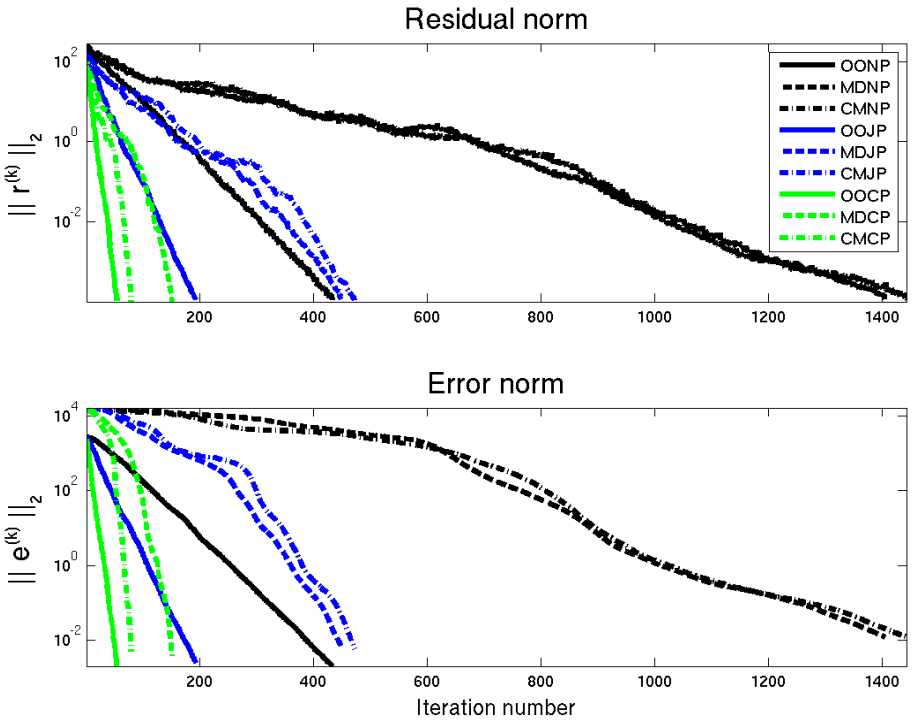

FIGURE 7.7: Conjugate Gradient convergence curves for solving the Pres_Poisson with 14,822 unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky; together with spectral deflation, $p = 500$.

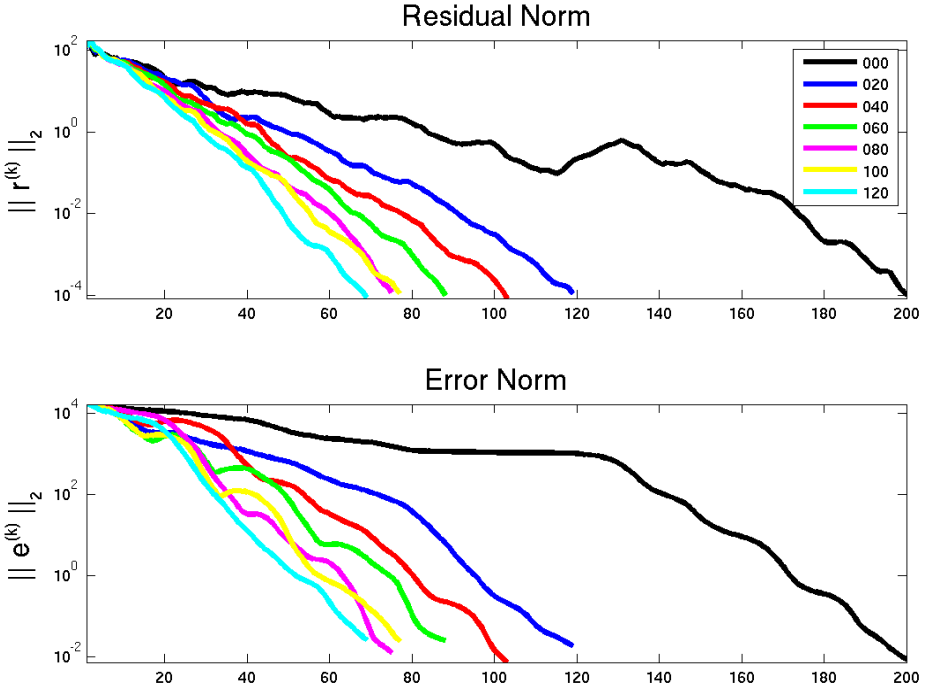

FIGURE 7.8: Convergence curves for $0, 20, 40, 60, 80, 100$ and 120 domain deflation vectors using Incomplete Cholesky preconditioning.

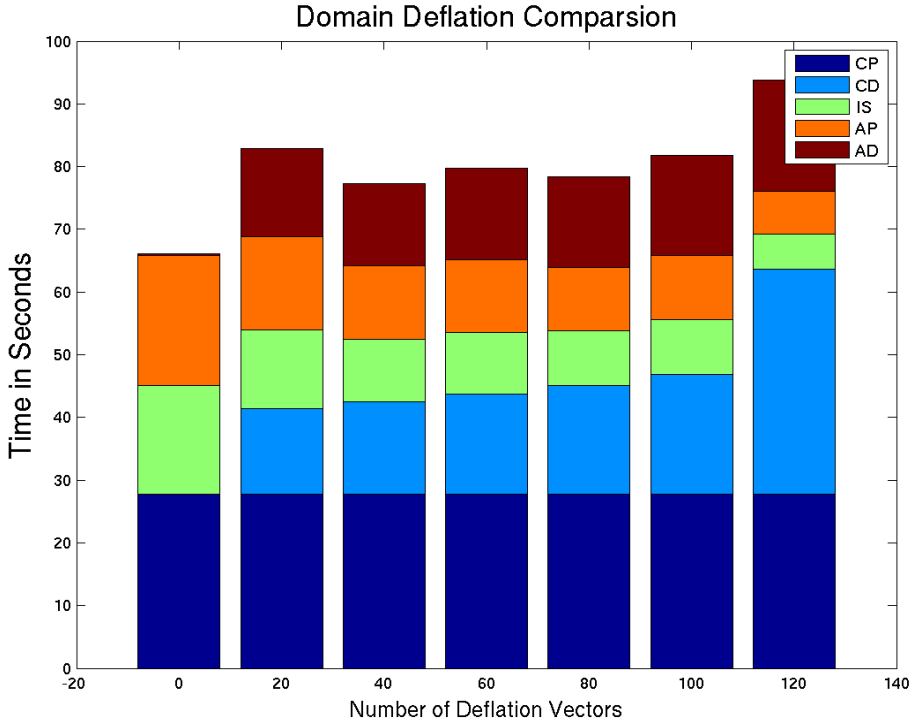

FIGURE 7.9: Elapsed time for $0, 20, 40, 60, 80, 100$ and 120 domain deflation vectors using Incomplete Cholesky preconditioning.

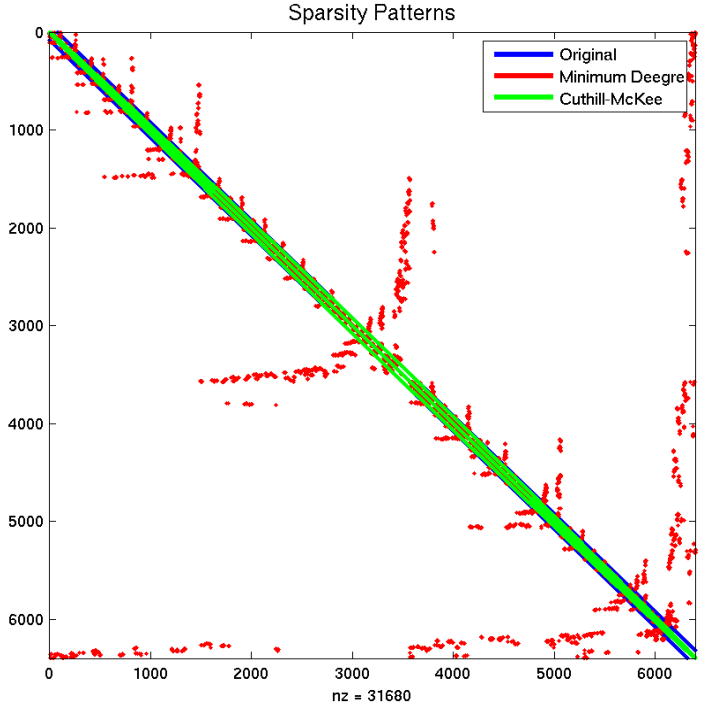

FIGURE 7.10: Sparsity pattern of ISIS Pressure Poisson system $6,400$ unknowns.

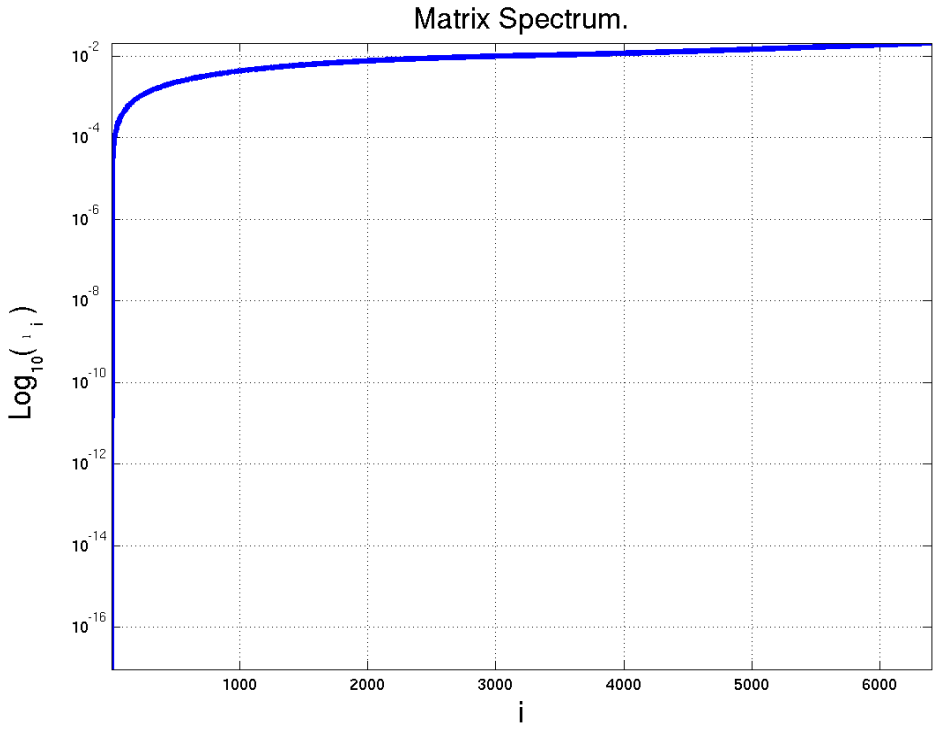

FIGURE 7.11: Logarithm of the spectrum for ISIS Pressure Poisson system $6,400$ unknowns.

Figure 7.13 shows the corresponding convergence curves. It is very important to note the flatness in the error curves even when a reduction in the residual norm curves is observed. This is a typical pathology of ill-conditioned systems.

Figures 7.14 and 7.15 show the corresponding results when in addition to reordering and preconditioning, deflation has been also applied. In this case we have used $m = 640$ spectral deflation vectors. In this case Jacobi preconditioning worse the performance compared with the unpreconditioned iteration. Again incomplete Cholesky has proven its effectiveness. Note the impressive reduction in the number of iterations. The number of iterations goes from 264 for the unpreconditioned and undeflated version to only 8 when incomplete Cholesky and spectral deflation is used.

Although, notice that the flatness in the error norm curve is worst that for the undeflated versions, indicating that annihilation of eigenvalues is causing an effect similar to make singular the deflated matrix.

Finally, figure 7.16 shows the convergence curves corresponding to take different numbers of deflation vectors, all using incomplete Cholesky preconditioning. Note that the number of iterations is always decresing with an stagnation when $m = 50$ is reached.

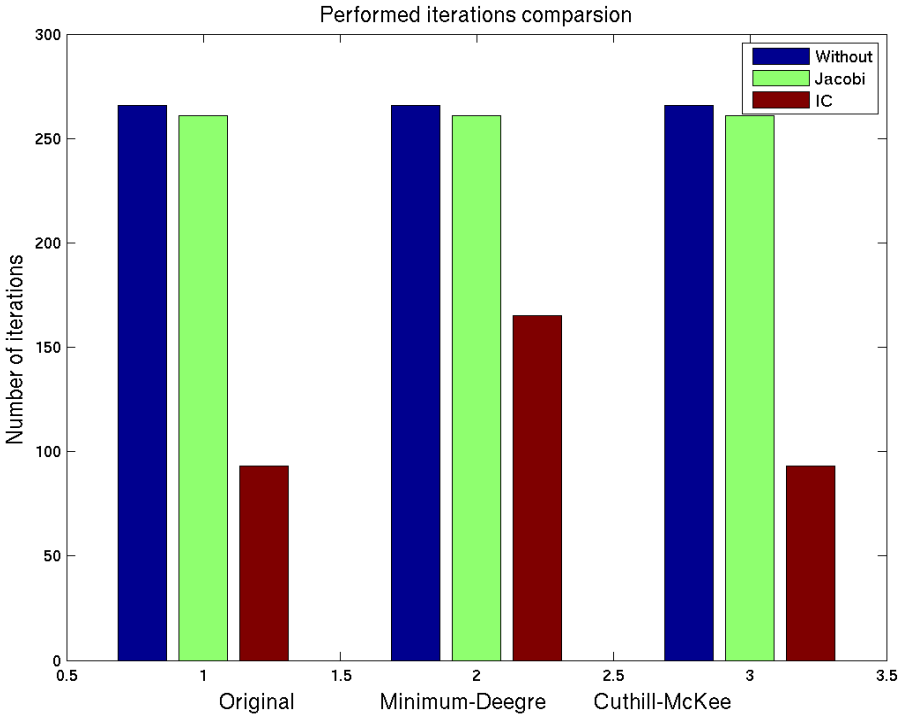

FIGURE 7.12: Number of performed Conjugate Gradient iterations for solving the ISIS Pressure Poisson system with $6,400$ unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky.

The flatness of the error norm curves is also explained with the same argument yet mentioned.

Figure 7.17 shows the elapsed time apportionment stage by stage. Notice that in this case the increasing of the time spent in computing the deflator fairly overcome any saving in the iterative solving.

In concluding the results for the matrix at hand we wish to add that when domain delfation has been used the reduction of the number of iterations is significantly inferior as those obtained with spectral deflation. This effect increses the overall time even more dramatically than in figure 7.17.

7.2 Unsymmetric Matrices

As stated in the previous section, our main interest when starting the present work was focused in the Poisson pressure equation which is essentially symmetric. Although, during developing the present work we became to be interested also in the solution of unsymmetric linear systems. In fact the most intensive development for preconditionig has been done for such unsymetric matrices.

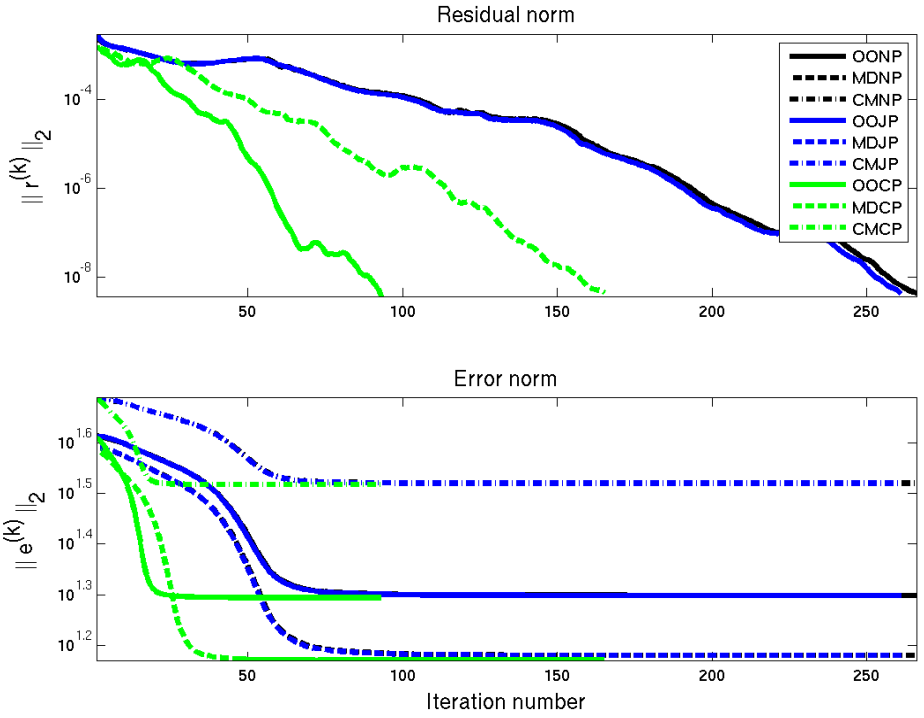

FIGURE 7.13: Conjugate Gradient convergence curves for solving the ISIS Pressure Poisson system with 6,400 unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky.

This interest have been motivated mainly by two reasons. The first one is the fact that in the ISIS software also the solution of unsymmetric linear systems is required. The second is due the situation of deflation methods for unsymmetric linear systems. As far as we know, there is a lack in the study of such deflation techniques for unsymetric matrices. It is only mentioned in a few of papers and we have not found numerical results with short term recurrence methods.

The first difficulty we can easily observe in the application of deflation for unsymmetric systems is the fact that, even if the matrix is real, its eigenvalues and eigenvectors can be complex. Here a word must be said about one of our limitations. The algorithms for eigenvalue computation we have currently available have been designed for symmetric matrices. Then, instead of providing the eigenvectors of the coefficient matrix as the deflation vectors we have computed those corresponding to the normalized system $\boldsymbol{A}^T\boldsymbol{A}$. Strictly speaking what we are computing are the left singular vectors of the singular values decomposition of $\boldsymbol{A}$.

Another point to be clarified before passing the specific examples is that for unsymmetric systems we have test, in addition to the two reaorderings used for symmetric matrices,

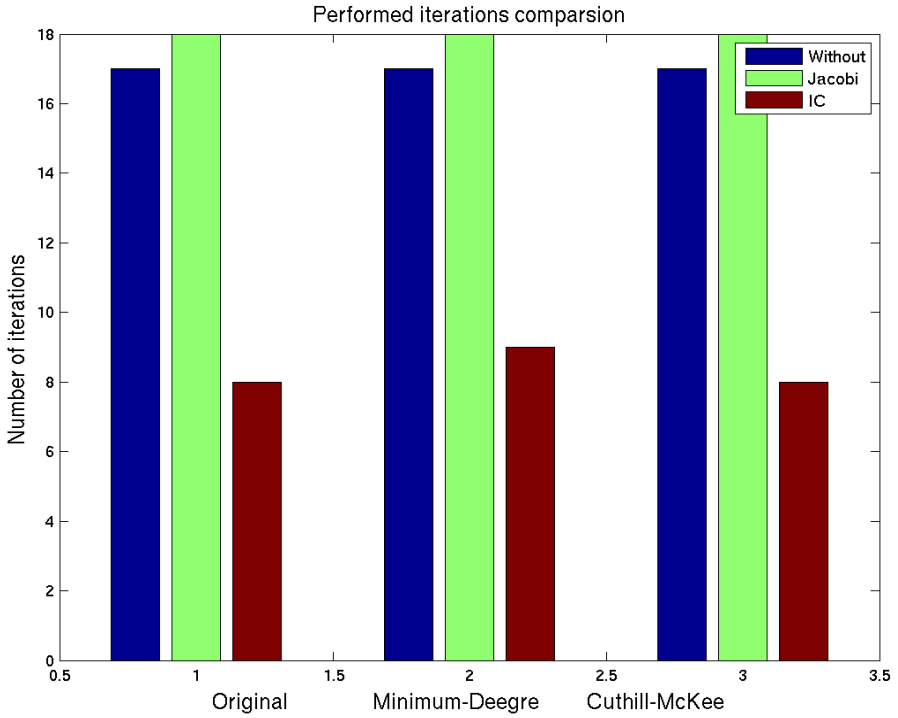

FIGURE 7.14: Number of performed Conjugate Gradient iterations for solving the ISIS Pressure Poisson system with $6,400$ unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky; together with spectral deflation, $p = 640$.

other four strategies. We list them as follows together their acronyms used for identifying them:

1. RCM.- Reverse Cuthill-McKee.
2. MID.- Minimum Degree.
3. MCO.- Multi Color.
4. OWD.- One Way Dissection.
5. NED.- Nested Dissection.
6. RQT.- Rooted Quotient Three.

Althoght when the matrix is not structurally symmetric, we apply the reordering to the sparsity of the symmetric matrix $\boldsymbol{A} + \boldsymbol{A}^T$.

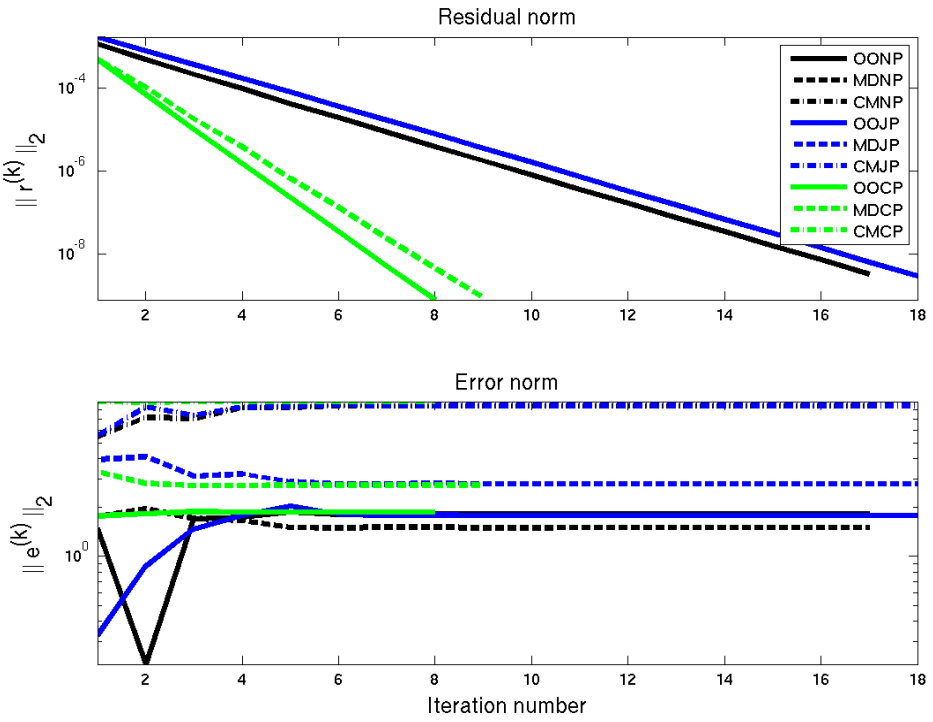

FIGURE 7.15: Conjugate Gradient convergence curves for solving the ISIS Pressure Poisson system with $6,400$ unknowns for the three orderings and no preconditioning, Jacobi and Incomplete Cholesky; together with spectral deflation, $p = 640$.

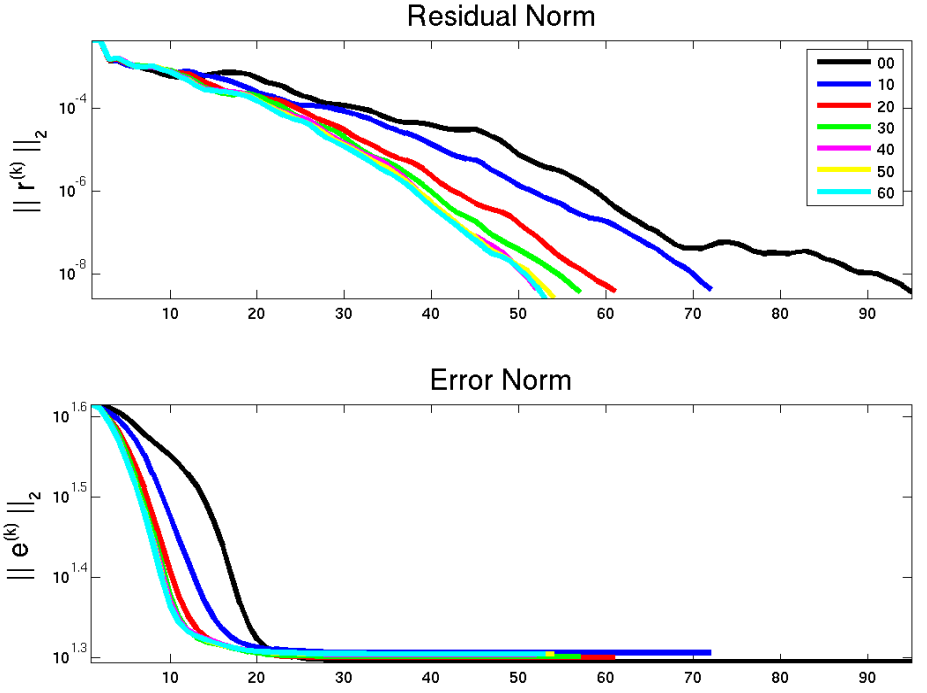

FIGURE 7.16: Convergence curves for $0, 10, 20, 30, 40, 50$ and 60 spectral deflation vectors using Incomplete Cholesky preconditioning.

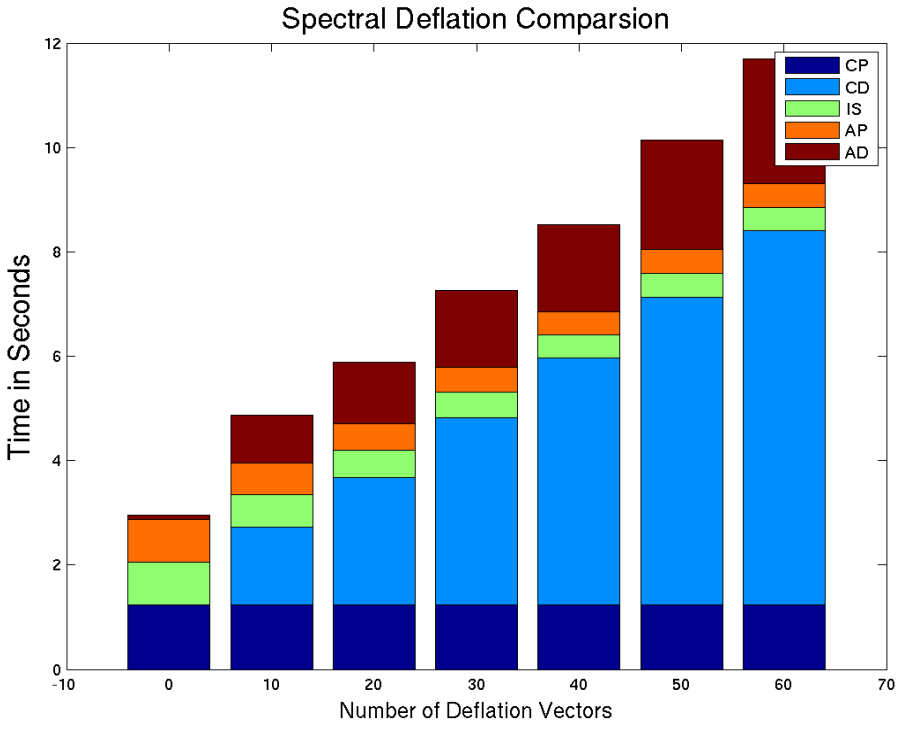

FIGURE 7.17: Elapsed time for $0, 10, 20, 30, 40, 50$ and 60 spectral deflation vectors using Incomplete Cholesky preconditioning.

7.2.1 Two Dimensional Convection-Diffusion

The first of the unsymmetric matrices is a slight modification of the first presented in the previous section for symmetric matrices. A little of convection has been added to the Laplacian in order to broke the symmetry of the problem, we have taken $\epsilon = 0.01$.

The best Krylov subspace method among those used in the present work has been the BiConjugate Gradient Stabilized, being the faster and the one that has performed the less number of iterations.

First we shall analyze the results obtained with the ILUTP preconditioner which has been the most efficient among those based in incomplete factorizations.

Figure 7.18 shows the sparsity patterns such incomplete factorization with the six different reordering used. Figure 7.19 shows the spectrum in the complex plane of the preconditioned systems with the six different orderings. Note that, even that in general the spectrum are clustered around one, some of them, particularly RQT, spreed it in an parallel line to the complex axis. The best clustered ones correspond tho the original reordering and RCM.

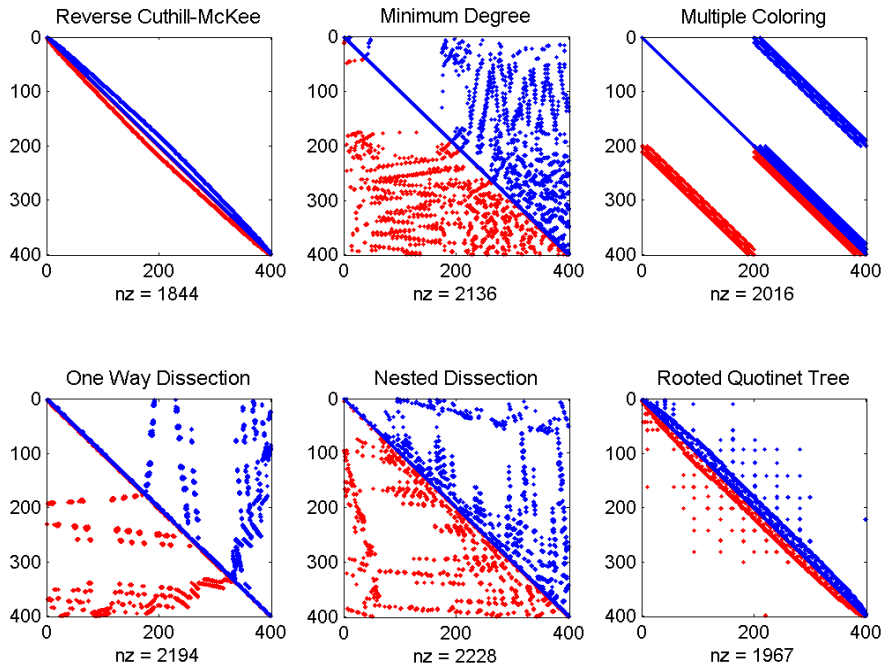

FIGURE 7.18: Sparsity patterns of $ILUTP(2\mu, 0.05; n, 0.05)$ preconditioner for the six reorderings computed.

Figure 7.20 shows the convergence curves when of the preconditioned systems together the unpreconditioned one for reference. Note the important reduction in the number of the iterations. From this we also see that the drop tolerance τ has no influence if low fill in is allowed.

In figure 7.21 we have compared the overall times for this preconditioner together the iterative solver. Note that the effectiveness of this preconditioner is at the high cost of being very time consuming for its computation. For this reason we consider feasible use this preconditioner only when other preconditioners fail or when it is possible to reuse it.

The last figures concerned with a preconditioner of the incomplete factorization family 7.22 and 7.23 corresponds when the mean performance case of the previously compared preconditioners is endowed with spectral deflation. Note that no important gains have been obtained in the reduction of the number of iterations. Moreover the flattening effect in the error curve is quite significant.

Figure 7.24 shows the sparsity patterns such approximate inverse with the six different reordering used. Figure 7.25 shows the spectrum in the complex plane of the preconditioned systems with the six different orderings. Note that, as expected AIMR is invariant

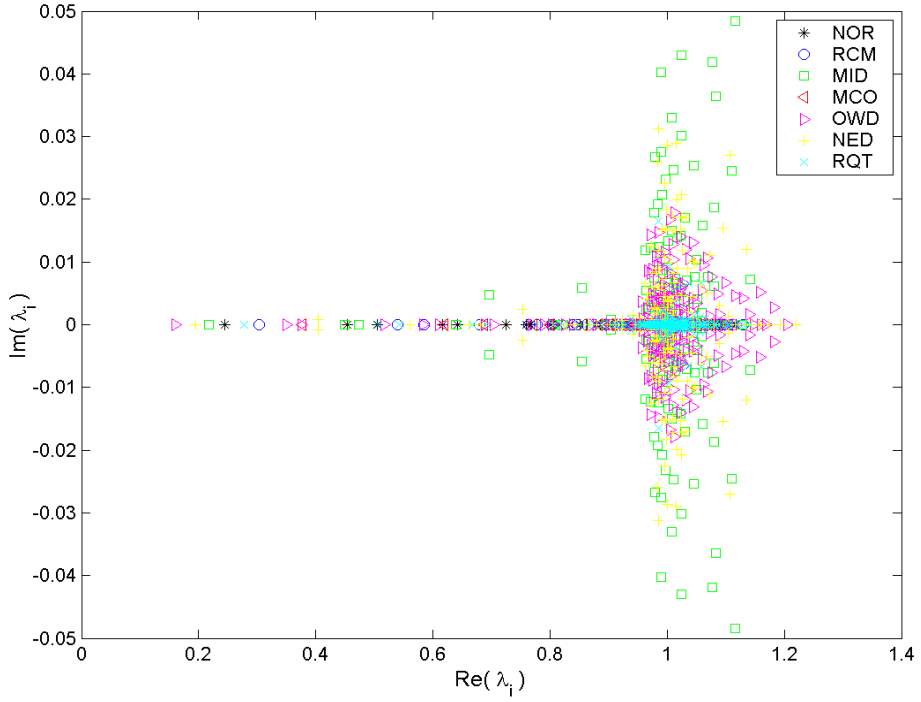

FIGURE 7.19: Spectrum of $ILUTP(2\mu, 0.05; n, 0.05)$ preconditioned systems for the Natural ordering and the six different reorderings.

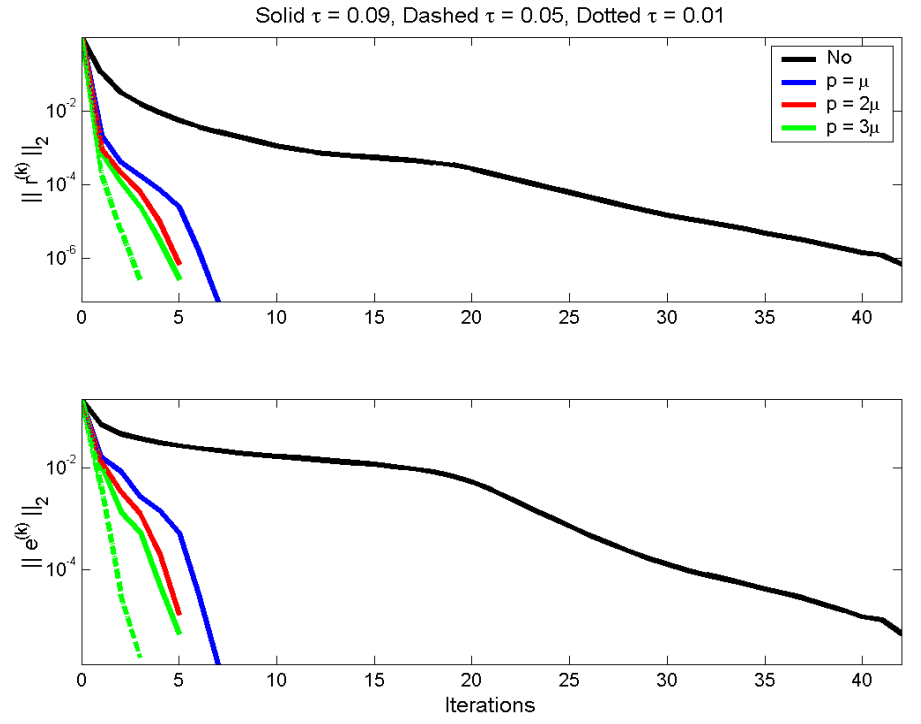

FIGURE 7.20: Convergence curves for $ILUTP(p, \tau; n, 0.05)$ preconditioners, together unpreconditioned system in black, with Natural ordering.

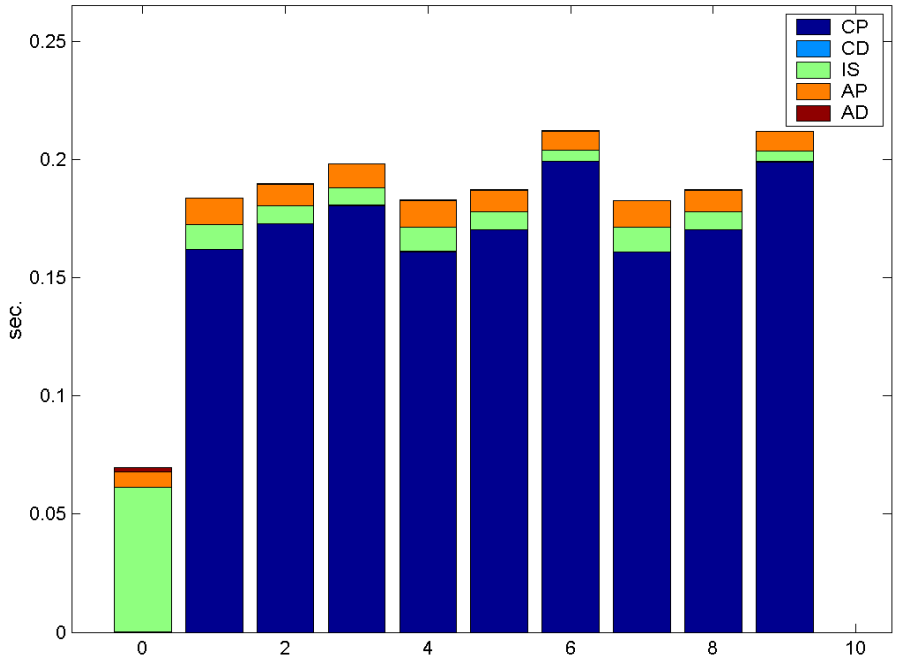

FIGURE 7.21: Time comparison for $ILUTP(p, \tau; n, 0.05)$ preconditioners, together unpreconditioned system at zero, with Natural ordering.

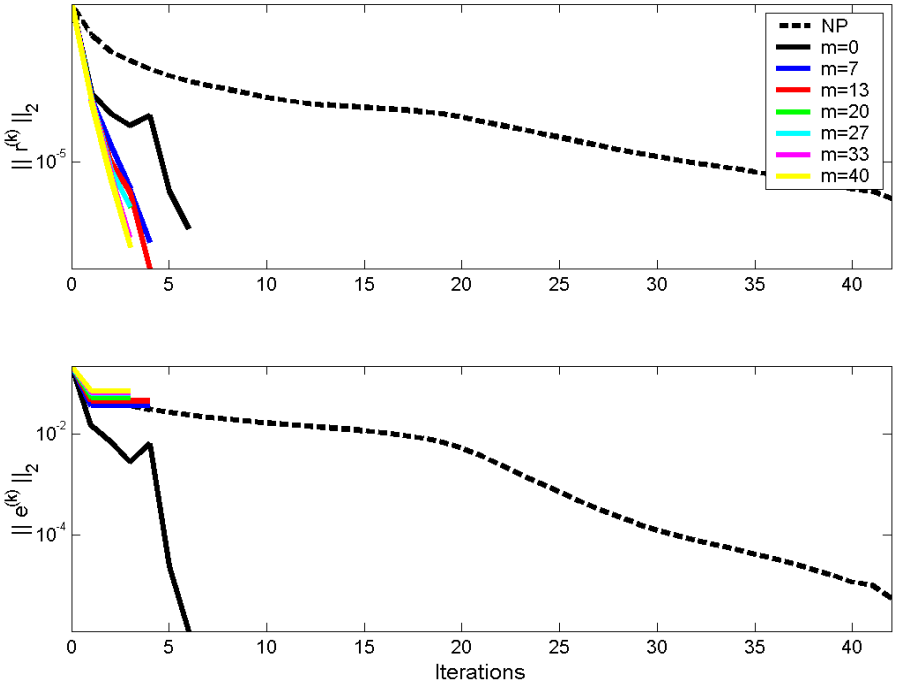

FIGURE 7.22: Convergence curves for undeflated and unpreconditioned systems, in dashed and solid black lines respectively, together with $ILUT(2\mu, 0.05)$ preconditioned and spectral deflated systems with Natural ordering.

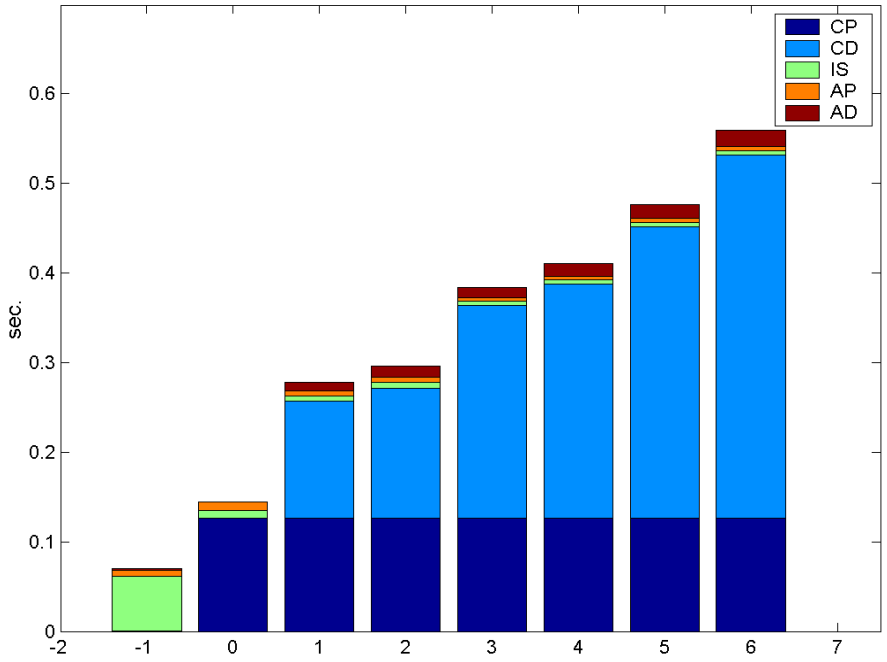

FIGURE 7.23: Time comparison for undeflated and unpreconditioned system (-1), and $ILUT(2\mu, 0.05)$ preconditioned and spectral deflated systems with Natural ordering.

to reordering.

Figure 7.26 shows the convergence curves when of the preconditioned systems together the unpreconditioned one for reference. Note the important reduction in the number of the iterations. For this preconditioner the effect observed with the previous preconditioner when the drop tolerance τ has no influence if low fill in is allowed is even more drastic than the previous case.

From the comparison of the overall times for this preconditioner together the iterative solver shown in figure 7.27 we see that the time required to compute this preconditioner dominates far away the overall process.

The last figures concerned with preconditioner of the approximate inverse family 7.22 and 7.23 corresponds when a the mean performance case of the previously compared preconditioners is endowed with spectral deflation. In thsi case slight gains have been obtained in the reduction of the number of iterations. The flattening effect in the error curve is still present. But what is more important, the overall time is still dominated by the computation of the preconditioner.

In closing the results for the matrix at hand we present in 7.30 the estimative if the condition number of the preconditioned by ILU(p) and deflated systems as functions

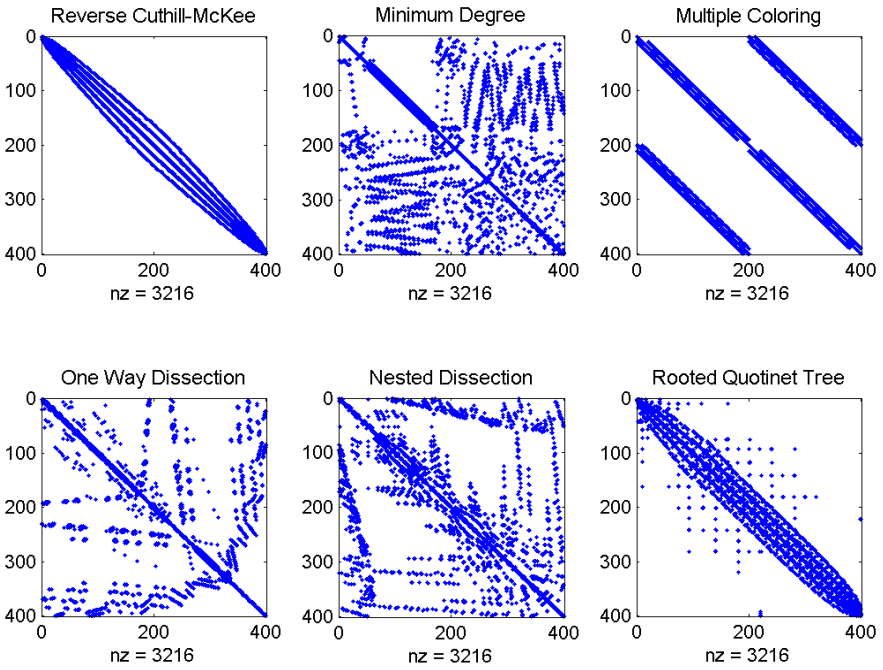

FIGURE 7.24: Sparsity patterns of $AIMR(2\mu, 0.05)$ preconditioner for the six reorderings computed.

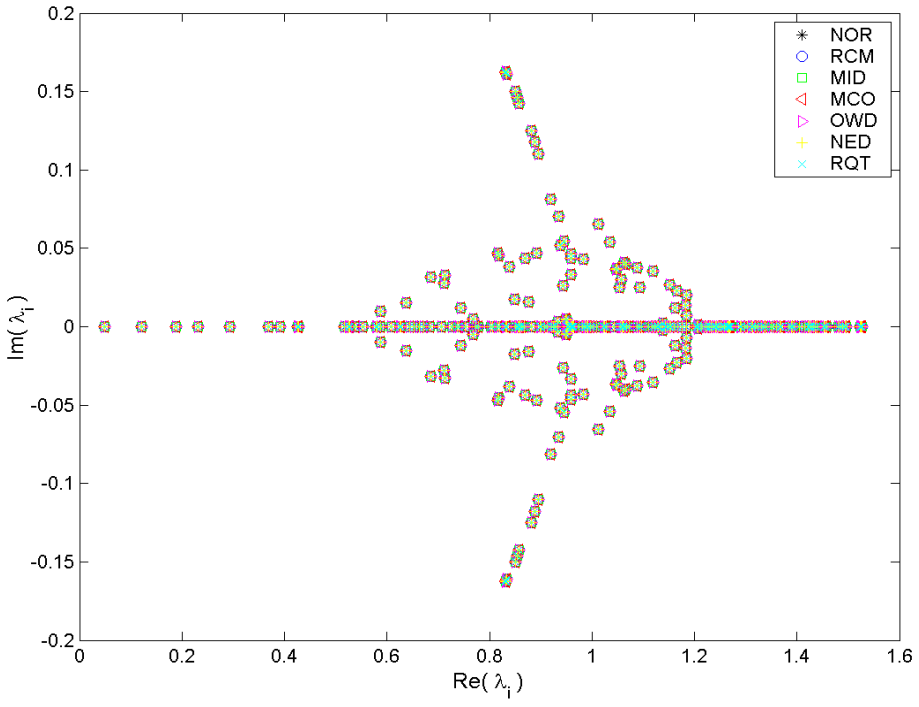

FIGURE 7.25: Spectrum of $AIMR(2\mu, 0.05)$ preconditioned systems for the Natural ordering and the six different reorderings.

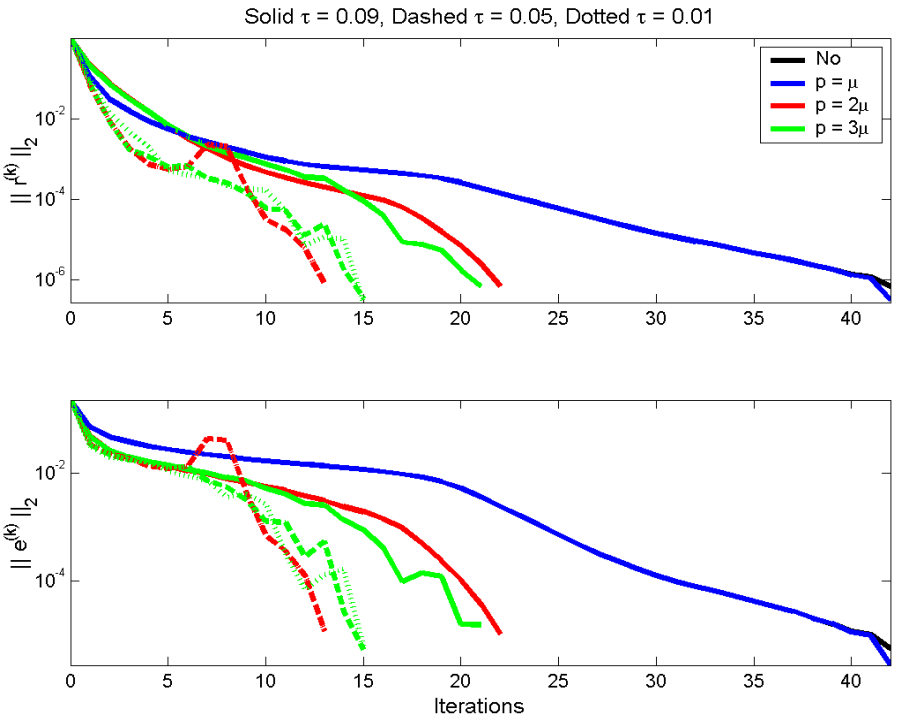

FIGURE 7.26: Convergence curves for $AIMR(p,\tau)$ preconditioners, together unpreconditioned system in black, with Natural ordering.

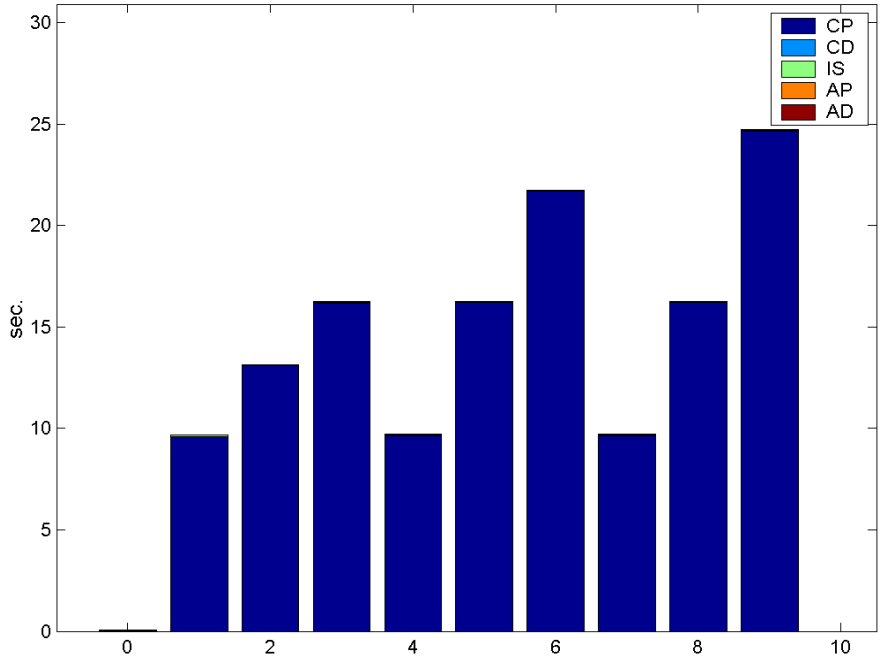

FIGURE 7.27: Time comparison for $AIMR(p,\tau)$ preconditioners, together unpreconditioned system at zero, with Natural ordering.

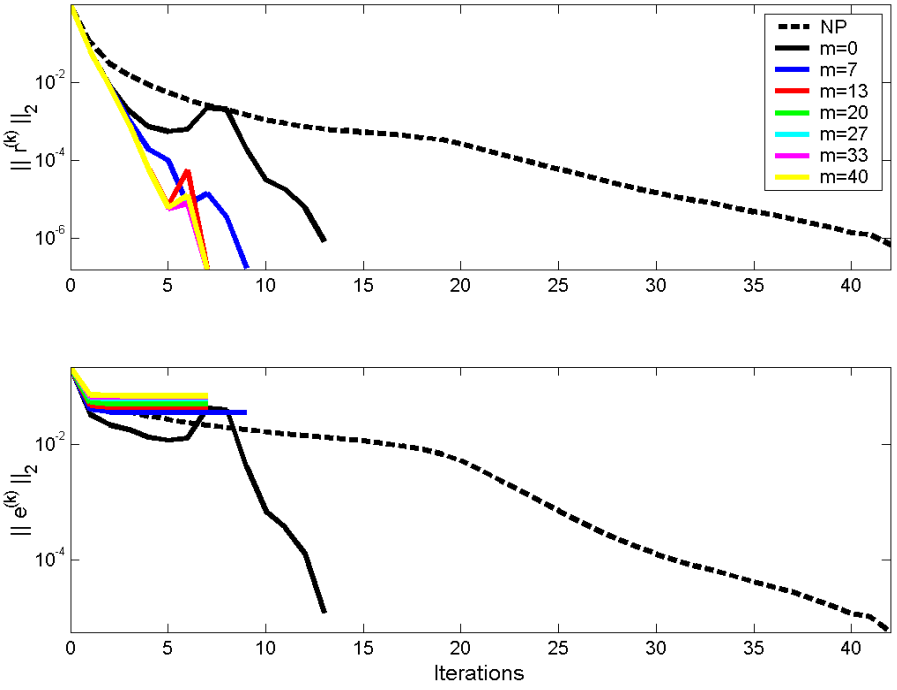

FIGURE 7.28: Convergence curves for undeflated and unpreconditioned systems, in dashed and solid black lines respectively, together with $AIMR(2\mu, 0.05)$ preconditioned and spectral deflated systems with Natural ordering.

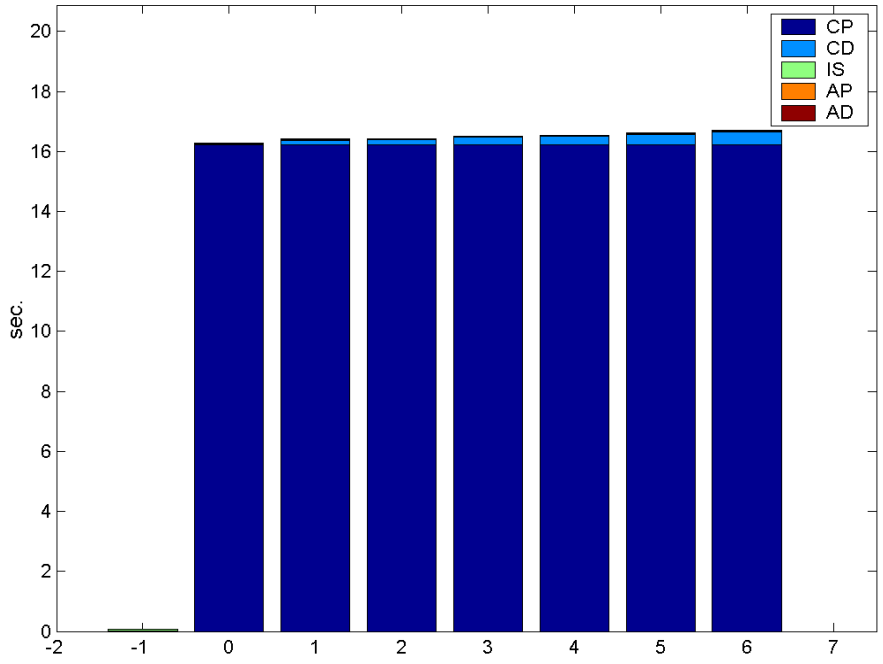

FIGURE 7.29: Time comparison for undeflated and unpreconditioned system (-1), and $AIMR(2\mu, 0.05)$ preconditioned and spectral deflated systems with Natural ordering.

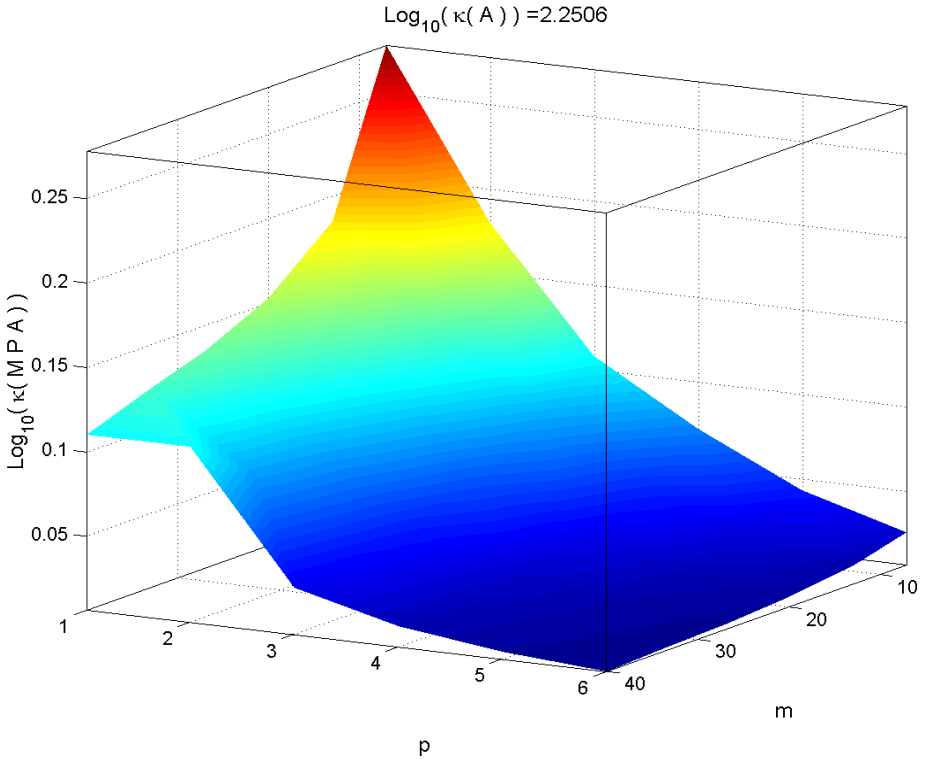

FIGURE 7.30: Condition numbers of preconditioned and deflated systems using $ILU(p)$ preconditioners and spectral deflation with Natural ordering.

of the fill in level p and the number of deflation vectors m. Figure 7.31 presents the analogous results when SPAI preconditioning has been used. For both, spectral deflation vectors were used.

7.2.2 LNS511 Matrix Market

The second unsymmetric example has been also taken from the University of Florida Sparse Matrix Collection. It corresponds to a linear systems arising from the discretization of the Navier-Stokes of the cavity flow problem using mixed finite elements. Since velocity and pressure are included in the unknown vector this matrix has saddle point structure.

In this case we have observed extremely erratic and irregular convergence behavior when BiCGStab has been used. Here we present the results corresponding to the QMR method, which was the most well behaved.

In difference of the previous case, this time the best preconditioner from the incomplete factorization family has been the ILUDP preconditioner.

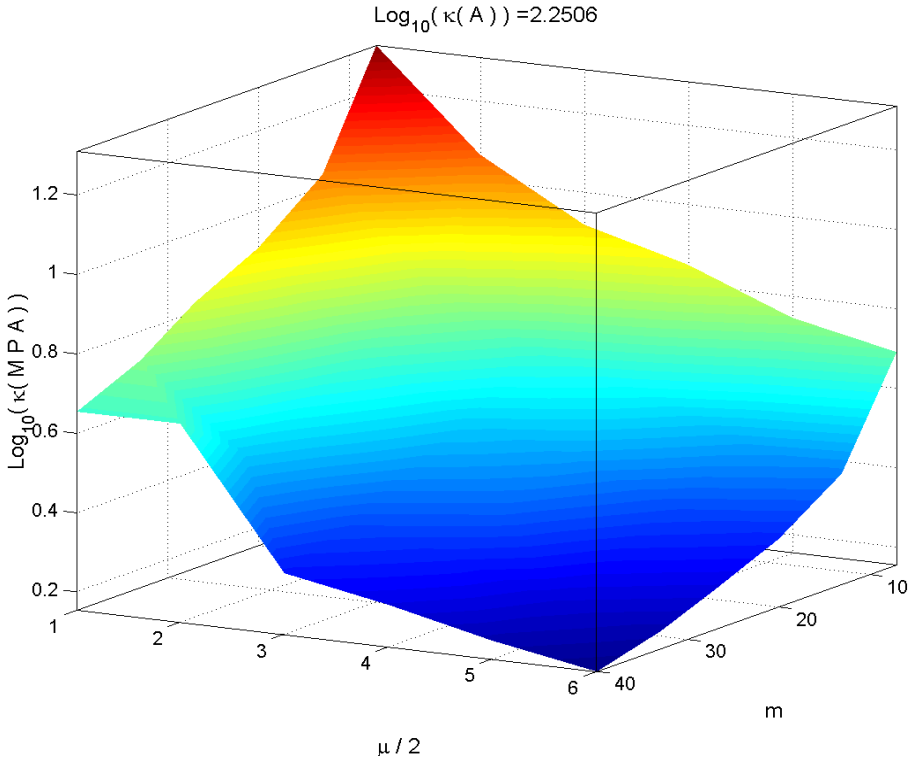

FIGURE 7.31: Condition numbers of preconditioned and deflated systems using $SPAI(p)$ preconditioners and spectral deflation with Natural ordering.

Figure 7.32 shows the sparsity patterns of such incomplete factorization with the six different reordering used.

Figure 7.33 shows the convergence curves when of the preconditioned systems together the unpreconditioned one for reference. Here it is important to mention that the minimum degree ordering is the same as the original one. Then all different reordering worse the convergence in this situation. More important to take into account is the fact that the unpreconditioned iteration did not succeed in reduce the residual norm until the required accuracy.

In figure 7.36 we have compared the overall times for this preconditioner together the iterative solver. For this case the situation is completely different to the previous cases. Even more, the column corresponding to the unpreconditined iteration is not strictly speaking comparable with the corresponding to the preconditioned iterations that have succeed because the former one did not. Then gains in the overall time can be obtained when the system to solve is really ill-conditioned so that using preconditioning is mandatory for convergence.

The last figures concerned with a preconditioner of the incomplete factorization family 7.37 and 7.38 corresponds when the mean performance case of the previously compared

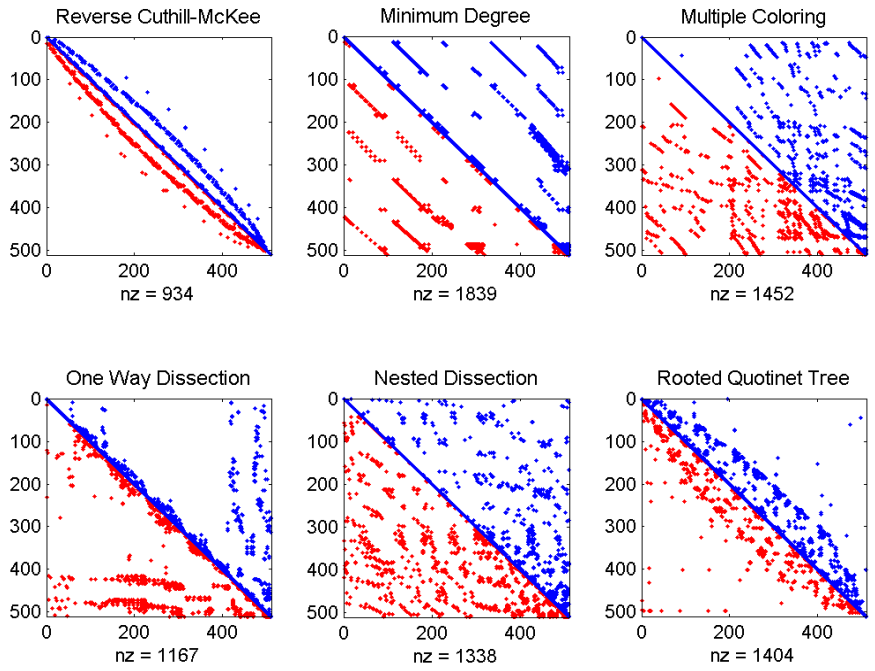

FIGURE 7.32: Sparsity patterns of $ILUDP(0.5, 0.05; n, 0.05)$ preconditioner for the six reorderings computed.

preconditioners is endowed with spectral deflation. Note that in this case, deflated iterations are in al cases very much worse that undeflated ones.

We shall omit to show any result concerned with the family of approximate inverses. In fact, the only method that succeed in the computation has been SPAI but for all cases the preconditioned iterations have been worse that the unpreconditioned ones. Any improvement with deflation was obtained.

7.2.3 ISIS Fully Coupled System

This last example concerning unsymmetric matrices corresponds to a matrix obtained from the discretization of the Navier-Stokes equations in a driven cavity flow using finite volumes with the *ISIS* software. But this time a fully coupled scheme is used in order to solve for the velocity and pressure at once.

The iterative solver having the best performance for this matrix we have been, in general terms, the BiCGStab. Although smoother and strictly decreasing converge curves have been observed more frequently with the QMR of QMRBiCGStab methods, these methods are in general more time consuming.

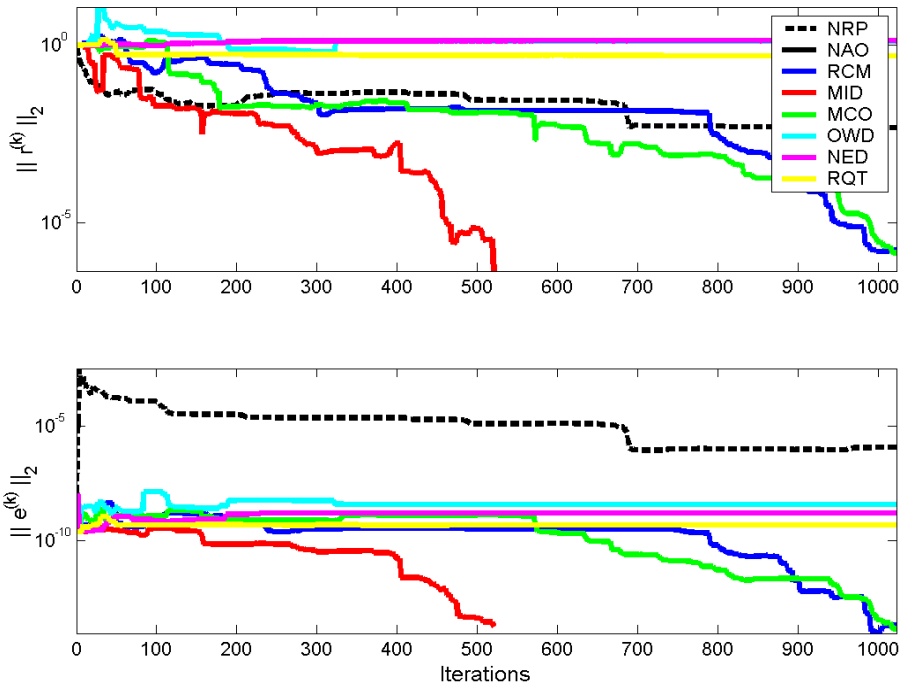

FIGURE 7.33: Convergence curves for unpreconditioned and preconditioned Natural ordering systems, dashed black and solid black respectively, and the six different reorderings using $ILUDP(0.5, 0.05; n, 0.05)$ preconditioning.

Due to the size of the matrix at hand, the times required for the computation of the preconditioners belonging to the class of approximate inverses have been completely prohibitive. For this reason we have avoided its computation and only those of the classical iterative methods and incomplete factorizations have been tested.

The best of such incomplete factorization preconditioners has been the ILUD one. It is quite amazing that for this matrix this preconditioner outperforms, sometimes faily, its pivoted version. This situation has been seldom observed.

In figure 7.39 we show the sparsity patterns of a mean performance ILUD factorization.

Due to the size and characteristics of the matrix it has been not possible to perform any computation with the whole spectrum of the matrix. Instead of information directly derived from such computations we present in the middle of figure 7.40 an estimative of the stability of the computed incomplete factorization. Together in this figures also appears the density of the preconditioner and the time spent in their computation. Note that in this case incomplete factorizations when based in a numerical dropping tolerance are denser that the factorization corresponding to the original ordering, having the exactly opposite effect for which they were developed.

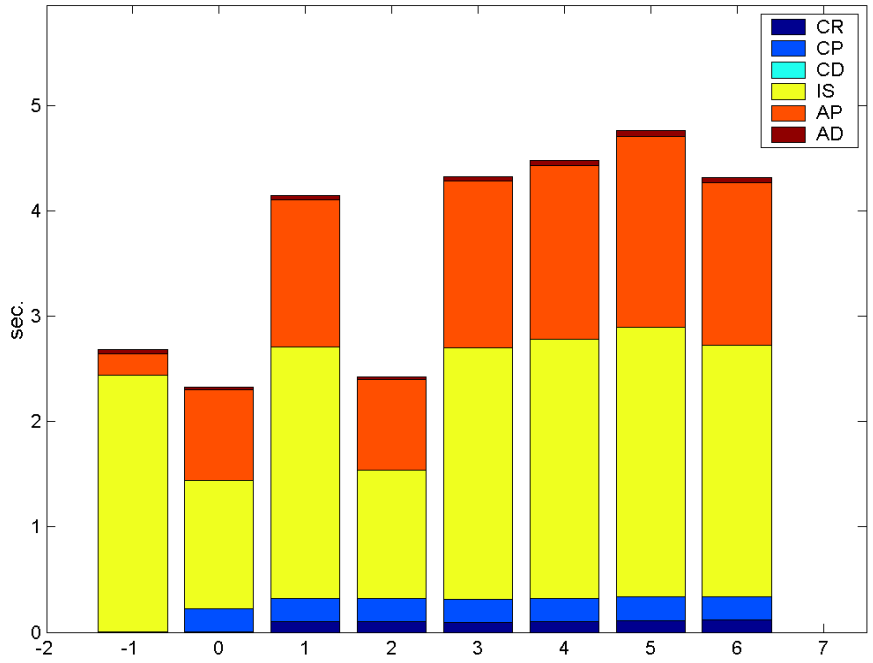

FIGURE 7.34: Time comparison for unpreconditioned and preconditioned Natural ordering systems, (-1) and (0) respectively, and the six different reorderings using $ILUDP(0.5, 0.05; n, 0.05)$ preconditioning.

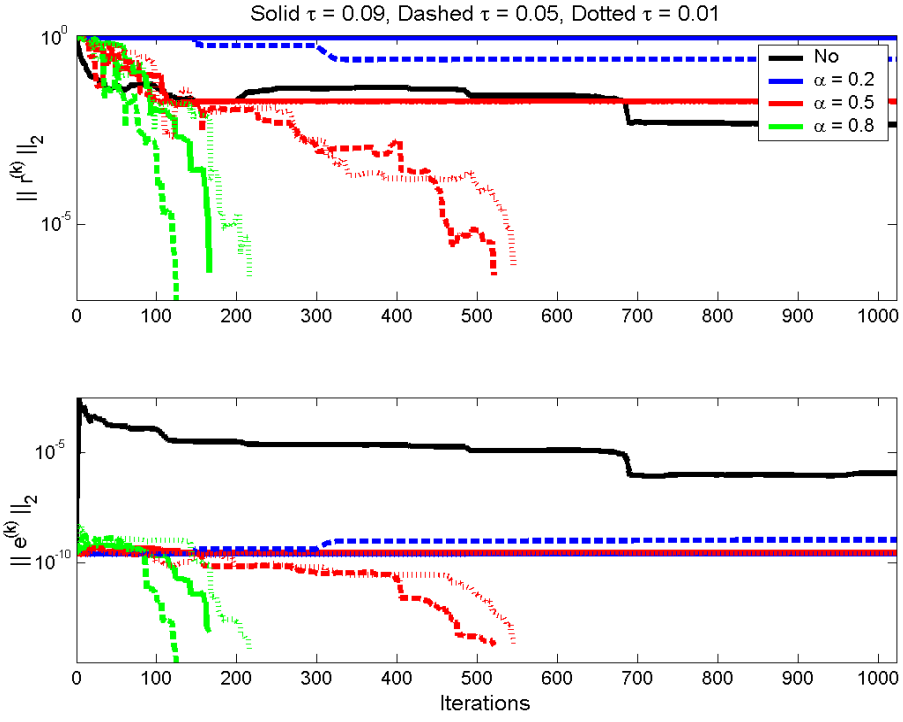

FIGURE 7.35: Convergence curves for $ILUDP(\alpha, \tau; n, 0.05)$ preconditioners, together unpreconditioned system in black, with Natural ordering.

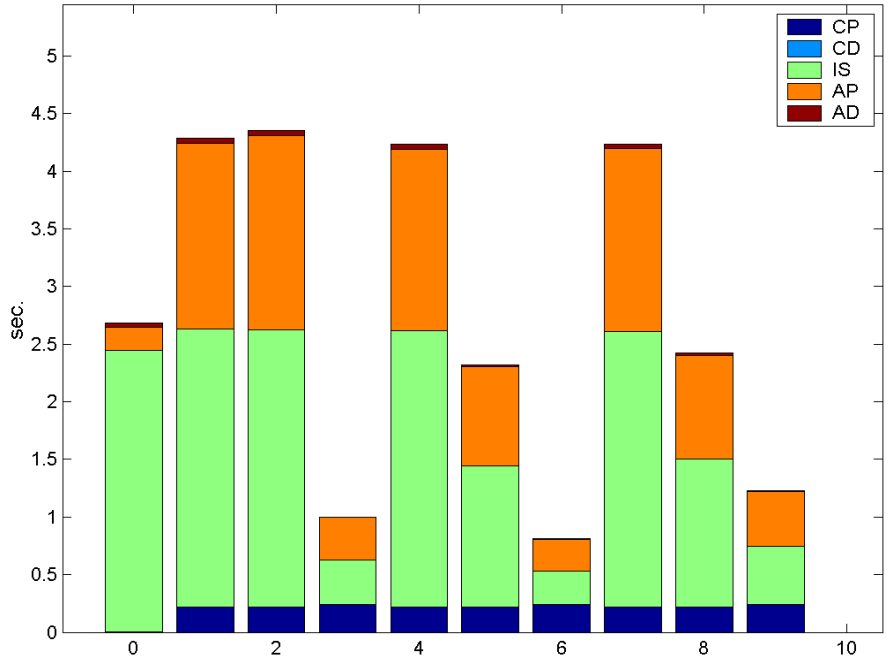

FIGURE 7.36: Time comparison for $ILUDP(\alpha, \tau; n, 0.05)$ preconditioners, together unpreconditioned system at zero, with Natural ordering.

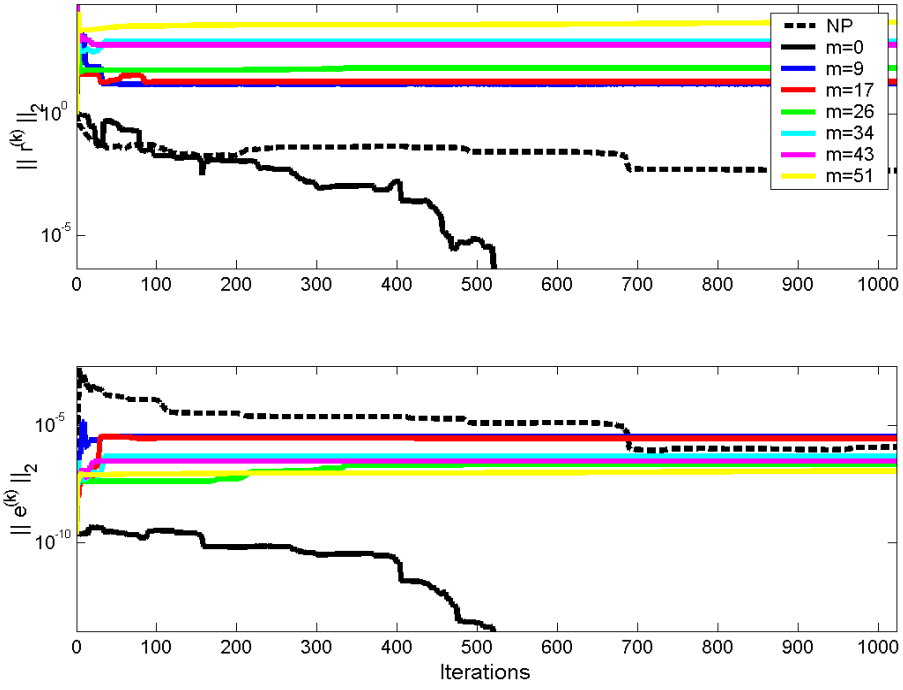

FIGURE 7.37: Convergence curves for undeflated and unpreconditioned systems, in dashed and solid black lines respectively, together with $ILUDP(0.5, 0.05; n, 0.05)$ preconditioned and spectral deflated systems with Natural ordering.

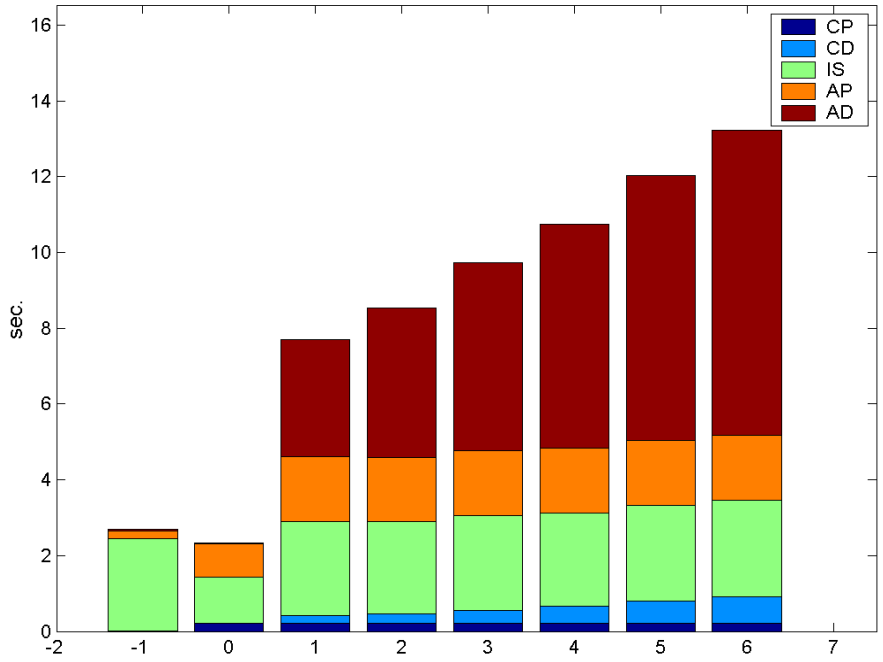

FIGURE 7.38: Time comparison for undeflated and unpreconditioned system (-1), and $ILUDP(0.5, 0.05; n, 0.05)$ preconditioned and spectral deflated systems with Natural ordering.

Figure 7.41 we show the sparsity patterns of all the ILUD factorizations we have computed with the natural ordering. It is important to mention that the fill in occurs mainly in the lower block. this can be substantial as we can observe in the first part of figure 7.40, the density goes from 2 to 12. Note that also the stability and time spent have the same pattern. All this significative differences depend more strongly on the numerical dropping τ.

Figure 7.43 shows the convergence curves when using a mean performance ILUD factorization. Note that only a slight difference can be observed depending in the reordering used, being the best one the natural reordering. Figure 7.44, which shows the overall elapsed time for the solution of the linear systems, we can finishes to confirm that the reorderings we have used are not convenient for this matrix.

Figures 7.45 and 7.46 the convergence curves and elapsed times, respectively, for all the ILUD factorizations. Note that, as in the previous matrix, when the conditioning of the matrix really represents a problem investing time in a hight quality preconditioner could reduce the overall solution time.

We finish the observations for this matrix with the comment that, as with the previous matrix, deflation in this case get worse results that the undeflated iteration.

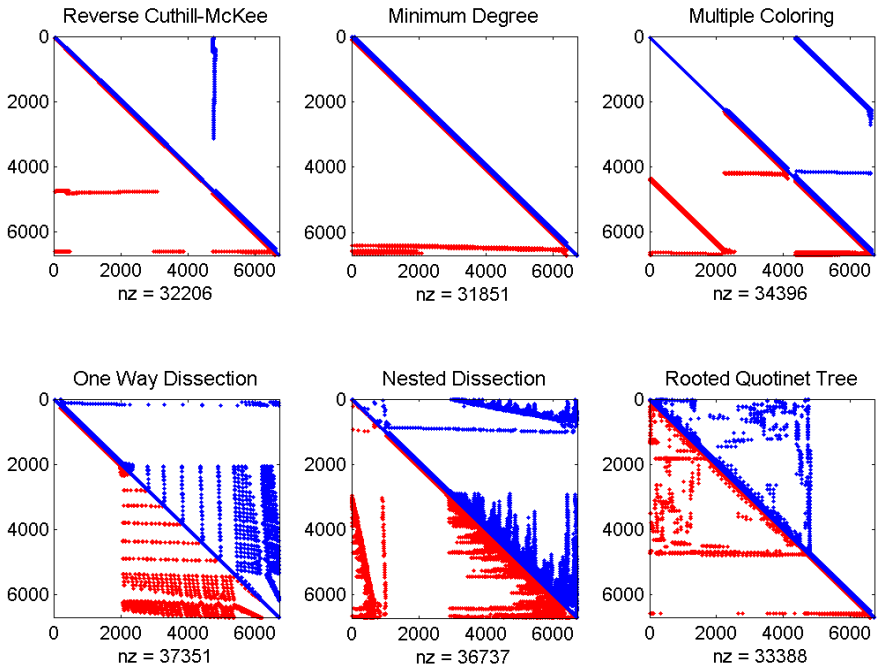

FIGURE 7.39: Sparsity patterns of $ILUD(0.5, 0.05)$ preconditioner for the six reorderings computed.

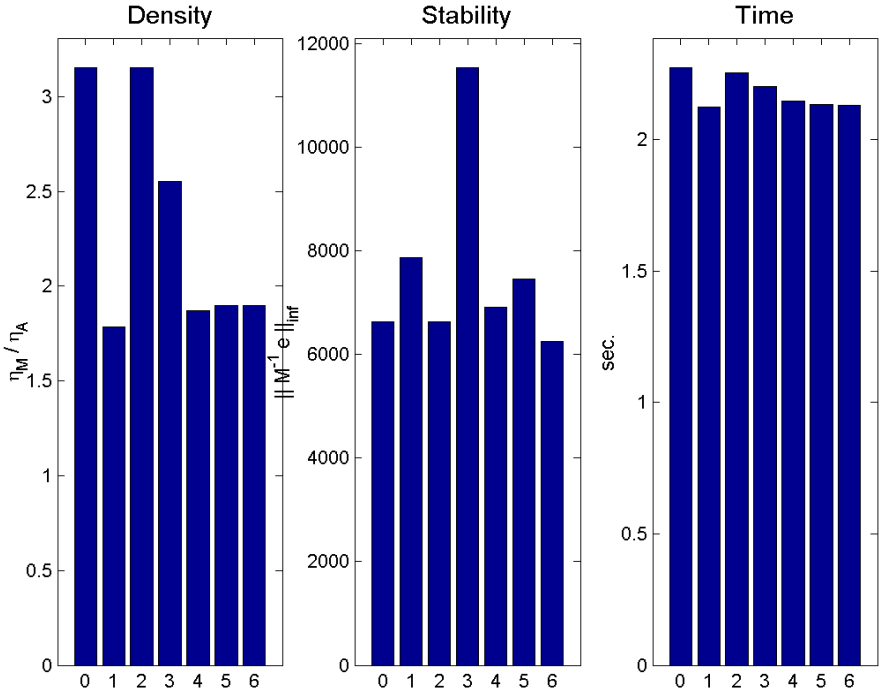

FIGURE 7.40: Comparison of density, stability estimative and elapsed time for computing $ILUD(0.5, 0.05)$ preconditioner for the Natural ordering, marked as zero, and the six reorderings computed.

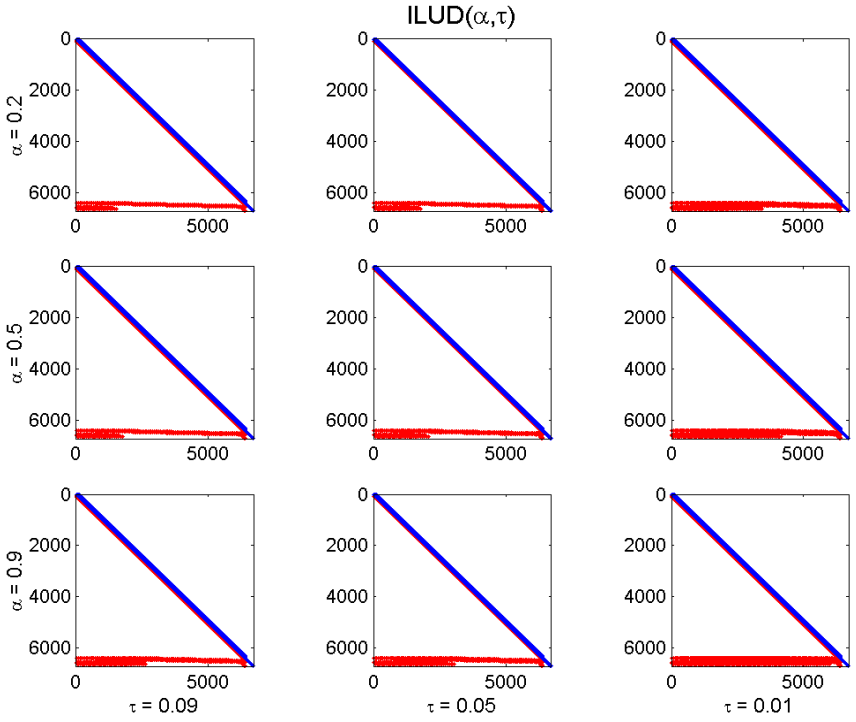

FIGURE 7.41: Sparsity patterns of $ILUD(\alpha, \tau)$ preconditioners for Natural ordering.

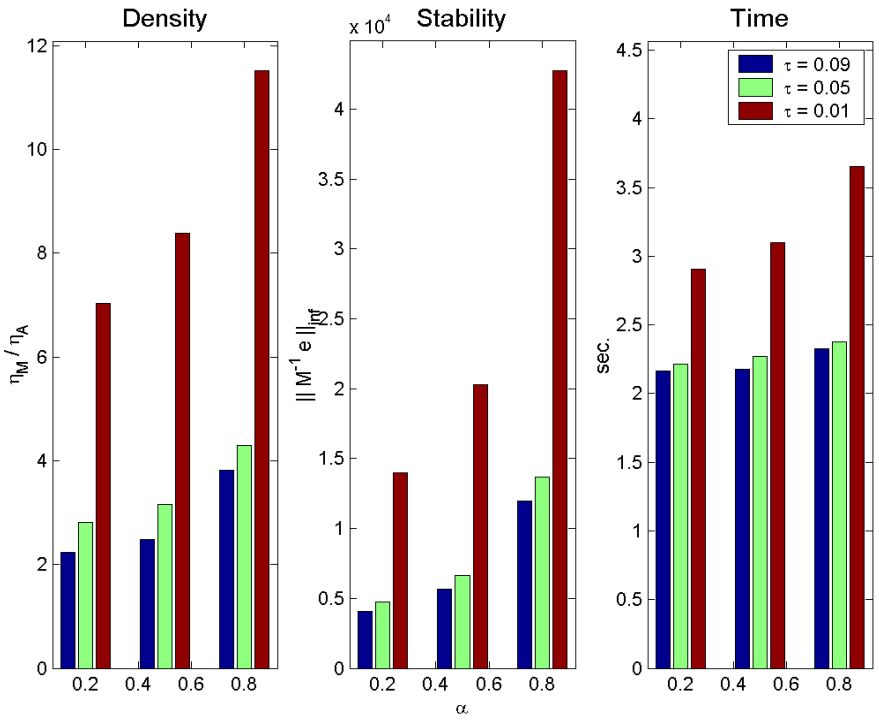

FIGURE 7.42: Comparison of density, stability estimative and elapsed time for computing $ILUD(\alpha, \tau)$ preconditioners for Natural ordering.

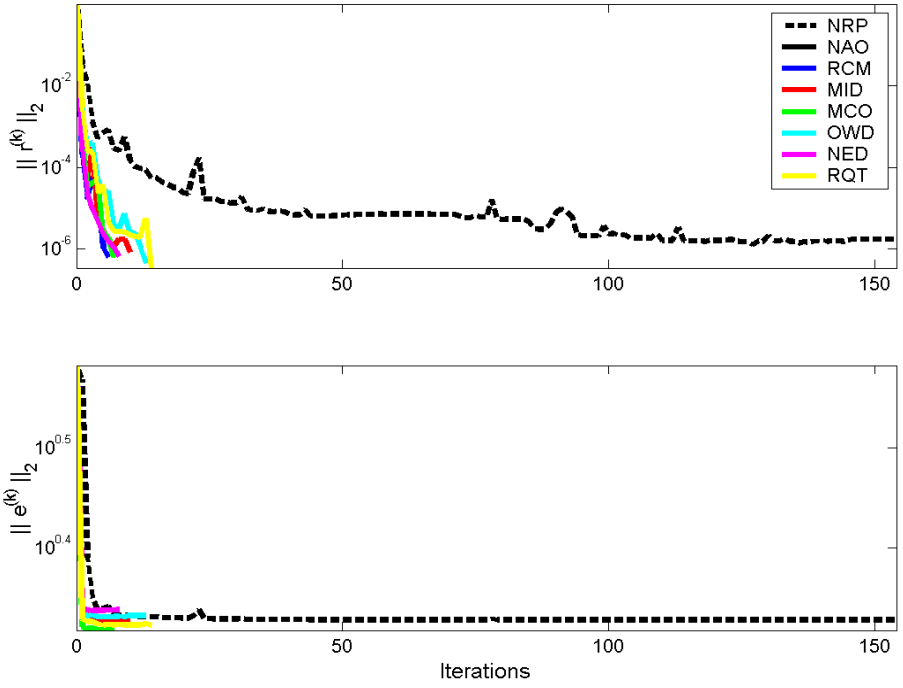

FIGURE 7.43: Convergence curves for unpreconditioned and preconditioned Natural ordering systems, dashed black and solid black respectively, and the six different reorderings using $ILUD(0.5, 0.05)$ preconditioning.

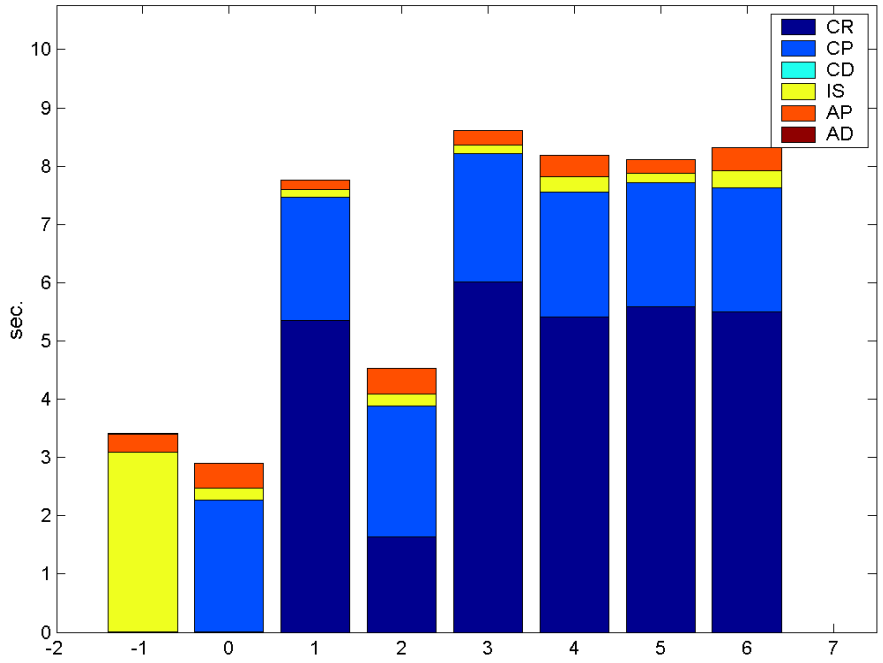

FIGURE 7.44: Time comparison for unpreconditioned and preconditioned Natural ordering systems, (-1) and (0) respectively, and the six different reorderings using $ILUD(0.5, 0.05)$ preconditioning.

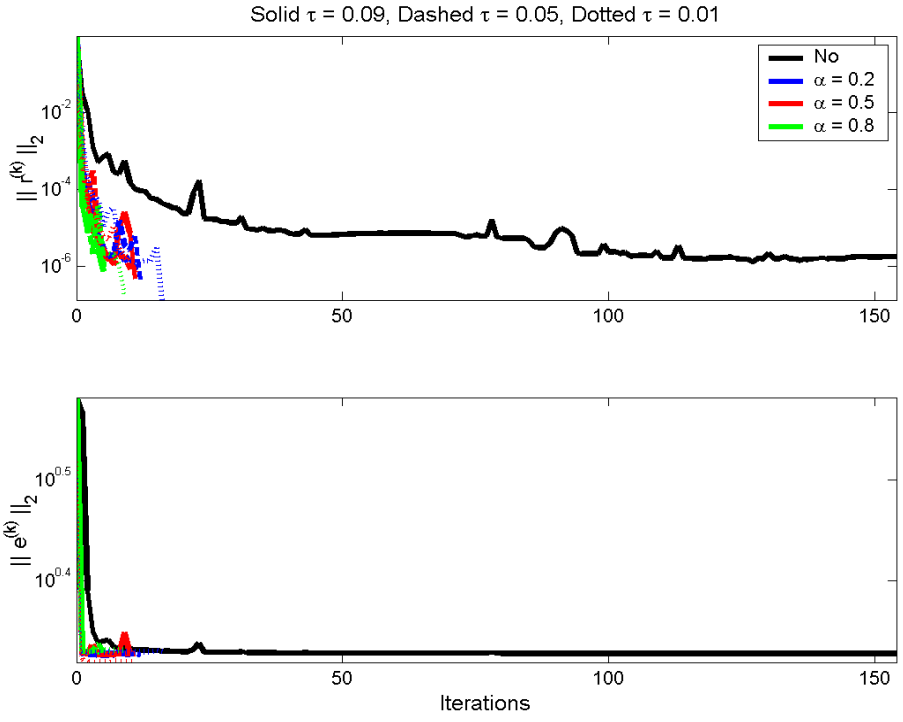

FIGURE 7.45: Convergence curves for $ILUD(\alpha, \tau)$ preconditioners, together unpreconditioned system in black, with Natural ordering.

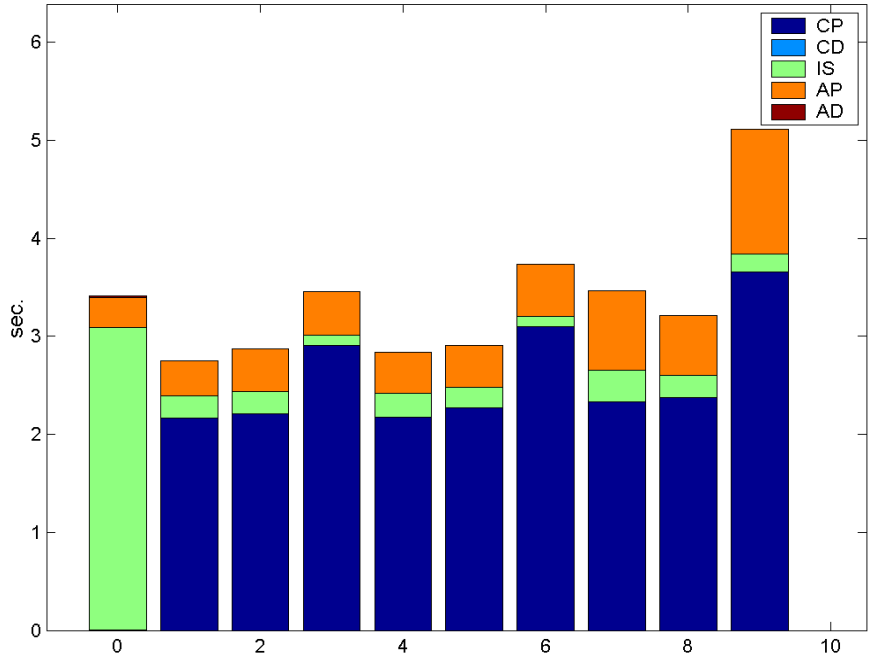

FIGURE 7.46: Time comparison for $ILUD(\alpha, \tau)$ preconditioners, together unpreconditioned system at zero, with Natural ordering.

Chapter 8

Conclusions

This thesis deals with the solution of linear systems where the coefficient matrix is presumed to be large and sparse. Particularly those arising in the simulation of fluid dynamics were here addressed. The objectives pursued in this work, introduced at chapter 1, have been fulfilled satisfactorily. A considerable variety of techniques for the solution of such linear systems have been developed in the context of the Krylov subspace, which has been used for the elucidating numerical tests described in the previous chapter.

8.1 Results Discussion

Throughout this work we have reviewed a large amount of references in order to obtain the some insight of the state of the art in the solution of large and sparse linear systems, a topic which is of scientific interests itself and that additionally have a wide applicability range in several branches of engineering and science. Also we have presented here the main ideas and developments in such a task.

The literature review also has been done in order to obtain suitable examples for validation our yet mentioned implementations. As result, an reliable and robust code has been developed.

In this code several well known Krylov subspace methods have been implemented for the two kinds of problems addressed in this work, namely symmetric and unsymetric matrices, in both the matrix at hand is large and sparse. For the acceleration of such iterative methods, several wide used *preconditioning techniques* have been discussed and implemented in order to analyze their effect while they are used in conjunction with several iterative methods. Also *reordering algorithms* have been implemented and we have tested its effect when used together preconditioning. Additionally the more novel

technique currently known as *deflation* have been discussed and implemented achieving a considerable reduction in the number of iterations for some important cases.

Standard problems are extensively used as benchmark problems to test and validate new proposed strategies and their implementations. In this work, some well known of such problems from the University of Florida Sparse Matrix Collection have been used. In such a task we have demonstrated the good performance of the formulations reviewed implemented in this work.

8.2 Applications and Limitations

In practice, the techniques we have discussed and implemented can be applied to a wide variety of linear systems. The applications are very important not only for practical reasons, but also for gaining experience and defining further development needs. The application of these techniques for linear systems arising from different areas should be straightforward. Although, while focusing our attention to a particular one, we claim even more significant gains can be obtained by enriching the techniques to linear systems with special characteristics.

The main limitations of the present work are those related when the coefficient matrix is unsymmetric, which is the common case if we take a slight review of the state of the art on this subject. The limitations were mainly observed in problems in which we have an important lack of diagonal dominance, with linear systems written for two or more different physical quantities, matrices having its spectrum on both sides of the complex axis and when saddle point systems were treated as a whole. Additionally, our code has a lack of robustness in the numerical treatment of the unsymmetric eigenvalues problem.

8.3 Further Developments

Further developments of this work are listed in sequel. They could include:

- *Parallel computing.* This further development is considered due the fact that, although deflation technique effectively reduces the number of iterations a Krylow subspace method needs to converge, the overall computational time is usually higher when a sequential computation is used. This is particularly applicable for deflation techniques for which we have observed high speed up and applications times. Although by reviewing its implementation in chapter 6 we can see that its

parallelization is quite intuitive. Almost the same can be say about at least three preconditioners of the approximate inverse framework.

- *Segregation and block strategies.* The developments and code implementation in this work have been made not taking into account if the linear system contains two or more different physical quantities. Exploiting block structures in general have demonstrated to increase the performance in some situations.

- *High performance computing.* This is maybe the first subsequent activity in continuing the present work. It seems almost mandatory to translate the current code to a high performance programming language like Fortran. As we have stated, this trend does not represent a high difficulty since it has been considered from the beginning of the present work.

- *Weighted reorderings.* All of the reordering strategies implemented and used in the tests are based solely in the sparsity structure of the coefficient matrix, or its symmetric part, and do not take into account individual entries contained in the matrix. Techniques taking into account such information have demonstrated to be more effective that the former ones. Also non symmetric permutations could be explored when the coefficient matrix is unsymmetric.

Bibliography

[1] Aldous J. and Wilson R. 2000. *Graphs and Applications An Introductory Approach.* Springer.

[2] Allarie G. and Mahmoud S. 2008. *Numerical Linear Algebra.* Springer.

[3] Amodei L. and Dedieu J. 2008. *Analyse Numérique Matricielle.* Dunod.

[4] Arnoldi W. 1951. *The Principle of Minimized Iteration in the Solution of the Matrix Eigenproblem.* Quarterly of Applied Mathematics, Volume 9, pages 17-29.

[5] Ashby S. and Gutknecht M. 1993. *A Matrix Analysis of Conjugate Gradient Algorithms.* Technical Report 113560 Lawrence Livermore National Laboratory.

[6] Aubry R., Mut F., Löhner R. and Cebral J. 2008. *Deflated Preconditioned Conjugate Gradient Solvers for the Pressure-Poisson Equation.* Journal of Computational Physics, Volume 227, Issue 24, pages 10196-10208.

[7] Axelsson O. 1987. *A Generalized Conjugate Gradient, Least Squares Method.* Numerische Mathematik, Volume 51, pages 209-227.

[8] Axelsson O. 1994. *Iterative Solution Methods.* Cambridge University Press.

[9] Barrett R., Berry M., Chan T., Demmel J., Donato J., Dongarra J., Eijkhout V., Pozo R., Romine C. and van der Vorst H. 1994. *Templates for the Solution of Linear Systems Building Blocks for Iterative Methods.* Society for Industrial and Applied Mathematics.

[10] Bathe K 1996. *Finite Element Procedures.* Prentice Hall.

[11] Benzi M. 2002. *Preconditioning Techniques for Large Linear Systems: A Survey.* Journal of Computational Physics, Volume 182, Issue 2, pages 418-477.

[12] Benzi M. and Tůma M. 1998. *A sparse approximate inverse preconditioner for nonsymmetric linear systems.* SIAM Journal on Scientific Computing, Volume 19, Issue 3, pages 968-994.

[13] Benzi M. and Tůma M. 1998. *Numerical Experiments with Two Approximate Inverse Preconditioners.* BIT Numerical Mathematics, Volume 38, Issue 2, pages 234-241

[14] Benzi M. and Tůma M. 1999. *A Comparative Study of Sparse Inverse Preconditioners.* Applied Numerical Mathematics, Volume 30, Issues 2-3, pages 305-340.

[15] Benzi M. and Tůma M. 2003. *A robust incomplete factorization preconditioner for positive definite matrices.* Numerical Linear Algebra with Applications, Volume 10, Issue 5-6, pages 385-400.

[16] Benzi M. and Tůma M. 2006. *Orderings for Factorized Approximate Inverse Preconditioners.* SIAM Journal on Scientific Computing, Volume 21 Issue 5, pages 1851-1868.

[17] Benzi M., Golub G. and Liesen J. 2005. *Numerical solution of saddle point problems.* Acta Numerica, Volume 14, pages 1-137.

[18] Bielawski S., Mulyarchik S. and Popov A. 1996. *The construction of an algebraically reduced system for the acceleration of preconditioned conjugate gradients.* Journal of Computational and Applied Mathematics, Volume 70, Issue 2, pages 189-200.

[19] Bollhöfer M. and Saad Y. 2002. *A Factored Approximate Inverse Preconditioner with Pivoting.* SIAM Journal on Matrix Analysis and Applications, Volume 23(3), pages 692-705.

[20] Brezinski C. and Redivo-Zaglia M. 1998. *Transpose-free Lanczos-type Algorithms for Nonsymmetric Linear Systems.* Numerical Algorithms, Volume 17, pages 67-103.

[21] Broyden C. and Vespucci M. 2004. *Krylov Solvers for Linear Algebraic Systems.* Elsevier.

[22] Bruaset A. 1995. *A Survey of Preconditioned Iterative Methods.* Longman Scientific and Technical.

[23] Bruaset A. and Tveito A. 1992. *RILU preconditioning a computational study.* Journal of Computational and Applied Mathematics, Volume 39, Issue 3, pages 259-275.

[24] Burrage K., Erhel J., Pohl B. and Williams A. 1998. *A Deflation Technique for Linear Systems of Equations.* SIAM Journal on Scientific Computing, Volume 19(4), pages 1245-1260.

[25] Carpentier M., Vuik C., Lucas P., van Gijzen M. and Bijl H. 2010. *A general algorithm for reusing Krylov subspace information. I. Unsteady Navier-Stokes.* NASA Technical Report 216190.

[26] Champan A. and Saad Y. 1997. *Deflated and augmented Krylov subspace techniques.* Numerical Linear Algebra with Applications, Volume 4, Issue 1, pages 43-66.

[27] Chan T., de Pillis L. and van der Vorst H. 1998. *Transpose-Free Formulations Of Lanczos-Type Methods For Nonsymmetric Linear Systems.* Numerical Algorithms, Volume 17, pages 51-66.

[28] Chan T., Gallopoulos E., Simoncini V., Szeto T. and Tong C. 1994. *QMRCGSTAB: A Quasi-Minimal Residual Variant of the Bi-CGSTAB Algorithm for Nonsymmetric Systems.* SIAM Journal on Scientific Computing, Volume 15, No. 2, pages 338-347.

[29] Chandra R. 1981. *Conjugate Gradient Methods for Partial Differential Equations.* PhD Thesis Department of Computer Science, Yale University.

[30] Chen G., Christenson J. and Yang D. 1997. *Application of Preconditioned Transpose Free Quasi Minimal Residual Method for Two Group Reactor Kinetics.* Annals of Nuclear Energy, Volume 24, Issue 5, pages 339-359.

[31] Chen K. 2005. *Matrix Preconditioning Techniques and Applications.* Cambridge University Press.

[32] Chen Y. 1975. *Iterative Methods for Linear Least Squares Problems.* Research Report CS-75-04, Department of Computer Science, University of Waterloo.

[33] Chow E. and Saad Y. 1995. *Approximate Inverse Preconditioners via Sparse Sparse Iterations.* SIAM Journal on Scientific, Volume 19 Issue 3, pages 995-1023.

[34] Coulaud O., Giraud L., Ramet P. and Vasseur X. 2013. *Deflation and augmentation techniques in Krylov subspace methods for the solution of linear systems.* INRIA Research Report No. 8265

[35] Craig E. 1955. *The n-step Iteration Procedure.* Journal of Mathematical Physics, Volume 34 pages 64-73.

[36] Cullum J. and Willoughby R. 2002. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol I Theory.* Society for Industrial and Applied Mathematics.

[37] D'Azevedo E., Forsyth W. and Tang W. 1992. *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems.* SIAM Journal on Matrix Analysis and Applications, Volume 13, pages 944-961 .

[38] Daniel J., Gragg W., Kaufmann L. and Stewart G. 1976. *Reorthogonalization and Stable Algorithms for Updating the Gram-Schmidt QR Factorization.* Mathematics of Computation, Volume 30, pages 772-795.

[39] Darnell D., Morgan R. and Wilcox W. 2008. *Deflated GMRES for systems with multiple shifts and multiple right-hand sides.* Linear Algebra and its Applications, Volume 429 pages 2425-2434.

[40] Davis T. 2006. *Direct Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics.

[41] Demel J. 1996. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics.

[42] Demko S., Moss W. and Smith P. 1984. *Decay Rates for Inverses of Band Matrices.* Mathematics of Computation, Volume 43, No. 168, pages 491-499.

[43] Dongarra J., Duff I., Sorensen D., and van der Vorst H. 1998. *Numerical linear Algebra for High Performance Computers.* Society for Industrial and Applied Mathematics.

[44] Duff I. and Meurant G. 1989. *The effect of ordering on preconditioned conjugate gradients.* BIT Numerical Mathematics, Volume 29, Issue 4, pages 635-657.

[45] Duff I., Erisman M. and Reid J. 1986. *Direct Methods for Sparse Matrices.* Clarendon Press.

[46] Ehrig R. and Deuflhard P. 1997. *GMERR - An Error Minimizing Variant of GMRES.* Technical Report 97-63 Konrad-Zuse-Zentrum für Informationstechnik Berlin.

[47] Eiermann M. and Ernst O. 1999. *Geometric aspects of the theory of Krylov subspace methods.* Acta Numerica, Volume 10 pages 251-312.

[48] Eisenstat S., Elman H. and Schultz M. 1983. *Variational Iterative Methods for Non-Symmetric Systems of Linear Equations.* SIAM Journal on Numerical Analysis Volume 20, pages 354-361.

[49] Eisenstat S., Gursky M., Schultz M. and Sherman A. 1982. *Yale sparse matrix package I: The symmetric codes.* International Journal for Numerical Methods in Engineering, Volume 18, Issue 8, pages 1145-1151.

[50] Engeli M., Ginsburg M., Rutishauser H. and Stiefel E. 1959. *Refined Iterative Methods for the Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems.* Mitteilungen aus dem Institut fr Angewandte Mathematik der ETH Zürich, Nr. 8.

[51] Erhel J. and Guyomarc'h F. 1997. *An Augmented Subspace Conjugate Gradient.* INRIA Research Report No. 3278.

[52] Erhel J., Burrage K. and Pohl B. 1996. *Restarted GMRES Preconditioned by Deflation.* Journal of Computational and Applied Mathematics, Volume 69 Issue 2, pPages 303-318 .

[53] Fletcher R. 1975. *Conjugate Gradient Methods for Indefinite Systems.* In Numencal Analysis Dundee 1975, Edited by Watson G., Lecture Notes in Mathematics, Volume 506. pages 73-89. Springer.

[54] Fokkema D. 1996. *Enhanced Implementation of BICGSTAB(l) for Solving Linear Systems of Equations.* Preprint 976, Department of Mathematics, Utrecht University.

[55] Fokkema D., Sleijpen G. and van der Vorst H. 1996. *Generalized Conjugate Gradient Squared.* Journal of Computational and Applied Mathematics, Volume 71, No. 1, pages 125-146.

[56] Frank J. and Vuik K. 2001. *On the Construction of Deflated Based Preconditioners.* SIAM Journal on Scientific Computing, Volume 23, pages 442-462.

[57] Freund R. 1993. *A Transpose Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems.* SIAM Journal on Scientific Computing, Volume 14, No. 2, pages 470-482.

[58] Freund R. and Nachtigal N. 1992. *QMR: A QuasiMinimal Residual Method for Non-Hermitian Linear systems.* Numerische Mathematik, Volume 60, Issue 1, pages 315-339 .

[59] Freund R. R. and Szeto T. 1992. *A Quasi Minimal Residual Squared Algorithm for Non-Hermitian Linear Systems.* Proceedings of the 1992 Copper Mountain Conference on Iterative Methods.

[60] Fridman V. 1963. *The Method of Minimum Iterations with Minimum Errors for a System of Linear Algebraic Equations with Symmetrical Matrix.* USSR Computational Mathematics and Mathematical Physics. Volume 2 pages 362-363.

[61] Gallopoulos E., Simoncini V., Szeto T., Tong C., Chan T. and Chan T. 1994. *A Quasi-Minimal Residual Variant of the BiCGSTab Algorithm for Nonsymmetric Systems.* SIAM Journal on Scientific Computing, Volume 115, pages 338-347.

[62] George A. and Liu J. 1981. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall.

[63] Golub G. and Concus P. 1976. *A Generalized Conjugate Gradient Method for Non-Symmetric Systems of Linear Equations.* Second International Symposium on Computing Methods in Applied Sciences and Engineering, Versailles 1975, Edited by Glowinski R. and Lions R., Lecture Notes in Economics and Methematical Systems, Volume 134, pages 56-65. Springer.

[64] Golub G. and Meurant G. 1983. *Résolition numérique des grandes syst'emes linéaires.* Editions Eyrolles.

[65] Golub G. and van Loan C. 1996. *Matrix Computations.* The John Hopkins University Press.

[66] Greenbaum A. 1997. *Iterative Methods for Solving Linear Systems.* Society for Industrial and Applied Mathematics.

[67] Guillard H. and Desideri J. 1990. *Iterative methods with spectral preconditioning for elliptic equations.* Computer Methods in Applied Mechanics and Engineering, Volume 80 Issue 1-3, pages 305-312 .

[68] Gustafsson I. 1978. *A class of first order factorization methods.* BIT Numerical Mathematics, Volume 18, Issue 2, pages 142-156.

[69] Gutknecht M. 1993. *Variants of BICGSTAB for Matrices with Complex Spectrum.* SIAM Journal on Scientific Computing, Volume 14, No. 5, pages 1010-1033.

[70] Gutknecht M. 2012. *Spectral Deflation in Krylov Solvers: A Theory of Coordinate Space Based Methods.* Electronic Transaction on Numerical Analysis, Volume 39, pages 156-185.

[71] Hackbusch W. 1994. *Iterative Solution of Large Sparse Systems of Equations.* Springer.

[72] Hageman L. and Young D. 1981. *Applied Iterative Methods.* Academic Press.

[73] Hegedüs C. 1991. *Generating Conjugate Directions for Arbitrary Matrices by Matrix Equations - I.* Computers and Mathematics with Applications, Volume 2l pages 71-85.

[74] Hestenes M. and Stiefel E. 1952. *Methods of Conjugate Gradients for Solving Linear Systems.* Journal of Research of the National Bureau of Standards, Volume 49, No. 6, pages 409-436.

[75] Higham N. 2002. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics.

[76] Jonsthovel T., van Gijzen M., MacLachlan S., Vuik C. and Scarpas A. 2012. *Preconditioned conjugate gradient method enhanced by deflation of rigid body modes applied to composite materials.* Computational Mechanics, Volume 50, pages 321-333.

[77] Joubert W. 1992. *Lanczos Methods for the Solution of Nonsymmetric Systems of Linear Equations.* SIAM Journal on Matrix Analysis and Applications, Volume 13, No. 3, pages 926-943.

[78] Kanzow C. 2000. *Numerik Linearer Gleichungssysteme: Direkte und Iterative Verfahren.* Springer.

[79] Kelley C. 1998. *Iterative Methods for Linear and Nonlinear Equations.* Society for Industrial and Applied Mathematics.

[80] Komzsik L. 2003. *The Lanczos Method.* Society for Industrial and Applied Mathematics.

[81] Kressner D. 2005. *Numerical Methods for General and Structured Eigenvalue Problems.* Springer.

[82] Lanczos C. 1952. *Solution of Systems of Linear Equations by Minimized Iterations.* Journal of Research of the National Bureau of Standards, Volume 49, pages 33-53.

[83] Lehoucq R., Sorensen D. and Yang C. 1998. *ARPACK Solution of Large Eigenvalue Problems with Arnoldi Methods.* Society for Industrial and Applied Mathematics.

[84] Luenberger D. 1967. *Hyperbolic Pairs in the Method of Conjugate Gradients.* SIAM Journal on Applied Mathematics, Volume 17, No. 6, pages 1263-1267

[85] Malandain M. 2013. *Simulation Massivement Parallele des Ecoulements Turbulents a Faible Nombre de Mach.* PhD Thesis L'Institute National des Sciences Appliquées de Rouen.

[86] Malandain M., Maheu N. and Moureau V. 2012. *Optimization of the Deflated Conjugate Gradient Algorithm for the Solving of Elliptic Equations on Massively Parallel Machines.* Journal of Computational Physics, Volume 238, pages 32-47.

[87] Meister A. 2008. *Numerik Linearer Gleichungssysteme.* Vieweg und Sohn.

[88] Meurant G. 1999. *Computer Solution of Large Linear Systems.* Elsevier.

[89] Meurant G. 2006. *The Lanczos and Conjugate Gradient Algorithms From Theory to Finite Precision Computations.* Society for Industrial and Applied Mathematics.

[90] Morgan R and Wilcox W. 2004. *Deflated iterative methods for linear equations with multiple right-hand sides.* arXiv:math-ph/0405053.

[91] Nabben R. 1999. *Decay Rates of the Inverse of Nonsymmetric Tridiagonal and Band Matrices.* SIAM Journal on Matrix Analysis and Applications, Volume 20 Issue 3, pages 820-837.

[92] Nabben R. and Vuik K. 2004. *A comparison of Deflation and Coarse Grid Correction applied to porous media flow.* SIAM Journal on Numerical Analysis, Volume 42, pages 1631-1647.

[93] Nabben R. and Vuik K. 2004. *A comparison of Deflation and the balancing Neumann-Neumann preconditioner.* Reports of the Department of Applied Mathematics, 04-09, Delft University of Technology.

[94] Nabben R. and Vuik K. 2006. *A Comparison of Deflation and the Balancing Preconditioner.* SIAM Journal on Scientific Computing, Volume 27, Number 5, pages 1742-1759.

[95] Nabben R. and Vuik K. 2008. *A comparison of abstract versions of deflation balancing and additive coarse grid correction preconditioners.* Numerical Linear Algebra with Applications, Volume 15 pages 355-372.

[96] Nicolaides R. 1987. *Deflation of conjugate gradients with applications to boundary value problems.* SIAM Journal on Numerical Analysis, Volume 24, No. 2, pages 355-365.

[97] Ortega J. 1989. *Introduction to parallel and vector solution of linear systems.* Plenum Press.

[98] Østerby O. and Zlatev Z. 1983. *Direct Methods for Sparse Matrices.* Springer.

[99] Paige C. and Saunders M. 1975. *Solution of Sparse Indefinite Systems of Linear Equations.* SIAM Journal on Numerical Analysis, Volume 12, No. 4, pages 617-629.

[100] Paige C. and Saunders M. 1982. *Algorithm 583 LSQR Sparse Linear Equations and Least Squares Problems.* ACM Transactions on Mathematical Software, Volume 8, No. 2, pages 195-209.

[101] Pissanestsky S. 1994. *Sparse Matrix Technology.* Academic Press.

[102] Poole G. and Neal L. 2000. *The Rook pivoting strategy.* Journal of Computational and Applied Mathematics, Volume 123, Issues 1-2, pages 353-369.

[103] Quarteroni A., Sacco R. and Saleri F. 2007. *Méthodes Numériques.* Springer.

[104] Robert Y. 1990. *The impact of vector and parallel architectures on the Gaussian elimination algorithm.* Manchester University Press.

[105] Saad Y. 1981. *Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems.* Mathematics of Computation, Volume 37, pages 105-126.

[106] Saad Y. 1994. *ILUT: A Dual Threshold Incomplete LU Factorization.* Numerical Linear Algebra with Applications, Volume 1, Issue 4, pages 387-402.

[107] Saad Y. 2000. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics.

[108] Saad Y. 2011. *Numerical Methods for Large Eigenvalue Problems.* Society for Industrial and Applied Mathematics.

[109] Saad Y. and Schultz M. 1986. *GMRES A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems.* SIAM Journal on Scientific and Statistical Computing, Volume 7, Issue 3, pages 856-869.

[110] Saad Y. and Wu K. 1996. *DQGMRES: a Direct Quasi Minimal Residual Algorithm Based on Incomplete Orthogonalization.* Numerical Linear Algebra with Applications, Volume 3, Issue 4, pages 329-343.

[111] Saunders M., Simon H. and Yip E. 1988. *Two Conjugate-Gradiente-Type Methods for Unsymmetric Linear Equations* SIAM Journal on Numerical Analysis, Volume 25, No. 4, pages 927-940.

[112] Segal A., ur Rehman M. and Vuik K. 2010. *Preconditioners for Incompressible Navier-Stokes Solvers.* Numerical Mathematics: Theory, Methods and Applications, Volume 3 Issue 3, page 245.

[113] Shores T. 2007. *Applied Linear Algebra and Matrix Analysis.* Springer.

[114] Sleijpen G. and Fokkema D. 1993. *BICGSTAB(L) for Linear Equations Involving Unsymmetric Matrices with Complex Spectrum.* Electronic Transactions on Numerical Analysis, Volume 1, pages 11-32.

[115] Sleijpen G., van der Vorst H. and Fokkema D. 1994. *Bi-CGSTAB(l) and Other Hybrid Bi-CG Methods.* Numerical algorithms, Volume 7, pages 75-109.

[116] Sonneveld P. 1989. *CGS: A Fast Lanczos-type Solver for Nonsymmetric Linear Systems.* SIAM Journal on Scientific and Statistical Computing, Volume 10, pages 36-52.

[117] Stewart G. 1973. *Introduction to matrix computations.* Academic Press.

[118] Stewart G. 1998. *Matrix Algorithms Vol 1 Basic Decompositions.* Society for Industrial and Applied Mathematics.

[119] Stewart G. 2001. *Matrix Algorithms Vol 2 Eigensystems.* Society for Industrial and Applied Mathematics.

[120] Stiefel E. 1955. *Relaxationsmethoden Bester Strategie zur Losung Linearer Gleichungssysteme.* Commentarii Mathematici Helvetici, Volume 29 pages 157-179.

[121] Strang G. 2003. *Introduction to Linear Algebra.* Wellesley-Cambridge Press.

[122] Tang J. 2008. *Two level preconditioned conjugate gradient methods with applications to bubbly flow problems.* PhD Thesis Delft University of Technology.

[123] Tang J. and Vuik K. 2007. *Efficient Deflation Methods applied to 3-D Bubbly Flow Problems.* Electronic Transactions on Numerical Analysis, Volume 26, pages 330-349.

[124] Tang J. and Vuik K. 2007. *New Variants of Deflation Techniques for Pressure Correction in Bubbly Flow Problems.* Journal of Numerical Analysis, Industrial and Applied Mathematics, Volume 2, pages 227-249.

[125] Tang J. and Vuik K. 2007. *On deflation and singular symmetric positive semi-definite matrices.* Journal of Computational and Applied Mathematics, Volume 206, Issue 2, pages 603-614.

[126] Tang J. and Vuik K. 2008. *Acceleration of preconditioned Krylov solvers for bubbly flow problems.* Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, Volume 4967, pages 1323-1332. Springer.

[127] Tang J. and Vuik K. 2008. *Fast deflation methods with applications to two-phase flows.* Delft Institute of Applied Mathematics Report 07-10. Delft University of Technology

[128] Tang J., MacLachlan S., Nabben R. and Vuik K. 2010. *A Comparison of Two-Level Preconditioners Based on Multigrid and Deflation.* SIAM Journal on Matrix Analysis and Applications, Volume 31, pages 1715-1739.

[129] Tang J., Nabben R., Vuik K. and Erlangga Y. 2007. *Theoretical and numerical comparison of various projection methods derived from deflation domain decomposition and multigrid methods.* Reports of the Department of Applied Mathematical Analysis, 07-04, Delft University of Technology.

[130] Tang J., Nabben R., Vuik K. and Erlangga Y. 2009. *Comparison of Two-Level Preconditioners Derived from Deflation Domain Decomposition and Multigrid Methods.* Journal of Scientific Computing, Volume 39, pages 340-370.

[131] Tewarson R. 1973. *Sparse Matrices.* Academic Press.

[132] Trefethen L. and Bau D. 1997. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics.

[133] ur Rehman M., Vuik K. and Segal A. 2006. *Solution of the incompressible Navier Stokes equations with preconditioned Krylov subspace methods.* Reports of the Department of Applied Mathematical Analysis Delft University 06-05.

[134] ur Rehman M., Vuik K. and Segal A. 2007. *A comparison of preconditioners for incompressible Navier-Stokes solvers.* International Journal for Numerical Methods in Fluids, Volume 57 pages 1731-1751.

[135] van der Vorst H. 1992. *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-symmetric Linear Systems.* SIAM Journal on Scientific and Statistical Computing, Volume 13, pages 631-644.

[136] van der Vorst H. 2003. *Iterative Krylov Methods for Large Linear Systems.* Cambridge University Press.

[137] van der Vorst H. and Melissen J. 1990. *A Petrov-Galerkin Type Method for Solving $Ax = b$, where A is Symmetric Complex.* IEEE Transactions on Magnetics, Volume 26, No. 2, pages 706-708.

[138] Varga R. 1963. *Matrix Iterative Analysis.* Prentice Hall.

[139] Verkaik J., Vuik K., Paarhuis B. and Twerda A. 2005. *The Deflation Accelerated Schwarz Method for CFD.* ICCS 2005 Proceedings of the 5th international conference on Computational Science - Volume Part I, pages 868-875, Springer.

[140] Vermolen F. and Vuik K. 2001. *The Influence of Deflation Vectors at Interfaces on the Deflated Conjugate Gradient Method.* Reports of the Department of Applied Mathematical Analysis, 01-13, Delft University of Technology.

[141] Vermolen F., Vuik K. and Segal A. 2004. *Deflation in Preconditioned Conjugate Gradient Methods for Finite Elements Problems.* In: Krizek M., Neittaanmaki

P., Glowinski R. and Korotov S. (eds.): Conjugate Gradient and Finite Element Methods, pages 103-129, Springer.

[142] Vuik K. and Frank J. 2000. *Deflated Incomplete Cholesky Conjugate Gradient Methods Applied to Problems with Extreme Contrats in the Coefficients.* In: M. Deville and R. Owens (eds.): Proceedings of the 16th IMACS World Congress Lausanne.

[143] Vuik K., Segal A. and Meijerink K. 1998. *An Efficient Conjugate Gradient Method for Layered Problems with Large Contrasts in the Coefficients.* Presented at Copper Mountain Conference on Iterative Methods.

[144] Vuik K., Segal A., Meijerink K. and Wijma G. 2001. *The Construction of Projection Vectors for a Deflated Incomplete Cholesky Conjugate Gradient Method Applied to Problems with Extreme Contrasts in the Coefficients.* Journal of Computational Physics, Volume 172, pages 426-450.

[145] Vuik K., Segal A., Yaakoubi L. and Dufour E. 2002. *A Comparison of Various Deflation Vectors Applied to Elliptic Problems with Discontinuous Coefficients.* Applied Numerical Mathematics, Volume 41, pages 219-233.

[146] Walker F. and Zhou L. 1994. *A simpler GMRES.* Numerical Linear Algebra with Applications, Volume 1, pages 571-581.

[147] Walker H. 1988. *Implementation of the GMRES Method using Householder Transformations.* SIAM Journal on Scientific and Statistical Computing, Volume 9, pages 152-163.

[148] Watkins D. 2002. *Fundamentals of Matrix Computations.* John Willey & Sons, Inc.

[149] Weiss R. 1994. *Error-Minimizing Krylov Subspace Methods.* SIAM Journal on Scientific and Statistical Computing, Volume 15, pages 511-527.

[150] Wilkinson J. 1988. *The Algebraic Eigenvalue Problem.* Oxford University Press.

[151] Wout E., van Gijzen M., Ditzel A., van der Ploeg A. and Vuik K. 2010. *The Deflated Relaxed Incomplete Cholesky CG method for use in a real-time ship simulator.* Procedia Computer Science ICCS 2010, Volume 1, Issue 1, pages 249-257

[152] Young D. and Jea K. 1980. *Generalized Conjugate-Gradient Acceleration of Nonsymmetrizable Iterative Methods.* Linear Algebra and its Applications, Volume 34, pages 159-194.

[153] Zhang S. 1997. *GP-BiCG: Generalized Product Type Methods Based on Bi-CG for Solving Nonsymmetric Linear Systems.* SIAM Journal on Scientific Computing, Volume 18, No. 2, pages 537-551.